6 ИЗДАНИЕ / **МИРОВОЙ БЕСТСЕЛЛЕР**

КОМПЬЮТЕРНЫЕ СЕТИ

НАСТОЛЬНАЯ КНИГА СИСТЕМНОГО АДМИНИСТРАТОРА

ДЖЕЙМС КУРОУЗ
КИТ РОСС

Всемирно известная книга, которая уже более 15 лет возглавляет рейтинги продаж по всему миру и пережила 6 переизданий. Несмотря на это, она не потеряла своей актуальности и продолжает оставаться незаменимым источником знаний для тех, чья работа связана с администрированием и защитой компьютерных сетей. Это прекрасное учебное пособие для студентов технических вузов.

ЧТО ВНУТРИ:

КАК ФУНКЦИОНИРУЮТ ИНТЕРНЕТ И ЛОКАЛЬНЫЕ СЕТИ?

КАК РАБОТАЮТ СЕТЕВЫЕ ПРОТОКОЛЫ И СЛУЖБЫ?

АЛГОРИТМЫ МАРШРУТИЗАЦИИ

СЕТЕВАЯ БЕЗОПАСНОСТЬ И ОСНОВЫ КРИПТОГРАФИИ

СЕТЕВОЕ АДМИНИСТРИРОВАНИЕ

«Принцип нисходящего изложения формирует перед читателем целостную картину – от общих принципов сетей и технологий до их технической и практической реализации. Различные примеры делают чтение интересным настолько, насколько это возможно для технической литературы

Дмитрий Кондратьев, ди
по бизнес-администрированию издательства «

ISBN 978-5-699-94358-6

Компьютерные сети

Нисходящий подход

6-е издание

ДЖЕЙМС КУРОУЗ | КИТ РОСС

Принципы работы модели OSI
Обеспечение сетевой безопасности
Беспроводные и мультимедийные (сетевые) технологии

Компьютерные сети

Нисходящий подход

JAMES KUROSE | KEITH ROSS

Computer Networking

A Top-Down Approach

sixth edition

ДЖЕЙМС КУРОУЗ | КИТ РОСС

Компьютерные сети

Нисходящий подход

6-е издание

Москва
2016

УДК 004.45
ББК 32.973-018.2
К93

James F. Kurose
Keith W. Ross
Computer Networking: A Top-Down Approach
Authorized translation from the English language edition, entitled
COMPUTER NETWORKING; A TOP-DOWN APPROACH, 6th Edition;
ISBN 0132856204 by KUROSE, JAMES F.; and by ROSS, KEITH W.;
published by Pearson Education, Inc, publishing as Addison-Wesley.

Куроуз, Джеймс.

К93 Компьютерные сети : Нисходящий подход / Джеймс Куроуз, Кит Росс. — 6-е изд. — Москва : Издательство «Э», 2016. — 912 с. — (Мировой компьютерный бестселлер).

ISBN 978-5-699-78090-7

Книга знакомит читателя с фундаментальными основами построения и функционирования компьютерных сетей на примере пятиуровневой архитектуры сети Интернет. Описаны базовые компоненты компьютерной сети, ключевые подходы к передаче данных в телекоммуникационных сетях, принципы взаимодействия сетей друг с другом, подробно рассмотрены важнейшие службы и протоколы всех уровней сетевой архитектуры. Отдельная глава посвящена беспроводным и мобильным сетям и их особенностям. Большое внимание уделено одной из самых развивающихся сегодня областей – мультимедийным сетевым технологиям, в частности, специфике передачи аудио- и видеоданных. Будут затронуты важные аспекты сетевой безопасности и разнообразные принципы, методы и приемы, обеспечивающие безопасный обмен информацией.

Весь материал книги снабжен интересными примерами, кроме того читателю доступны дополнительные материалы для выполнения упражнений.

Книга будет полезна всем, кто специализируется в области технологий компьютерных сетей — от студентов до системных администраторов.

УДК 004.45
ББК 32.973-018.2

ISBN 978-5-699-78090-7

ОГЛАВЛЕНИЕ

Джулии и трем нашим чудесным детям — Крису, Чарли и Нине.

Джеймс Куроуз

Большое СПАСИБО моим преподавателям,
коллегам и студентам со всего мира.

Кит Росс

ПРЕДИСЛОВИЕ

Перед вами шестое издание книги «Компьютерные сети. Нисходящий подход». Первое вышло более десятилетия назад, с тех пор книга была адаптирована для изучения в сотнях колледжей и университетов, переведена на 14 языков и проработана более чем ста тысячами студентов и специалистов во всем мире. Мы получили от этих читателей массу положительных отзывов, многие из которых очень нас воодушевили.

Что нового в данном издании

Полагаем, успех книги объясняется не в последнюю очередь тем, что в каждом издании она обновляется и всегда предлагает свежий и актуальный материал для изучения компьютерных сетей. Мы внесли определенные изменения в шестое издание, но также сохранили в нем те аспекты, которые кажутся нам (а также преподавателям и студентам, работавшим с предыдущими версиями) наиболее важными. Среди таких черт следует отметить нисходящий метод изложения материала, акцент на Интернете и современном понимании компьютерных сетей, внимание как к теоретическим, так и к практическим принципам, академический и в то же время доступный стиль изложения материала. Тем не менее данное издание было существенно переработано и дополнено.

- В главе 1 был модернизирован материал о сетях доступа, существенно пересмотрено описание экосистемы интернет-провайдеров. Учтены недавно появившиеся сети доставки контента — например, такой сетью располагает компания Google. Мы существенно реорганизовали текст о коммутации каналов и пакетов, акцентировав не исторические, а тематические аспекты этого материала.
- В главе 2, где речь идет о программировании сокетов, вместо Java использован язык Python. Мы по-прежнему явно описываем базовые идеи, лежащие в основе API-сокетов, но код Python будет лучше понятен программистам-новичкам, чем код Java. Кроме того, Python, в отличие от Java, предоставляет доступ к сырым (простым) сокетам,

благодаря чему студенты могут программировать самые разные сетевые приложения. Лабораторные работы по сокетам, где использовался язык Java, были заменены соответствующими работами с применением языка Python; кроме того, появилась новая лабораторная работа с Python, в которой исследуется программа ICMP Ping. Как обычно, если материал исключается из книги (в данном случае, речь идет о лабораторных работах по программированию сокетов на языке Java), он доступен в электронном виде на сайте (подробнее об этом ниже).

- В главе 3 мы упростили описание протоколов надежной передачи данных, а также добавили новую врезку о разделении TCP-соединений — данная технология широко применяется для оптимизации производительности облачных сервисов.
- В главе 4 был существенно дополнен раздел об архитектурах маршрутизаторов, в нем мы постарались отразить последние разработки и практики из этой области. В главе добавились несколько новых интегративных врезок, где рассматриваются технологии DNS, BGP и OSPF
- Глава 5 была реорганизована и «причесана»; в ней мы учли повсеместное распространение коммутируемых Ethernet-соединений в локальных сетях и, как следствие, все более активное применение технологии Ethernet в двухточечных сценариях. Кроме того, в этой главе появился новый раздел о сетях для дата-центров.
- Глава 6 была обновлена с учетом новейших разработок в области беспроводных сетей, в частности, сотовых сетей, а также сервисов и архитектуры 4G.
- Глава 7, рассказывающая о мультимедийных сетях, была значительно переработана. Теперь в ней вы найдете подробное обсуждение потокового видео; в частности, мы поговорим об адаптивном потоковом вещании. Кроме того, в главе появился совершенно новый раздел о сетях доставки контента (CDN). Также мы поговорим о системах потокового видео Netflix, YouTube и Kankan. Материал, от которого пришлось отказаться в пользу этих новых тем, по-прежнему доступен на сайте.
- В главе 8 появилось подробное обсуждение аутентификации конечной точки
- Значительно обогатился материал в заданиях, приведенных в конце каждой главы. Как и в предыдущих изданиях, некоторые упражнения для самостоятельной работы были пересмотрены, добавлены или удалены.

Целевая аудитория

Данная книга предназначена для изучения на вводном курсе по компьютерным сетям. Ее можно использовать как на факультете информатики, так и на электротехническом. Что касается языков программирования, читатель должен иметь представление о C, C++, Java или Python (эти знания потребуются всего в нескольких разделах). Хотя эта книга более имеет более детальный и аналитический характер, чем другие вводные тексты по компьютерным технологиям, в ней почти не используются математические концепции, выходящие за рамки курса старших классов. Мы специально старались обходиться без сложного математического анализа, теории вероятности или стохастических процессов (хотя и даем несколько таких заданий для студентов, разбирающихся в этих темах). Соответственно, книга подходит для изучения на старших курсах университета и на первых курсах аспирантуры. Кроме того, она должна быть полезна для практикующих специалистов, работающих в сфере телекоммуникаций.

Что особенного в этой книге?

Тема компьютерных сетей исключительно сложна. Она включает в себя множество концепций, протоколов и технологий, которые переплетены воедино самым нетривиальным образом. Поэтому, чтобы было легче освоить столь непростую и объемную информацию, многие тексты о компьютерных сетях структурированы в порядке изучения отдельных уровней сетевой архитектуры. Такая организация помогает студентам оценить сложность данной тематики. Слушатели курса изучают концепции и протоколы, применяемые на конкретном уровне архитектуры, но в то же время видят и общую картину — как отдельные уровни подогнаны друг к другу. С педагогической точки зрения должны отметить, что, по нашему опыту, такой многоуровневый подход действительно замечательно воспринимается. С другой стороны, мы полагаем, что традиционный восходящий способ изучения этой темы — от физического уровня к прикладному, то есть снизу вверх — не слишком подходит для современного курса по компьютерным сетям.

Нисходящий подход

Когда мы выпустили первое издание этой книги 12 лет назад, мы применили новый на тот момент подход, рассказав о компьютерных се-

тях по нисходящему принципу — то есть начиная с прикладного уровня и рассматривая стек по направлению к нижнему физическому уровню. Отклики, которые мы получили от преподавателей и студентов, убедили нас, что такой нисходящий подход действительно обладает массой преимуществ и очень удобен с педагогической точки зрения. Во-первых, сразу делается акцент на прикладном уровне (который является в сфере сетевых технологий своеобразной «зоной роста»). Действительно, многие из недавних революционных изменений в компьютерных сетях — в частности, появление Всемирной Паутины, одноранговых сетей для обмена файлами, а также потоковых медиа — произошли именно на прикладном уровне. Его изучение на раннем этапе курса — необычный подход. В большинстве книг о компьютерных сетях прикладному уровню посвящается небольшой раздел, где вкратце рассказывается о сетевых приложениях, предъявляемых к ним требованиях, парадигмах сетевого уровня (в частности, о клиент-серверной и об одноранговой), а также об интерфейсах программирования приложений. Во-вторых, по нашему преподавательскому опыту (а также по опыту многих наших коллег, работавших с этим текстом), изучение сетевых приложений в начале книги сильно мотивирует студентов. Им не терпится узнать, как именно работают сетевые приложения — например, электронная почта и всемирная паутина, — так как большинство слушателей курса пользуются ими практически ежедневно. Когда студент поймет приложения, он сможет понять и сетевые сервисы, необходимые для поддержки этих приложений. В свою очередь, сами студенты могут исследовать несколько способов предоставления и реализации таких сервисов на более низких уровнях. В итоге изучение уровня приложений на раннем этапе оказывается очень полезным для понимания всего последующего текста.

В-третьих, нисходящий подход позволяет преподавателю на раннем этапе познакомить студентов с разработкой сетевых приложений. Слушатели не только наблюдают популярные приложения и протоколы в действии, но и узнают, насколько просто создавать собственные сетевые приложения и протоколы сетевого уровня. Описываемый подход позволяет студентам на самых ранних этапах освоить принципы программирования сокетов, модели обслуживания и протоколы. Все эти концепции очень важны, и к ним приходится регулярно обращаться при изучении всех последующих уровней. Давая упражнения по программированию сокетов на языке Python, мы подчеркиваем основные идеи, не запутывая студента сложными примерами с кодом. Студенты старших курсов, обучающиеся на электротехнических и кибернетических специальностях, должны без труда воспринимать код на Python.

Интернет как центральная тема книги

Хотя мы и убрали слово «Интернет» из названия книги еще в четвертом издании, это вовсе не означает, что данная тема стала интересовать нас меньше. Нет, совершенно наоборот! Действительно, поскольку Интернет сегодня проникает буквально повсюду, мы полагаем, что в любой книге о компьютерных сетях ему должно быть уделено пристальное внимание, поэтому упоминание Интернета в заголовке становится излишним. Мы продолжаем рассматривать архитектуру и протоколы Интернета как основной материал для изучения фундаментальных концепций в сфере компьютерных сетей. Разумеется, в книге также затрагиваются концепции и протоколы из других сетевых архитектур. Но в центре внимания остается Интернет, поэтому книга композиционно строится вокруг именно тех пяти сетевых уровней, которые в нем присутствуют. Речь пойдет о прикладном, транспортном, сетевом, канальном и физическом уровнях.

Немаловажно и то, что большинство студентов с факультетов информатики и электротехнических факультетов жаждут подробнее изучить Интернет и его протоколы. Они знают, что это революционная и прорывная технология, которая прямо на наших глазах коренным образом меняет мир. Учитывая огромное значение Интернета, слушатели, естественно, желают знать и понимать его внутреннее строение. Соответственно, преподавателю также не составляет труда научить студентов всем основополагающим принципам, если они рассматриваются на основе Интернета.

Обучение принципам функционирования сетей

Две уникальные черты книги — нисходящий подход к изложению материала и акцент на Интернете — фигурировали в заголовках ее изданий. Если бы мы могли вместить в подзаголовок и *третью* фразу, то, вероятно, в ней было бы слово «принципы». Сфера компьютерных сетей уже достаточно зрелая, в ней можно выделить ряд фундаментальных проблем. Например, на транспортном уровне примерами таких проблем являются: обеспечение надежной коммуникации на базе ненадежного сетевого уровня, установление/разрыв соединения и рукопожатие, управление перегрузками и потоком данных, а также мультиплексирование. Две фундаментальных проблемы сетевого уровня — это нахожде-

ние «хороших» путей между двумя маршрутизаторами и взаимное соединение большого количества разнородных сетей. Фундаментальная проблема канального уровня — это совместное использование канала для множественного доступа. Говоря о сетевой безопасности, необходимо отметить различные приемы обеспечения конфиденциальности, аутентификации и целостности сообщения — в основе всех этих аспектов лежат фундаментальные криптографические принципы. В тексте рассматриваются основополагающие сетевые проблемы и исследования, направленные на их разрешение. Читатель, изучающий эти материалы, приобретает знания с большим «сроком годности». Современные сетевые стандарты и протоколы когда-нибудь устареют, но принципы, которые в них воплощаются, останутся актуальными и важными. Мы полагаем, что методическая комбинация изучения Интернета с рассмотрением фундаментальных проблем и их решений не только позволяет заполнить аудиторию любознательными студентами, но и помогает слушателям быстро понять практически любую сетевую технологию.

Примеры к книге

Читателю предоставляется доступ к сайту с дополнительными материалами к книге по адресу http://eksmo.ru/upload/Networks_Primers.rar. В загруженном архиве вы найдете:

- *Ссылки на Java-апплеты*. Вы найдете файл со ссылками на Java-апплеты, необходимыми для выполнения упражнений из книги.
- *Презентации PowerPoint*. Для всех девяти глав мы подготовили презентации PowerPoint. В 6-м издании все слайды обновлены. В них подробно рассматривается материал каждой главы. На слайдах применяется графика и анимация (а не только скучные маркированные списки), что помогает сделать материал более интересным и визуально привлекательным. Мы предоставляем оригиналы презентаций, чтобы вы могли дорабатывать их в соответствии с деталями вашего курса. Некоторые слайды были предоставлены другими преподавателями, читавшими нашу книгу.
- *Решения задач для самостоятельной работы*. Мы предоставляем справочник с решениями к задачам для самостоятельной работы, заданиям по программированию и лабораторным работам Wireshark. Как было указано выше, в первых пяти главах книги мы добавили множество новых задач.

- *Лабораторные работы (Java и Python).* Для более полного понимания сетевых протоколов очень важно рассмотреть их на практике. На сайте есть ряд заданий по работе с программой Wireshark — они помогут студенту непосредственно наблюдать последовательность сообщений, которыми обмениваются два устройства, взаимодействующие по протоколу. На сайте вы найдете лабораторные работы Wireshark по протоколам HTTP, DNS, TCP, UDP, IP, ICMP, Ethernet, ARP, Wi-Fi, SSL и по трассировке всех протоколов, участвующих в удовлетворении запроса по получению веб-страницы.

Кроме того, по отдельным ссылкам, указанным в книге, вы получите доступ к дополнительным материалам, необходимым для выполнения упражнений.

Педагогические аспекты

Оба автора посвятили преподаванию теории компьютерных сетей более чем по 20 лет каждый. Вместе мы вкладываем в эту книгу более 50 лет педагогического опыта, за которые мы успели обучить тысячи студентов. На протяжении всего этого времени мы активно занимались исследованиями в области компьютерных технологий (Джим и Кит познакомились еще в 1979 году, когда учились в магистратуре по курсу компьютерных сетей у Миши Шварца). Полагаем, такой опыт позволяет нам уверенно судить о состоянии и перспективах развития компьютерных сетей. Тем не менее мы старались не переполнять эту книгу материалами, которые были бы связаны с нашими собственными исследовательскими проектами. Если они вас интересуют, вы можете ознакомиться с ними на персональном сайте каждого из авторов. Итак, вы держите в руках книгу о современных компьютерных сетях — технологиях и протоколах, а также о базовых принципах, лежащих в их основе. Мы также уверены, что курс компьютерных сетей может быть увлекательным как для студента, так и для преподавателя. Надеемся, что юмор, живые аналогии и примеры из жизни, которые встретятся вам на страницах этой книги, сделают материал значительно интереснее.

Взаимосвязи между главами

Первая глава этой книги представляет собой самодостаточный обзор современных компьютерных сетей. В ней вы познакомитесь с многочис-

ленными ключевыми концепциями и терминологией, она задает контекст для остальных глав. Все они непосредственно связаны с первой. Завершив работу над главой 1, рекомендуем преподавателям последовательно изучить главы 2–5, придерживаясь избранного нами нисходящего подхода. Каждая из этих глав строится на материале предыдущих. Закончив первые пять глав, преподаватель сможет достаточно гибко строить последующий курс. Между последними четырьмя главами нет взаимных зависимостей, их можно изучать в любом порядке. Однако содержание каждой из этих глав строится на материале первых пяти. Многие преподаватели сначала разбирают первые пять глав, оставляя «на закуску» одну из последних четырех.

Благодарности

Мы начали работу над этой книгой еще в 1996 году, и с тех пор многие люди оказали нам неоценимую помощь, дали массу ценных рекомендаций о том, как лучше всего организовать и читать университетский курс о компьютерных сетях. Мы хотим сказать БОЛЬШОЕ СПАСИБО всем, кто помог нам — от вычитки первых черновиков вплоть до пятого издания. Мы также *очень* благодарны сотням читателей со всего мира — студентам, преподавателям, специалистам, — которые присылали нам комментарии о первых изданиях книги и пожелания относительно новых изданий. Отдельно хотелось бы поблагодарить следующих людей:

Ави Рубин (Университет Джона Хопкинса), Аль Ахо (Колумбийский университет), Альберт Хуань (бывший студент Пенсильванского университета), Ардаш Сетхи (Университет штата Делавэр), Арнод Легу (INRIA), Бен Кю Чой (Мичиганский технологический университет), Бешан Кулапала (Аризонский государственный университет), Боб Меткалф (International Data Group), Бобби Бхаттачарджи (Университет штата Мэриленд), Боджи Шу (бывший студент Нью-Йоркского политехнического института), Брайан Левин (Массачусетский университет), Брэм Коэн (компания BitTorrent. Inc), Брюс Харви (Флоридский аграрно-технический университет, Флоридский государственный университет), Ван Якобсон (компания Xerox PARC), Верн Пэксзон (Калифорнийский университет, Беркли), Винтон Серф (компания Google), Ву-Чи Фэнь (Орегонский институт последипломного образования), Вэнь Син (Университет Парк), Георгий Полизос (Афинский университет экономики и бизнеса), Дебора Эстрин (Калифорнийский университет, Лос-Анджелес), Деспина Сапарилья (компания Cisco Systems),

Дж. Дж. Гарсия-Луна-Асевес (Калифорнийский университет, Санта-Круз), Дженни Мойер (компания Comcast), Дженнифер Рексфорд (Принстонский университет), Джефф Кейз (исследовательская группа SNMP Research International), Джефф Чолтас (компания Sprint), Джитендра Падхье (компания Microsoft Research), Джобин Джеймс (Калифорнийский университет, Риверсайд), Джон Дейгл (Университет штата Миссисипи), Джон Шанц (компания Comcast), Джош Мак-Кинзи (Университет Парк), Ди Ву (Университет Сунь Ятсена), Дип Медхи (Миссурийский университет, Канзас-Сити), Дон Таусли (Массачусетский университет), Дуглас Сэлейн (Колледж Джона Джея), Дэвид Гудмен (Политехнический институт штата Нью-Йорк), Дэвид Коц (Дартмутский колледж), Дэвид Тернер (Калифорнийский государственный университет, Сан-Бернардино), Дэвид Уэзеролл (Университет штата Вашингтон), Дэн Рубенштайн (Колумбийский университет), Дэниэл Бруштейн (ранее — студент Пенсильванского университета), Жан Боло (компания Technicolor Research), Ира Уинстон (Пенсильванский университет), Йехиам Йемини (Колумбийский университет), К. Сэм Шэнмуган (Канзасский университет), Карл Хаузер (Вашингтонский государственный университет), Кевин Филлипс (компания Sprint), Кен Колверт (Университет штата Кентукки), Кен Рик (Рочестерский технологический институт), Кин Сун Там (Государственный университет штата Нью-Йорк, Олбани), Клей Шилдс (Университет Джорджтауна), Клифф К. Зу (Университет Центральной Флориды), Константин Кутрас (Университет Пейс), Крейг Партридж (компания BBN Technologies), Кристоф Дио (компания Technicolor Research), Леон Резник (Рочестерский технологический институт), Леонард Клейнрок (Калифорнийский университет, Лос-Анджелес), Ли Лейтнер (Университет Дрекселя), Ликсинь Гао (Университет штата Массачусетс), Ликсья Чжан (Калифорнийский университет, Лос-Анджелес), Личунь Бао (Калифорнийский университет, Ирвин), Макс Хальперин (Колледж Густава Адольфа), Марио Герла (Калифорнийский университет, Лос-Анджелес), Мартин Рейслейн (Аризонский государственный университет), Мигель А. Лабрадор (Университет Южной Флориды), Минь Ю (Государственный университет штата Нью-Йорк, Бингемптон), Михаил Л. Сихитиу (Государственный университет Северной Каролины), Михалис Фалоустос (Калифорнийский университет, Риверсайд), Миша Швартц (Колумбийский университет), Мишель Уэйгл (Клемсонский университет), Мэнь Чжан (бывший студент Нью-Йоркского политехнического института), Ник Мак-Кеоун (Стэндфордский университет), Нитин Вайдья (Университет штата Иллинойс), Пабло Родригес (ком-

пания Telefonica), Парвиз Кермани (бывший сотрудник компании IBM Research), Пинак Джайн (бывший студент Нью-Йоркского политехнического института), Питер Стинкисте (Университет Карнеги-Меллона), Пол Бэрфорд (Университет штата Висконсин), Пол Френсис (Институт Макса Планка), Пол Эймер (Университет штата Делавэр), Правин Бхагват (компания Wibhu), Пратима Аккунур (Аризонский государственный университет), Прашант Шеной (Массачусетский университет), Притхула Дхунгхель (компания Akamai), Радж Яваткар (компания Intel), Радья Перлман (компания Intel), Ракеш Кумар (компания Bloomberg), Рамачандран Рамджи (компания Microsoft Research), Рашель Хеллер (Университет Джорджа Вашингтона), Саймон Лэм (Университет Техаса), Салли Флойд (ICIR, Калифорнийский университет, Беркли), Самит Рой (Университет штата Вашингтон), Снеха Касера (Университет штата Юта), Стив Лай (Государственный университет штата Огайо), Стивен Белловин (Колумбийский университет), Субин Шестра (Пенсильванский университет), Сугих Джамин (Мичиганский университет), Супратик Бхаттачарья (ранее — компания Sprint), Сью Мун (научно-исследовательский институт KAIST), Сяодун Чжан (Государственный университет штата Огайо), Тацуя Суда (Калифорнийский университет, Ирвин), Тим Бернерс-Ли (Консорциум Всемирной Паутины), Тим Гриффин (Кембриджский университет), Том Ла-Порта (Пенсильванский государственный университет), Уиллис Марти (Техасский аграрно-технический университет), Уильям Лянь (бывший студент Пенсильванского университета), Фил Циммерман (независимый консультант), Филипп Хошка (компания INRIA/W3C), Филиппе Декуэтос (Институт Eurecom), Хариш Сетху (Университет Дрекселя), Хеннинг Шульцринне (Колумбийский университет), Хийоджин Ким (бывшая студентка Пенсильванского университета), Хишам Аль-Мубайд (Хьюстонский университет — Клир Лейк), Хонгган Чжан (Саффолкский университет), Христос Пападопулос (Государственный университет Колорадо), Ху Чжан (Университет Карнеги-Меллона), Чжи Ли Чжан (Университет штата Миннесота), Чунчунь Ли (бывший студент Нью-Йоркского политехнического института), Чэнь Хуань (компания Microsoft Research), Шамуил Эйзом (Аризонский государственный университет), Шахид Бохари (Инженерно-технологический университет Лахора), Шивкумар Кальянараман (компания IBM Research, Индия), Ширли Уинн (Нью-Йоркский политехнический институт), Шрирам Раджагопалан (Аризонский государственный университет), Шучун Чжан (бывший студент Пенсильванского университета), Эвандро Канту (Федеральный университет Санта-Катарины), Эдмундо А. де Соуза

э Сильва (Федеральный университет Рио-де-Жанейро), Эллен Зегура (Технологический институт штата Джорджия), Эрих Нахум (компания IBM Research), Эрнст Бирзак (Институт Eurecom), Эстер А. Хьюз (Университет штата Виргиния), Юнь Лю (студент Нью-Йоркского политехнического института), Юсси Кангашарью (Университет Хельсинки), Янь Гуо (компании Alcatel/Lucent Bell Labs)

Также мы хотим поблагодарить весь коллектив издательства Addison-Wesley, в частности, Майкла Хирша, Мэрилин Ллойд и Эмму Снайдер. Они проделали над шестым изданием титаническую работу (и смогли поладить с двумя очень капризными авторами, которые, по-видимому, от природы неспособны укладываться в заданные сроки). Большое спасибо нашим иллюстраторам Джанет Тойрер и Патрису Росси Калкину, авторам симпатичных рисунков к нашей книге. Особой благодарности заслуживают Андреа Стефанович и ее команда из PreMediaGlobal — за их замечательную работу по подготовке этого издания к выходу. Наконец, отдельно благодарим Майкла Хирша, нашего редактора из Addison-Wesley, и Сьюзен Хартманн — нашего бывшего редактора из Addison-Wesley. Эта книга не получилась бы без их искусного руководства, постоянного подбадривания, практически бесконечного терпения, чувства юмора и настойчивости.

Глава 1
КОМПЬЮТЕРНЫЕ СЕТИ И ИНТЕРНЕТ

Возможно, современный Интернет является крупнейшей инженерной системой, когда-либо созданной человечеством. Она состоит из сотен миллионов соединенных компьютеров, линий связи и коммутаторов; с миллиардами пользователей, которые подключаются через ноутбуки, планшеты и смартфоны; и с множеством новых подключенных устройств, таких как датчики, веб-камеры, игровые консоли, фоторамки и даже стиральные машины. Принимая во внимание, что Интернет настолько велик и имеет так много разнообразных компонентов и применений, можно ли разобраться в том, как он работает? Существуют ли руководящие принципы и структура, которые способны служить основой для понимания такой поразительно огромной и сложной системы? И если это так, может ли изучение компьютерных сетей на самом деле быть интересным *и* увлекательным? К счастью, на все эти вопросы мы, не сомневаясь, отвечаем ДА! В самом деле, наша цель в данной книге — предложить современное введение в динамично развивающуюся область компьютерных сетей, изложив принципы и практические сведения, необходимые для понимания не только сегодняшних, но и завтрашних технологий.

В этой главе дается широкий обзор организации компьютерных сетей и Интернета. Наша цель здесь — нарисовать общую картину и определить контекст для остальной части книги, чтобы увидеть лес сквозь деревья. Мы охватим немало основных сведений и обсудим большинство компонентов компьютерной сети, не упуская из вида общей картины.

Мы систематизируем наш обзор компьютерных сетей в этой главе следующим образом. После введения базовой терминологии и некоторых концепций мы сначала рассмотрим основные компоненты аппаратного и программного обеспечения, из которых состоит сеть. Мы начнем с периферии сети и рассмотрим конечные системы и сетевые приложения, выполняющиеся в сети. Затем мы исследуем ядро компьютерной сети, проанализировав соединения и коммутаторы, которые передают данные, а также изучим сети доступа и физические среды передачи, которые соединяют конечные системы с ядром сети. Мы узнаем, что Интернет — это сеть, состоящая из сетей, а также узнаем, как они соединяются друг с другом.

Завершив обзор периферии и ядра компьютерной сети, во второй части этой главы мы займемся более широким и абстрактным изучением сети. Мы исследуем задержки, потери и пропускную способность в компьютерной сети и рассмотрим простые количественные модели, которые учитывают задержки, связанные с передачей, распространением данных и организацией очередей. Затем мы введем некоторые из основных принципов построения архитектуры в компьютерных сетях, а именно разделение протоколов на уровни и модели обслуживания. Мы также узнаем, что компьютерные сети уязвимы для множества атак разных типов. Мы сделаем обзор некоторых из этих атак и рассмотрим возможные способы повысить защищенность сетей. Наконец, в заключительной части главы будет кратко изложена история развития компьютерных сетей.

1.1. Что такое Интернет

В этой книге мы будем иметь дело с общедоступным Интернетом, особой компьютерной сетью, которая является основным средством передачи данных, и используем его как пример для обсуждения компьютерных сетей и их протоколов. Но что собой *представляет* Интернет? Существует два ответа на этот вопрос. Во-первых, мы можем описать внутреннюю механику, «винтики» Интернета, то есть основные аппаратные и программные компоненты, из которых он состоит. Во-вторых, мы можем описать Интернет на языке сетевой инфраструктуры, предоставляющей службы для распределенных приложений. Давайте начнем с описания внутреннего устройства Интернета, пользуясь для иллюстрации обсуждаемых вопросов рис. 1.1.

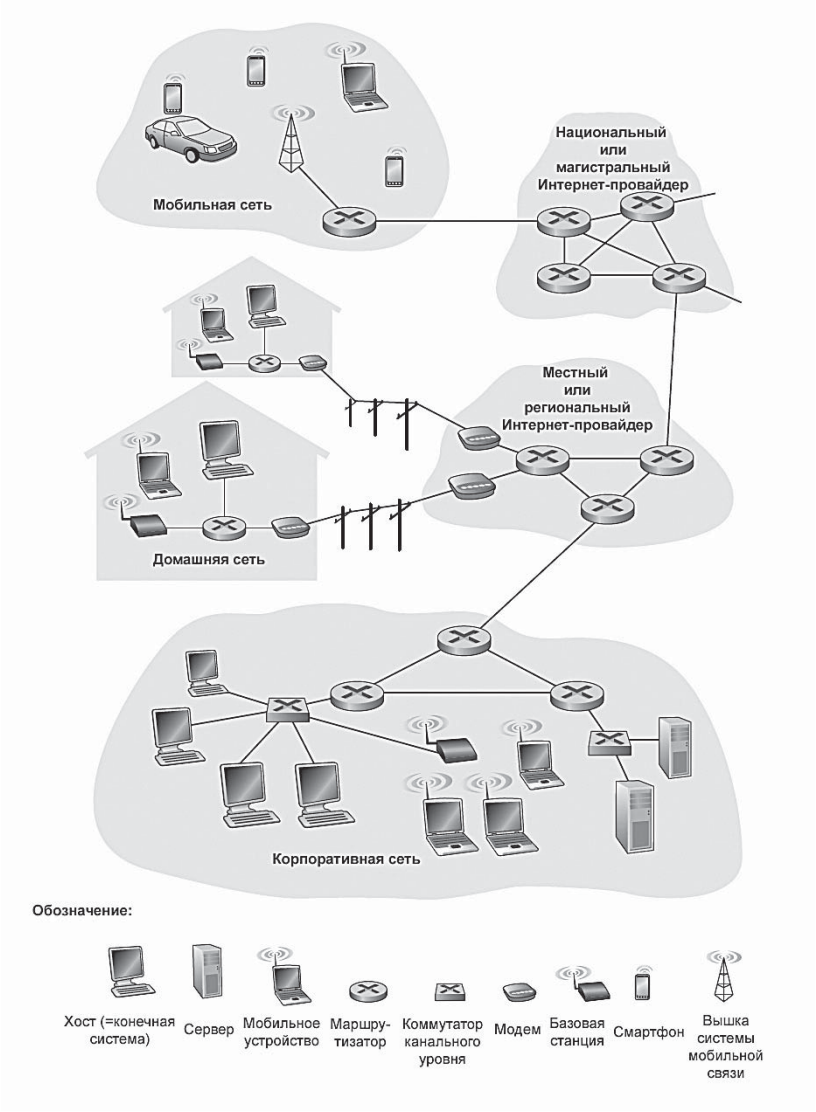


Рис. 1.1. Некоторые основные элементы Интернета

1.1.1. Внутреннее устройство Интернета

Интернет — это компьютерная сеть, которая связывает между собой сотни миллионов вычислительных устройств по всему миру. Не так давно эти вычислительные устройства были в основном обычными настольными персональными компьютерами, рабочими станциями

с операционной системой Linux и так называемыми серверами, которые хранили и передавали информацию, например веб-страницы и сообщения электронной почты. Однако к Интернету подключают все больше и больше нетрадиционных конечных систем, таких как ноутбуки, смартфоны, планшеты, телевизоры, игровые консоли, веб-камеры, автомобили, датчики для определения состояния окружающей среды, фоторамки, а также домашние электрические и охранные системы. На самом деле термин *компьютерная сеть* несколько устаревает, принимая во внимание множество новых устройств, которые подключаются к Интернету. На профессиональном жаргоне все эти устройства называются **хостами,** или **конечными системами**. По состоянию на июль 2011 г. к Интернету было подключено около 850 миллионов конечных систем[245], не считая смартфонов, ноутбуков и других устройств, которые подключаются к Интернету эпизодически. В общем, по оценкам, число пользователей Интернета насчитывает более 2 миллиардов[255–256].

Конечные системы соединяются вместе с помощью сети **линий связи** и **коммутаторов пакетов**. В разделе 1.2 мы увидим, что линии связи бывают множества типов, с различными физическими носителями, включая коаксиальный кабель, медный провод, оптическое волокно и радиочастотный спектр. По разным линиям данные могут передаваться с разными скоростями, причем **скорость передачи** линии измеряется в битах в секунду (бит/с). Когда в одной конечной системе имеются данные для отправки в другую, передающая конечная система сегментирует их и добавляет в каждый сегмент байты заголовка. Затем полученные порции информации, известные как **пакеты**, посылаются через сеть в конечную систему назначения, где снова собираются в исходные данные.

Коммутатор пакетов получает пакет по одной из своих входных линий связи и направляет его по одной из своих исходящих линий связи. Коммутаторы поставляются в разных формах и разновидностях, но двумя наиболее известными типами в современном Интернете являются **маршрутизаторы** и **коммутаторы канального уровня**. Коммутаторы обоих типов направляют пакеты в их конечные пункты назначения. Коммутаторы канального уровня обычно применяются в сетях доступа, тогда как маршрутизаторы обычно используются в ядре сети. Последовательность линий связи и коммутаторов пакетов, через которые проходит пакет из отправляющей конечной системы в принимающую, называется **маршрутом** или **путем** в сети. Точный объем данных, передаваемых в Интернете, трудно подсчитать, но по оценкам компа-

нии Cisco[108], мировой трафик составлял в 2012 году приблизительно 40 эксабайт в месяц.

Сети с коммутацией пакетов во многих отношениях сходны с транспортными сетями шоссе, дорог и перекрестков, по которым ездят автомобили. Например, рассмотрим завод, которому требуется перевезти большое количество груза на склад, расположенный за тысячи километров. Груз на заводе делится на отдельные части и загружается на несколько грузовиков. Затем каждый из грузовиков независимо путешествует по сети шоссе, дорог и перекрестков к пункту назначения. На складе груз разгружается и группируется с остальным прибывающим грузом из той же партии. Таким образом, пакеты во многих отношениях аналогичны грузовикам, линии связи напоминают шоссе и дороги, коммутаторы пакетов сходны с перекрестками, а конечные системы похожи на здания. Так же как грузовик перемещается по пути в транспортной сети, так и пакет проходит путь в компьютерной сети.

Конечные системы получают доступ к Интернету через поставщиков услуг Интернета, или **Интернет-провайдеров** (Internet Service Providers, **ISP**). Существуют провайдеры, которые обеспечивают услуги Интернета по месту жительства (местные кабельные и телефонные компании); корпоративные, университетские, а также тех, что предоставляют доступ Wi-Fi в аэропортах, гостиницах, кофе-барах и других общественных местах. Каждый Интернет-провайдер сам по себе имеет сеть, состоящую из коммутаторов пакетов и линий связи. Провайдеры Интернета предоставляют для конечных систем различные типы доступа в сеть, включая широкополосный доступ по месту жительства, например DSL- или кабельный модем, высокоскоростной доступ к локальной сети, беспроводной доступ и доступ через модем для коммутируемых телефонных линий на скорости 56 Кбит/с. Интернет-провайдеры предоставляют также доступ во всемирную сеть для поставщиков контента, подключая веб-сайты непосредственно к глобальной сети. Интернет — это конечные системы, соединяемые друг с другом, так что предоставляющие к ним доступ организации также должны быть соединены между собой. Интернет-провайдеры, находящиеся на нижнем уровне, соединяются друг с другом через национальных и международных Интернет-провайдеров верхнего уровня, таких как Level 3 Communications, AT&T, Sprint и NTT. Сети провайдеров верхнего уровня состоят из маршрутизаторов высокой производительности, взаимосвязанных с помощью высокоскоростных оптоволоконных каналов. Сеть каждого поставщика услуг Интернета верхнего или нижнего уровня имеет независимое

управление, использует протокол IP (о протоколах см. ниже) и соответствует определенным соглашениям по именованию и адресации. Мы рассмотрим поставщиков услуг Интернета и их взаимосвязи более подробно в разделе 1.3.

Конечные системы, коммутаторы пакетов и другие части Интернета выполняют **протоколы**, которые управляют передачей и получением информации в Интернете. Двумя самыми важными протоколами в Интернете являются **протокол управления передачей** (Transmission Control Protocol, **TCP**) и **протокол Интернета** (Internet Protocol, **IP**). В протоколе IP задается формат пакетов, которые передаются между маршрутизаторами и конечными системами. Совокупность (стек) основных протоколов Интернета известна как **TCP/IP**. Мы начнем изучать протоколы в вводной главе, которую вы сейчас читаете. Но это только начало — бóльшая часть нашей книги посвящена протоколам компьютерных сетей!

В связи со значимостью протоколов для Интернета важно, чтобы все договорились о том, что делает каждый протокол, тогда пользователи могут создавать взаимодействующие системы и изделия. Вот где вступают в действие стандарты. **Интернет-стандарты** разрабатываются инженерным советом Интернета (IETF, Internet Engineering Task Force)[238]. Документы стандартов IETF называются запросами на отзывы, или **рабочими предложениями** (request for comments, **RFC**). Документы RFC появились в виде общих предложений к обсуждению (отсюда их название) для решения проблем, с которыми сталкивался при разработке сетей и протоколов предшественник Интернета[21]. Документы RFC отличаются технической направленностью и подробностью изложения. В них определяются протоколы, такие как TCP, IP, HTTP (для Всемирной паутины) и SMTP (для электронной почты). В настоящее время существует более 6000 документов RFC. В других организациях также определяют стандарты для компонентов сетей, в особенности для сетевых линий связи. Например, комитетом по стандартам локальных и городских сетей (IEEE 802 LAN/MAN Standards Committee)[227] разрабатываются стандарты по Ethernet, беспроводной связи Wi-Fi.

1.1.2. Описание служб

В приведенных выше рассуждениях мы идентифицировали отдельные части, составляющие Интернет. Однако мы можем также описать его, рассматривая его под совершенно другим углом — а именно как *ин-*

фраструктуру, которая предоставляет службы для приложений. Эти приложения включают электронную почту, веб-серфинг, социальные сети, обмен мгновенными сообщениями, передача речи по IP-протоколу (VoIP), потоковое видео, сетевые игры, одноранговый совместный доступ к файлам, телевидение через Интернет (IP TV), удаленный доступ и многое-многое другое. Все они считаются **распределенными приложениями**, поскольку вовлекают в работу множество конечных систем, которые обмениваются данными друг с другом. Важно, что Интернет-приложения выполняются на конечных системах, а не на коммутаторах пакетов в ядре сети. Хотя коммутаторы пакетов обеспечивают обмен данными между конечными системами, они не имеют отношения к конкретным приложениям, являющимся источниками или получателями данных.

Давайте проанализируем чуть глубже то, что мы подразумеваем под инфраструктурой, предоставляющей службы для приложений. С этой целью предположим, что у вас есть потрясающая новая идея для распределенного Интернет-приложения, которая облагодетельствует человечество или просто может принести вам богатство и известность. Как бы вы могли трансформировать эту идею в реальное Интернет-приложение? Так как приложения выполняются на конечных системах, вам требуется создать программы, которые выполняются на конечных системах. Например, вы могли бы написать программы на Java, C или Python. Теперь, поскольку вы разрабатываете распределенное Интернет-приложение, программы, выполняющиеся на разных конечных системах, должны передавать данные друг другу. И здесь мы подходим к главной теме, которая ведет к альтернативному способу описания Интернета как платформы для приложений. Каким образом программа, выполняющаяся на одной конечной системе, отдает команду Интернету доставить данные в другую программу, выполняющуюся на другой конечной системе?

Конечные системы, подсоединенные к Интернету, предоставляют **интерфейс программирования приложений** (Application Programming Interface, **API**), который определяет, как программа, выполняющаяся на одной конечной системе, запрашивает инфраструктуру Интернета для доставки данных в конкретную целевую программу, выполняющуюся на другой конечной системе. API Интернета является набором правил, которые должны выполняться, чтобы Интернет мог доставлять данные в целевую программу. Мы подробно рассмотрим функции интерфейса программирования приложений в главе 2. А пока давайте проведем про-

стую аналогию, которую будем часто использовать в этой книге. Предположим, Алиса хочет отправить письмо Бобу, пользуясь почтовой службой. Само собой разумеется, что наша героиня не может просто написать письмо (или данные) и бросить его из окна. Вместо этого почтовая служба требует, чтобы Алиса поместила письмо в конверт, написала полностью фамилию и имя Боба, его адрес и почтовый код, заклеила конверт, приклеила марку в верхнем правом углу и, наконец, бросила конверт в почтовый ящик. Таким образом, у почтовой службы есть свой собственный «API почтовой службы», или набор правил, которые должна соблюдать Алиса, чтобы ее письмо доставили Бобу. Аналогично для Интернета имеется API, правила которого должна соблюдать программа, отправляющая данные, чтобы эти данные были доставлены через Интернет принимающей их программе.

Конечно, почтовая служба предоставляет своим клиентам более чем одну услугу: срочная доставка, подтверждение получения, обычная почта и многое другое. Подобным образом в Интернете предлагается множество служб для приложений. Когда вы разрабатываете Интернет-приложение, вы также можете выбрать для него одну из служб Интернета. Мы рассмотрим службы Интернета в главе 2.

Мы только что привели два описания Интернета; одно с точки зрения его аппаратных и программных компонентов, другое в терминах инфраструктуры для предоставления служб распределенным приложениям. Возможно, вы все еще путаетесь в том, что такое Интернет. Что такое коммутация пакетов и протокол TCP/IP? Что собой представляют маршрутизаторы? Какие типы линий связи существуют в Интернете? Что такое распределенное приложение? Как можно подсоединить к Интернету тостер или датчик погодных условий? Если вы чувствуете, что информации слишком много, не волнуйтесь — цель нашей книги состоит в том, чтобы ознакомить вас с базовыми элементами Интернета и принципами, лежащими в основе его работы. Эти важные понятия и ответы на поставленные вопросы будут излагаться в следующих разделах и главах.

1.1.3. Что такое протокол?

Теперь, когда вы немного разобрались с тем, что такое Интернет, давайте рассмотрим другой важный специальный термин компьютерных сетей: *протокол*. Что такое протокол? Какую функцию он выполняет?

Аналогия с человеком

Возможно, легче всего разобраться с понятием протокола компьютерных сетей, проведя сначала некоторые аналогии с человеком, поскольку мы, люди, выполняем протоколы постоянно. Рассмотрим, что мы делаем, когда хотим спросить у кого-нибудь, который час. Типичный обмен информацией показан на рис. 1.2. Социальный протокол (или, по крайней мере, правила хорошего тона) диктует, что первый из общающихся приветствует собеседника (первый «Привет» на рис. 1.2), чтобы установить контакт. Типичным ответом на это приветствие является точно такое же сообщение «Привет». Само собой разумеется, что сердечный ответ «Привет» воспринимается как признак того, что инициатор контакта может продолжить и спросить, который час. Другой ответ на первоначальную реплику «Привет» (например, «Не беспокойте меня!», или «Я не говорю по-русски», или что-нибудь непечатное) может указывать на нежелание или невозможность общения. В этом случае, согласно социальному протоколу, последующий вопрос о времени будет неуместен.

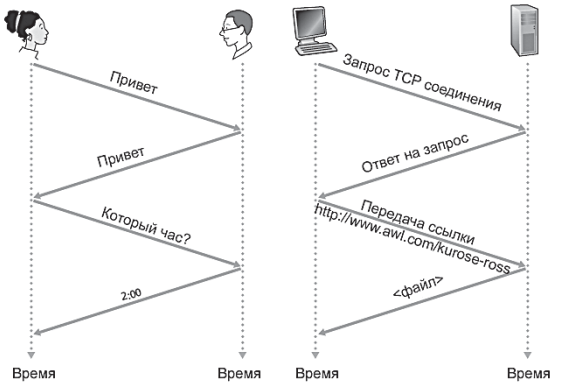


Рис. 1.2. Социальный протокол и протокол компьютерной сети

Иногда спрашивающий вообще не получает ответа на вопрос, и тогда обычно прекращает попытки узнать время у этого человека. Обратите внимание, что в нашем социальном протоколе *существуют специальные сообщения, передаваемые нами, и определенные действия, выполняемые в ответ на полученные ответные сообщения или другие события* (например, отсутствие ответа в течение некоторого заданного времени). Несомненно, передаваемые и получаемые сообщения, а также действия, предпринимаемые, когда отправляются или принимаются эти сообще-

ния либо возникают другие события, играют центральную роль в социальном протоколе. Если люди выполняют разные протоколы (например, если один человек воспитан, а другой нет или если один знает, что такое время, а другой не имеет о нем представления), протоколы не взаимодействуют, и никакую полезную работу выполнить невозможно. То же самое справедливо в сетях — для выполнения задачи двум (или более) взаимодействующим объектам необходимо выполнять один и тот же протокол.

Давайте рассмотрим второй пример аналогии с людьми. Предположим, вы находитесь в классе учебного заведения (например, в компьютерном классе школы!). Учитель что-то бубнит о протоколах, а вы ничего не понимаете. Учитель останавливает свой монолог, чтобы задать вопрос: «Есть вопросы?» (сообщение, которое передается всем не спящим студентам и принимается ими). Вы поднимаете руку (передавая неявное сообщение учителю). Ваш учитель подтверждает улыбкой, что увидел ваш сигнал, говоря: «Да...» (переданное сообщение, поощряющее вас задать свой вопрос — учитель *любит*, когда ему задают вопросы), и затем вы задаете свой вопрос (т. е. передаете свое сообщение учителю). Учитель слышит ваш вопрос (принимает сообщение-вопрос) и отвечает (передает ответ вам). Еще раз мы видим, что передача и прием сообщений, а также набор выполняемых при этом общепринятых действий составляют суть данного протокола вопросов-и-ответов.

Сетевые протоколы

Сетевой протокол похож на социальный протокол, только объектами, обменивающимися сообщениями и выполняющими действия, в данном случае являются аппаратные или программные компоненты некоторого устройства (например, компьютера, смартфона, планшета, маршрутизатора или любого другого, обладающего возможностью работы в сети). Все действия в Интернете, предпринимаемые двумя или более взаимодействующими удаленными объектами, регулируются протоколом. Например, аппаратным образом реализованные протоколы в двух физически соединенных компьютерах управляют потоком данных по «проводу» между двумя сетевыми интерфейсными платами; протоколы, отслеживающие перегрузки сети в конечных системах, управляют скоростью передачи пакетов между отправителем и получателем; протоколы в маршрутизаторах определяют путь пакетов от источника к приемнику. Протоколы выполняются в Интернете повсе-

местно, и поэтому бóльшая часть данной книги посвящена протоколам компьютерных сетей.

В качестве примера протокола компьютерной сети, который вам, вероятно, знаком, рассмотрим, что происходит, когда вы отправляете запрос на веб-сервер, то есть, когда вы набираете URL-адрес веб-страницы в своем веб-браузере. Сценарий иллюстрирует правая часть рис. 1.2. Сначала ваш компьютер передает сообщение с запросом о подключении на веб-сервер и ожидает ответа. Веб-сервер в итоге получает ваше сообщение с запросом на подключение и возвращает ответное сообщение о подключении. Зная, что теперь можно запросить веб-документ, ваш компьютер посылает затем имя веб-страницы, которую он хочет получить с веб-сервера, в сообщении `GET`. Теперь, наконец, веб-сервер возвращает веб-страницу (файл) в ваш компьютер.

Как показано в приведенных выше примерах человеческого и сетевого протоколов, обмен сообщениями и действия, предпринимаемые при передаче и приеме этих сообщений, являются ключевыми определяющими элементами протокола:

> ***Протокол*** *определяет формат и порядок сообщений, которыми обмениваются два или более взаимодействующих объектов, а также действия, предпринимаемые при передаче и/или приеме сообщения либо при возникновении другого события.*

Интернет и компьютерные сети в целом широко используют протоколы. Разные протоколы применяются для выполнения разных задач связи. По мере чтения этой книги вы узнаете, что некоторые протоколы простые и очевидные, тогда как другие сложные и глубоко продуманные. Освоение области компьютерных сетей эквивалентно поиску ответов на вопросы что, почему и как в отношении сетевых протоколов.

1.2. Периферия сети

В предыдущем разделе мы представили схематичный обзор Интернета и сетевых протоколов. Сейчас мы собираемся «копнуть» чуть глубже в компоненты компьютерной сети (и Интернета в частности). Мы начнем в этом разделе с границы сети и взглянем на компоненты, которые нам наиболее знакомы — а именно компьютеры, смартфоны и другие устройства, используемые нами ежедневно. В следующем раз-

деле мы перейдем от периферии к ядру сети и рассмотрим коммутацию и маршрутизацию в компьютерных сетях.

ИСТОРИЯ

Ошеломляющее множество конечных систем Интернета

Не так давно устройствами конечных систем, подключенными к Интернету, были в основном обычные компьютеры, такие как настольные ПК и мощные серверы. Со второй половины 1990-х и вплоть до нашего времени множество интересных устройств подключаются к Интернету, используя свои возможности передавать и получать цифровые данные. Принимая во внимание вездесущность Интернета, его хорошо определенные (стандартизованные) протоколы и доступность современного бытового оборудования, вполне естественно стремление подключить все эти устройства в сеть и к серверам, уже соединенным с Интернетом.

Многие из этих устройств находятся дома — игровые видеоприставки (например, Xbox компании Microsoft), телевизоры с функцией Smart TV, цифровые фоторамки с функцией загрузки изображений, стиральные машины, холодильники и даже тостеры, которые получают метеорологический прогноз (например, смешанные облака и солнце) и запекают его на утренний тост[44]. В IP-телефонах, оснащенных GPS-функциями, зависящих от места нахождения службы (карты, информация о ближайших сервисах или людях) находятся под кончиками пальцев. Объединенные в сеть датчики, встроенные в физическую среду, позволяют отслеживать состояния зданий, мостов, сейсмическую активность, ареалы проживания диких животных, производить мониторинг устьев рек и предоставлять информацию о погоде. Биомедицинские устройства могут встраиваться и объединяться в сеть внутри человеческого тела. При таком разнообразии устройств, соединенных вместе, Интернет на самом деле становится «Интернетом вещей»[252].

Вспомните из предыдущего раздела, что на профессиональном жаргоне компьютеры и другие устройства, подключенные к Интернету, часто называют конечными системами. Это объясняется тем, что, как показано на рис. 1.3, они находятся на внешнем краю Интернета. К конечным системам относятся настольные компьютеры (под управлением операционных систем Windows, Mac и Linux), серверы (например, почтовые и веб-серверы), а также мобильные компьютеры (включая ноутбуки, смартфоны и планшеты). Кроме того, в качестве конечных систем к Интернету присоединяется растущее число нетрадиционных устройств (см. вставку на странице).

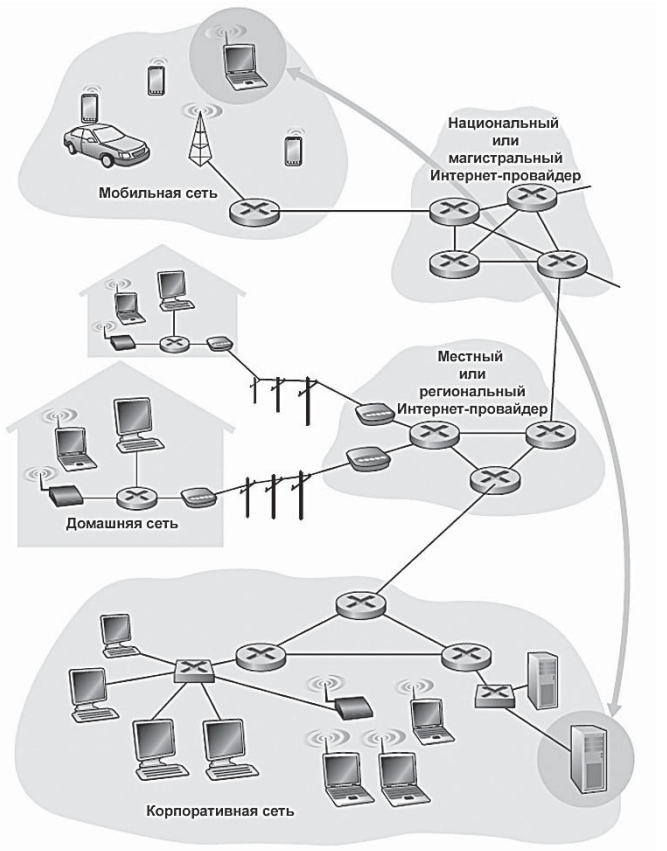


Рис. 1.3. Взаимодействие конечных систем

Конечные системы называются также *хостами*, так как на них находятся (т. е. выполняются) прикладные программы, такие как веб-браузер, приложение веб-сервера и клиентская программа электронной почты или программное обеспечение почтового сервера. В этой книге термины «хосты» и «конечные системы» взаимозаменяемы; а именно *хост = конечная система*. Хосты иногда дополнительно подразделяются на две категории: **клиенты** и **серверы**. Проще говоря, клиентами, как правило, являются настольные и мобильные персональные компьютеры, смартфоны и другие устройства, тогда как серверы — это обычно более мощные машины, которые хранят и рассылают веб-страницы, передают потоковое видео, перенаправляют электронную почту и выполняют другие операции. Сегодня большинство серверов, с которых мы получаем результаты поиска, электронную почту, веб-страницы и видео, находятся в больших **центрах**

обработки данных (дата-центрах). Например, Google имеет 30–50 центров обработки данных, причем во многих из них находится более сотни тысяч серверов.

1.2.1. Сети доступа

Рассматривая приложения и конечные системы на «границе» сети, давайте далее проанализируем сеть доступа — сеть, которая физически соединяет конечную систему с первым маршрутизатором (также известным как «граничный маршрутизатор») на пути от конечной системы к любой другой удаленной конечной системе.

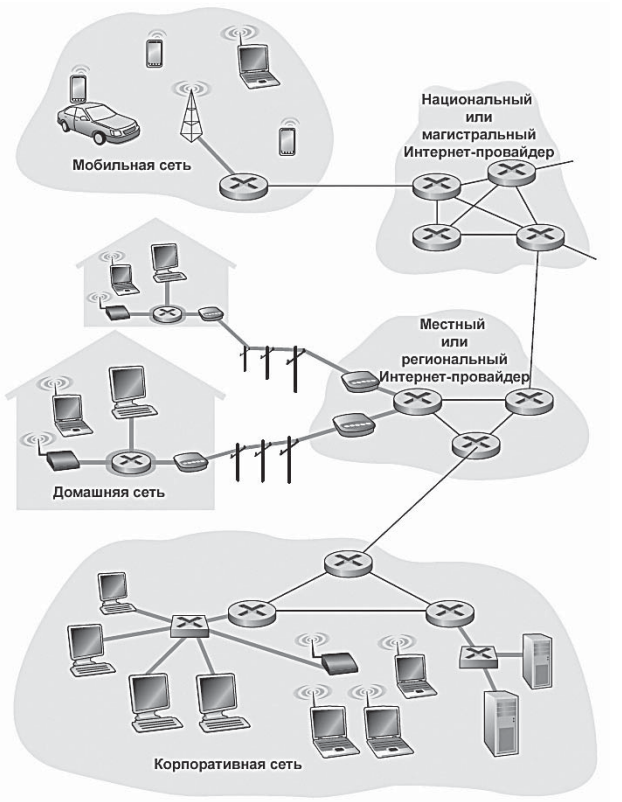


Рис. 1.4. Сети доступа

На рис. 1.4 показано несколько типов сетей доступа с жирными, закрашенными линиями и средой (домашняя, корпоративная и глобальная сеть беспроводной мобильной связи), в которой они используются.

Домашний доступ: DSL, кабельный, FTTH, коммутируемый и спутниковый

В развитых странах сегодня более 65% семей имеют доступ в Интернет, причем Корея, Нидерланды, Финляндия и Швеция лидируют с показателем 80%, причем почти все пользователи подключены через высокоскоростные широкополосные соединения[255, 256]. Недавно в Финляндии и Испании объявили, что высокоскоростной доступ в Интернет должен быть «законным правом» жителей этих стран. Принимая во внимание такую сильную заинтересованность в домашнем доступе, давайте начнем наш обзор сетей доступа с того, что рассмотрим способы подключения домов к Интернету.

Сегодня двумя доминирующими типами широкополосного доступа в Интернет по месту жительства являются **абонентская цифровая линия** (digital subscriber line**, DSL**) и кабель. Обычно квартира (или другое местопребывание) получает DSL-доступ в Интернет от той же телефонной компании, которая предоставляет проводную местную связь. Таким образом, когда используется технология DSL, телефонная компания клиента является также и поставщиком услуг Интернета. Как показано на рис. 1.5, DSL-модем каждого клиента использует существующую телефонную линию (медный провод в виде витой пары, который будет рассмотрен в разделе 1.2.2) для обмена данными с мультиплексором доступа по абонентской цифровой линии (DSLAM), находящимся в местном центральном офисе телефонной компании. Домашний DSL-модем принимает цифровые данные и преобразует их в высокочастотные тональные сигналы для передачи по телефонным проводам в центральный офис; аналоговые сигналы из множества таких домов преобразуются обратно в цифровой сигнал в DSLAM.

По абонентской телефонной линии одновременно передаются как данные, так и традиционные телефонные сигналы, которые кодируются с разными частотами:

- высокоскоростной входной канал, в диапазоне от 50 кГц до 1 МГц;
- среднескоростной выходной канал в диапазоне от 4 кГц до 50 кГц;
- обычный двухпроводной телефонный канал в диапазоне от 0 до 4 кГц.

При таком подходе один DSL-канал делится на три отдельных для того, чтобы его могли одновременно совместно использовать телефонные вызовы и Интернет-подключение. (Мы обсудим эту технологию частотного мультиплексирования в разделе 1.3.1.)

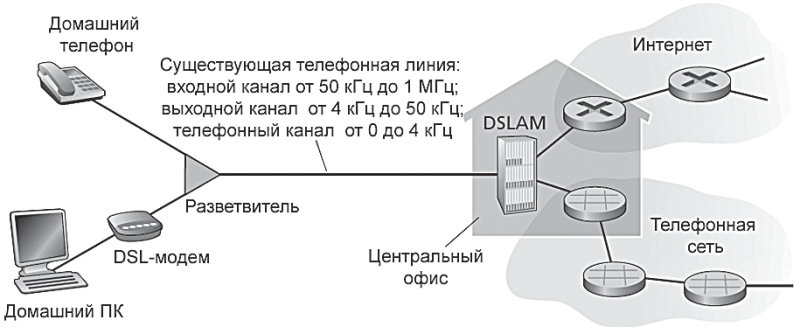


Рис. 1.5. DSL-доступ в Интернет

На стороне клиента разветвитель разделяет поступающие в квартиру сигналы данных и телефонных вызовов и направляет сигнал данных в DSL-модем. На стороне телефонной компании, в центральном офисе, мультиплексор доступа по цифровой абонентской линии (DSLAM) разделяет сигналы данных и телефонных вызовов и передает данные в Интернет. К одному мультиплексору DSLAM подключены сотни или даже тысячи семей[137].

В стандартах DSL определены скорости передачи, равные 12 Мбит/с на входе и 1,8 Мбит/с на выходе[249], а также 24 Мбит/с для входных данных и 2,5 Мбит/с для выходных данных[250]. Так как скорости передачи исходящих и входящих данных разные, говорят, что доступ асимметричный. Фактические скорости передачи входных и выходных данных могут быть меньше скоростей, приведенных выше, так как поставщик DSL-служб может намеренно ограничивать скорости обмена данными домашних пользователей, если предоставляются многоуровневые услуги (разные скорости, доступные по разным ценам) либо по причине ограничения максимальной скорости передачи в зависимости от расстояния между домом и центральным офисом, пропускной способности витой пары и уровня электрических помех. Разработчики рассчитывали, что DSL будет применяться на коротких расстояниях между квартирами и центральным офисом; в целом, если место проживания находится далее 8–16 км от центрального офиса, следует использовать альтернативные способы доступа в Интернет.

В то время как технологией DSL используется существующая местная инфраструктура телефонной компании, **кабельный доступ в Интернет** организуется с помощью существующей кабельной телевизионной инфраструктуры, предоставленной компанией кабельного телевиде-

ния. Именно от нее пользователь получает кабельный доступ в Интернет. Как показано на рис. 1.6, головная кабельная станция посредством оптоволоконного кабеля подключается к разветвлениям уровня района или квартала, от которых дальше идет обычный коаксиальный кабель, используемый для подключения отдельных домов и квартир. Каждым районным разветвлением обычно поддерживается от 500 до 5000 домов. Так как в этой системе используются и волоконно-оптические, и коаксиальные кабели, ее часто называют **гибридной оптико-коаксиальной** (hybrid fiber-coaxial, **HFC**) кабельной сетью.

Рис. 1.6. Гибридная оптико-коаксиальная сеть доступа

Для кабельного доступа в Интернет требуются специальные модемы, называемые *кабельными модемами*. Как и в случае с DSL-модемом, кабельный модем обычно является внешним устройством и подсоединяется к домашнему персональному компьютеру через Ethernet-порт. (Мы подробно рассмотрим технологию Ethernet в главе 5.) **Терминальная станция кабельных модемов** (Cable modem termination system, **CMTS**) выполняет аналогичные функции, что и мультиплексор DSLAM в DSL-сети — преобразует исходящий аналоговый сигнал множества домашних кабельных модемов в цифровой формат. Кабельные модемы делят гибридную оптокоаксиальную кабельную сеть на два канала, нисходящий и восходящий. Как и в случае DSL, доступ обычно асимметричный: для нисходящего канала, как правило, выделяется более высокая скорость передачи данных, чем для восходящего. Стандартом DOCSIS 2.0 определяются стандартные скорости для нисходящего потока данных — до 42,8 Мбит/с и для восходящего потока данных — до 30,7 Мбит/с. Как

в случае DSL-сетей, максимально доступная скорость может быть нереализуема из-за более низких скоростей передачи данных, обусловленных договором или повреждениями средств коммуникации.

Важная особенность кабельного доступа в Интернет состоит в том, что он является совместно используемым средством вещания. В частности, каждый пакет, передаваемый головной станцией, путешествует в нисходящем направлении по каждому каналу во все дома, а каждый пакет, передаваемый домом, путешествует по восходящему каналу в головную станцию. По этой причине, если несколько пользователей одновременно загружают видеофайл по нисходящему каналу, фактическая скорость передачи данных, с которой каждый пользователь принимает свой видеофайл, значительно меньше, чем суммарная скорость передачи нисходящих данных по кабелю. С другой стороны, если имеется лишь несколько активных пользователей и все они путешествуют по Интернету, то каждый может фактически загружать веб-страницы при максимальной скорости передачи нисходящих данных по кабелю, так как веб-страницы редко запрашиваются точно в одно время. Поскольку канал восходящих данных также используется совместно, необходим протокол распределенного множественного доступа, чтобы координировать передачи данных и избегать конфликтов. (Некоторые аспекты проблемы конфликтов мы рассмотрим в главе 5.)

Хотя в США в настоящее время DSL и кабельные сети составляют более 90 процентов домашнего широкополосного доступа в Интернет, перспективной технологией, которая обеспечивает еще более высокие скорости передачи данных, является развертывание **оптоволоконных линий до самой квартиры** (Fiber To The Home**, FTTH**)[175]. Как следует из самого названия технологии, идея FTTH проста — предоставление волоконно-оптического соединения от центрального офиса напрямую в дом. В Соединенных Штатах Америки компания Verizon особенно настойчиво предлагает FTTH со своей службой FIOS[646].

Существует несколько конкурирующих технологий, использующих оптические соединения между центральным офисом и домами. Простейшая оптическая распределительная сеть называется прямой волоконно-оптической сетью, с одним волоконно-оптическим кабелем, выходящим из центрального офиса, для каждого дома. Более распространенный вариант, когда каждый волоконно-оптический кабель, выходящий из центрального офиса, на практике совместно используется множеством зданий; только после того, как кабель доходит достаточно близко до домов, он разделяется на отдельные для каждого конкретного клиента. Су-

ществует две конкурирующие сетевые архитектуры с распределением данных по оптическим каналам, которые выполняют такое разделение: **активные оптические сети** (active optical networks, **AON**) и **пассивные оптические сети** (passive optical networks, **PON**). По существу AON — это переключаемый Ethernet, который рассматривается в главе 5.

Здесь мы кратко обсудим технологию PON, которая используется в службе FIOS компании Verizon. На рис. 1.7 показана технология развертывания волоконной оптики до квартиры (FTTH), использующая архитектуру распределения PON.

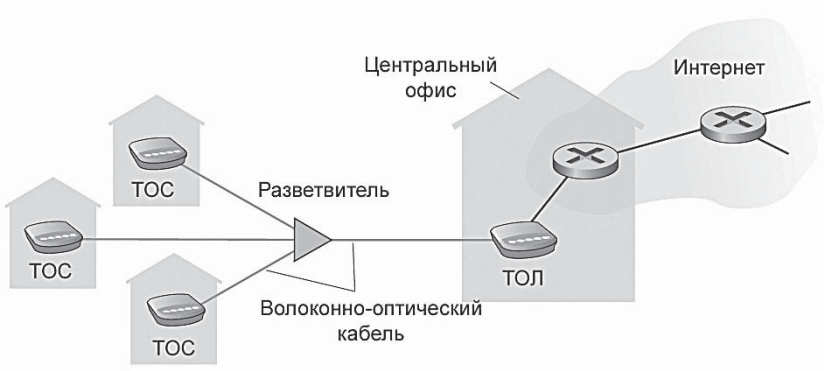


Рис. 1.7. Доступ в Интернет по технологии FTTH

В каждом доме имеется терминатор оптической сети (ТОС), который соединяется по выделенному волоконно-оптическому кабелю с районным разветвителем. Данное устройство соединяет ряд домов (обычно менее 100) с одним совместно используемым оптическим кабелем, который, в свою очередь, подключен к терминатору оптической линии (ТОЛ) в центральном офисе телекоммуникационной компании. ТОЛ, обеспечивающий преобразование между оптическими и электрическими сигналами, соединяется с Интернетом через маршрутизатор телекоммуникационной компании. В доме пользователи подключают к ТОС домашний маршрутизатор (обычно беспроводной) и получают доступ к Интернету через него. В архитектуре PON все пакеты, передаваемые из ТОЛ в разветвитель, дублируются в разветвителе (аналогично головной кабельной станции).

Технология FTTH позволяет предоставлять доступ в Интернет со скоростями вплоть до нескольких гигабит/с. Однако большинство Интернет-провайдеров, работающих по технологии FTTH, предлагают

варианты доступа с разной скоростью, при этом, чем выше скорость, тем больше цена. Средняя скорость передачи входящих данных для пользователей технологии FTTH в США составляла в 2011 году 20 Мбит/с (по сравнению с 13 Мбит/с для сетей с кабельным доступом и менее чем 5 Мбит/с для технологии DSL)[176].

Для обеспечения домов доступом в Интернет используются также еще две технологии сетей доступа. Там, где отсутствуют DSL, кабель и FTTH (например, в сельской местности), может использоваться спутниковый канал для подключения к Интернету на скорости свыше 1 Мбит/с. К поставщикам такого спутникового доступа относятся StarBand и HughesNet. Коммутируемый доступ по традиционным телефонным линиям основан на той же модели, что и DSL — домашний модем подключается по телефонной линии к модему Интернет-провайдера. По сравнению с DSL и другими широполосными сетями коммутируемый доступ является невыносимо медленным, обеспечивая максимальную скорость всего лишь 56 Кбит/с.

Доступ на предприятии (и дома): Ethernet и Wi-Fi

В корпоративных и университетских городках и все в большей степени в домашних условиях для подключения конечных систем к граничным маршрутизаторам используются локальные вычислительные сети или ЛВС (Local Area Networks, или LAN). Хотя существует много типов технологий ЛВС, Ethernet является безоговорочно самой распространенной в корпоративных, университетских и домашних сетях. Как показано на рис. 1.8, пользователи Ethernet применяют витую медную пару для подключения к Ethernet-коммутатору.

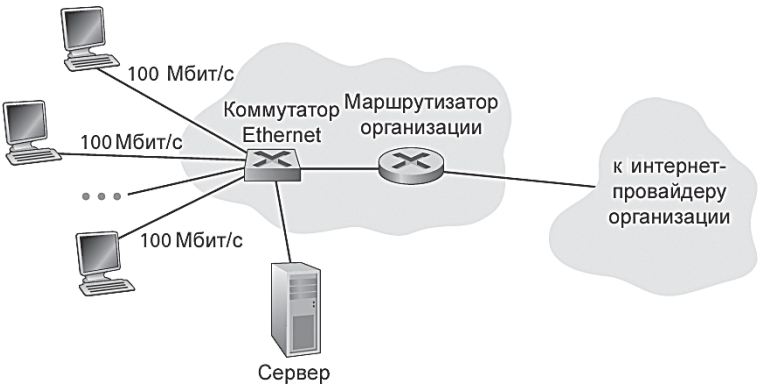


Рис. 1.8. Доступ в Интернет по технологии Ethernet

Эта технология подробно рассматривается в главе 5. Ethernet-коммутатор или сеть таких взаимосвязанных коммутаторов затем в свою очередь соединяется с более крупной сетью Интернет. Используя Ethernet, пользователи обычно имеют доступ к Ethernet-коммутатору на скорости 100 Мбит/с, тогда как для серверов может обеспечиваться доступ на скорости 1 Гбит/с или даже 10 Гбит/с.

Тем не менее пользователи все в большей степени получают доступ в Интернет по беспроводной связи с ноутбуков, смартфонов, планшетов и других устройств (см. выше врезку «Ошеломляющее множество конечных систем Интернета»). В окружении беспроводных ЛВС пользователи беспроводной связи передают (принимают) пакеты в точку доступа, которая соединена с корпоративной сетью (обычно содержащей также и проводной Ethernet), подключенной в свою очередь к проводному Интернету. Пользователь беспроводной ЛВС обычно должен находиться в пределах десятков метров от точки доступа. В наше время беспроводной доступ к ЛВС, основанный на технологии IEEE 802.11, которую чаще называют Wi-Fi, присутствует повсеместно — в университетах, бизнес-центрах, кафе, аэропортах, домах и даже в самолетах. Во многих городах человек, стоя на углу улицы, будет находиться в диапазоне действия десяти или двадцати базовых станций (вы можете поглядеть глобальную карту сети базовых станций 802.11, сделанную энтузиастами, которые любят такие вещи[666]). Технология 802.11 на сегодняшний день обеспечивает разделяемый доступ на скоростях до 54 Мбит/с. Подробнее об этом говорится в главе 6.

Несмотря на то, что Ethernet и Wi-Fi-сети первоначально были разработаны как корпоративные, в последнее время они стали достаточно широко применяться в качестве компонентов домашних сетей. Во многих домах абонентам предоставляется широкополосный доступ по кабельным модемам или DSL совместно с этой достаточно недорогой технологией ЛВС для создания мощных домашних сетей[148]. На рис. 1.9 представлена типичная домашняя сеть, состоящая из беспроводного ноутбука, а также проводного персонального компьютера, базовой станции (точки доступа к беспроводной сети), которая соединяется с ноутбуком, кабельного модема, предоставляющего широкополосный доступ в Интернет, и маршрутизатора, связывающего базовую станцию и стационарный персональный компьютер с кабельным модемом. Такая сеть позволяет членам семьи иметь широкополосный доступ в Интернет, как со стационарного персонального компьютера, так и с ноутбука, который можно перемещать при этом по комнатам.

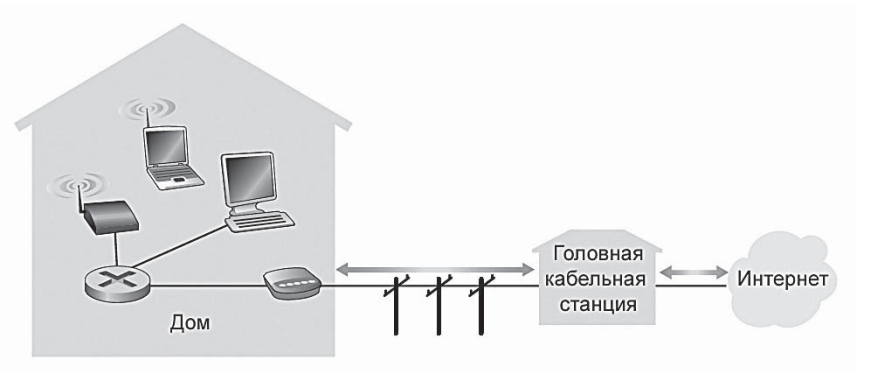


Рис. 1.9. Типичная домашняя сеть

Беспроводной мобильный доступ: 3G и LTE

Все больше растет количество таких устройств, как смартфоны и планшеты под управлением операционных систем iOS, BlackBerry и Android. Такие аппараты применяются для отправки почты, просмотра веб-страниц, общения в Twitter, загрузки музыки и используют ту же самую беспроводную инфраструктуру, которая применяется в сотовой телефонии для отправки/получения пакетов через базовую станцию, обслуживаемую оператором мобильной связи. Но, в отличие от технологии Wi-Fi, в данном случае пользователь может находиться на расстоянии нескольких десятков километров от базовой станции (а не десятков метров).

Телекоммуникационные компании вложили огромные инвестиции в беспроводные мобильные сети третьего поколения (3G), которые обеспечивают доступ в Интернет, используя беспроводные технологии доступа с коммутацией пакетов на скоростях, превышающих 1 Мбит/с. Более того, уже разработаны сети четвертого поколения (4G), использующие технологии мобильного доступа на более высоких скоростях.

Технология LTE (Long-Term Evolution, буквально с англ. — долговременное развитие) своими корнями уходит в 3G-технологию и может потенциально предоставлять скорости, превышающие 10 Мбит/с. И уже в некоторых рекламных объявлениях сообщается о входящих скоростях в несколько десятков Мбит/с. Мы изучим основные принципы беспроводных сетей, а также технологии мобильного доступа, такие как Wi-Fi, 3G, LTE и другие в главе 6.

1.2.2. Физические среды передачи данных

В предыдущем подразделе мы сделали краткий обзор некоторых наиболее важных технологий сетевого доступа в Интернет. Когда мы описывали эти технологии, мы также упоминали используемые физические среды передачи данных. Например, мы сказали, что технология HFC использует комбинацию оптоволоконного и коаксиального кабелей. Мы также упомянули, что в DSL и Ethernet применяется медный кабель (или медную витую пару), а в сетях мобильного доступа — радиочастотный сигнал. В этом подразделе мы дадим краткое описание этих и других сред передачи, которые чаще всего используются в Интернете.

Для того чтобы определить, что подразумевается под понятием физической среды, давайте рассмотрим короткую жизнь одного бита. Представим себе бит, путешествующий от одной конечной системы к другой сквозь сеть каналов и маршрутизаторов. Несчастный бит толкают и передают несчетное число раз. Исходная конечная система вначале передает бит, вскоре после этого первый маршрутизатор на пути получает его; затем он передает бит дальше, вскоре после этого второй маршрутизатор его получает и так далее. Таким образом, наш бит, путешествуя от источника к месту назначения, проходит несколько пар «отправитель-получатель». В каждой такой паре он пересылается с помощью распространяемых электромагнитных волн либо с помощью оптического импульса через **физическую среду**. Последняя может принимать различные формы, в том числе меняться для каждой пары «отправитель-получатель». Примеры физической среды — это витая пара медной проволоки, коаксиальный кабель, многомодовый оптоволоконный кабель, наземный радиочастотный канал и спутниковый радиоканал. Физическая среда делится на две категории: **проводная среда** и **беспроводная среда**. В проводной среде волны распространяются по твердому носителю, такому, как оптоволоконный кабель, медная витая пара или коаксиальный кабель. В беспроводной среде волны распространяются в атмосфере или в окружающем пространстве, например в беспроводной ЛВС или в цифровом спутниковом канале.

Но перед тем как рассмотреть некоторые характеристики различных типов сред, давайте скажем пару слов об их стоимости. Фактическая стоимость физического соединения (медного кабеля, оптоволоконного кабеля и так далее) обычно значительно меньше, чем у других компонентов сети. По этой причине при строительстве зданий очень часто одновременно прокладываются все виды кабелей: медная пара, оптово-

локно, коаксиальный кабель, и даже если первоначально используется один из носителей, не исключена возможность, что в ближайшем будущем может возникнуть потребность в другом и будут сэкономлены немалые средства, так как нужный кабель уже проложен.

Медная витая пара

Наиболее дешевой и распространенной средой передачи является медная витая пара. На практике более 99% кабелей, соединяющих различные телефонные устройства с телефонными коммутаторами, представляют собой именно ее. Многие из нас встречались с ней либо дома, либо на работе. Витая пара состоит из двух изолированных медных проводов, каждый из которых имеет толщину около 1 мм, заключенных в обычную спиральную оболочку. Провода переплетены друг с другом для уменьшения электрических помех, идущих от находящихся рядом проводов. Обычно несколько пар проводов объединяют вместе в один кабель и помещают их в защитный экран. Каждая пара представляет собой одно соединение связи. **Неэкранированная витая пара** (Unshielded twisted pair, **UTP**) используется обычно в локальных сетях внутри здания, то есть в ЛВС. Скорость передачи данных для ЛВС, использующих витую пару, на сегодняшний момент варьируется в пределах от 10 Мбит/с до 10 Гбит/с. Зависит скорость от толщины провода, а также от расстояния между источником и приемником.

С появлением оптоволоконной технологии в 80-е годы многие стали критиковать витую пару за ее относительно низкие скорости передачи. Некоторые даже считали, что оптоволокно ее полностью вытеснит. Но витая пара оказалась гораздо более живучей. Современные технологии такого рода, в частности, кабели категории 6а, позволяют передавать информацию со скоростью до 10 Гбит/с в радиусе до 100 метров. В конце концов витая пара стала доминирующей технологией в создании высокоскоростных локальных сетей.

Как было указано выше, витая пара широко используется и при стационарном подключении к Интернету. Также упоминалось, что технология коммутируемого доступа через модем позволяет передавать данные по кабелю «витая пара» со скоростью до 56 Кбит/с. Кроме того, мы видели, что технологии DSL (digital subscriber line) используют витую пару, обеспечивая для абонентов доступ в Интернет на скоростях в десятки мегабит в секунду (когда пользователи находятся недалеко от модема Интернет-провайдера).

Коаксиальный кабель

Так же как и витая пара, коаксиальный кабель состоит из двух медных проводников, только эти проводники расположены не параллельно, а концентрически (или коаксиально). С помощью такой конструкции, а также благодаря специальной изоляции и экранирования, коаксиальный кабель позволяет достичь высоких скоростей передачи данных. Он часто используется в системах кабельного телевидения. Как мы уже видели раньше, системы кабельного телевидения в сочетании с кабельными модемами могут обеспечивать для абонентов доступ в Интернет на скоростях в десятки мегабит в секунду. В кабельном телевидении, а также в кабельных сетях доступа передатчик переносит цифровой сигнал в определенную полосу частот, и затем результирующий аналоговый сигнал посылается от передатчика к одному или нескольким приемникам. Коаксиальный кабель может использоваться как **разделяемая проводная среда**. К кабелю могут быть непосредственно подключены несколько конечных систем, и каждая из них может принимать сигнал, передаваемый другими конечными системами.

Оптоволоконный кабель

Оптоволокно — это тонкий, гибкий кабель, по которому распространяются световые импульсы, представляющие собой биты информации. Один оптоволоконный кабель может передавать данные на очень значительных скоростях: от десятков до сотен гигабит в секунду. Они не подвержены электромагнитным помехам, имеют очень низкий уровень затухания сигнала на расстояниях до 100 километров, а также устойчивы к механическим воздействиям. Эти характеристики сделали оптоволоконный кабель, в частности, предпочтительной средой передачи данных для межконтинентальных линий связи. Во многих телефонных линиях связи огромной протяженности в Соединенных Штатах Америки и других странах используются исключительно оптоволоконные кабели, однако высокая стоимость оптических устройств — таких как передатчики, приемники и коммутаторы — делает невыгодным их применение для передачи данных на короткие расстояния, например, в ЛВС, либо в сетях домашнего доступа. Оптические носители (Optical Carrier, OC) предлагают скорости передачи от 51,8 Мбит/с до 39,8 Гбит/с. В стандартах они обычно описываются как OC-*n*, где скорость соединения равна *n*×51,8 Мбит/с. На сегодняшний день используются такие стандарты, как OC-1, OC-3, OC-12, OC-24, OC-48, OC-96, OC-192, OC-768. Источники[359, 408] раскрывают различные аспекты оптических сетей.

Наземные радиоканалы

В радиоканалах сигналы передаются посредством электромагнитных волн радиодиапазона. Такая среда передачи очень привлекательна тем, что она не требует физического носителя, может обеспечивать соединение с мобильными пользователями, передачу сигнала на достаточно дальние расстояния, при этом сигнал способен проникать сквозь стены и другие препятствия. Характеристики радиоканала в значительной степени зависят от среды распространения и от расстояния, на которое передается сигнал. Факторами среды обусловлены такие явления, как потери при распространении и затухание сигнала (когда уровень сигнала уменьшается в результате прохождения большого расстояния, а также огибания препятствий), многолучевое затухание (вследствие отражения сигнала от других объектов) и интерференция (наложение других электромагнитных сигналов).

Наземные радиоканалы могут быть классифицированы по трем группам: функционирующие на очень коротких расстояниях (1 или 2 метра); каналы локального распространения, обычно работающие на расстояниях от десяти до нескольких сотен метров; каналы дальнего распространения сигнала, действующие на расстояниях в десятки километров. В первом диапазоне сверхкоротких каналов работают такие устройства, как наушники, клавиатура, некоторые медицинские устройства; технологии беспроводных ЛВС, описанные в разделе 1.2.1, используют каналы локального распространения; технологии мобильного доступа — каналы дальнего распространения. Подробнее мы обсудим радиоканалы в главе 6.

Спутниковые радиоканалы

Спутниковая связь соединяет два или более наземных приемопередатчика сверхвысокочастотного (СВЧ) диапазона, известных как наземные станции. Спутник принимает сигнал на одной полосе частот, восстанавливает его с использованием ретранслятора (обсуждается ниже) и передает на другой частоте. Используются два типа спутников: **геостационарные** и **низкоорбитальные**.

Геостационарные спутники постоянно находятся над одной и той же точкой Земли. Это достигается за счет размещения спутника на орбите в 36 000 километров над поверхностью Земли. Такое огромное расстояние от земной станции до спутника и обратно к другой земной станции приводит к существенной задержке распространения сигнала до 280 мс.

Тем не менее спутниковые каналы связи, которые могут работать на скоростях в сотни мегабит в секунду, часто используются в районах, где нет DSL- или кабельного доступа в Интернет.

Низкоорбитальные спутники размещаются намного ближе к Земле и вращаются вокруг нее (так же, как Луна). Они могут общаться как друг с другом, так и с земными станциями. Чтобы обеспечить непрерывное покрытие, на орбите необходимо разместить достаточно много спутников. В настоящее время разрабатывается целый ряд проектов низкоорбитальных спутниковых систем связи. На тематической вебстранице Ллойда Вуда[673] размещена информация о так называемом созвездии спутников, использующихся для коммуникаций. Спутниковые низкоорбитальные технологии могут быть широко использованы для доступа в Интернет в будущем.

1.3. Ядро сети

Изучив периферию Интернета, давайте углубимся дальше и обратимся к ядру сети — набору коммутаторов пакетов и каналов связи, которые взаимодействуют с конечными системами Интернета. На рис. 1.10 элементы ядра сети выделены жирными линиями.

1.3.1. Коммутация пакетов

Конечные системы обмениваются друг с другом **сообщениями**, используя сетевые приложения. Сообщения могут содержать все что угодно, любую информацию, которую разработчик приложения пожелает туда поместить. Иногда они выполняют функции управления (например, сообщение «Привет» в нашем примере с рукопожатием на рис. 1.2), иногда содержат данные, например почтовое сообщение, рисунок в формате JPEG, либо аудиофайл в формате MP3. Для того чтобы отослать сообщение от конечной системы-источника в конечную систему-приемник, оно разбивается на более мелкие порции данных, называемые **пакетами**. На пути от источника к приемнику каждый пакет проходит через линии связи и **коммутаторы** (среди которых основными типами являются **маршрутизаторы** и **коммутаторы канального уровня**). Пакеты передаются по каждой линии связи с *максимальной* скоростью, которую может обеспечить данная линия. Поэтому, если исходная конечная система либо коммутатор отсылает пакет длиной L бит через соединение со скоростью R бит/с, то время передачи пакета равно L/R секунд.

Передача с промежуточным накоплением

Большинство коммутаторов пакетов используют так называемую **передачу с промежуточным накоплением**. Промежуточное накопление означает, что коммутатор пакетов должен сначала принять пакет целиком перед тем, как он начнет передавать в выходную линию связи его первый бит.

Чтобы изучить передачу с накоплением более детально, рассмотрим простую сеть, состоящую из двух конечных систем, соединенных одним маршрутизатором, как показано на рис 1.11.

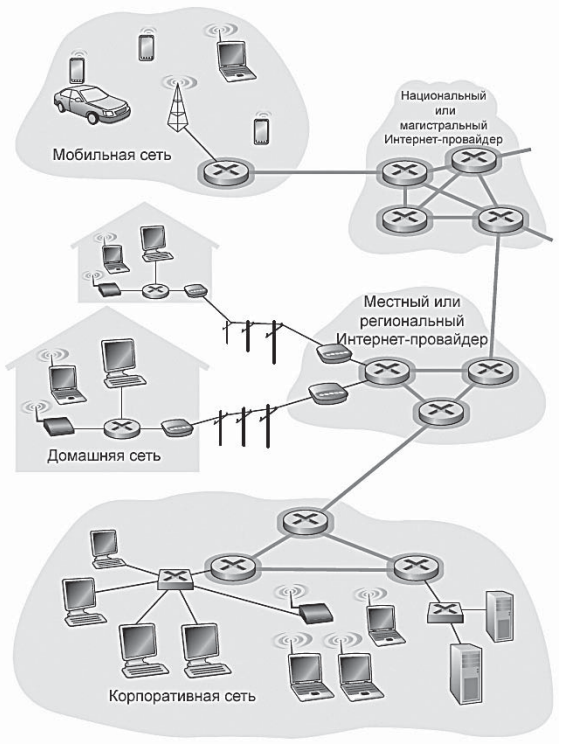


Рис. 1.10. Ядро сети

Рис. 1.11. Коммутация с промежуточным накоплением

Каждый маршрутизатор обычно имеет несколько соединений, так как его задача заключается в том, чтобы перенаправить входящий пакет в исходящее соединение; в данном примере задача маршрутизатора довольно проста: передача пакета из одной линии связи (входящей) в единственную исходящую. Здесь источник отправляет три пакета, каждый из них содержит L бит. В момент времени, показанный на рис. 1.11, источник отправил часть пакета 1, и она уже прибыла на маршрутизатор. Так как маршрутизатор работает по методу передачи с промежуточным накоплением, в данный момент он не может отправить биты, которые получил; вместо этого он должен сначала их сохранить (буферизовать). Только после того, как маршрутизатор получит все биты пакета, он может начать передачу (перенаправление) пакета в исходящую линию связи. Чтобы лучше разобраться, давайте подсчитаем количество времени, которое пройдет от того момента, когда источник начал отправку пакетов, до того момента, когда приемник получил весь пакет целиком. (Здесь мы пренебрегаем задержкой распространения — временем, которое требуется битам для прохождения по проводнику со скоростью, приблизительно равной скорости света, — что будет обсуждено в разделе 1.4). Источник начинает передачу в момент времени 0; в момент времени L/R секунд источник завершил передачу всего пакета, и пакет полностью получен и сохранен в маршрутизаторе (задержка распространения нулевая). В момент времени L/R, так как маршрутизатор только что получил весь пакет, он может начать его передачу в выходную линию связи в направлении адресата; в момент времени $2L/R$ маршрутизатор завершает передачу всего пакета, и тот полностью прибывает к месту назначения. Таким образом, общая задержка будет равна $2L/R$. Если коммутатор перенаправлял бы биты сразу же, как они прибывают, не дожидаясь получения всего пакета, общая задержка была бы равна L/R, так как не тратилось бы время на хранение битов в маршрутизаторе. Но маршрутизаторам необходимо получать, хранить и обрабатывать пакет перед тем, как его отправить.

Теперь давайте посчитаем время от того момента, когда источник начинает отсылать первый пакет, до того момента, когда приемник получит все три. Как и раньше, в момент времени L/R маршрутизатор начинает передавать первый пакет. Но в это же время L/R источник начинает отсылать второй пакет, так как первый он уже отправил целиком. Таким образом, в момент времени $2L/R$ приемник получит первый пакет, а маршрутизатор получит второй. Аналогично, в момент времени $3L/R$ приемник получит первые два пакета, а маршрутизатор получит третий. Наконец, в момент времени $4L/R$ приемник получит все четыре пакета!

Давайте теперь рассмотрим общий случай отправки одного пакета от источника к приемнику по пути, состоящему из N соединений, имеющих каждый скорость R (то есть между источником и приемником $N-1$ маршрутизатор). Применяя тот же метод, что и раньше, мы увидим, что общая (сквозная) задержка прохождения от источника к приемнику равна

$$d_{\text{сквозн}} = N\frac{L}{R} \tag{1.1}$$

Теперь вы можете сами посчитать время задержки, которое необходимо для передачи P пакетов через сеть из N соединений.

Задержки ожидания и потери пакетов

Каждый коммутатор пакетов может иметь несколько соединений. Для каждого соединения у коммутатора есть **выходной буфер** (также называемый **выходной очередью**), в котором будут храниться пакеты для отправки в данную линию связи. Выходные буферы играют ключевую роль в коммутации пакетов. Если, например, прибывающий пакет нужно отправить в линию связи, но она занята передачей другого пакета, то прибывающий пакет должен встать в очередь в выходном буфере. Таким образом, в дополнение к **задержкам накопления**, пакеты, вставая в очередь, испытывают задержки ожидания. Эти задержки являются переменными и зависят от степени перегруженности сети. Так как размер буфера маршрутизатора не бесконечен, может наступить момент, когда он полностью заполнен прибывшими пакетами, а пакеты все поступают и поступают. В таком случае происходит **потеря пакетов**. Отбрасывается либо один из прибывающих пакетов, либо один из тех, которые уже находятся в очереди.

На рис. 1.12 представлена простая сеть с коммутацией пакетов. Так же, как и на рис. 1.11, пакеты изображены в виде трехмерных плиток. Ширина плитки представляет число битов в пакете. На этом рисунке все пакеты одной ширины, то есть имеют один размер. Предположим, что хосты А и Б отсылают пакеты на хост Д. Хосты А и Б сначала отсылают пакеты по Ethernet-каналу (10 Мбит/с) на первый маршрутизатор. Маршрутизатор перенаправляет эти пакеты в канал со скоростью передачи 1,5 Мбит/с. Если в короткий промежуток времени скорость прибытия пакетов на маршрутизатор (в бит/с) превышает 1,5 Мбит/с, на нем происходит перегрузка, и пакеты встают в очередь перед тем, как их отправят в выходную линию связи. Например, если хосты А и Б каждый отправят серию из пяти пакетов в одно и то же время, то боль-

шинство из этих пакетов будут некоторое время ждать очереди. Ситуация на самом деле полностью аналогична многим каждодневным ситуациям из жизни, например, когда мы стоим в очереди перед банкоматом либо в кассе магазина. В разделе 1.4 мы подробнее рассмотрим задержку ожидания.

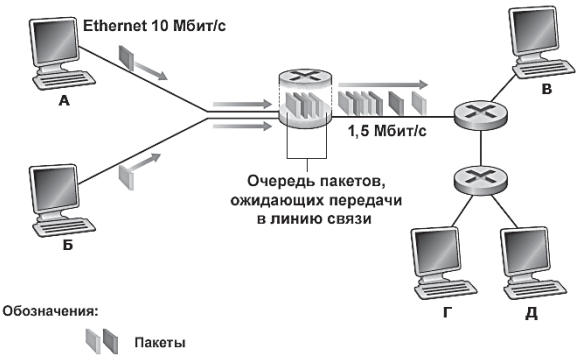


Рис. 1.12 Коммутация пакетов

Таблица маршрутизации и протоколы маршрутизации

Ранее мы упомянули, что маршрутизатор принимает пакеты, поступающие на одно из его соединений, и перенаправляет их на другое. Но как маршрутизатор определяет, куда направить пакет? На самом деле в различных видах компьютерных сетей перенаправление пакетов происходит различными методами. Здесь мы вкратце опишем, как это делается в Интернете.

Каждая конечная система в Интернете имеет свой адрес, называемый IP-адресом. Когда одна конечная система (источник) пытается отправить пакет на другую, то она включает в заголовок пакета IP-адрес места назначения. Как и в случае с почтовыми адресами, он имеет иерархическую структуру. Когда пакет прибывает на маршрутизатор, находящийся в сети, тот проверяет часть пакета, содержащую адрес места назначения и в соответствии с ним направляет пакет по необходимому пути. Если углубиться дальше, каждый маршрутизатор имеет **таблицу маршрутизации**, которая ставит в соответствие адреса места назначения (либо часть адресов места назначения) с исходящими соединениями маршрутизатора. Когда пакет прибывает на маршрутизатор, тот проверяет адрес и находит в таблице маршрутизации соответствующее исходящее соединение, куда и направляет данный пакет.

Процесс маршрутизации от источника к приемнику аналогичен ситуации, когда автолюбитель вместо того, чтобы воспользоваться картой, предпочитает спросить, как проехать. Предположим, например, Джо, который живет в Филадельфии, хочет навестить бабушку, живущую в другом штате по адресу 156, Лэйксайд-Драйв, Орландо, Флорида. Первым делом он едет на ближайшую заправочную станцию и спрашивает, как добраться до дома 156, Лэйксайд-Драйв, Орландо, Флорида. Сотрудник заправочной станции извлекает из адреса часть со словом «Флорида» и говорит Джо, что ему нужно на шоссе I-95 Юг, которое проходит рядом с АЗС. Также он сообщает Джо, что как только тот въедет во Флориду, ему следует там кого-нибудь спросить, куда направляться дальше. Джо едет по I-95 Юг, пока не добирается до местечка Джексонвилл во Флориде, где спрашивает еще одного работника заправочной станции, как проехать дальше. Работник выделяет слово «Орландо» из адреса и сообщает Джо, что ему нужно продолжать движение по шоссе I-95 до Дайтона-Бич, а там уже спросить еще кого-нибудь. Работник следующей АЗС в Дайтона-Бич говорит Джо, что ему нужно ехать по I-4, и он попадет прямо в Орландо. Джо едет по I-4, попадает в Орландо и опять направляется на заправочную станцию. На этот раз сотрудник выделяет часть адреса «Лэйксайд-Драйв» и указывает дорогу к шоссе «Лэйксайд-Драйв». Как только Джо выезжает на «Лэйксайд-Драйв», он спрашивает ребенка с велосипедом, как добраться до места назначения. Ребенок выделяет в адресе часть «156» и указывает Джо на нужный ему дом. Наконец Джо достигает места назначения. В указанной выше аналогии все работники заправочных станций, а также ребенок на велосипеде являются аналогами маршрутизаторов.

Мы только что узнали, что маршрутизатор использует адрес места назначения в пакете, чтобы найти его в таблице маршрутизации и определить подходящее выходное соединение. Но возникает вопрос: откуда берутся таблицы маршрутизации? Конфигурируются ли они вручную на каждом из маршрутизаторов либо в Интернете используются какие-то автоматические процедуры? Это мы изучим подробнее в главе 4. Но чтобы удовлетворить ваше любопытство прямо сейчас, мы отметим, что в Интернете существуют специальные **протоколы маршрутизации**, которые используются для автоматической генерации таблиц маршрутизации. Протокол маршрутизации может, например, определять кратчайший путь от маршрутизатора до любого места назначения и использовать эти результаты для конфигурации таблиц в маршрутизаторах.

Вы действительно хотите узнать маршрут, по которому пакеты проходят через Интернет от источника к приемнику? Тогда мы приглашаем вас попробовать поработать с программой Traceroute. Посетите сайт **www.traceroute.org**, выберите источник в определенной стране и проследите маршрут от этого источника до вашего компьютера. (Обсуждение Traceroute см. в разделе 1.4.)

1.3.2. Коммутация каналов

Существует два фундаментальных подхода к методам передачи данных по сетям: **коммутация каналов** и **коммутация пакетов**. Рассмотрев сети коммутации пакетов в предыдущем разделе, давайте обратим наше внимание на сети с коммутацией каналов.

В сетях с коммутацией каналов ресурсы, необходимые для обеспечения взаимодействия между конечными системами (буфер, скорость передачи), *резервируются* на время соединения между системами. В сетях с коммутацией пакетов эти ресурсы *не резервируются*; сеанс взаимодействия использует ресурсы по запросу и как следствие может ожидать (то есть вставать в очередь) доступа к освободившемуся соединению. В качестве простой аналогии, рассмотрим пример с двумя ресторанами, в одном из которых требуется предварительное резервирование столиков, а в другом приглашаются все желающие, без резервирования. Чтобы заказать обед в первом ресторане, мы должны предварительно договориться об этом по телефону, прежде чем туда идти. Но зато, когда мы прибудем в ресторан, мы можем немедленно занять резервированное место и заказать обед. Чтобы пообедать во втором ресторане, нам не нужно туда звонить, но придя, мы можем обнаружить, что все столики заняты, и нам придется ожидать освободившегося места.

Традиционные телефонные сети являются примером сетей с коммутацией каналов. Посмотрим, что происходит, когда один человек желает послать информацию (голосовое или факсимильное сообщение) другому через телефонную сеть. Перед тем как отправитель посылает информацию, в сети должно установиться соединение между получателем и отправителем. Это *добросовестное* соединение, которое поддерживается коммутаторами на всем его протяжении. На жаргоне телефонии такое соединение называется **каналом**. Когда устанавливается в сети канал, также резервируется постоянная скорость передачи в линии соединения (которая представляет собой часть пропускной способности этого соединения) на все время соединения. Так как данная скорость пе-

редачи резервируется для этого соединения между передатчиком и приемником, то в дальнейшем данные могут передаваться с *гарантированной* постоянной скоростью.

На рис.1.13 показана сеть с коммутацией каналов, в которой четыре коммутатора соединены между собой линиями связи. Каждая из этих линий связи имеет четыре канала, то есть поддерживает одновременно четыре соединения. Хосты (персональные компьютеры или рабочие станции) соединены напрямую с одним из коммутаторов. Когда два хоста хотят обмениваться информацией, между ними устанавливается выделенное **сквозное соединение**. Таким образом, для того чтобы хост А мог взаимодействовать с хостом Б, нужно зарезервировать один канал на каждой из двух линий связи. В данном примере выделенное соединение использует второй канал первой линии связи и четвертый канал второй линии связи. Так как в линиях связи содержится по четыре канала, то для каждой линии, используемой в выделенном соединении, используется полоса пропускания, равная полосы пропускания линии связи, на все время соединения. Таким образом, например, если каждая линия связи между соседними коммутаторами имеет скорость передачи 1 Мбит/с, то любому из сквозных коммутируемых соединений доступна скорость 250 Кбит/с.

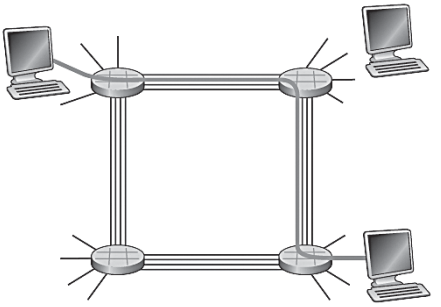


Рис. 1.13. Простейшая сеть с коммутацией каналов, состоящая из четырех линий связи и четырех коммутаторов

В противоположность этому рассмотрим, что происходит в сетях с коммутацией пакетов, таких как Интернет, когда один хост пытается отправить пакет другому хосту. Как и в случае с коммутацией каналов, пакет также проходит через последовательность линий связи. Но существенное отличие здесь в том, что пакет отсылается в сеть без резервирования каких-либо ресурсов линий соединения. Если одна из линий связи перегружена вследствие передачи по ней других пакетов в это же

время, то пакет вынужден ожидать в буфере на передающем конце соединения и испытывать, таким образом, задержку. Отсюда следует, Интернет прикладывает все усилия для своевременной доставки пакетов, но не может ее гарантировать.

Мультиплексирование в сетях с коммутацией каналов

Канал в линии связи организуется при помощи **мультиплексирования с частотным разделением** (frequency-division multiplexing, **FDM**), либо с **временным разделением** (time-division multiplexing, **TDM**). В случае с частотным разделением спектр частот всей линии делится между установленными каналами, то есть каждому каналу на все время соединения отводится определенная полоса частот. В телефонных сетях, например, эта полоса имеет ширину 4 кГц (то есть 4000 Гц или 4000 циклов в секунду). Эта ширина называется **полосой пропускания**. Радиостанции FM-диапазона тоже используют принцип частотного разделения и делят частоты от 88 МГц до 108 МГц.

В случае с временным разделением время разбивается на фиксированные промежутки, называемые *кадрами*, а кадры, в свою очередь, делятся на фиксированное число слотов. Когда в сети устанавливается соединение, то ему выделяется один временно́й слот в каждом кадре. Эти временны́е слоты используются для передачи данных только для одного соединения.

На рис. 1.14 показан принцип функционирования линий связи, поддерживающей до 4 каналов для случаев мультиплексирования с разделением по частоте и по времени. Для частотного мультиплексирования весь частотный диапазон разделен на четыре полосы по 4 кГц каждая. Для случая с временным разделением диапазон времени делится на кадры, содержащие по 4 слота; каждому каналу связи назначен один и тот же слот в сменяющихся кадрах. Скорость передачи данных в этом случае равна произведению частоты смены кадров и числа битов в слоте. Например, если по линии передается 8000 кадров в секунду, а слот содержит 8 бит, то скорость каждого канала связи составит 64 Кбит/с.

Сторонники коммутации пакетов всегда утверждали, что коммутация каналов является очень расточительной технологией, потому что выделенные каналы вынуждены простаивать во время так называемых **периодов тишины**. Например, когда в ходе телефонного разговора человек перестает говорить, то простаивающие сетевые ресурсы (частотная полоса или временные диапазоны) не могут быть использованы для

других соединений. В качестве еще одного примера представим себе врача-рентгенолога, который использует сеть с коммутацией каналов для удаленного доступа к базе рентгеновских снимков. Рентгенолог устанавливает соединение, запрашивает снимок, изучает его, затем запрашивает снова. Во время изучения снимка и размышлений сетевые ресурсы для соединения выделены, но не используются, а значит расходуются впустую. Также сторонники коммутации пакетов любят подчеркивать тот факт, что создание соединений и резервирование каналов — непростая задача, требующая сложного программного обеспечения для управления и синхронизации.

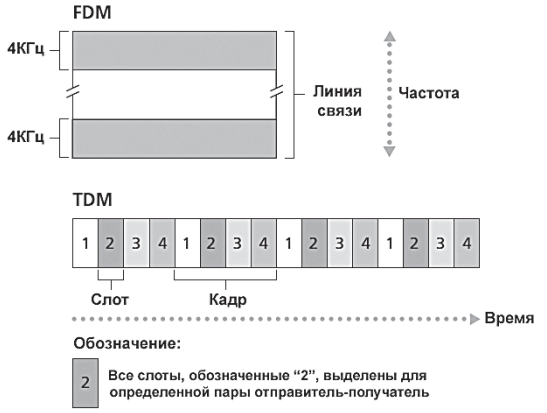


Рис. 1.14. При частотном разделении каждый канал постоянно занимает свою полосу частот. При временно́м канал периодически использует всю полосу пропускания (в выделенные ему временные слоты)

Перед тем как завершить обсуждение коммутации каналов, давайте рассмотрим числовой пример, который поможет еще лучше понять суть технологии коммутации пакетов. Давайте определим, сколько времени понадобится, чтобы отослать файл размером 640 000 бит с хоста А на хост Б в сети с коммутацией каналов. Предположим, что все линии связи в сети используют метод временного разделения с 24 слотами в кадре, и скорость передачи составляет 1,536 Мбит/с. Также предположим, что на установление соединения перед тем, как хост А начнет передавать файл, уйдет 500 мс. Каково тогда время отправки файла? Каждый канал имеет скорость передачи, равную 1,536 Мбит/с : 24 = 64 Кбит/с. Поэтому время передачи файла равно 640 000 бит : 64 Кбит/с = 10 с. К этим 10 с прибавляем время на установление соединения в канале и получаем 10,5 с — общее время передачи файла. Отметим, что время передачи не зависит от числа линий связи и бу-

дет равно примерно 10 с и в случае с одной линией, и в случае со ста линиями (на самом деле реальная задержка при передаче источник-приемник включает еще и задержку распространения, которую опишем в разделе 1.4).

Коммутация пакетов или коммутация каналов

Описав методы коммутации пакетов и коммутации каналов, давайте теперь сравним их. Противники коммутации пакетов часто утверждают, что она не подходит для приложений, работающих в реальном времени, например телефонии и видеоконференций, из-за того, что задержки передачи от источника к приемнику непостоянны и непредсказуемы (имеются в виду задержки ожидания). Сторонники же пакетной коммутации возражают и говорят, что данный метод дает возможность лучшего разделения емкости канала между пользователями, чем коммутация каналов. Также он проще, эффективнее и менее затратен. Очень интересное обсуждение коммутации пакетов против коммутации каналов можно посмотреть вот тут[349]. Вообще люди, которые не любят резервировать столики в ресторане, предпочитают коммутацию пакетов, а не коммутацию каналов.

Почему коммутация пакетов является более эффективной? Посмотрим на простой пример. Предположим, что пользователи делят канал 1 Мбит/с. Допустим также, что каждый пользователь чередует периоды активности, во время которых передает данные с постоянной скоростью 100 Кбит/с, и периоды бездействия, во время которых никаких данных не передает. Предположим также, что пользователь активен 10% всего времени, а все остальные 90%, например, пьет кофе. В случае с коммутацией каналов для *каждого* пользователя должно быть *зарезервировано* 100 Кбит/с на все время сеанса связи. Например, при мультиплексировании с временным разделением, если односекундный кадр делится на 10 временных слотов по 10 мс каждый, то каждому пользователю выделится 1 временной слот в каждом кадре.

Таким образом, соединение линии связи в сети с коммутацией каналов может поддерживать только 10 одновременных пользователей (равно 1 Мбит/с:100 Кбит/с). В случае с коммутацией пакетов вероятность того, что определенный пользователь активен, равна 0,1 (то есть 10%). Если в сети 35 пользователей, то вероятность того, что 11 из них одновременно активны, равна приблизительно 0,0004 (как получено данное значение, узнаете в упражнении 8). Когда одновременно активно 10 или меньше пользователей (что случается с вероятностью

0,9996), то общая скорость поступления данных меньше либо равна 1 Мбит/с, то есть не превышает пропускной способности линии связи, и в этом случае пользовательские пакеты проходят по ней без задержек, как в случае с коммутацией каналов. Когда в сети будет более чем 10 активных пользователей одновременно, то общая скорость поступления пакетов превысит пропускную способность линии, и начнет увеличиваться выходная очередь пакетов (она будет продолжать расти до тех пор, пока общая скорость входного потока не понизится до 1 Мбит/с). Поскольку вероятность одновременной активности 10 пользователей в данном примере очень мала, то метод с коммутацией пакетов покажет ту же производительность, что и метод с коммутацией каналов, *но при этом разрешит работать в сети втрое бóльшему количеству пользователей*.

Теперь давайте рассмотрим еще один простой пример. Предположим, есть 10 пользователей в сети, и один из них внезапно генерирует 1000 1000-битных пакетов, в то время как другие пользователи не являются активными, не генерируя ничего. В сети с коммутацией каналов и временным разделением, когда каждый кадр делится на 10 слотов, и в каждом слоте содержится 1000 бит, активный пользователь может использовать один временной слот на кадр, чтобы передать данные, в то время как оставшиеся 9 временных слотов в каждом кадре будут простаивать. Тогда, чтобы передать весь миллион бит активного пользователя, уйдет 10 секунд. В случае с коммутацией пакетов активный пользователь сможет непрерывно посылать свои пакеты в канал связи с максимальной скоростью передачи в 1 Мбит/с, так как больше никто не передает пакеты в сеть. В этом случае все данные активного пользователя будут переданы за 1 с.

Указанная пара примеров демонстрирует нам преимущество в производительности метода коммутации пакетов перед методом коммутации каналов. Это также подчеркивает серьезную разницу между двумя формами того, как разделяется на несколько потоков данных скорость передачи в линии связи. Коммутация каналов резервирует емкость независимо от запросов, и поэтому зарезервированное, но неиспользуемое время остается потраченным впустую. Коммутация пакетов, с другой стороны, использует канал *по запросу*. Скорость передачи делится среди тех пользователей, кому нужно передать пакеты по каналу связи.

Хотя в нынешних телекоммуникационных системах работают обе технологии коммутации, тенденция направлена в сторону коммутации

пакетов. На нее постепенно переходят даже многие телефонные сети, и очень часто используют данный метод коммутации для межконтинентальных телефонных вызовов.

1.3.3. Сеть сетей

Мы видели ранее, что конечные системы (персональные компьютеры, смартфоны, веб-серверы, почтовые серверы и так далее) соединяются с Интернетом, пользуясь услугами Интернет-провайдеров. Интернет-провайдер обеспечивает либо проводной, либо беспроводной доступ, используя ряд технологий, включающих DSL, кабельный доступ, FTTH, Wi-Fi, а также мобильные сотовые сети. Заметим, что Интернет-провайдером необязательно бывают телефонные компании либо компании кабельных сетей; им может быть, например, университет (предоставляющий доступ в Интернет своим студентам и персоналу факультетов) либо коммерческая компания (обеспечивающая доступ в Интернет своим сотрудникам). Но соединение конечных пользователей, провайдеров контента, сетей доступа, Интернет-провайдеров — это только малая часть сложной мозаики, состоящей из миллионов конечных систем, которые образуют Интернет. Чтобы закончить эту мозаику, сети доступа Интернет-провайдеров должны быть сами взаимосвязаны между собой. Это делается путем создания *сети сетей* — понимание этой фразы является ключом к пониманию Интернета.

По прошествии многих лет сеть сетей, которая сформировала Интернет, развилась в очень сложную структуру. Развитие это во многом определялось не фактором производительности, а экономикой и национальной политикой. Для того чтобы лучше понять структуру сегодняшнего Интернета, давайте построим по шагам серию сетевых структур, на каждом шаге приближаясь к сложному Интернету, который мы имеем сегодня. Вспомним, что главной целью является связать все сети доступа таким образом, что все конечные системы могли отсылать пакеты друг другу. Самым простым и достаточно наивным подходом было бы соединить каждую сеть доступа *напрямую* со всеми другими сетями доступа. Получившаяся в результате сеть была бы очень дорогая, так как потребовала бы отдельной линии связи для каждого соединения сети доступа с сотнями тысяч других таких же сетей по всему миру.

Первый вариант, *Сетевая Структура 1,* объединяет все сети доступа Интернет-провайдеров с *одним магистральным Интернет-провайдером*.

Наш (воображаемый) магистральный провайдер — это сеть маршрутизаторов и коммуникационных каналов связи, которая охватывает не только весь земной шар, но и имеет по крайней мере по одному маршрутизатору для каждой из сотен тысяч сетей доступа. Конечно, было бы очень затратно для провайдера построить такую широкую сеть. Чтобы получать прибыль, провайдер должен брать плату за соединение с каждой из сетей доступа, и ее размер должен отражать (но необязательно пропорционально) объем трафика, которым сеть доступа обменивается с провайдером. Так как сеть доступа оплачивает прохождение трафика провайдеру, то сеть доступа можно назвать **заказчиком**, а провайдера **поставщиком**.

Если какая-то компания строит такую глобальную сеть, которая приносит прибыль, вполне естественно, что другие компании захотят тоже стать магистральными провайдерами. Мы приходим к *Сетевой Структуре 2*, состоящей из сотен тысяч сетей доступа и *нескольких* магистральных провайдеров. Естественно, сети доступа как заказчики будут предпочитать Сетевую Структуру 2, а не 1, так как в этом случае у них есть выбор между провайдерами, которых они могут предпочесть и посчитать более выгодными по цене и предлагаемым услугам. Отметим, что сети провайдеров в данном случае должны быть связаны между собой, а иначе клиенты разных магистральных провайдеров не смогли бы взаимодействовать друг с другом.

Сетевая Структура 2 представляет собой двухуровневую иерархию, в которой магистральные провайдеры занимают верхний уровень, а сети доступа лежат на нижнем. Такая структура предполагает, что магистральный провайдер не только имеет возможность соединяться со всеми сетями доступа, но и находит экономически выгодным делать это. Несмотря на то, что магистральные провайдеры имеют очень внушительное сетевое покрытие и связаны с очень многими сетями доступа, не все из них присутствуют в любом городе на планете. Вместо этого существуют **региональные Интернет-провайдеры**, которые осуществляют подключение сетей доступа в каждом регионе, а те, в свою очередь, соединены с **Интернет-провайдерами первого уровня**. Существует приблизительно дюжина провайдеров первого уровня, среди которых Level 3 Communications, ATMT, Sprint и NTT. Интересно отметить, что ни один из провайдеров прямо не заявляет о своей принадлежности к первому уровню. Как говорится, если вы сомневаетесь, входите ли в узкий круг — то, вероятно, вы в него не входите.

Возвращаясь к этой сети сетей, заметим, что конкурируют не только провайдеры первого уровня, но и многочисленные местные Интернет-провайдеры в своем регионе. В такой иерархической структуре каждая сеть доступа вынуждена платить за соединение местному Интернет-провайдеру, а каждый местный Интернет-провайдер — провайдеру первого уровня (сеть доступа может быть соединена напрямую с провайдером первого уровня и в этом случае платить за подключение непосредственно ему). Таким образом, на каждом из уровней иерархии существуют отношения заказчик-поставщик. Заметим также, что провайдеры первого уровня не платят никому, так как они находятся на вершине иерархии. На самом деле структура может быть еще сложнее, например, в некоторых регионах существуют более крупные местные Интернет-провайдеры (иногда охватывающие целую страну), к которым подсоединяются более мелкие местные провайдеры, а крупные соединяются с провайдерами первого уровня. Например, в Китае в каждом городе есть сети доступа, которые соединяются с провайдерами провинции, а те, в свою очередь, с национальными Интернет-провайдерами, соединяющимися с провайдерами первого уровня[636]. Таким образом, мы пришли к *Сетевой Структуре 3*, которая является все еще неточным приближением сегодняшнего Интернета.

Для построения сетевой структуры, которая наиболее близко отражает сегодняшний Интернет, мы должны добавить еще точки присутствия (Points of Presents или PoP), пиринг, точки обмена трафиком (IXP), а также возможность использования многоинтерфейсного режима. Точки присутствия существуют на всех уровнях иерархии, исключая нижний (сети доступа). **Точка присутствия** — это группа из одного или нескольких маршрутизаторов (расположенных в одном и том же месте) сети провайдера, к которым могут подключаться маршрутизаторы сети заказчика. Чтобы подключиться к Интернет-провайдеру, заказчик может арендовать высокоскоростные линии связи какой-нибудь телекоммуникационной компании для соединения своих маршрутизаторов с маршрутизаторами провайдера в точке присутствия. Любой Интернет-провайдер (за исключением провайдера первого уровня) может выбрать так называемое **многоинтерфейсное** подключение, то есть соединение с двумя или более провайдерами верхнего уровня. Так, например, сеть доступа может иметь многоинтерфейсное подключение к двум местным Интернет-провайдерам или, как вариант, к двум местным и к одному провайдеру первого уровня. Аналогично, местный Интернет-провайдер может иметь многоинтерфейсное подключение к нескольким провайдерам первого уровня. Используя такое подключение, провайдер гаранти-

рует для себя отправку и получение пакетов, если у одного из его провайдеров проблемы на линии связи.

Как мы уже выяснили, заказчики платят Интернет-провайдерам за доступ в сеть, и размер этой платы, как правило, отражает объем трафика, которым заказчик обменивается с поставщиком. Для уменьшения затрат пара соседствующих Интернет-провайдеров одного уровня иерархии может установить между собой, так называемое **пиринговое** соединение, то есть соединить свои сети напрямую таким образом, чтобы трафик между ними не шел через промежуточные каналы связи. Обычно по соглашению сторон такое соединение является бесплатным. Как уже упоминалось, провайдеры первого уровня также устанавливают между собой пиринговые соединения. Обсудить пиринг и взаимоотношения провайдеров, заказчиков Интернет-услуг можно здесь[643]. Компании, не являющиеся Интернет-провайдерами (так называемые третьи стороны), могут создавать **точки обмена Интернет-трафиком** (Internet Exchange Point, **IXP**), обычно в отдельно стоящем здании, своими собственными коммутаторами. Через такие точки провайдеры могут устанавливать пиринговое взаимодействие между собой. На сегодняшний день в Интернете существует примерно 300 точек обмена трафиком[31]. Таким образом, мы приходим к нашей Сетевой Структуре 4, состоящей из точек доступа, региональных провайдеров, провайдеров первого уровня, точек присутствия, многоинтерфейсного режима, пиринга и точек обмена трафиком (IXP).

Наконец, мы пришли к *Сетевой Структуре 5*, которая описывает сегодняшний Интернет (по состоянию 2012 года). Сетевая структура, показанная на рис. 1.15, происходит из Структуры 4 с добавлением **сетей провайдеров контента**. Одним из ярких представителей таких провайдеров контента является компания Google. На момент написания данной книги, по приблизительным оценкам, Google имела от 30 до 50 центров обработки данных, размещенных по всей Северной Америке, Европе, Азии, Южной Америке и Австралии. Многие из этих центров включали в себя сотни серверов, а некоторые и до сотни тысяч. Все центры обработки данных компании Google взаимосвязаны частной сетью Google (TCP/IP сетью), которая охватывает весь земной шар, но, тем не менее, отделена от публичного Интернета. Важно заметить, что трафик в частной сети Google идет только между серверами Google. Как мы видим на рис. 1.15, частная сеть Google пытается выступать в качестве провайдера первого уровня, устанавливая пиринговое соединение (бесплатное) с Интернет-провайдерами нижнего уровня, либо связываясь с ними на-

прямую, либо через точки обмена трафиком[303]. Однако из-за того, что подключиться ко многим сетям доступа можно только через провайдера первого уровня, Google также устанавливает соединение с провайдерами первого уровня и платит им за трафик, которым с ними обменивается. Создавая собственные сети, провайдер контента не только сокращает расходы, связанные с подключением провайдера более высокого уровня, но также получает возможность оптимизировать управление своими сервисами, предоставляемыми конечным пользователям. Более подробно инфраструктуру сети Google мы опишем в разделе 7.2.4.

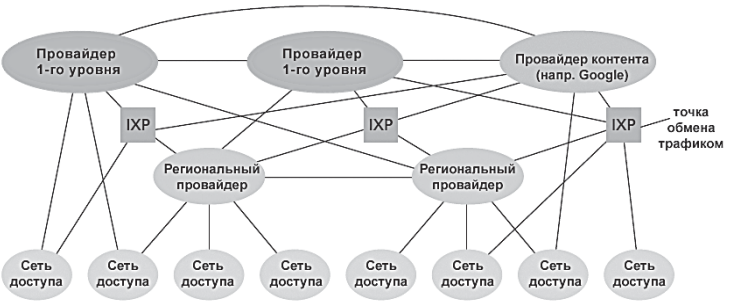


Рис. 1.15. Взаимодействие сетей

Резюмируя все вышесказанное, заметим, что сегодняшний Интернет — сеть сетей — это сложная структура, состоящая примерно из дюжины Интернет-провайдеров первого уровня и сотен тысяч провайдеров нижних уровней. Некоторые из них охватывают небольшие географические области, а некоторые — многие континенты и океаны. Провайдеры нижнего уровня соединяются с провайдерами верхнего уровня, а также взаимодействуют друг с другом. Пользователи и провайдеры контента являются заказчиками провайдеров нижних уровней, а те, в свою очередь, заказчиками провайдеров верхних уровней. В последние годы большинство провайдеров контента создали свои собственные сети, которые соединяются с сетями провайдеров нижних уровней там, где это возможно.

1.4. Задержки, потери и пропускная способность в сетях с коммутацией пакетов

Как мы упомянули в разделе 1.1, Интернет можно рассматривать как инфраструктуру, которая предоставляет службы распределенным приложениям, выполняющимся на конечных системах. В идеале мы бы,

конечно, хотели, чтобы Интернет-службы были в состоянии передавать данные между двумя системами без какой-либо потери. Но в реальности это недостижимо. На самом деле при передаче данных в компьютерных сетях происходят задержки и потери пакетов. В связи с этим существует понятие пропускной способности сети — количество передаваемых в секунду данных, которое способно пропустить сеть. С одной стороны, объясняемые законами физики задержки и потери вызывают сложности, но с другой, подобные проблемы являются источником множества интересных тем для курсов по компьютерным сетям и мотивируют тысячи ученых в этой области. В данном разделе мы начнем изучать и рассчитывать задержки, потери и пропускную способность в компьютерных сетях.

1.4.1. Обзор задержек в сетях с коммутацией пакетов

Напомним, что пакет начинает свой путь на хосте-источнике, проходит через последовательность маршрутизаторов и заканчивает его на хосте-приемнике. Проходя путь от одного узла (хоста или маршрутизатора) до другого, пакет испытывает на *каждом* из этих узлов несколько видов задержек: **задержка обработки в узле, задержка ожидания, задержка передачи и задержка распространения**; сумма этих задержек называется общей узловой задержкой. Производительность многих Интернет-приложений, таких, как поисковая система, веб-браузеры, почтовые клиенты, системы мгновенных сообщений, IP-телефония напрямую зависит от сетевых задержек. Чтобы глубоко понять процесс коммутации пакетов и общего функционирования компьютерных сетей необходимо четко представлять природу этих задержек и их важность.

Виды задержек

Давайте изучим эти задержки на примере участка сети, представленного на рис. 1.16. Частью маршрута пакета от источника до места назначения является фрагмент сети от исходящего узла через маршрутизатор А к маршрутизатору Б. Нашей целью является охарактеризовать узловую задержку на маршрутизаторе А. Отметим, что маршрутизатор А имеет исходящую линию связи до маршрутизатора Б. Эта линия связи в своем начале имеет входной буфер маршрутизатора А (там, где хранятся пакеты, находящиеся в очереди). Когда пакет прибывает с передающего узла на маршрутизатор Ас передающего узла, он проверяет заголовок пакета для того, чтобы определить исходящую линию связи

и направить в нее пакет. В нашем примере исходящей линией связи для пакета будет являться та, которая ведет к маршрутизатору Б.

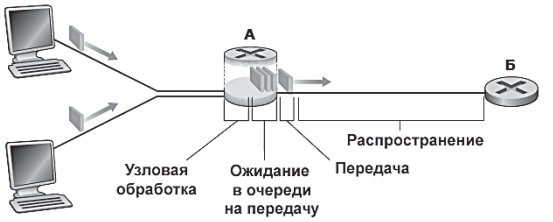


Рис. 1.16. Узловая задержка на маршрутизаторе А

Пакет может быть передан в линию связи лишь при условии, что по ней не передаются или не находятся в очереди другие пакеты; если же линия связи в данный момент занята либо есть ожидающие в очереди пакеты, то в нее встают и вновь прибывающие.

Задержка обработки

Время, необходимое на проверку заголовка пакета и определение дальнейшего его маршрута, является частью **задержки обработки**. Данная задержка может включать также затраты на проверку ошибок, происходящих из-за искажения отдельных битов сообщения при передаче его от передающего узла на маршрутизатор А. Обычное время задержки в типичных современных высокоскоростных маршрутизаторах составляет порядка нескольких микросекунд или менее. После такой обработки в узле маршрутизатор направляет пакет в очередь линии связи с маршрутизатором Б. (Подробнее о том, как функционируют маршрутизаторы, мы расскажем в главе 4.)

Задержка ожидания

Находясь в очереди, пакет испытывает **задержку ожидания**, так как он ждет, пока его передадут дальше в линию связи. Время задержки ожидания для определенного пакета будет зависеть от количества пакетов, прибывших до него и стоящих в очереди на передачу в линию связи. Если же очередь пуста и нет пакетов для передачи, то задержка ожидания для нашего пакета будет равна нулю. В случае же большого объема трафика задержки ожидания могут быть достаточно продолжительными. Вскоре мы увидим, что количество пакетов, находящихся в очереди к моменту прибытия нового пакета, зависит от интенсивности

и природы трафика прибывающих пакетов. На практике задержки ожидания обычно составляют от нескольких микросекунд до нескольких миллисекунд.

Задержка передачи

Предположим, что пакеты передаются в линию связи в порядке очередности (первый пришел — первого обслужили), как это распространено в сетях с коммутацией пакетов. Тогда наш пакет будет передан только после того, как закончится передача всех тех, которые прибыли перед ним. Пусть длина пакета равна L бит, а скорость передачи по линии связи от маршрутизатора А к маршрутизатору Б равна R бит/с. Например, для 10-мегабитного Ethernet-соединения скорость R равна 10 Мбит/с, для 100-мегабитного R равно 100 Мбит/с. Задержка передачи будет равна L/R — время, которое требуется, чтобы протолкнуть, то есть передать все биты пакета в линию связи. На практике задержка передачи составляет от нескольких микросекунд до нескольких миллисекунд.

Задержка распространения

После того как бит попал в линию связи, он распространяется по ней до маршрутизатора Б. Время, необходимое ему до достижения маршрутизатора В, называется **задержкой распространения**. Бит движется со скоростью распространения в данной линии связи, зависящей от физической среды передачи (оптоволокна, медной витой пары и т.д.) и лежит в пределах от 2×10^8 до 3×10^8 м/с, что немногим меньше, чем скорость света. Задержка распространения тогда будет равна расстоянию между двумя маршрутизаторами, деленному на скорость распространения, то есть d/s, где d — расстояние между маршрутизаторами А и Б, s — скорость распространения по линии связи. Когда последний бит пакета достигает маршрутизатора В, то все биты пакета сохраняются в маршрутизаторе. Затем весь процесс повторяется на маршрутизаторе Б. Задержки распространения обычно составляют порядка нескольких миллисекунд.

Сравнение задержки передачи и задержки распространения

Новички в области компьютерных сетей часто испытывают трудности в понимании разницы между задержками передачи и задержками распространения. Эта разница хотя и не слишком очевидна, но достаточно важна. Задержка передачи — это время, необходимое маршрутизатору,

чтобы протолкнуть пакет в линию связи. Зависит оно от размера пакета и скорости передачи по линии связи и никак не связано с расстоянием между двумя маршрутизаторами. Задержка распространения — это время, требуемое для передачи бита от одного маршрутизатора к другому, и зависящее от расстояния между этими маршрутизаторами, но не связанное ни с размером пакета, ни со скоростью передачи в линии.

Пояснить разницу между передачей и распространением нам поможет некоторая аналогия. Представим себе скоростное шоссе, на котором через каждые 100 км расположены пункты взимания пошлины (ПВП). Участки шоссе между такими пунктами будут играть роль линий связи, а сами пункты сбора роль маршрутизаторов. Предположим, что автомобили двигаются по шоссе (то есть «распространяются») со скоростью 100 км/ч (пренебрежем временем разгона и будем считать, что после прохождения пункта взимания пошлины автомобиль мгновенно ускоряется до 100 км/ч и продолжает движение с этой скоростью до следующего ПВП). Предположим теперь, что по шоссе движется колонна из 10 автомобилей в определенном порядке. Автомобиль будет играть роль бита, а колонна — роль пакета. Предположим, что каждый пункт взимания пошлины обслуживает (то есть передает) один автомобиль за 12 с. А также что дело происходит ночью и другого движения, кроме нашей колонны на шоссе нет. Наконец, предположим, что, когда первый автомобиль колонны прибывает к ПВП, он останавливается перед въездом и ждет, пока не подъедут оставшиеся девять и встанут за ним в очередь. Таким образом, вся колонна должна собраться перед ПВП перед тем, как начнется ее обслуживание. Время, требуемое для обслуживания всей колонны и прохождения ее на следующий участок шоссе, равно 10 автомобилей/5 автомобилей в минуту = 2 минуты. Это время — аналог задержки передачи в маршрутизаторе. Время, которое требуется автомобилю, что доехать от одного ПВП до следующего ПВП, равно 100 км/100 км/ч = 1 ч. Это время — аналог задержки распространения. Следовательно, промежуток от момента сбора всей колонны перед въездом в один ПВП до момента сбора у въезда в другой ПВП будет равен сумме задержки передачи и задержки распространения. В данном примере составит 62 минуты.

Рассмотрим дальше эту аналогию. Что произойдет, если время на обслуживание колонны в пункте взимания пошлины будет больше, чем время, требуемое автомобилю для прохождения расстояния между ПВП? Предположим, например, что автомобиль движется со скоростью 1000 км/ч, а ПВП обслуживает транспорт со скоростью 1 автомобиль

в минуту. Тогда колонна будет проходить расстояние между двумя ПВП за 6 минут, а время на ее обслуживания составит 10 минут. В этом случае первые несколько автомобилей колонны будут приезжать ко второму ПВП до того момента, как вся колонна покинет первый ПВП. Такая ситуация часто встречается в сетях с коммутацией пакетов — первые несколько бит пакета могут достигнуть маршрутизатора, в то время как оставшаяся часть битов пакета все еще ожидает передачи на предыдущем маршрутизаторе.

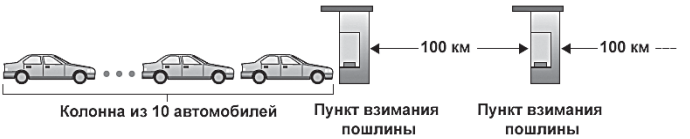


Рис. 1.17. Аналогия с автомобильной колонной

Говорят, лучше один раз увидеть, чем сто раз услышать. Так вот, еще лучше посмотреть анимацию, которая в сто раз лучше, чем статическое изображение. На веб-сайте **tinyurl.com/lgnn7bl** вы можете увидеть интерактивный Java-апплет, который прекрасно демонстрирует разницу между задержкой передачи и задержкой распространения. Самые воодушевленные читатели могут посетить эту страничку. Источник[606] также предлагает очень интересное обсуждение задержек распространения, ожидания и передачи.

Если мы обозначим задержку обработки, ожидания, передачи и распространения как $d_{\text{обработка}}$, $d_{\text{ожидание}}$, $d_{\text{передача}}$ и $d_{\text{распростр}}$ соответственно, то общая узловая задержка будет вычисляться по формуле

$$d_{\text{узловая}} = d_{\text{обработка}} + d_{\text{ожидание}} + d_{\text{передача}} + d_{\text{распростр}}$$

Влияние каждого из компонентов общей задержки значительно варьируется. Например, $d_{\text{распростр}}$ может быть очень незначительным (пара микросекунд) для линии связи, соединяющей два маршрутизатора в одном корпусе учебного заведения; наоборот, для двух маршрутизаторов, связанных геостационарной спутниковой линией связи, эта величина может равняться сотням миллисекунд и составлять максимальную долю общей узловой задержки. Аналогично, $d_{\text{передача}}$ может меняться от пренебрежимо малых до значительных величин. Ее влияние обычно очень незначительно для скоростей передачи 10 Мбит/с и выше (например, в сетях ЛВС); однако для больших пакетов, передаваемых в Интернет через низкоскоростное коммутируемое соединение, эта величи-

на может составлять сотни миллисекунд. Задержка обработки $d_{\text{обработка}}$ очень часто незначительна. Однако она оказывает сильное влияние на максимальную пропускную способность маршрутизатора, то есть максимальную скорость, с которой маршрутизатор может переправлять пакеты в сеть.

1.4.2. Задержка ожидания и потеря пакетов

Наиболее сложной и интересной составляющей общей узловой задержки является задержка ожидания, $d_{\text{ожидание}}$. В действительности эта тема настолько важна и интересна, что ей посвящены тысячи публикаций в прессе и очень много книг[50,122,288,289,290,572]. Здесь мы предлагаем лишь поверхностное интуитивное обсуждение задержки ожидания; наиболее любопытный читатель может сам изучить книги по этому вопросу (либо даже написать кандидатскую диссертацию). В отличие от трех других видов задержек ($d_{\text{обработка}}$, $d_{\text{передача}}$ и $d_{\text{распростр}}$), задержка ожидания может изменяться от пакета к пакету.

Если 10 пакетов одновременно прибывают в пустую очередь, то первый переданный пакет не будет испытывать задержки ожидания, в то время как для последнего она будет достаточно большой (пока передаются 9 других пакетов). Чтобы охарактеризовать задержки ожидания, обычно используют статистические величины, такие как средняя задержка ожидания, дисперсия задержки ожидания, вероятность превышения задержки ожидания какого-то конкретного значения.

Когда же задержка ожидания является большой величиной, а когда незначительной? Ответ на этот вопрос зависит от скорости, с которой пакеты поступают в очередь, скорости передачи линии связи, характера поступающего трафика, то есть, является ли он периодическим либо поступает пачками. Чтобы глубже изучить эту тему, давайте обозначим среднюю скорость прибытия пакета в очередь как a (измеряется в пакетах в секунду). R — скорость передачи по линии связи (в бит/с). Также предположим для простоты, что все пакеты имеют размер L бит. Тогда средняя скорость, с которой бит поступает в очередь, равна La бит/с. Наконец, предположим, что очередь может быть очень большой (размер буфера маршрутизатора), то есть содержать бесконечное количество бит информации. Величина La/R, которая называется **интенсивностью трафика**, часто играет важную роль в оценке задержек ожидания. Если $La/R > 1$, то скорость прибытия пакетов

в очередь превышает скорость передачи пакетов из очереди. К сожалению, в такой ситуации очередь будет бесконечно расти, и задержка ожидания достигнет огромных размеров, поэтому одно из золотых правил построения при организации трафика в компьютерных сетях гласит: *проектируйте ваши системы таким образом, чтобы интенсивность трафика не превышала единицу*.

Теперь рассмотрим случай, когда $La/R \leq 1$. Тогда на задержку ожидания будет влиять характер трафика. Например, если пакеты прибывают периодически, с равными интервалами — то есть по одному через L/R секунд, — то каждый пакет будет поступать в пустую очередь, и задержка ожидания окажется нулевой. Если же пакеты прибывают периодически, но не по одному, а пачками, то возникнет значительная средняя задержка ожидания. Предположим, например, что каждые $(L/R)N$ секунд прибывают одновременно N пакетов. Тогда задержка ожидания для первого пакета будет нулевая, для второго пакета — равная L/R секунд и в общем случае для n-го пакета — $(n - 1)L/R$ секунд. Подсчитать среднюю задержку ожидания в этом случае вы можете самостоятельно в качестве упражнения.

Два рассмотренных выше примера с правильными интервалами между прибытием пакетов являются скорее теоретическими. В реальности процесс поступления пакетов в очередь является *случайным*, то есть не описывается никаким законом. Поэтому величина интенсивности трафика La/R не может достаточно полно характеризовать задержки ожидания, но помогает получить представление о их степени. В частности, если интенсивность трафика близка к нулю, то пакетов прибывает мало и поэтому маловероятно, что по прибытии пакета очередь будет заполнена. Следовательно, средняя задержка ожидания будет близка к нулю. С другой стороны, когда интенсивность трафика близка к единице, появятся интервалы времени, когда скорость прибытия пакетов превысит скорость передачи и начнет формироваться очередь из пакетов; в случае, если скорость поступления меньше скорости передачи, длина очереди будет уменьшаться. В любом случае, когда интенсивность трафика близка к единице, то средняя задержка ожидания становится все больше и больше. Зависимость средней задержки ожидания от интенсивности трафика показана на рис. 1.18.

Рисунок 1.18 демонстрирует один очень важный момент: если интенсивность трафика приближается к единице, то средняя задержка

ожидания очень быстро растет. Тогда незначительное в процентном отношении увеличение интенсивности приводит к гораздо большему росту задержек. Вы наверняка наблюдали этот феномен при езде на автомобиле по шоссе. Если вы едете по дороге, которая перегружена автомобилями, это значит, что интенсивность трафика близка к единице, и в этом случае, если небольшое ДТП приводит к незначительному увеличению интенсивности движения, то задержки при этом вы можете испытать достаточно серьезные.

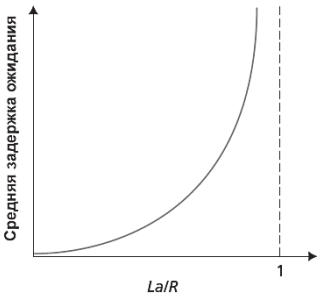


Рис. 1.18. Зависимость средней задержки ожидания от интенсивности трафика

Чтобы нагляднее познакомиться с задержками ожидания, вы можете посетить веб-сайт **tinyurl.com/83vwapv**, на котором расположен Java-апплет. Если вы зададите скорость прибытия пакетов достаточно большой так, что интенсивность трафика превысит единицу, вы будете наблюдать медленное возрастание очереди.

Потеря пакетов

В предыдущем разделе мы с вами предположили, что в очереди может содержаться бесконечное число пакетов. На практике же очередь (буфер маршрутизатора) обычно имеет ограниченный объем и зависит в основном от конструкции и стоимости маршрутизатора. Это означает, что задержки пакетов в реальности не могут быть бесконечными, когда интенсивность трафика приближается к единице. Вместо этого в момент прибытия пакета к маршрутизатору очередь может оказаться заполненной и, так как негде сохранить пакет, маршрутизатор его **отбросит**, то есть пакет будет **потерян**. Демонстрацию переполнения очереди вы опять же можете посмотреть с помощью Java-апплета, задав интенсивность трафика больше единицы.

С точки зрения конечной системы, потеря пакета будет означать, что переданные в сеть данные не достигнут получателя. Количество потерянных пакетов, очевидно, возрастает по мере увеличения интенсивности трафика. Поэтому очень часто производительность в узлах характеризуется не только величиной задержек, но и вероятностью потери пакетов. Как мы обсудим в следующих главах, для того, чтобы гарантированно передавать данные от источника к приемнику, в сетях производится повторная отправка потерянных пакетов.

1.4.3. Общая задержка

До этого момента мы с вами обсуждали узловые задержки, то есть, задержки, вызванные одним маршрутизатором. Теперь же давайте рассмотрим общие задержки при передаче от источника к приемнику. Предположим, что на пути между хостом-источником и хостом-приемником находятся $N - 1$ маршрутизатор. Допустим также, что нагрузка в сети невысокая (задержки ожидания незначительные), задержка обработки на каждом маршрутизаторе и на хосте-источнике равна $d_{\text{обработка}}$. Скорость передачи при отправке с каждого из маршрутизаторов, а также с хоста-источника равна R бит/с, а задержка распространения на каждой линии связи составляет $d_{\text{распростр}}$. Тогда сумма всех узловых задержек даст нам общую (сквозную) задержку передачи на маршруте источник-приемник

$$d_{\text{сквозн}} = N\,(d_{\text{обработка}} + d_{\text{передача}} + d_{\text{распростр}}) \qquad (1.2)$$

где $d_{\text{передача}} = L/R$ (L — длина пакета). Заметим, что частным случаем уравнения 1.2 является уравнение 1.1, в котором не учитывались задержки обработки и распространения. Мы предлагает вам обобщить уравнение 1.2 для случая, когда узловые задержки непостоянны, а задержки ожидания на каждом узле не равны нулю.

Traceroute

Чтобы лучше понять, как происходят сквозные задержки (задержки на пути источник-приемник) в компьютерных сетях, мы можем использовать программу Traceroute. Это простейшая утилита, которую можно запустить на любом хосте в Интернете. Пользователь указывает имя хоста-приемника, а программа на хосте-источнике отсылает специальные пакеты в сторону приемника. На своем пути к получателю эти пакеты проходят через серию маршрутизаторов. Когда маршрутизатор

получает один из этих пакетов, он отправляет обратно хосту-источнику короткое сообщение, в котором содержатся имя и адрес этого маршрутизатора.

Предположим, что на пути между источником и приемником расположены $N - 1$ маршрутизаторов. В таком случае хост-источник будет отсылать в сеть N специальных пакетов, каждый из которых адресован своему приемнику. Эти N пакетов маркируются от 1 до N в порядке возрастания. Когда n-ный маршрутизатор получает n-ный пакет, маркированный n, он не переправляет его, а вместо этого отсылает сообщение обратно отправителю. То же самое делает хост-приемник, получая N-ный пакет. Хост-источник регистрирует время, прошедшее с момента отправки пакета до момента получения соответствующего обратного сообщения; также он фиксирует имя и адрес маршрутизатора (или хоста-приемника), которые возвращают сообщение. Таким способом хост-отправитель может воспроизвести весь маршрут пакетов от источника к приемнику и определить задержки до всех маршрутизаторов, через которые проходят пакеты (задержка в две стороны — туда и обратно). Traceroute повторяет описанную передачу три раза, то есть на самом деле хост-отправитель посылает $3 \times N$ пакетов хосту-получателю. Более подробно утилита Traceroute описывается в документе RFC 1393.

Перед вами пример вывода программы Traceroute, отслеживающей маршрут от хоста-источника **gaia.cs.umass.edu** (в Массачусетском университете) до хоста-приемника **cis.poly.edu** (в Бруклинском политехническом университете). Выводимые строки содержат шесть полей: в первом находится описанное выше значение n, то есть порядковый номер маршрутизатора на маршруте; второе занимает имя маршрутизатора; в третьем стоит адрес этого маршрутизатора (записанный в форме xxx.xxx.xxx.xxx); в последних трех полях содержится значение двусторонних задержек для трех попыток. В случае если источник получает меньше, чем три сообщения от какого-нибудь маршрутизатора (из-за потери пакета в сети), то программа Traceroute ставит звездочку после номера соответствующего маршрутизатора и информирует о менее чем трех величинах задержки, соответствующих этому маршрутизатору.

```
1 cs-gw (128.119.240.254) 1.009 ms 0.899 ms 0.993 ms
2 128.119.3.154 (128.119.3.154) 0.931 ms 0.441 ms 0.651 ms
3 border4-rt-gi-1-3.gw.umass.edu (128.119.2.194) 1.032 ms 0.484 ms 0.451 ms
```

```
4 acr1-ge-2-1-0.Boston.cw.net (208.172.51.129) 10.006 ms 8.150 ms 8.460 ms
5 agr4-loopback.NewYork.cw.net (206.24.194.104) 12.272 ms 14.344 ms 13.267 ms
6 acr2-loopback.NewYork.cw.net (206.24.194.62) 13.225 ms 12.292 ms 12.148 ms
7 pos10-2.core2.NewYork1.Level3.net (209.244.160.133) 12.218 ms 11.823 ms 11.793 ms
8 gige9-1-52.hsipaccess1.NewYork1.Level3.net (64.159.17.39) 13.081 ms 11.556 ms 13.297 ms
9 p0-0.polyu.bbnplanet.net (4.25.109.122) 12.716 ms 13.052 ms 12.786 ms
10 cis.poly.edu (128.238.32.126) 14.080 ms 13.035 ms 12.802 ms
```

В данном примере мы видим, что на маршруте между источником и приемником находятся 9 маршрутизаторов. Большинство из них представлены именем, и у всех есть IP-адрес. Например, имя третьего маршрутизатора `border4-rt-gi-1-3.gw.umass.edu`, а его адрес `128.119.2.194`. Если мы посмотрим на данные, представленные для этого маршрутизатора, мы увидим, что в первой попытке двусторонняя задержка между источником и маршрутизатором была 1,03 мс. Задержки в двух других попытках составили соответственно 0,48 и 0,45 мс. Эти двусторонние задержки включают в себя все обсужденные нами виды задержек, то есть задержки передачи, распространения, обработки на маршрутизаторе и ожидания. Из-за того, что задержки ожидания могут меняться во времени, то задержка пакета *n*, посланного маршрутизатору *n*, может иногда быть больше, чем задержка пакета $n + 1$, посланного маршрутизатору $n + 1$. В действительности мы можем наблюдать этот феномен в представленном выше примере: задержка до маршрутизатора 6 больше, чем до маршрутизатора с номером 7!

Хотите сами попробовать утилиту Traceroute? Тогда мы *очень* рекомендуем вам посетить сайт **www.traceroute.org**, который представляет веб-интерфейс с широким набором источников для отслеживания маршрута. Вы выбираете источник и сами предлагаете имя хоста для получателя. Остальную работу делает Traceroute. Существует множество свободно распространяемых программ, которые предлагают графический интерфейс для утилиты Traceroute; одна из наших любимых среди них — PingPlotter[396].

Задержки на конечных системах, приложениях и другие виды задержек

Кроме изученных нами задержек обработки, передачи и распространения пакеты могут испытывать значительные задержки на конечных системах. Например, конечная система при передаче пакета в разделяемую среду (например, в сетях с кабельным либо Wi-Fi доступом) может

целенаправленно задерживать свою передачу, потому что это предусмотрено протоколом для разделения среды с другими конечными системами; мы рассмотрим такие протоколы более подробно в главе 5. Еще одним важным видом задержек является задержка пакетизации медиаресурсов (например, аудио- или видеофайла), которая присутствует, например, в приложениях IP-телефонии (VoIP). В этом случае передающая сторона вначале собирает в пакет закодированную цифровую речь, перед тем как отправить этот пакет в Интернет. Время на упаковку или сборку этого пакета — называемое задержкой пакетизации — может достигать значительной величины и, следовательно, влиять на качество IP-вызова, заметное пользователям. Эту тему мы обсудим в домашнем задании в конце этой главы.

1.4.4. Пропускная способность в компьютерных сетях

Кроме задержек и потерь пакетов еще одной важной характеристикой производительности компьютерных сетей является сквозная пропускная способность (сквозная, то есть от источника к приемнику). Чтобы дать ей определение, рассмотрим передачу большого файла от хоста А к хосту Б по компьютерной сети. В качестве примера можно рассмотреть передачу видеоклипа между двумя системами в одноранговой сети. **Мгновенная пропускная способность** в любой момент времени — это скорость (в бит/с), с которой хост Б получает файл. Многие приложения, включая большинство систем в одноранговых сетях с разделением файлового доступа, отображают мгновенную пропускную способность в интерфейсе пользователя во время загрузки — возможно, вы с этим встречались. Если файл содержит F бит, а передача занимает T секунд, то **средняя пропускная способность** для передачи файла равна F/T бит/с. Для некоторых приложений, таких как IP-телефония, желательно, чтобы уровень задержек был низким, а мгновенная пропускная способность превышала определенное пороговое значение (например, не ниже 24 Кбит/с для ряда приложений IP-телефонии и не ниже 256 Кбит/с для приложений, обрабатывающих видеопоток в реальном времени). Для других приложений наоборот, задержки не так критичны, но, с другой стороны, желательна высокая пропускная способность, например для упомянутой выше передачи файлов.

Чтобы получить более глубокое представление о том, как важно понятие пропускной способности, давайте рассмотрим несколько при-

меров. На рис. 1.19а показаны две конечные системы, сервер и клиент, соединенные двумя коммуникационными линиями связи и маршрутизатором. Рассмотрим пропускную способность, необходимую для передачи файла от сервера к клиенту. Обозначим через R_S скорость передачи данных на линии между сервером и маршрутизатором, а через R_C — скорость передачи между маршрутизатором и клиентом. Предположим также, что любой другой трафик в этой сети, кроме трафика от сервера к клиенту, отсутствует. Возникает вопрос, какова пропускная способность в таком идеальном случае по маршруту сервер-клиент? Чтобы ответить на этот вопрос, представим биты информации как *жидкость*, а линии связи как *трубы*. Очевидно, что сервер не может передавать («качать») биты через свое соединение со скоростью выше, чем R_S бит/с; то же самое справедливо и для маршрутизатора: он не может перенаправлять биты со скоростью выше, чем R_C бит/с. Если $R_S < R_C$, то биты, «качаемые» сервером, будут течь через маршрутизатор и прибывать к клиенту со скоростью R_S бит/с, что и составит пропускную способность, равную R_S. Если же $R_C < R_S$, то маршрутизатор не сможет перенаправлять биты так же быстро, как он их получает. В этом случае биты будут покидать маршрутизатор со скоростью R_C, определяя пропускную способность как R_C. Отметим также, что если биты продолжают прибывать на маршрутизатор со скоростью R_S и покидать его со скоростью R_C, то остаток битов, ожидающих своей очереди для передачи клиенту, будет все возрастать и возрастать — очень нежелательная ситуация! Таким образом, для простой сети с двумя линиями связи пропускная способность равна $\min\{R_C, R_S\}$, то есть скорости передачи **соединения типа бутылочного горла**. Определив пропускную способность, мы можем теперь подсчитать время, необходимое на передачу большого файла из F бит от сервера к клиенту как $F/\min\{R_C, R_S\}$. Если взять конкретный пример, предположим, вы загружаете MP3-файл размером F, равным 32 Мбит, скорость передачи со стороны сервера R_S равна 2 Мбит/с, и скорость передачи до клиента R_C равна 1 Мбит/с, то время на передачу будет составлять 32 секунды. Конечно, данное значение пропускной способности и времени передачи достаточно приближенное, так как не учитываются задержки обработки и некоторые виды задержек, обусловленные протоколом передачи.

На рис. 1.19б теперь представлена сеть, содержащая N линий связи между сервером и клиентом, со скоростями передачи данных для линий, равными $R_1, R_2, ..., R_N$. Применяя те же рассуждения, что и для сети с двумя линиями связи, мы находим, что пропускная способность для передачи файла от сервера к клиенту равна $\min\{R_1, R_2, ..., R_N\}$, что, опять же,

является скоростью передачи линии с бутылочным горлышком между сервером и клиентом.

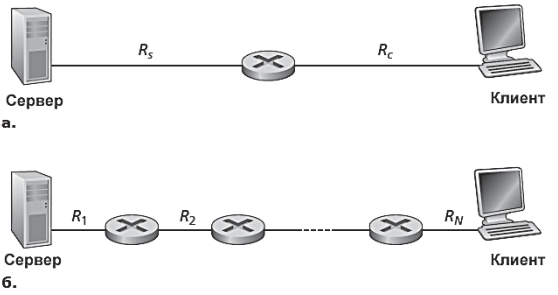


Рис. 1.19. Пропускная способность для передачи файла от сервера клиенту

Рассмотрим еще один пример, вызванный необходимостью передачи данных в Интернете. На рис. 1.20а мы видим две конечных системы — сервер и клиент, соединенные компьютерной сетью. Определим пропускную способность для передачи файла от сервера к клиенту. Сервер соединен с сетью линией, имеющей скорость передачи R_S. Клиент соединен с сетью линией со скоростью R_C.

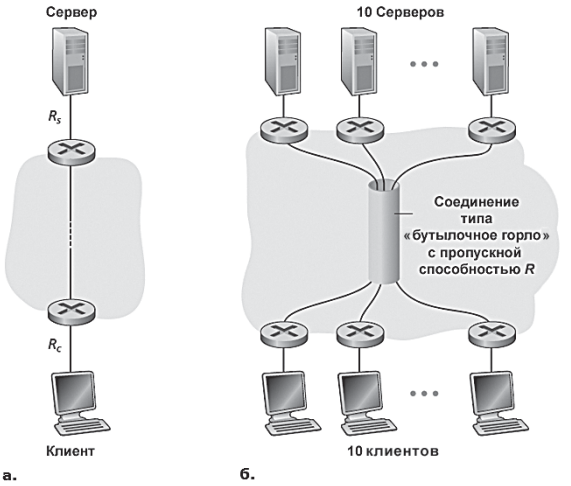


Рис. 1.20. Сквозная пропускная способность: (а) клиент загружает файл с сервера; (б) 10 клиентов загружают данные с 10 серверов

Предположим теперь, что все линии связи в ядре коммуникационной сети имеют очень высокие скорости передачи, гораздо выше, чем R_S и R_C. Предположим опять, что в сети нет никакого трафика, кроме дан-

ных, передаваемых от сервера к клиенту. Так как ядро компьютерных сетей в нашем примере подобно широкой трубе, позволяющей данным течь от источника к приемнику, скорость их движения снова будет равна $\min\{R_C, R_S\}$. Следовательно, мы видим, что сдерживающим фактором для пропускной способности в сегодняшнем Интернете являются сети доступа, так как ядро компьютерных сетей составляют достаточно производительные высокоскоростные маршрутизаторы.

В качестве заключительного примера рассмотрим рис. 1.20б, на котором мы видим, что 10 серверов и 10 клиентских рабочих мест соединены с ядром компьютерной сети. В этом случае 10 пар клиент-сервер производят одновременно 10 загрузок файлов. Опять предположим, что кроме этих 10 загрузок никакого трафика в сети в данный момент не существует. Как показано на рисунке, существует какая-то линия связи в ядре сети, через которую проходят все 10 загрузок. Обозначим через R скорость передачи этой линии связи. Предположим также, что все соединения со стороны серверов имеют одинаковую скорость R_S, все соединения со стороны клиента имеют одну и ту же скорость R_C, а скорость передачи данных во всех линиях связи в ядре сети — за исключением той, где скорость равна R, гораздо больше, чем R_S, R_C и R. Какова же будет пропускная способность для загрузок файлов? Очевидно, что если скорость общей линии R достаточно велика — скажем, в сотни раз больше, чем оба значения R_S и R_C — то пропускная способность для каждой загрузки опять составит $\min\{R_C, R_S\}$. Но что будет, если скорость передачи в общем соединении одного порядка с R_S и R_C? Какова окажется пропускная способность в этом случае? Возьмем более конкретный пример. Предположим, R_S равно 2 Мбит/с, R_C равно 1 Мбит/с, R равно 5 Мбит/с, а общий канал связи разделяет эту скорость между 10 загрузками. В данном случае узким местом для каждой загрузки будет уже не сеть доступа, а общее соединение в ядре сети, которое может обеспечить загрузку со скоростью лишь 500 Кбит/с. Таким образом, сквозная пропускная способность для каждой загрузки уменьшилась до 500 Кбит/с.

Рассмотренные выше примеры показывают нам, что пропускная способность зависит от скоростей передачи по линиям связи, через которые идут данные. Мы видели, что при отсутствии другого трафика пропускную способность можно упрощенно считать равной минимальной скорости передачи по линии между источником и приемником. Пример на рис. 1.20 (б) показывает, что в более общем случае пропускная способность зависит не только от скоростей передачи по линии связи, но также от влияющего постороннего трафика. В частности, линия связи с доста-

точно высокой скоростью передачи также может оказаться узким местом для передачи файла, если через эту линию проходят еще и другие потоки данных. Более детально мы проверим пропускную способность в компьютерных сетях в домашних заданиях и в последующих главах.

1.5. Уровни протоколов и модели их обслуживания

Из нашего с вами обсуждения можно сделать вывод, что Интернет — *очень* сложная система. Она состоит из многих компонентов: многочисленных приложений и протоколов, различных типов конечных систем, коммутаторов пакетов, а также различных сред передачи. Учитывая такую необычайную сложность, возникает вопрос, возможно ли как-либо организовать сетевую архитектуру или, по крайней мере, организовать обсуждение сетевой архитектуры. К счастью, на эти оба вопроса мы можем ответить «да».

1.5.1. Многоуровневая архитектура

Перед тем как попытаться систематизировать наши знания архитектуры Интернета, давайте обратимся к аналогии из мира людей. В нашей повседневной жизни мы все имеем дело со сложными системами. Представьте себе, что кто-нибудь попросил вас описать, например, как организована система воздушных сообщений. Как бы вы стали описывать такую сложную структуру, которая включает в себя агентство по продаже авиабилетов, службу по проверке багажа, сотрудников терминалов, летный персонал, парк воздушных судов, а также международные диспетчерские службы? Один из возможных способов состоит в том, чтобы показать набор действий, которые вы производите (или которые производятся для вас), когда вы совершаете авиаперелет. Такие действия включают в себя, например, покупку билета, регистрацию багажа, проход через выход терминала и посадку на борт самолета. После этого самолет взлетает и движется по маршруту до места назначения. Затем вы приземляетесь, покидаете самолет через выход терминала и получаете свой багаж. Если вы остались недовольны полетом, то подаете жалобу авиаперевозчику. Сценарий такого авиаперелета показан на рис. 1.21.

Тут мы сразу замечаем аналогию с функционированием компьютерных сетей: авиакомпания доставляет вас от исходного пункта до пункта назначения. Пакет в Интернете тоже доставляется от хоста-источника

до хоста-приемника. Но этим сходство не ограничивается. Здесь просматривается некоторая *структура*. Мы замечаем, что на обоих концах данной структуры находятся функции, связанные с билетами; с багажом для пассажиров, которые уже получили билеты, а также со стойкой регистрации и выходом для тех пассажиров, которые уже приобрели билет и прошли регистрацию багажа. Для пассажиров, которые уже прошли регистрацию, то есть купили билеты, сдали багаж и миновали ворота, есть функция, связанная со взлетом и посадкой, а также присутствуют функции в течение полета, относящиеся к движению лайнера по маршруту. Можно взглянуть на всю эту функциональность, представленную на рис. 1.21, и изобразить ее в виде *горизонтальных* уровней.

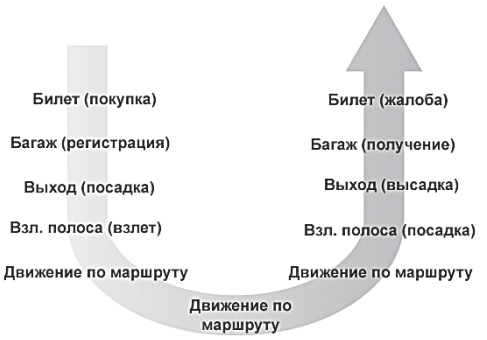


Рис. 1.21. Действия при воздушном путешествии

На рис. 1.22 функциональность, предоставляемая авиакомпанией, разделена на уровни, и эту структуру мы будем обсуждать, рассматривая воздушные путешествия. Обратим внимание, что каждый уровень в комбинации с находящимися под ним, предоставляет некоторую функциональность или некоторые *службы*. Например, на билетном уровне и ниже пассажир проходит путь от кассира авиакомпании до кассира авиакомпании (от покупки билета до возможного получения компенсации по жалобе). На багажном уровне и ниже производится обслуживание пассажира от регистрации до получения багажа. Обратим внимание, что багажный уровень обслуживает только тех пассажиров, которые уже имеют билеты. На уровне выхода выполняется передача пассажира и багажа от стойки регистрации вылета до стойки регистрации прибытия. На уровне взлет-посадка производится обслуживание пассажиров и их багажа от взлетной полосы до взлетной полосы. Следовательно, на каждом уровне, во-первых, выполняются определенные действия, относящиеся к этому уровню (например, на уровне выхода производятся по-

садка на самолет и высадка с него) и, во-вторых, используются службы, предоставляемые уровнями ниже (например, уровень выхода включает также передачу пассажиров от взлетной полосы до взлетной полосы, которую предоставляет уровень взлет/посадка).

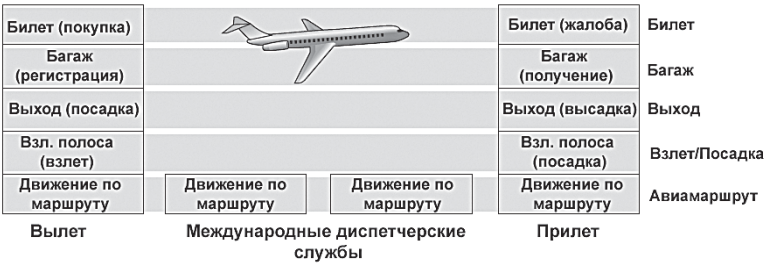


Рис. 1.22. Функциональность авиакомпании в виде горизонтальных уровней

Такая многоуровневая архитектура позволяет нам обсуждать более детально отдельные части большой и сложной системы. Это упрощение само по себе очень ценно, так как дает возможность модульной организации, позволяя намного проще изменять реализацию функций и служб, предоставляемых уровнем. Так как один уровень предлагает одни и те же услуги и функции уровню над ним и использует те же самые услуги и функции с уровня ниже, то изменение реализации одного уровня не затронет всю остальную систему. Заметим, что изменение реализации предоставляемых услуг сильно отличается от изменения самой услуги, то есть, например, если функции выхода поменялись (например, посадка пассажиров производится по росту), то вся остальная система воздушного сообщения не претерпит изменений, так как данный уровень выхода все так же выполняет свои функции (посадка и высадка пассажиров); он просто реализует эту функцию немного по-другому после внесения изменений. Для больших и сложных систем, которые постоянно обновляются, способность изменять реализацию предоставляемых услуг (сервисов), не затрагивая другие компоненты системы, является еще одним важным преимуществом многоуровневой структуры.

Уровни протоколов

Но достаточно об авиаперелетах. Давайте обратим наше внимание на сетевые протоколы. Структура сетевых протоколов — аппаратное и программное обеспечение, реализующее эти протоколы — организована с помощью **уровней**. Каждый протокол принадлежит какому-

то определенному одному уровню, точно так же, как каждая функция в архитектуре воздушной линии на рис. 1.22. Мы будем рассматривать **услуги**, которые уровень предоставляет уровню выше — так называемую **модель обслуживания** уровня. Как в случае с нашим примером авиалинии, каждый уровень предоставляет свои услуги, во-первых, выполняя определенные действия внутри себя, и, во-вторых, используя услуги уровня, находящегося ниже. Например, услуги, уровня *n*, могут включать надежную доставку сообщений с одного конца сети на другой. Это может быть реализовано с помощью ненадежной сквозной доставки сообщения на уровне $n - 1$ с добавлением функциональности уровня *n* для обнаружения и передачи потерянных сообщений.

Уровни протоколов могут быть реализованы в программном, в аппаратном обеспечении либо в их комбинации. Протоколы прикладного уровня — такие, как HTTP и SMTP — почти всегда реализованы в программном обеспечении конечных систем; то же самое можно сказать о протоколах транспортного уровня. Поскольку физический и канальный уровень отвечают за коммуникации по линиям связи, они обычно реализованы в сетевых интерфейсных картах, например Ethernet или Wi-Fi, соединенных с линией связи. Сетевой уровень обычно использует как аппаратную, так и программную реализацию. Обратим также внимание, что, как и функции в многоуровневой архитектуре воздушных сообщений перераспределены между различными аэропортами и диспетчерскими службами, образующими систему, так и протокол уровня *n распределен* между конечными системами, коммутаторами и другими компонентами, образующими компьютерную сеть. Таким образом, в каждой компьютерной сети существуют компоненты протоколов уровня *n*.

Разделение протоколов на уровни имеет свои преимущества[513]. Как мы уже видели, такой подход позволяет разложить структуру на компоненты. Принцип модульности облегчает обновление и модернизацию составляющих частей системы. Однако следует сказать, что некоторые специалисты сетевых коммуникаций выступают против уровней[653]. Одним из потенциальных недостатков такой структуры является то, что один уровень может дублировать функции других, например, нижележащих. Допустим, обработка ошибок во многих стеках протоколов может выполняться на нескольких уровнях на сквозной основе. Еще одним недостатком является то, что на одном уровне может потребоваться информация (например, значение времени), которая представлена только на другом, а это нарушает принцип изолированности.

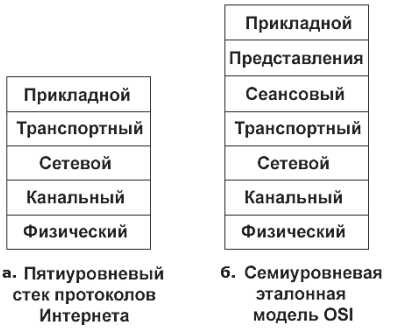


Рис. 1.23. Стек протоколов Интернета (а) и эталонная модель OSI (б)

Набор протоколов различных уровней называется **стеком протоколов**. Он состоит из пяти уровней: физического, канального, сетевого, транспортного и прикладного, как показано на рис. 1.23а. Если вы взглянете на содержание нашей книги, вы заметите, что мы организовали ее структуру в порядке, соответствующем стеку протоколов Интернета, применив **нисходящий подход**, начиная с прикладного уровня и спускаясь вниз.

Прикладной уровень

Прикладной уровень (иначе называемый уровнем приложений) поддерживает сетевые приложения и их протоколы. Прикладной уровень Интернета включает множество протоколов, таких как HTTP (обеспечивающий запрос и передачу веб-документов), SMTP (отвечающий за сообщения электронной почты) и FTP (для обмена между двумя конечными системами).

Мы с вами увидим, что определенные сетевые функции, такие, как трансляция понятных человеку имен конечных систем в Интернете, например **www.ietf.org** в 32-разрядные сетевые адреса также выполняются при помощи специального протокола прикладного уровня, называемого DNS (domain name system, система доменных имен). В главе 2 мы увидим, что можно разработать свои собственные протоколы прикладного уровня.

Протокол прикладного уровня обслуживает множество конечных систем, при этом приложение одной конечной системы обменивается порциями данных с приложением другой конечной системы. Порцию данных прикладного уровня назовем **сообщением**.

Транспортный уровень

Транспортный уровень Интернета осуществляет передачу сообщений прикладного уровня между конечными приложениями. Два транспортных протокола, существующих в Интернете и организующих передачу сообщений прикладного уровня, — это TCP и UDP. Протокол TCP предлагает приложениям службы с установлением соединения. Эти службы обеспечивают надежную доставку сообщений прикладного уровня получателям, а также контроль переполнения (то есть регулирование скорости потока). TCP также разбивает длинные сообщения на более короткие сегменты и обеспечивает механизм для контроля перегрузок таким образом, что при перегрузке сети источник снижает свою скорость передачи. Протокол UDP предоставляет приложениям службы без установления соединения. При этом не гарантируется надежность передачи, нет контроля переполнения и контроля перегрузок. Мы назовем порцию данных транспортного уровня **сегментом**.

Сетевой уровень

Сетевой уровень Интернета отвечает за передачу порций данных, известных как **дейтаграммы**, от одного хоста сети к другому. Протоколы транспортного уровня (TCP и UDP) передают сегмент транспортного уровня и адрес назначения на сетевой уровень точно так же, как вы отправляете письмо на почту с указанием адреса доставки. Сетевой уровень, в свою очередь, обеспечивает службу для доставки этого сегмента на транспортный уровень хоста-получателя.

Сетевой уровень Интернета включает протокол IP, который определяет поля дейтаграмм, а также действия, которые должны производить конечные системы и маршрутизаторы с этими полями. Протокол IP един для всего Интернета, и все компоненты, работающие на сетевом уровне, должны исполнять его. Сетевой уровень содержит также протоколы маршрутизации, которые определяют маршруты прохождения дейтаграмм между хостами-источниками и хостами-приемниками. Таких протоколов маршрутизации в Интернете довольно много. Как мы видели в разделе 1.3, Интернет является сетью сетей, и каждая сеть, входящая в него, может по желанию администратора использовать свой собственный протокол маршрутизации. Несмотря на то, что сетевой уровень содержит кроме протокола IP еще и многочисленные протоколы маршрутизации, их объединяют в одном протоколе IP, потому что на самом деле именно он является связующим звеном Интернета.

Канальный уровень

Сетевой уровень обеспечивает передачу дейтаграммы по цепочке маршрутизаторов от источника к приемнику в Интернете. Чтобы переместить пакет от одного узла (хоста или маршрутизатора) к следующему на маршруте, сетевой уровень использует службы канального уровня; в частности, на каждом узле передает дейтаграмму ниже на канальный уровень, который доставляет ее к следующему узлу на маршруте, а затем канальный уровень передает дейтаграмму вверх на сетевой.

Службы, предоставляемые канальным уровнем, зависят от конкретного протокола канального уровня, который используется на определенной линии связи. Например, некоторые протоколы канального уровня обеспечивают надежную доставку по линии связи от передающего узла к принимающему. Важно отметить, что надежность доставки на канальном уровне отличается от той, что предлагается в протоколах TCP и обеспечивает надежную доставку от одной конечной системы до другой. Примером протоколов канального уровня могут служить Ethernet, Wi-Fi, а также протокол кабельных сетей доступа DOCSIS. Когда дейтаграммы на пути следования от источника к приемнику проходят несколько линий связи, то на каждой из них они могут быть обработаны различными протоколами канального уровня. Например, на одном участке маршрута дейтаграмму обрабатывает протокол Ethernet, а на другом — протокол PPP. Таким образом, различные протоколы канального уровня предоставляют различные службы для сетевого уровня. Мы назовем порции данных канального уровня **кадрами**.

Физический уровень

В то время как работой канального уровня является передача кадров между соседними узлами сети, физический уровень предназначен для передачи отдельных битов кадра между этими узлами. Протоколы физического уровня опять же зависят от используемой линии связи и от реальной среды передачи этой линии (медная витая пара, одномодовое оптоволокно и т.д.). Например, Ethernet поддерживает множество протоколов физического уровня: один для витой медной пары, другой для коаксиального кабеля, третий для оптоволоконного и так далее. В каждом из этих случаев биты передаются по линии связи различными способами.

Модель OSI

Мы с вами подробно обсудили стек протоколов Интернета. Но стоит упомянуть, что это не единственный набор протоколов для компьютерных сетей. В конце 70-х годов международная организация по стандартизации (International Organization for Standardization, ISO) предложила так называемую эталонную **модель взаимодействия открытых систем** (Open Systems Interconnection model или просто **модель OSI**[247]. В ней предлагается организовать компьютерные сети с помощью семи уровней. Модель OSI формировалась в то время, когда будущие протоколы Интернета только начинали свое развитие среди множества других протоколов; в действительности разработчики первоначальной модели OSI, создавая ее, не думали об Интернете. Тем не менее начиная с конца 70-х годов все образовательные программы по компьютерным сетям строились на базе концепции семиуровневой модели. Поскольку такая модель оказала значительное влияние на образование в области компьютерных сетей, она до сих пор не потеряла свое значение и упоминается во многих учебниках и обучающих курсах по сетевым технологиям.

Семь уровней модели OSI, представленные на рис. 1.23б — это прикладной уровень, уровень представления, сеансовый уровень, транспортный уровень, сетевой уровень, канальный уровень и физический уровень. Функциональность пяти из этих уровней та же самая, что и у аналогичных уровней стека протоколов Интернета. Давайте рассмотрим два дополнительных уровня, представленных в эталонной модели OSI — уровень представления и сеансовый уровень. Основная роль уровня представления — обеспечить сервис, позволяющий взаимодействующим приложениям интерпретировать данные, которыми они обмениваются. Это включает сжатие данных, их шифрование (название говорит само за себя), а также описание (мы увидим в главе 9, что эти службы позволяют приложениям не заботиться о внутренних форматах, в которых данные представлены/сохранены, и которые могут отличаться от одного компьютера к другому). Сеансовый уровень обеспечивает разграничение и синхронизацию данных в процессе обмена, предоставляет средства для контроля за сеансом и его восстановление в случае разрыва соединения.

Отсутствие этих двух уровней эталонной модели OSI в стеке протоколов Интернета вызывает пару интересных вопросов: разве службы, предоставляемые этими уровнями, не так важны? Что если приложению *необходимы* эти службы? Интернет отвечает на эти вопросы одинако-

во — на усмотрение разработчика приложения. Он сам должен решить, важна конкретная служба или нет, и в случае, если она *действительно* важна, он же решает, как встроить такую функциональность в приложение.

1.5.2. Инкапсуляция

На рис. 1.24 показан путь, который проходят данные от отправляющей конечной системы вниз по стеку протоколов этой системы, затем вверх и вниз по протоколам коммутатора и маршрутизатора, и затем вверх по уровням протоколов принимающей конечной системы. Как мы с вами обсудим позднее в этой книге, и маршрутизаторы, и коммутаторы канального уровня занимаются коммутацией пакетов. Подобно конечным системам, аппаратное и программное обеспечение данных устройств также организовано в виде уровней. Но, в отличие от конечных систем, маршрутизаторы и коммутаторы не используют *все* уровни стеков протоколов. Они обычно работают на нижних уровнях. Как мы видим из рис. 1.24, коммутатор канального уровня использует уровни 1 и 2; в маршрутизаторах реализованы уровни с 1 по 3. Это означает, например, что маршрутизаторы в Интернете могут работать с протоколом IP (с протоколом третьего уровня), в то время как коммутаторы канального уровня не способны это делать. Они не могут распознавать IP-адреса, но зато работают с адресами второго уровня, такими как адреса Ethernet. Обратим внимание, что хосты поддерживают все пять уровней, то есть можно сделать вывод, что вся сложность архитектуры Интернета ложится на плечи конечных устройств сети.

Рисунок 1.24 демонстрирует также важное понятие **инкапсуляции**. На отправляющем хосте **сообщение прикладного уровня** (M на рис. 1.24) передается на транспортный уровень. В простейшем случае транспортный уровень принимает сообщение и добавляет к нему дополнительную информацию (так называемый заголовок транспортного уровня H_t на рис. 1.24), которая будет впоследствии использоваться транспортным уровнем на принимающей стороне. Сообщение прикладного уровня вместе с информацией заголовка транспортного уровня составляют **сегмент транспортного уровня**. Таким образом, сегмент транспортного уровня *инкапсулирует* сообщение прикладного уровня. Дополнительная информация, включаемая в заголовок, может содержать данные, позволяющие транспортному уровню принимающей стороны доставлять сообщения необходимому приложению. Сюда же

включаются биты контроля ошибок, с помощью которых получатель определяет количество измененных битов сообщения на маршруте. Затем транспортный уровень передает сегмент на сетевой уровень, который, в свою очередь, добавляет информацию заголовка своего уровня (H_n на рис. 1.24), например, адреса конечных систем источника и приемника, создавая таким образом **дейтаграмму сетевого уровня**. Дейтаграмма затем пересылается на канальный уровень, который (естественно) также добавляет свой собственный заголовок и создает **кадр канального уровня**. Таким образом, мы видим, что пакет на каждом уровне содержит два типа поля — поле заголовка и **поле данных**, которые обычно содержат пакет из уровня, расположенного над ним.

Рис. 1.24. Хосты, маршрутизаторы и канальные коммутаторы реализуют различные наборы уровней, что отражает разницу в их функциональности

Уместным будет провести аналогию с пересылкой сообщения из одного офиса компании в другой с помощью почтовой службы. Предположим, Алиса, работающая в одном офисе, хочет отправить Бобу в другой офис сообщение. Оно будет аналогично *сообщению прикладного уровня*. Алиса помещает письмо в корпоративный конверт и пишет на его лицевой стороне имя Боба и название подразделения, в котором он работает. *Корпоративный конверт* аналогичен *сегменту транспортного уровня*. Он содержит информацию заголовка (имя Боба и номер подразделения), а также инкапсулирует (упаковывает) сообщение прикладного уровня (письмо от Алисы). Когда этот конверт приходит к работникам офиса, отвечающим за отправку корреспонденции, они помещают его внутрь еще одного конверта, который можно послать по обычной го-

родской почте. Работники административного отдела пишут почтовый адрес отправляющего и принимающего офисов на почтовом конверте. В данном случае *почтовый конверт* — это аналог *дейтаграммы* — он инкапсулирует сегмент транспортного уровня (корпоративный конверт), который, в свою очередь, инкапсулирует исходное сообщение (сообщение от Алисы). Почтовая служба доставляет почтовый конверт в офис принимающей стороны — в отдел корреспонденции. Там начинается процесс деинкапсуляции. Сотрудники отдела по работе с корреспонденцией извлекают из почтового конверта корпоративный и направляют его Бобу. Наконец Боб открывает конверт и извлекает сообщение.

Конечно, процесс инкапсуляции может быть гораздо сложнее, чем в описанном выше примере. Например, большое сообщение может быть разделено на ряд сегментов транспортного уровня (а те, в свою очередь, на множество дейтаграмм сетевого уровня). На принимающем конце такие сегменты должны быть заново собраны из входящих в него дейтаграмм.

1.6. Атаки на сети

Интернет стал играть большую роль в деятельности сегодняшних организаций, включая большие и малые компании, образовательные учреждения и органы власти. Многие люди также полагаются на Интернет как на помощника в профессиональной деятельности, в общественной и личной жизни. Но, несмотря на все достоинства Интернета и удобства, которые он предоставляет, существует и другая — темная сторона, когда «плохие парни» пытаются внести хаос в нашу повседневную жизнь, нанося вред нашим компьютерам, парализуя работу служб сети, от которых мы стали сильно зависимы, а также вторгаясь в нашу личную жизнь.

Задача безопасности сетей заключается в том, чтобы знать, как эти плохие парни могут атаковать компьютерные сети, и как мы, будущие эксперты, можем противостоять этим атакам, а лучше всего, как мы можем построить новую архитектуру сетей, которая в первую очередь бы подразумевала защиту от таких угроз. Учитывая частоту и разнообразие существующих атак, а также угрозу новых, более разрушительных в будущем, безопасность сети становится важнейшей темой в области информационных технологий. Актуальность сетевой безопасности — это один из ключевых моментов данной книги.

Так как мы еще недостаточно опытны в области компьютерных сетей и протоколов Интернета, мы начнем с обзора самых распространенных сегодняшних проблем, связанных с безопасностью. Это подогреет наш аппетит для более предметных обсуждений в следующих главах, так что мы начнем с простых вопросов: что может произойти плохого? Насколько уязвимы компьютерные сети? Каковы на сегодняшний день самые основные типы атак на сети?

Злоумышленники могут устанавливать вредоносное ПО на вашем компьютере, используя Интернет

Мы с вами подключаем различные устройства к Интернету для того, чтобы получить какую-нибудь информацию из сети либо передать ее туда. Наша активность в Интернете связана с различными полезными видами деятельности, включая просматривание веб-страниц, получение сообщений электронной почты, скачивание музыкальных файлов, работу в поисковых системах, телефонные вызовы, видео в реальном времени и так далее. Но, к сожалению, Интернет может оказать и вредное влияние, например, инфицировать наши с вами компьютеры широко известным **вредоносным программным обеспечением**. Однажды попав на наш компьютер, такое вредоносное ПО способно на различные пакости, включая удаление файлов и установку шпионских программ, которые могут собирать нашу личную информацию, такую, как персональные пароли, а затем пересылать ее (естественно, используя Интернет) злоумышленникам. Наш подвергшийся атаке компьютер может стать частью сети из тысяч подобных устройств. Такую сеть, известную как **ботнет**, злоумышленники контролируют и используют для рассылки спама, а также проведения атак типа отказа в обслуживании (вскоре будет обсуждаться) против целевых хостов.

Большинство вредоносных программ на сегодняшний день являются **саморазмножающимися**. Заразив один хост, такая программа ищет способ попасть на другие устройства, используя Интернет, а, заразив их, пытается распространиться дальше. Таким образом, распространение вредоносного ПО может происходить с достаточно большой скоростью. Такое ПО обычно бывает в форме вируса или червя. **Вирусы** — это вредоносные программы, которые требуют некоторой формы взаимодействия с пользователем для заражения устройства этого пользователя. Классическим примером вируса является вложение почтового сообщения, которое содержит вредоносный исполняемый код. Когда пользователь получает электронную почту и открывает такое

вложение, то он непроизвольно запускает вредоносное ПО на своем устройстве. Как правило, такие почтовые вирусы размножаются сами по себе: после первого запуска вирус способен послать идентичное сообщение с тем же самым вредоносным вложением, например, каждому получателю из адресной книги пользователя зараженного компьютера. **Черви** — это разновидность вредоносного ПО, которая попадает на устройство без явного взаимодействия с пользователем. Например, если тот запускает какое-нибудь сетевое приложение, где существует уязвимость, используя которую злоумышленник может отправить вредоносную программу. В некоторых случаях приложение без вмешательства пользователя может принимать вредоносную программу из Интернета и запускать ее, создавая тем самым червя. Затем червь в новом зараженном устройстве сканирует сеть в поисках другого хоста, на котором запущено то же сетевое приложение с той же уязвимостью. Найдя такие уязвимые хосты, червь посылает на них копию самого себя. На сегодняшний день распространение вредоносного ПО стало повсеместным, и защититься от него достаточно непросто. В процессе работы с книгой постарайтесь подумать, как ответить на вопрос: что могут предпринять разработчики компьютерных сетей для защиты устройств, подсоединенных к Интернету, от атак вредоносного программного обеспечения?

Злоумышленники могут атаковать сервера и сетевую инфраструктуру

Еще один широкий класс угроз безопасности представляют собой **DoS-атаки** (denial-of-service, отказ в обслуживании). Как следует из названия, DoS-атаки перегружают ресурсы сети и делают их недоступными для пользователей. Объектами DoS-атак могут быть веб-серверы, почтовые серверы, DNS-серверы (обсуждаемые в главе 2), а также сети любой организации. DoS-атаки в Интернете очень распространены: каждый год их происходят тысячи[351, 345]. Большинство DoS-атак в Интернете можно разделить на три категории:

- *Атака на уязвимость.* Данный вид атаки предполагает отправку нескольких хорошо продуманных сообщений приложению или операционной системе, запущенным на сетевом хосте и имеющим уязвимость. Если послать определенную последовательность пакетов уязвимому приложению или операционной системе, то это может привести к остановке работы служб или, что еще хуже, к выводу из строя всего устройства.

- *Переполнение полосы пропускания.* Злоумышленник посылает огромное количество пакетов на целевой хост — такое огромное, что соединение целевого хоста становится перегруженным и, как следствие, сервер становится недоступным для пакетов пользователей.
- *Переполнение запросов на соединение.* Атакующий устанавливает большое количество полуоткрытых TCP-соединений (TCP-соединения обсуждаются в главе 3) на целевом хосте. В результате хост настолько перегружается этими фиктивными соединениями, что прекращает принимать соединения законных пользователей.

Давайте более подробно рассмотрим атаку с помощью переполнения полосы пропускания. Если вспомнить наше обсуждение задержек и потерь в разделе 1.4.2, то будет очевидно, что если сервер имеет скорость R бит/с, то злоумышленнику необходимо отправлять трафик со скоростью приблизительно равной R, чтобы причинить ущерб серверу. При достаточно высоком значении R единственный атакующий источник не способен сгенерировать для этого достаточно трафика. Более того, если трафик исходит из одного источника, то принимающий маршрутизатор вполне может обнаружить атаку и блокировать все пакеты от данного источника, и они не дойдут до сервера. При использовании **распределенных DoS-атак** (distributed DoS или **DDoS**), показанных на рис. 1.25, злоумышленник управляет множественными источниками, и каждый из них направляет огромный трафик на целевой компьютер.

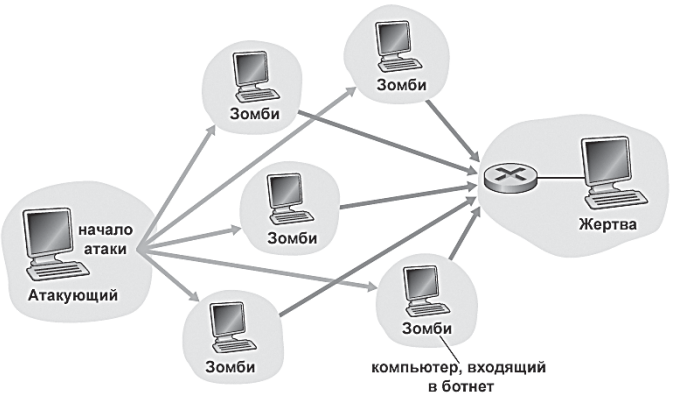


Рис. 1.25. Распределенная DoS-атака

В таком случае совокупная скорость трафика от всех источников должна быть приблизительно равна R, чтобы навредить службам серверов. DDoS-атаки, использующие ботнеты с тысячами зараженных хо-

стов — это широко распространенное явление в наши дни[345]. По сравнению с DoS-атаками с одного хоста, DDoS-атаки представляют большую угрозу, так как их сложнее обнаружить и от них труднее защититься.

В процессе работы с книгой предлагаем вам подумать еще над одним вопросом: как разработчики компьютерных сетей могут защититься от DoS-атак? Позднее мы увидим, что для трех типов DoS-атак применяются различные средства защиты.

Злоумышленники могут перехватывать пакеты

Сегодня многие пользователи выходят в Интернет, используя беспроводные устройства, как, например, ноутбуки, подключенные через Wi-Fi, или портативные устройства типа планшетов или смартфонов, получающие доступ с помощью операторов мобильной связи (обсуждается в главе 6). В то время как повсеместный доступ в Интернет — исключительно удобная вещь, позволяющая мобильным пользователям работать с многочисленными приложениями, это же, с другой стороны, создает большую уязвимость — если в непосредственной близости от беспроводного передатчика расположить пассивный приемник, то можно получить копию любого переданного пакета! Такие пакеты могут содержать в том числе любые виды конфиденциальной информации, включая пароли, PIN-коды, другую важную секретную информацию, а также личные сообщения. Такой пассивный приемник, делающий копию каждого пакета, называется **снифферером (анализатором) пакетов**.

Снифферы могут быть развернуты также и в проводных средах. Например во многих ЛВС на базе Ethernet снифер может получать копии широковещательных пакетов, посланных через ЛВС. Как обсуждалось в главе 1.2, технологии кабельного доступа также используют широковещательные пакеты и, следовательно, тоже подвержены перехвату пакетов. Более того, злоумышленник, получив доступ к маршрутизатору или линии соединения с Интернетом какой-то организации, способен заставить анализатор копировать любой отправленный или полученный пакет. Перехваченные пакеты впоследствии могут быть проанализированы на предмет конфиденциальной информации в другом месте. Снифферы пакетов доступны как свободное ПО на различных вебсайтах, а также в качестве коммерческих продуктов. Профессора, преподающие курсы по компьютерным сетям, часто используют в своих уроках упражнения, включающие написание анализатора пакетов и программ для восстановления данных прикладного уровня. На самом

деле в лабораторной работе с использованием Wireshark[670] (см примеры к книге) вы встретите в точности такой анализатор пакетов! Поскольку программы такого рода являются пассивным программным обеспечением — то есть они не генерируют никаких пакетов, — их очень сложно обнаружить. Следовательно, когда мы посылаем пакеты в беспроводную линию, мы должны понимать, что существует вероятность перехвата их злоумышленниками. Как вы, наверно, догадались, один из лучших методов защиты от перехвата пакетов — это криптография. Ее мы изучим, когда будем проходить безопасность сетей в главе 8.

Злоумышленники могут маскироваться под кого-то, кому вы доверяете

Существует на удивление простой способ (вы вскоре его освоите) создать пакет с произвольным адресом источника, содержимым и адресом приемника и затем отправить этот самодельный пакет в Интернет, который послушно перенаправит его по адресу назначения. Представьте себе ничего не подозревающего получателя (например, Интернет-маршрутизатор), который примет такой пакет, воспримет адрес источника (ложный) как истинный и затем выполнит некоторые команды, встроенные в содержимое пакетов (которые, например, модифицируют его таблицы маршрутизации). Такая возможность внедрить в сеть пакеты с ложным адресом источника называется **IP-спуффингом** (или подменой адреса) и является примером способа, с помощью которого один пользователь может маскироваться под другого.

Чтобы решить данную проблему, нам потребуется *аутентификация конечного пользователя*, то есть механизм, который позволит нам с уверенностью определить, что сообщение исходит именно оттуда, откуда мы думаем. Еще раз призываем вас самих анализировать в процессе работы над главами книги, как это может быть сделано. Мы изучим механизмы конечной аутентификации в главе 8.

Завершая данный раздел, стоит задать вопрос, почему же Интернет стал таким небезопасным местом? Ответ, в сущности, кроется в первоначальной задумке, в том, что он был придуман как модель «прозрачной сети, которая объединяет группу доверяющих друг другу пользователей»[58] — модель, в которой (по определению) нет необходимости для организации безопасности. Многие аспекты исходной архитектуры Интернета четко отражают эту концепцию взаимного доверия. Например, возможность посылать пакеты любому другому пользователю просто так, а не по схеме «запросил/предоставил», а также то, что

идентификация пользователя рассматривается как вещь сугубо добровольная, а не по принципу обязательной аутентификации.

Но сегодняшний Интернет определенно уже не является такой сетью «доверяющих друг другу пользователей». Тем не менее, им по-прежнему нужно обмениваться данными. Пользователи могут взаимодействовать анонимно, а также через третьи стороны (например, веб-кэш, который будет изучен в главе 2, или агенты мобильной связи, которые рассматриваются в главе 6). Они могут также не доверять программному обеспечению, оборудованию и даже эфиру, через который они связываются. По ходу изучения книги, мы уже обозначили достаточно много проблем, связанных с безопасностью: нам надо защищаться от перехвата пакетов, от подмены адресов конечных систем, от DDoS-атак, от вредоносного ПО и т.д. Мы всегда должны помнить, что взаимодействие между доверяющими друг другу пользователями — это скорее исключение, чем правило. Добро пожаловать в мир современных компьютерных сетей!

1.7. История компьютерных сетей и Интернета

В предыдущих разделах этой главы мы сделали обзор технологии компьютерных сетей и Интернета. Теперь у вас достаточно знаний, чтобы произвести впечатление на вашу семью и близких друзей! Однако если вы хотите быть в центре внимания на ближайшей вечеринке, не мешало бы украсить ваш рассказ о компьютерных сетях увлекательными фактами из истории Интернета[596].

1.7.1. Развитие коммутации пакетов: 1961–1972

Компьютерные сети и сегодняшний Интернет берут свои истоки в начале 60-х годов, когда телефонная сеть была основным средством связи в мире. Как вы помните из раздела 1.3, телефонная сеть использует коммутацию каналов для передачи информации от отправителя к получателю — оптимальный выбор для голосового сигнала. Учитывая растущую важность вычислительных машин в начале 60-х годов, а также появление компьютеров, использующих принцип разделения времени, естественно, возникла потребность найти способ объединить их таким образом, чтобы можно было поделить ресурсы между территориально удаленными пользователями. Трафик, создаваемый такими пользователями, был *неравномерным*: периоды активности, когда удаленному ком-

пьютеру посылалась команда, сменялись периодами бездействия, когда ожидался ответ.

Три группы исследователей независимо друг от друга[316] начали разработку технологии коммутации пакетов как эффективной и надежной альтернативы технологии коммутации каналов. Первой опубликованной работой по методу коммутации пакетов была работа Клейнрока[286, 287], который в то время заканчивал Массачусетский технологический институт. Используя теорию очередей, Клейнрок в своей работе продемонстрировал эффективность метода коммутации пакетов для источников пульсирующего (неравномерного) трафика. В 1964 году Пол Бэран[39] в институте корпорации RAND начал исследование на предмет использования коммутации пакетов для безопасной передачи голоса в сетях Министерства обороны США. В это же время в национальной физической лаборатории в Англии также разрабатывали свои идеи пакетной коммутации Дональд Дэвис (Donald Davies) и Роджер Скэнтлбери (Roger Scantlebury).

Проекты ученых в Массачусетском технологическом институте, а также корпорации RAND и национальной физической лаборатории заложили основы сегодняшнего Интернета. Но и другие разработки, датируемые началом 60-х годов, внесли большой вклад в развитие Интернета. В частности, коллеги Клейнрока по Массачусетскому институту Джозеф Ликлайдер (Joseph Licklider)[126] и Лоуренс Робертс (Lawrence Roberts) в начале 60-х годов руководили программой развития компьютерных технологий в агентстве по перспективным научно-исследовательским разработкам (Advanced Research Projects Agency, ARPA) в США. Робертс опубликовал принципиальную схему компьютерной сети ARPAnet[565] — первой компьютерной сети с коммутацией пакетов, которая является прямым предком сегодняшнего Интернета. В День Труда в 1969 году под непосредственным руководством Леонарда Клейнрока первый коммутатор пакетов был установлен в Калифорнийском университете в Лос-Анджелесе. Еще три таких коммутатора чуть позже появились в Стэндфордском исследовательском институте (Stanford Research Institute, SRI), в Калифорнийском университете в Санта-Барбаре (UC Santa Barbara) и в университете Юты (University of Utah) (рис. 1.26). Молодая сеть — предшественник Интернета — включала в себя к концу 1969 года всего 4 узла. Клейнрок вспоминает самую первую попытку использовать сеть для удаленного доступа, которая потерпела неудачу и завершилась выводом из строя системы[291].

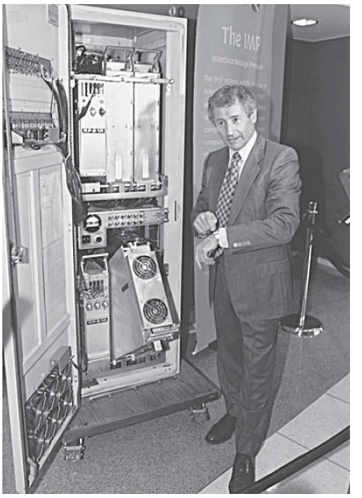


Рис. 1.26. Один из первых коммутаторов пакетов

Когда в 1972 году Роберт Кан (Robert Kahn) организовал первую публичную демонстрацию сети ARPAnet, она уже насчитывала в своем составе 15 узлов. После того как был разработан первый протокол обмена между хостами сети ARPAnet, известный как протокол управления сетью (network-control protocol, NCP[416]), стало возможным написание использующих его приложений. Тогда же, в 1972 году, была написана первая программа для работы с электронной почтой, разработчиком которой стал Рэй Томлинсон (Ray Tomlinson).

1.7.2. Развитие частных сетей и Интернета: 1972–1980

Первоначально ARPAnet являлась закрытой изолированной сетью. Для взаимодействия с любым ее узлом нужно было подключаться к интерфейсному процессору сообщений (IMP). К середине 70-х годов появились другие отдельно стоящие сети с коммутацией пакетов кроме ARPAnet: ALOHANet — коротковолновая сеть, объединившая университеты на Гавайских островах[5] с сетями агентства DARPA[424], которому пакеты передавались через спутниковую и радиосвязь[272]; Telenet — коммерческая сеть компании BBN, построенная на технологии ARPAnet; Cyclades — французская сеть с коммутацией пакетов, представленная впервые Луи Пузеном (Louis Pouzin)[635]; сети с разделением времени, такие как Tymnet и сеть GE Information Services[592]; сеть SNA от компании IBM (1969–1974), которая также была аналогична сети ARPAnet[592].

Количество сетей росло, необходимо было разрабатывать такую архитектуру, которая бы объединила все их вместе. Первые разработки по взаимодействию сетей были проведены Винтоном Серфом и Робертом Каном при спонсорстве агентства DARPA[80]. Эти разработки явились, в сущности, основой создания *сети сетей*, и именно в это время был применен термин «*Интернет*» для обозначения такой архитектуры.

Эти принципы архитектуры сетей были реализованы в первом протоколе TCP, который, конечно, отличался от TCP сегодняшнего дня. Ранние версии протокола TCP совмещали надежную последовательную доставку данных, используя ретрансляцию между конечными системами (которая и сейчас применяется в TCP) с транспортными функциями (которые сегодня выполняет протокол IP). Ранние эксперименты с протоколом TCP, обозначившие важность ненадежной транспортной службы без управления потоком для таких приложений, как передача пакетов голосовыми сообщениями, привели к отделению IP-протокола от TCP и разработке протокола UDP. Следовательно, три основных протокола — TCP, UDP и IP — начали играть ключевую роль в сетевом взаимодействии уже к концу 1970-х годов.

Наряду с исследованиями агентства DARPA велась и другая активная деятельность в области компьютерных сетей. На Гавайских островах Норман Абрамсон (Norman Abramson) разработал ALOHAnet — сеть на базе пакетной передачи по радиосигналу, которая позволяла обмениваться между собой множественным удаленным пользователям[5]. Протокол ALOHA, использовавшийся в этой сети, явился первым протоколом множественного доступа, который позволял географически распределенным пользователям разделять между собой одну среду передачи (а именно радиочастоту). Меткалф (Metcalfe) и Боггс (Boggs), разрабатывая протокол Ethernet для проводных сетей вещания, опирались именно на принцип работы протокола множественного доступа, созданного Абрамсоном[342]. Заметим, что разработка протокола Ethernet Меткалфом и Боггсом была вызвана необходимостью обеспечить соединение множественных компьютеров, принтеров и общих дисков[386]. Таким образом, Меткалф и Боггс заложили фундамент сегодняшних компьютерных локальных вычислительных сетей 25 лет назад, задолго до революции в компьютерах и сетях.

1.7.3. Рост компьютерных сетей: 1980–1990

К концу 70-х годов сеть ARPAnet включала в себя уже около двухсот конечных систем, а к концу 80-х годов число хостов, объединенных

в сеть, достигло сотни тысяч, и структура этой сети, соединявшей множество других, уже напоминала сегодняшний Интернет. Именно 80-е стали временем стремительного роста сетей.

Большое влияние на этот рост оказали попытки создать компьютерные сети, которые бы объединяли локальные сети университетов. Например, сеть BITNET, которая обеспечила услуги электронной почты и обмена файлами между несколькими университетами на северо-западе Соединенных Штатов; сеть CSNET (научная компьютерная сеть), сформированная для объединения исследовательских учреждений, у которых не было подключения к сети ARPAnet. В 1986 году при спонсорстве национального фонда науки США (NSF) была создана сеть NSFNET для обеспечения доступа к суперкомпьютерным центрам. Начав с пропускной способности в 56 Кбит/с, скорость магистрали сети NSFNET к концу десятилетия достигла 1,5 Мбит/с, и NSFNET стала главной магистралью, объединившей региональные сети.

Большинство элементов архитектуры сегодняшнего Интернета были заложены в сети ARPAnet. В качестве стандартного протокола обмена между хостами для сети ARPAnet 1 января 1983 года был официально утвержден в протокол TCP/IP (заменив собой протокол NCP). Переход от NCP к TCP/IP[422] стал знаковым событием. С того дня все конечные устройства в Интернете стали использовать именно TCP/IP для передачи данных. В конце 1980-х в протокол были внесены важные дополнения, целью которых являлось внедрение контроля перегрузки при передаче данных между хостами[260]. Также была разработана система доменных имен (DNS), позволившая связать имена хостов в понятном человеку виде (например, gaia.cs.umass.edu) и их 32-разрядные IP-адреса[430].

Параллельно с развитием сети ARPAnet в США в начале 1980-х годов во Франции был запущен проект Minitel, который имел амбициозную цель — провести компьютерную сеть в каждый дом. Спонсируемый французским правительством, проект представлял собой открытую сеть с коммутацией пакетов (основанную на протоколе X.25) и включал серверы Minitel и недорогие терминалы для пользователей со встроенными низкоскоростными модемами. Большого успеха удалось достичь в 1984 году, когда правительство Франции стало раздавать каждому желающему бесплатный терминал Minitel. Сеть предоставляла как бесплатные услуги, так и те, за пользование которыми бралась ежемесячная абонентская плата. В середине 1990-х годов на пике своего развития Minitel предлагала более 20 тысяч различных услуг — от удаленного банков-

ского обслуживания до предоставления доступа к специализированным исследовательским базам данных. Большая часть жителей во Франции пользовалась услугами сети Minitel за 10 лет до того, как американцы впервые услышали слово «Интернет».

1.7.4. Интернет-взрыв: 1990-е

Девяностые годы прошлого века сопровождались рядом событий, которые знаменовали продолжение развития и коммерциализацию Интернета. Прекратила свое существование сеть ARPAnet — прародитель Интернета. В 1991 году NSFNET наложила ограничение на ее использование в коммерческих целях, а спустя четыре года она сама прекратила свое существование, передав функции по обслуживанию магистрального трафика коммерческим Интернет-провайдерам. Главным событием 90-х годов, очевидно, стало появление Всемирной паутины (World Wide Web или просто Web), которая принесла Интернет в каждый дом миллионам людей по всему миру. Паутина послужила платформой для разработки и внедрения сотен новых приложений, без которых мы не обходимся и сейчас, включая поисковые системы (например, Google или Bing), электронную коммерцию (Amazon или eBay), а также социальные сети (например, Facebook или Одноклассники).

Проект всемирной паутины создал Тим Бернерс-Ли, между 1989 и 1991 годами, когда он работал в лаборатории по ядерным исследованиям (CERN)[48]. Он опирался на идеи, предложенные в ранних работах по гипертексту в 40-х годах Вэнивером Бушем[66] и в 60-х годах Тедом Нельсоном (Ted Nelson)[675]. Бернерс-Ли и его коллеги разработали начальные версии HTML, HTTP, веб-сервера и браузера — четыре ключевых компонента всемирной паутины. Примерно к концу 1993 года в мире насчитывалось более 200 веб-серверов, и они были всего лишь предвестниками того, что ожидалось впереди. Примерно в это же время несколько исследователей разрабатывали веб-браузеры с пользовательскими интерфейсами. Среди них был Марк Андрессен (Marc Andreessen), совместно с Джимом Кларком (Jim Clark) организовавший компанию Mosaic Communications, которая позже получила название Netscape Communications Corporation[120, 402]. К 1995 году студенты университетов уже использовали браузер Netscape для просмотра веб-страниц. Примерно в это же время большие и малые компании стали применять веб-серверы в коммерческих целях. В 1996 году компания Microsoft начала производить браузеры, тем самым положив начало

войне между Netscape и Microsoft, которая завершилась победой последней[120].

Вторая половина 1990-х годов знаменует собой период небывалого роста и огромных инноваций в глобальной сети. Тысячи компаний и различных проектов разрабатывают продукты и всевозможные службы для работы в сети. К концу тысячелетия Интернет поддерживал сотни популярных приложений, включая четыре основные группы:

- Электронная почта, включая пересылку файлов, сообщений, а также доступ к почте через веб-интерфейс
- Веб-приложения, включая просмотр веб-сайтов и Интернет-коммерцию
- Службы мгновенных сообщений со списками контактов
- Одноранговый совместный доступ к файлам, например, в формате MP3; первым из таких приложений явилась программа Napster

Интересно, что первые два вида приложений были разработаны сообществом ученых-исследователей, в то время как два последних — молодыми предпринимателями.

Годы с 1995 по 2001 были периодом «американских горок» для Интернета на финансовых рынках. Сотни новых Интернет-проектов появились на фондовом рынке, и в результате многие компании были оценены в миллиарды долларов, не имея до этого никаких значительных доходов. Рынок Интернет-акций рухнул в 2000–2001 годах, и многие проекты были закрыты. Тем не менее ряд компаний, включая Microsoft, Cisco, Yahoo, E-Bay, Google и Amazon, имели огромный успех.

1.7.5. Новое тысячелетие

Инновации в области компьютерных сетей продолжают внедряться быстрыми темпами. Продвижение сетевых технологий в настоящее время происходит на всех фронтах, в том числе в развертывании высокопроизводительных маршрутизаторов и увеличении скоростей передачи данных, как в магистральных сетях, так и в сетях доступа. Но следующие события заслуживают особого внимания:

- С начала нового тысячелетия мы наблюдаем активное развертывание широкополосного домашнего доступа в Интернет, включая использование не только кабельных и DSL-модемов, но и оптово-

локонных технологий (как описано в разделе 1.2). Как результат, наличие высокоскоростного доступа в Интернет подготовило почву для использования богатого набора видео-приложений, в том числе для размещения создаваемого пользователями видео (например, YouTube), предоставляемого по запросу контента с потоковыми видеоматериалами и телевизионными шоу (Netflix), а также для организации видеоконференций с большим числом участников (Skype).

- Повсеместное распространение высокоскоростных (54 Мбит/с и выше) общественных беспроводных сетей и среднескоростной (до нескольких Мбит/с) доступ в Интернет через 3G- и 4G-сети операторов сотовой связи не только дают возможность быть постоянно онлайн, но и расширяют многообразие разрабатываемых приложений. Количество беспроводных устройств, подключенных к Интернету, в 2011 году уже превзошло число проводных. В результате высокоскоростные беспроводные технологии привели к быстрому становлению мобильных компьютерных устройств (смартфонов и планшетов под управлением операционной системы iOS, Android и т.п.), которые позволяют своим владельцам наслаждаться постоянным мобильным доступом к Всемирной сети.
- Социальные сети, такие как Facebook и Twitter, — это целое общественное явление, объединившее огромные группы людей. Многие пользователи Интернета сегодня, можно сказать, «живут» в социальных сетях, благодаря чему все больше возрастает спрос на разработку новых сетевых приложений и многопользовательских игр.
- Как обсуждалось в разделе 1.3.3, поставщики онлайн-сервисов, такие как Google и Microsoft, развернули свои собственные обширные частные сети, которые не только соединяют их многочисленные распределенные центры обработки данных, но и сами выступают в качестве поставщиков Интернет-услуг, устанавливая пиринговые соединения с провайдерами нижних уровней. Как результат, Google предоставляет пользователю результаты поиска и доступ к электронной почте почти мгновенно — так быстро, как если бы центры обработки данных были расположены в компьютере пользователя.
- Многие компании, занимающиеся Интернет-коммерцией, в настоящее время запускают свои приложения в «облаке» — таком, как EC2 от Amazon, Google Application Engine, или в Azure от компании Microsoft. Облачными технологиями уже успешно пользуются мно-

гие коммерческие компании и образовательные учреждения, размещая свои Интернет-приложения (например, электронную почту или веб-хостинг). Компании, предоставляющие облачные услуги, не только обеспечивают приложениям масштабируемые вычисления и среду хранения, но и доступ к своим высокопроизводительным частным сетям.

1.8. Заключение

В этой главе мы изучили огромное количество материала! Мы познакомились с различными элементами аппаратного и программного обеспечения, которые составляют Интернет в частности и компьютерные сети вообще. Мы начали с периферии сети, рассмотрев конечные системы и приложения, а также транспортные услуги, предоставляемые приложениям, исполняющимся в конечных системах. Мы также рассмотрели технологии канального уровня и физическую среду, использующуюся обычно в сетях доступа. Затем мы заглянули глубже внутрь — в ядро сети, познакомившись с коммутацией пакетов и коммутацией каналов — двумя ключевыми подходами к передаче данных в телекоммуникационных сетях, рассмотрели сильные и слабые стороны каждого подхода. Мы также изучили структуру глобальной сети Интернет, которая представляет собой сеть сетей. Мы увидели, что иерархическая структура, состоящая из провайдеров высших и низших уровней, позволяет Интернету легко масштабироваться и включать в себя тысячи новых сетей.

Во второй части этой главы мы рассмотрели несколько важных тем. Сначала мы обсудили причины задержек и потерь пакетов, а также пропускную способность в сети с коммутацией пакетов. Мы разработали простые количественные модели для определения задержек передачи, распространения и ожидания, а также для пропускной способности; мы будем широко использовать эти модели задержек в домашних упражнениях на протяжении всей книги. Далее мы рассмотрели уровни протоколов и модели обслуживания, ключевые архитектурные принципы в области сетевых технологий; мы будем также к ним обращаться на протяжении нашей книги. Кроме того, мы сделали обзор некоторых из наиболее распространенных сегодня сетевых атак в Интернете и закончили наше введение в тему компьютерных сетей краткой историей их развития. Первая глава сама по себе является мини-экскурсом в область компьютерных сетей.

Итак, мы с вами действительно охватили огромный объем материала! Если вы немного перегружены, не волнуйтесь. В следующих главах мы будем возвращаться ко всем этим идеям, раскрывая их более подробно (и это не угроза, а обещание!). На данный момент вы завершаете начальную главу, и мы надеемся, что ваша интуиция будет двигать вас дальше, к подробному изучению составных частей компьютерной сети, к новой сетевой лексике (не стесняйтесь, кстати, возвращаться к этой главе, чтобы освежить в памяти те или иные термины и понятия). Постоянно растущее желание знать о сетевых технологиях больше — вот что будет двигать нас вперед в остальной части этой книги.

План этой книги

Перед началом любого путешествия не мешает взглянуть на карту, чтобы ознакомиться с маршрутом, изучить главные и второстепенные дороги и перекрестки на пути к месту назначения. В нашем с вами путешествии конечным пунктом является глубокое понимание того, *как*, *что* и *почему* в компьютерных сетях. Наша карта представляет собой последовательность глав этой книги:

1. Компьютерные сети и Интернет
2. Прикладной уровень
3. Транспортный уровень
4. Сетевой уровень
5. Канальный уровень и локальные сети
6. Беспроводные и мобильные сети
7. Мультимедийные сетевые технологии
8. Сетевая безопасность
9. Администрирование вычислительной сети

Главы со 2 по 5 являются в этой книге основными. Вы, очевидно, заметили, что они соответствуют четырем главным уровням стека протоколов Интернета. Отметим, что наше путешествие начнется с верхней части, а именно с прикладного уровня, и пойдет вниз по стеку. Причиной такого подхода, основанного на делении на уровни, является то, что как только мы вникаем в суть приложений, мы можем понять, какие сетевые службы необходимы для их поддержки. Мы можем затем, в свою очередь, изучить различные способы реализации таких служб в рамках

сетевой архитектуры. Таким образом, освоение одного уровня мотивирует к изучению следующего и т.д.

Вторая половина книги — главы с 6 по 9 — фокусируется на четырех чрезвычайно важных (и, в некоторой степени, независимых) темах в современных компьютерных сетях. В главе 6 мы рассмотрим беспроводные и мобильные сети, в том числе беспроводные локальные сети (включая Wi-Fi и Bluetooth), сети сотовой связи (в том числе GSM, 3G и 4G), а также мобильные коммуникации вообще (как в IP-, так и в GSM-сетях). В главе 7 (мультимедийные сетевые технологии) мы изучим аудио- и видео-приложения, такие как IP-телефония, видеоконференции, а также потоковое вещание мультимедийных данных. Мы также рассмотрим, как организовать сеть с коммутацией пакетов, чтобы обеспечить стабильное качество обслуживания аудио- и видеоприложений. В главе 8 (сетевая безопасность) мы сначала затронем основы шифрования данных и сетевой безопасности, а затем рассмотрим, как основные принципы безопасности применяются в Интернете. Последняя глава (администрирование вычислительной сети) раскрывает ключевые принципы администрирования вычислительной сети, а также описывает используемые для этих целей основные Интернет-протоколы.

Глава 2
ПРИКЛАДНОЙ УРОВЕНЬ

В сетевых приложениях заключается *весь смысл* существования компьютерных сетей — если бы не было полезных приложений, нам бы не нужны были сетевые протоколы, их поддерживающие. С самого начала существования Интернета было создано множество как серьезных и полезных, так и развлекательных сетевых приложений, которые, без сомнения, явились основным движущим фактором успеха всемирной сети, мотивируя людей сделать ее частью своей повседневной деятельности на работе, дома и на учебе.

Интернет-приложения включают в себя классические текстовые приложения, которые стали популярны еще в 70-х и 80-х годах: текстовая электронная почта, удаленный доступ к устройствам по сети, передача файлов, сообщения групп новостей. Среди них также ключевые приложения середины 90-х годов, которые легли в основу всемирной паутины: веб-обозреватели, поисковые системы, системы электронной торговли. В этот же ряд можно добавить приложения для систем мгновенных сообщений и одноранговых файлообменных сетей, представленные в конце тысячелетия. Начиная с 2000 года мы наблюдали взрыв популярности аудио- и видеоприложений, включая IP-телефонию, а также видеоконференции по IP-сетям, такие как Skype, приложений для размещения пользовательского контента, например, YouTube, для загрузки видеоконтента по запросу, таких как NetFlix. В это же время распространяется всеобщее увлечение многопользовательскими онлайн-играми, такими как Second Life или World Of Warcraft. Совсем недавно появилось новое поколение приложений для социальных сетей, таких как Facebook или Twitter, которые создали своеобразную надстройку из общественных сетей над сетями из маршрутизаторов и линий связи. Очевидно, что эта сфера будет развиваться и в дальнейшем, и вполне возможно, что некоторые из наших читателей станут создателями следующего поколения важнейших Интернет-приложений!

В этой главе мы изучим теоретические и практические аспекты сетевых приложений. Мы начнем с определения ключевых понятий сетевого уровня, таких как сетевые службы, требуемые для приложений,

клиенты и серверы, процессы и интерфейс транспортного уровня. Некоторые из сетевых приложений мы рассмотрим детально, например веб-приложения, электронную почту, службу DNS, а также приложения для файлообменных сетей (глава 8 фокусируется на мультимедийных приложениях, включая потоковое видео и IP-телефонию). Затем мы познакомимся с разработкой сетевых приложений с использованием протоколов TCP и UDP. В частности, мы изучим API (интерфейс программирования приложений) сокетов и познакомимся с некоторыми простыми клиент-серверными Интернет-приложениями, написанными на языке Python. Также в конце этой главы мы рассмотрим некоторые интересные и даже забавные примеры программирования сокетов.

Начинать наше изучение протоколов именно с прикладного уровня очень удобно. Мы будем базироваться на аналогии: так как мы знакомы со многими из приложений, основанных на протоколах прикладного уровня, которые мы будем изучать, это даст нам почувствовать, что такое протокол вообще, а затем мы по тому же принципу приступим к изучению протоколов других уровней: транспортного, сетевого и канального.

2.1. Принципы сетевых приложений

Предположим, у вас есть идея нового сетевого приложения. Возможно, что оно принесет величайшую пользу человечеству или просто порадует вашего учителя, сделает вас миллионером или просто займет на досуге — какой бы ни была ваша мотивация, давайте подумаем, как вам воплотить идею в жизнь.

Ключевую часть разработки сетевых приложений составляет написание программ, которые работают на различных конечных системах и общаются друг с другом по сети. Например, веб-приложение — это две различные программы, взаимодействующие друг с другом: браузер, запущенный на хосте пользователя (настольном компьютере, ноутбуке, планшете, смартфоне и так далее), и веб-сервер, работающий на серверном хосте. Другой пример — это одноранговые системы совместного доступа к файлам, где на каждом из хостов, который участвует в файловом обмене, запущена такая программа. В этом случае программы на различных хостах могут быть аналогичными или даже идентичными.

Таким образом, для разработки нового приложения вам нужно написать программное обеспечение, которое бы работало на различных

конечных системах. Вы можете использовать, например, языки Си, Java или Python. Важно отметить здесь, что вам не нужно писать программное обеспечение для устройств, составляющих ядро сети, таких как маршрутизаторы или коммутаторы канального уровня. Но даже если бы вы и задались такой целью, вы бы не смогли это сделать, ведь, как мы узнали в главе 1 и как было показано ранее на рис. 1.24, данные устройства функционируют не на прикладном уровне, а на сетевом и более низких уровнях.

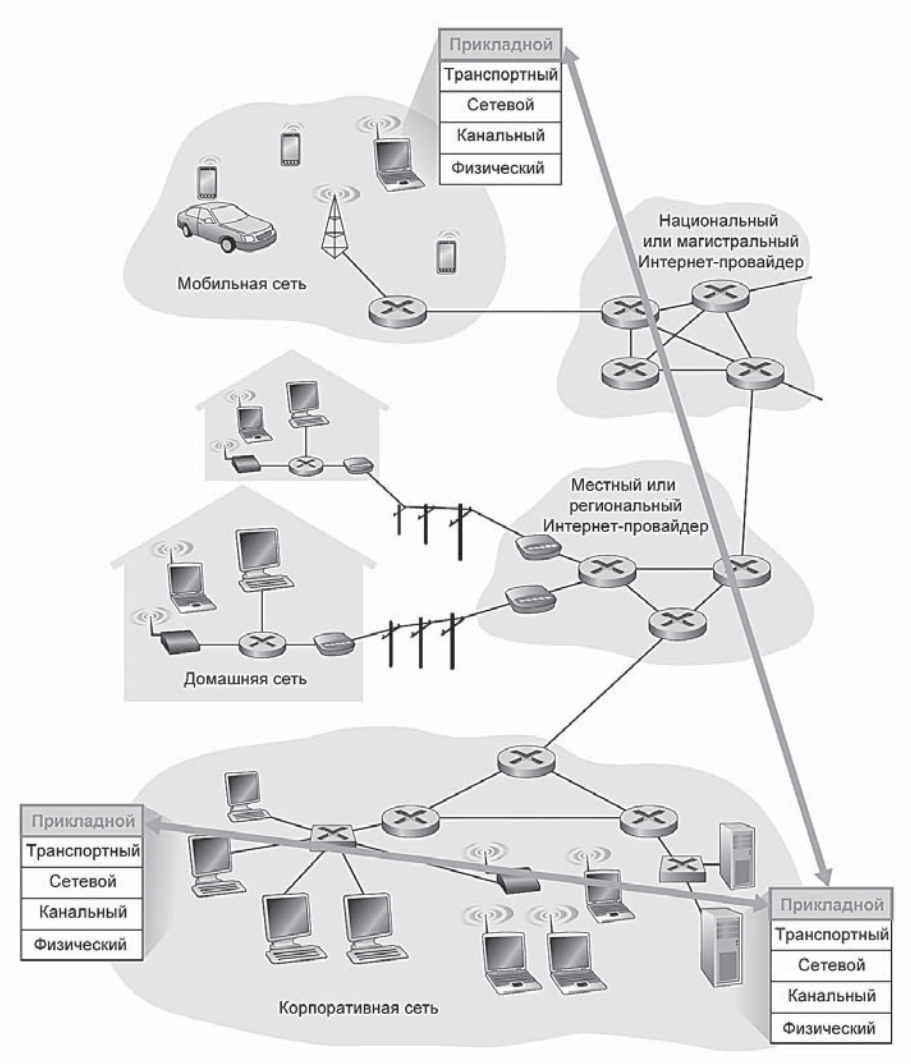


Рис. 2.1. Взаимодействие сетевых приложений происходит между конечными системами на прикладном уровне

Именно тот принципиальный факт, что программное обеспечение приложений для сети относится исключительно к конечным системам (как показано на рис. 2.1), способствовал быстрому развитию и распространению огромной массы сетевых приложений.

2.1.1. Архитектура сетевых приложений

Перед тем как углубляться в разработку программного обеспечения, нам нужно набросать план построения вашего приложения. Необходимо помнить, что архитектура приложений и архитектура сети (например, пятиуровневая архитектура Интернета, обсуждаемая в главе 1) — это разные вещи. С точки зрения разработчика приложения, архитектура сети постоянна и предлагает определенный набор служб приложениям. С другой стороны, **архитектура приложения** создается разработчиком этого приложения и определяет, каким образом оно будет строиться на различных конечных системах. Выбирая архитектуру приложения, разработчик, скорее всего, выберет одну из двух доминирующих архитектурных парадигм, используемых в современной разработке сетевых приложений: клиент-серверная или одноранговая (P2P).

В **клиент-серверной архитектуре** существует один хост, называемый *сервером*, который постоянно находится в режиме онлайн и обслуживает запросы от других многочисленных хостов, называемых *клиентами*. Классическим примером является веб-приложение, для которого постоянно работающий веб-сервер обслуживает запросы, поступающие от браузеров, запущенных на клиентских хостах. Когда веб-сервер получает запрос объекта от клиентского хоста, он в ответ отправляет запрашиваемый объект этому хосту. Заметим, что в данном виде архитектуры клиенты непосредственно не связываются друг с другом; например, в веб-приложении два браузера напрямую не обмениваются информацией. Еще одной характеристикой клиент-серверной архитектуры является то, что сервер имеет фиксированный, известный всем адрес, называемый IP-адресом (обсудим вскоре). Поскольку сервер имеет постоянный, известный адрес и всегда включен, клиент может всегда взаимодействовать с ним, отправляя пакеты на IP-адрес этого сервера. Некоторые из хорошо известных приложений клиент-серверной архитектуры — это Всемирная паутина, FTP, Telnet и электронная почта. Данная архитектура показана на рис. 2.2а.

Очень часто при работе клиент-серверных приложений серверный хост не способен в одиночку обрабатывать многочисленные запросы

от клиентов. Например, если бы веб-сайт популярной социальной сети включал в себя один сервер, то он быстро был бы перегружен запросами. По этой причине очень часто используются **центры обработки данных (дата-центры)**, содержащие в себе большое количество хостов и образующие мощный виртуальный сервер. Наиболее популярные Интернет-службы — такие как поисковые системы (например, Google или Bing), сервисы продаж через Интернет (например, Amazon или e-Bay), электронная почта с доступом через веб-интерфейс (например, Gmail или Yahoo Mail), приложения социальных сетей (например, Facebook или Twitter) — используют один или более центров обработки данных. Как обсуждалось в разделе 1.3.3, компания Google имеет от 30 до 50 таких центров по всему миру, которые совместно обрабатывают поисковые запросы, обеспечивают работу Gmail, YouTube и других служб. Современные центры обработки данных (ЦОД) могут насчитывать сотни тысяч серверов, а значит, провайдеру услуг потребуются значительные расходы не только на поддержку и обслуживание ЦОДов, но и дополнительные затраты, связанные с передачей данных из их центров обработки.

В **одноранговой (P2P) архитектуре** применение серверов или центров обработки сведено до минимума или вообще до нуля. Вместо них приложения используют непосредственное взаимодействие между парой соединенных хостов, называемых *пирами (а также партнерами или узлами)*. Пирами являются обычные настольные компьютеры или ноутбуки, контролируемые пользователями и размещаемые в учебных заведениях, офисах или домах. Так как пиры взаимодействуют без выделенного сервера, такая архитектура называется одноранговой (peer-to-peer). Многие из сегодняшних популярных приложений, использующих наиболее интенсивный трафик, основываются на такой одноранговой архитектуре. Сюда можно отнести файлообменные приложения, например, BitTorrent, ускорители загрузок (например, Xunlei), IP-телефонию (например, Skype) и IP-телевидение (например, Kankan и PPstream). Одноранговая архитектура показана на рис. 2.2б. Отметим, что некоторые приложения построены с использованием гибридной архитектуры, где совмещаются клиент-серверный и одноранговый подходы. Например, во многих приложениях систем мгновенных сообщений серверы используются для отслеживания IP-адресов пользователей, но сообщения от пользователя к пользователю посылаются напрямую между двумя хостами (не проходя через промежуточные серверы).

Одним из наиболее выигрышных свойств одноранговой архитектуры является ее **самомасштабируемость**. Например, в файлообменном

приложении каждый пир не только нагружает систему своими запросами файлов, но и, раздавая файлы другим узлам, в то же время увеличивает скоростной ресурс системы. Такая архитектура эффективна еще и с точки зрения стоимости, так как обычно не требует значительных расходов на серверную инфраструктуру (в отличие от дорогих центров обработки данных в случае с клиент-серверной архитектурой).

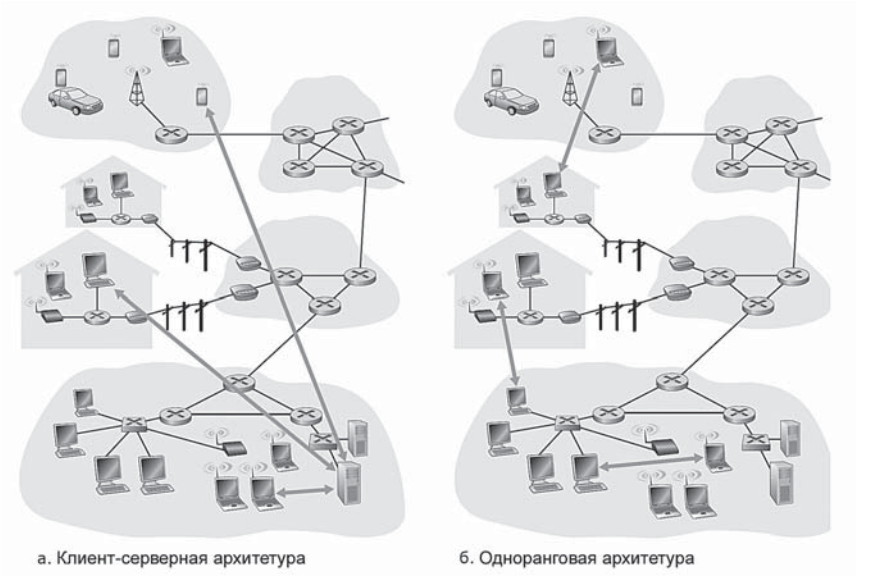


Рис. 2.2. (а) Клиент-серверная архитектура; (б) Одноранговая архитектура

Однако использование одноранговых приложений связано с темя довольно значительными проблемами:

1. *Адаптация к сетям доступа*. Большинство резидентных сетей доступа (включая DSL и кабельные сети) построены по асимметричному принципу, когда скорость входящего трафика значительно превышает скорость исходящего. Поскольку одноранговые приложения для обработки видеопотока и обмена файлами смещают исходящий трафик от серверов к сетям доступа, нагрузка на эти сети значительно увеличивается. Поэтому в будущем одноранговые сети нужно проектировать таким образом, чтобы они были адаптированы к сетям доступа в плане балансировки нагрузки[677].

2. *Безопасность*. Одноранговые сети в связи с их широким распространением и открытой природой представляют собой проблему безопасности[143, 681, 320, 363, 131, 313, 314].

3. *Стимулирование*. Успешное будущее одноранговых сетей связано с задачей привлечения пользователей, которые добровольно будут предлагать свои ресурсы вычисления, хранения и передачи данных для приложений. Организация такой системы стимулирования также является одной из проблем будущего одноранговых сетей[159, 394, 25, 325].

2.1.2. Взаимодействие процессов

Перед тем как начать разрабатывать сетевое приложение, у вас должно быть базовое понимание того, как программы, запущенные на многочисленных конечных системах, взаимодействуют друг с другом. С точки зрения операционных систем, на самом деле взаимодействуют не программы, а **процессы**. Процесс можно рассматривать как программу, запущенную на конечной системе. Когда процессы работают на одной конечной системе, они могут общаться друг с другом с помощью средств межпроцессного взаимодействия, используя набор правил, регулируемый операционной системой конкретного устройства. Но в данной книге нас не интересует, как общаются между собой процессы на одном хосте. Вместо этого мы будем рассматривать взаимодействие процессов, запущенных на *различных* хостах (и, вполне вероятно, отличающимися операционными системами).

Процессы на двух различных конечных системах взаимодействуют друг с другом, обмениваясь **сообщениями** через компьютерную сеть. Процесс на хосте-источнике создает и отправляет сообщения в сеть; процесс на хосте-приемнике получает эти сообщения и, возможно, отправляет в ответ обратные. Рисунок 1.1 показывает, что процессы взаимодействуют друг с другом на прикладном уровне пятиуровневого стека протоколов.

Клиентский и серверный процессы

Сетевое приложение состоит из пар процессов, которые отправляют сообщения друг другу по сети. Например, в веб-приложении процесс браузера клиента обменивается сообщениями с процессом веб-сервера. В системе однорангового файлового обмена файл передается из процесса одного хоста в процесс другого хоста. Для каждой пары взаимодействующих процессов обычно обозначают один из них как **клиентский**, а другой как **серверный**. Например, браузер — это клиентский процесс, а веб-сервер — это серверный процесс. В системе файлового обмена

пир, который загружает файл, является клиентом, а пир, выгружающий файл, считается сервером.

Вы, возможно, замечали, что в некоторых приложениях (например, одноранговый файловый обмен) процесс может быть как клиентским, так и серверным. В действительности в такой системе процесс может как выгружать, так и загружать файлы. Тем не менее в контексте любого конкретного сеанса взаимодействия между парой процессов мы всегда обозначаем один процесс как клиент, а другой как сервер. Определим понятия клиентского и серверного процессов следующим образом:

В контексте сеанса взаимодействия между парой процессов тот, который инициирует это взаимодействие (то есть первоначально контактирует с другим процессом в начале сеанса), обозначается как ***клиентский****. Процесс, который ожидает контакта для начала сеанса, обозначаем как* ***серверный****.*

Итак, процесс браузера инициирует контакт с процессом веб-сервера; следовательно, процесс браузера является клиентским, а процесс веб-сервера является серверным. При одноранговом файловом обмене, когда пир А запрашивает файл у пира Б, то пир А — клиент, а пир Б — сервер в контексте этого конкретного сеанса взаимодействия. Иногда мы используем терминологию «клиентская и серверная сторона (или часть) приложения». В конце этой главы мы рассмотрим простой пример для случаев клиентской и серверной частей сетевых приложений.

Интерфейс между процессом и компьютерной сетью

Как было отмечено выше, большинство приложений состоят из пар взаимодействующих процессов, отправляющих друг другу сообщения. Любое из них должно проходить через нижележащую сеть. Процессы отправляют и принимают сообщения через программный интерфейс, называемый **сокетом**. Рассмотрим аналогию для лучшего понимания процессов и сокетов. Процесс аналогичен дому, а его сокет подобен двери в этом доме. Когда процесс пытается отправить сообщение другому процессу на удаленном хосте, он «проталкивает» сообщение через свою дверь (сокет). Такая отправка предполагает, что с другой стороны этой двери существует некоторая транспортная инфраструктура, которая организует доставку сообщения до двери процесса назначения. Когда сообщение прибывает на хост назначения, оно проходит через дверь принимающего процесса (через сокет). Принимающий процесс затем обрабатывает это сообщение.

Рисунок 2.3 демонстрирует взаимодействие между двумя процессами через сокеты по Интернету. (Этот рисунок предполагает, что базовый транспортный протокол, используемый процессами — это протокол TCP.) Как мы видим из рисунка, сокет — это интерфейс между прикладным и транспортным уровнями внутри хоста. Его определяют также как **интерфейс программирования приложений (Application Programming Interface**, **API)** между приложением и сетью, так как сокет — это программный интерфейс, с помощью которого строятся сетевые приложения. Разработчик приложения управляет всем, что находится со стороны прикладного уровня сокета, но со стороны транспортного уровня он способен только контролировать, во-первых, выбор транспортного протокола и, во-вторых, возможность фиксировать некоторые параметры транспортного уровня, такие как максимальный буфер и максимальный размер сегмента (это рассматривается в главе 3). Когда разработчик приложения выбирает транспортный протокол (если такой выбор доступен), то приложение создается с использованием служб транспортного уровня, предоставляемых этим протоколом. Подробнее мы изучим сокеты в разделе 2.7.

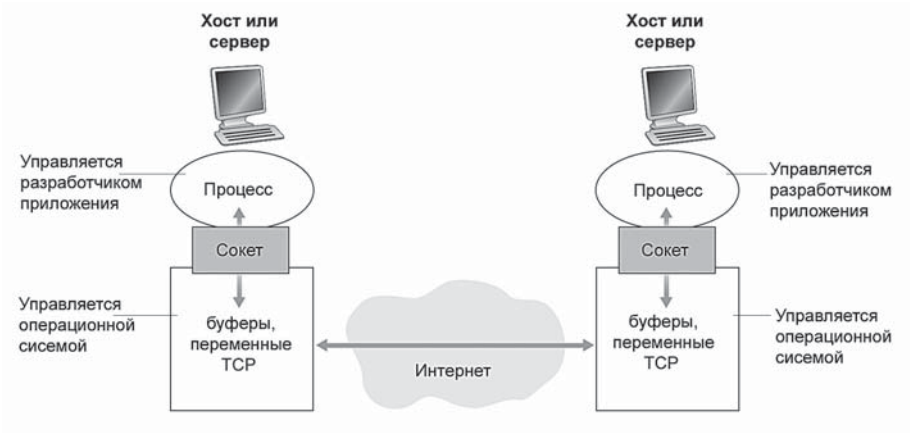


Рис. 2.3. Сокеты — интерфейс между процессами приложений и транспортным протоколом

Адресация процессов

Для того чтобы отправить письмо конкретному адресату, нужно иметь его адрес. То же самое можно сказать и о процессах. Когда запущенный на одном хосте процесс пытается отправить пакеты процессу, работающему на другом хосте, то необходимо знать адрес второго процесса. Для идентификации процесса-получателя необходима информа-

ция о двух вещах: во-первых, об адресе хоста назначения и, во-вторых, об идентификаторе, который определяет нужный нам процесс на хосте назначения.

В Интернете хост можно идентифицировать по его **IP-адресу**. Подробнее мы поговорим об адресах в главе 4, а пока все, что нам нужно знать — это то, что IP-адрес представляет собой 32-разрядное число, которое однозначно определяет хост. В дополнение к адресу хоста отправляющий процесс должен идентифицировать процесс принимающий (а точнее, принимающий сокет), запущенный на хосте-получателе. Такая информация необходима, потому что в общем случае на хосте могут быть запущены многие сетевые приложения. **Номер порта** назначения решает эту задачу. Наиболее популярным сетевым приложениям назначены определенные номера портов. Например, веб-сервер идентифицируется портом 80. Процесс почтового сервера, использующий протокол SMTP, использует порт 25. Список номеров широко известных портов для всех стандартных протоколов Интернета можно посмотреть на сайте **www.iana.org**. Подробно мы изучим номера портов в главе 3.

2.1.3. Транспортные службы, доступные приложениям

Вспомним, что сокет является интерфейсом между процессом приложения и протоколом транспортного уровня. Приложение на передающей стороне проталкивает сообщения через сокет. С другой стороны сокета протокол транспортного уровня отвечает за доставку сообщений к сокету принимающего процесса.

Во многих сетях, включая Интернет, используется более чем один транспортный протокол. Когда вы разрабатываете приложение, вам нужно выбрать один из доступных протоколов транспортного уровня. Как сделать этот выбор? Вероятнее всего, вы изучите службы, которые предлагают доступные протоколы, и затем выберете протокол с теми службами, которые наиболее подходят для нужд вашего приложения. Ситуация похожа на ту, когда вы, собираясь в путешествие, выбираете вид транспорта — самолет или поезд. Вам нужно выбрать один из двух, причем каждый из них предлагает различные виды доставки (например, поезд может предложить промежуточные остановки, в то время как самолет — более короткое время путешествия).

Какие службы протокол транспортного уровня предоставляет использующим его приложениям? Мы можем классифицировать возмож-

ные службы по четырем критериям: надежная передача данных, пропускная способность, время доставки и безопасность.

Надежная передача данных

Как обсуждалось в главе 1, в компьютерных сетях происходит потеря пакетов. Например, пакет может переполнить буфер маршрутизатора или быть отброшен хостом или маршрутизатором после получения поврежденных битов. Потеря данных во многих приложениях — таких как электронная почта, передача файлов, удаленный доступ, финансовые программы — может иметь довольно значительные последствия (в последнем случае — как для банка, так и для клиента!). Таким образом, чтобы решить эту проблему, нужна какая-то гарантия, что отправленные данные с одного конца приложения полностью и без ошибок будут доставлены на другой конец. Если протокол обеспечивает гарантированную службу доставки, то говорят, что обеспечивается **надежная передача данных**. Одна из важнейших служб, которые транспортный протокол может предоставить приложению — это как раз надежная передача данных от процесса к процессу. Когда транспортный протокол предоставляет такую службу, то передающий процесс может просто отправить данные в сокет и быть уверенным, что они прибудут в принимающий процесс без ошибок.

В случае, когда транспортный протокол не может обеспечить надежную передачу, то часть данных, отправленных передающим процессом, могут никогда не достигнуть процесса-получателя. Такое вполне может быть приемлемо для ряда **приложений, устойчивых к потерям** данных, например, некоторых мультимедиа-приложений, где потеря данных не является критической, а может привести всего-навсего к незначительным помехам.

Пропускная способность

В первой главе мы познакомились с понятием доступной пропускной способности, которая в контексте сеанса взаимодействия между двумя процессами в сети является ничем иным, как скоростью, с которой передающий процесс может доставлять биты процессу принимающему. Из-за того, что доступную полосу пропускания будут разделять и другие сеансы, а также по причине того, что количество этих сеансов будет непостоянно, значение доступной пропускной способности может изменяться во времени. Учитывая это, можно прийти к естествен-

ному выводу, что транспортный протокол должен также предоставлять еще одну службу, а именно гарантированную доступную пропускную способность, то есть доставку данных с определенной минимальной скоростью. Используя такую службу, приложение может запрашивать гарантированную пропускную способность *r* бит/с, и транспортный протокол должен будет заботиться о том, чтобы доступная скорость передачи данных была не меньше *r* бит/с. Такая гарантированная пропускная способность необходима многим приложениям. Например, если приложение IP-телефонии кодирует голосовые сообщения со скоростью 32 Кбит/с, то необходимо, чтобы данные отправлялись в сеть и доставлялись принимающему приложению с этой скоростью. Если транспортный протокол не может предоставить такую пропускную способность, приложению нужно будет осуществлять кодирование на более низкой скорости (и получать достаточную пропускную способность для поддержки этой низкой скорости кодировки), либо приложение завершит свою работу, так как не сможет использовать, например, имеющуюся в его распоряжении половину необходимой пропускной способности. Приложения, которым требуется определенная пропускная способность, называются **чувствительными к скорости передачи данных**. Такими являются многие мультимедиа-приложения, хотя в некоторые из них уже встроены инструменты для адаптации кодировки голоса или видео на скорости, соответствующей доступной пропускной способности.

В отличие от чувствительных к скорости передачи приложений, **эластичные приложения** используют доступную пропускную способность. Эластичными являются: электронная почта, передача файлов, веб-приложения. В любом случае, чем шире полоса пропускания канала, тем лучше. Ведь как говорится «ширины канала много не бывает».

Время доставки

Протокол транспортного уровня может также обеспечивать гарантии относительно времени доставки сообщений. Временные гарантии тоже предоставляются в различной форме. Например, протокол может гарантировать, что каждый бит, отправленный передающей стороной в сокет, приходит на сокет получателя не более чем через 100 мс. Служба такого рода будет полезна интерактивным приложениям реального времени, таким как IP-телефония, виртуальное окружение, телеконференции, а также многопользовательские игры. Все эти виды приложе-

ний для эффективной своей работы требуют ограничений по времени доставки. (См. главу 7 и публикации[181, 409]). Из-за продолжительных задержек в IP-телефонии, например, могут возникать паузы в разговоре; в многопользовательской игре или в виртуальном интерактивном окружении долгая задержка между действием и ответом на него (например, от другого игрока на другом конце соединения) может привести к потере реалистичности. Для приложений, не являющихся приложениями реального времени, низкие задержки, конечно, тоже предпочтительны, но жестких ограничений по времени нет.

Безопасность

Наконец, транспортный протокол может предоставлять приложениям одну или несколько служб, относящихся к безопасности. Например, транспортный протокол на передающем хосте способен шифровать все данные, отправленные процессом-источником, а затем на принимающем хосте расшифровывать их перед доставкой процессу-получателю. Это обеспечит конфиденциальность между двумя процессами. В дополнение к конфиденциальности транспортный протокол может предлагать службы, обеспечивающие целостность данных, а также конечную аутентификацию, о чем мы подробнее поговорим в главе 8.

2.1.4. Транспортные службы, предоставляемые Интернетом

До этого момента мы рассматривали транспортные службы, которые *могли* бы обеспечить компьютерные сети вообще. Теперь давайте обратимся к основным типам транспортных служб, предоставляемых Интернетом. Интернет (и TCP/IP сети вообще) предоставляет приложениям два транспортных протокола — UDP и TCP. Каждый из этих протоколов предлагает разнообразный набор служб исполняющимся приложениям. На рис. 2.4 представлены требования к этим службам для некоторых выбранных приложений.

Службы протокола TCP

Модель обслуживания протокола TCP включает службу установления логического соединения, а также службу надежной передачи данных. Когда приложение использует протокол TCP, то обе эти службы предоставляются приложению.

Приложение	Потери данных	Пропускная способность	Чувствительность к потере данных
Передача файлов/ загрузка	Не допускаются	Эластичное	Нет
Электронная почта	Не допускаются	Эластичное	Нет
Веб-документы	Не допускаются	Эластичное (несколько Кбит/с)	Нет
IP-телефония/ Видео-конференции	Допускаются	Аудио: От нескольких Кбит/с до 1 Мбит/с Видео: От 10 Кбит/с до 5 Мбит/с	Да, сотни миллисекунд
Потоковый аудио и видеоконтент	Допускаются	См. предыдущее	Да, несколько секунд
Интерактивные игры	Допускаются	От нескольких до 10 Кбит/с	Да, сотни миллисекунд
Мгновенные сообщения	Не допускаются	Эластичное	Да и нет

Рис. 2.4. Требования к службам, предъявляемые определенными сетевыми приложениями

- *Передача с установлением соединения.* Перед тем как начинают передаваться сообщения прикладного уровня, протокол TCP обеспечивает обмен управляющей информацией между клиентом и сервером на транспортном уровне. Эта так называемая процедура рукопожатия предупреждает клиентскую и серверную сторону, позволяя им подготовиться к началу обмена пакетами. После этапа рукопожатия говорят, что между сокетами клиентского и серверного процессов установлено **TCP-соединение**. Соединение является дуплексным — это означает, что оба процесса могут передавать сообщения друг другу в одно и то же время. Когда приложение завершит передачу сообщения, соединение должно быть разорвано. Передачу с установлением соединения и ее реализацию мы подробно рассмотрим в главе 3.

О БЕЗОПАСНОСТИ

Безопасность протокола TCP

Ни один из протоколов TCP и UDP не предоставляет службу шифрования — данные, которые передающий процесс отправляет в свой сокет, идентичны данным, которые проходят через сеть к принимающему процессу. Так, например, если процесс-источник отправляет пароль открытым текстом (то есть незашифрованным) в свой сокет,

то этот открытый текстовый пароль будет проходить все линии связи на пути между отправителем и получателем, в любой из них являясь потенциальной мишенью для перехвата. С некоторых пор соображения конфиденциальности и безопасности информации при передаче по сетям стали более актуальными, и Интернет-сообщество разработало криптографический протокол **уровень защищенных сокетов** (Secure Sockets Layer, **SSL**) как доработку протокола TCP. Такой вариант TCP, расширенный протоколом SSL, предлагает не только традиционные службы TCP, но также обеспечивает безопасность передачи данных от процесса к процессу, включая шифрование, контроль их целостности, а также конечную аутентификацию. Подчеркнем, что SSL — это не третий транспортный протокол в добавление к TCP и UDP, а лишь дополнение к TCP, с доработками, реализованными на прикладном уровне. В частности, если приложению необходимо использовать службы протокола SSL, то нужно включить поддержку SSL (задействовать оптимизированные библиотеки и классы) в приложении, как на клиентской, так и на серверной стороне. Протокол SSL имеет свой собственный API сокета, похожий на API сокета протокола TCP. При использовании протокола SSL передающий процесс отправляет обычные текстовые данные в сокет; затем SSL на передающем хосте шифрует данные и передает их в сокет TCP. Зашифрованные данные отправляются по сети в сокет TCP к принимающему процессу. Сокет получателя передает зашифрованные данные в SSL, затем расшифровывает их. Наконец, SSL передает обычные текстовые данные через свой сокет в принимающий процесс. Детально мы изучим протокол SSL в главе 8.

- *Надежная передача данных*. Взаимодействующие процессы могут полагаться на протокол TCP, который доставит все отправленные данные без ошибок и в строго определенном порядке. Когда одна сторона приложения отправляет поток байтов в сокет, она может быть уверена, что TCP доставит этот самый поток в принимающий сокет без потерь или дублирования.

Протокол TCP кроме всего прочего предоставляет механизм контроля перегрузки, который, правда, выгоден больше не взаимодействующим процессам, а самой коммуникационной сети Интернет. Когда участок сети между отправителем и получателем перегружен, данный механизм заставляет передаваемый процесс (либо клиентский, либо серверный) снижать нагрузку на сеть. В главе 3 мы с вами увидим, что контроль перегрузки в TCP пытается наложить ограничения на каждое TCP-соединение, чтобы распределить пропускную способность сети более равномерно.

Службы протокола UDP

UDP представляет собой простой протокол транспортного уровня, предлагающий минимальный набор служб. UDP является протоколом без установления соединения, то есть процедуры рукопожатия перед тем, как два процесса начинают взаимодействовать, не происходит. UDP обеспечивает ненадежную передачу данных — другими словами, когда процесс отправляет сообщение в сокет UDP, *нет* никакой гарантии, что сообщение будет получено принимающим процессом. Более того, сообщения могут поступать в принимающий процесс в произвольном порядке.

Механизма контроля перегрузок в протоколе UDP тоже нет. Поэтому отправляющая сторона может передавать данные в нижележащий уровень (сетевой) с любой скоростью. (Заметим, что реальная сквозная пропускная способность может быть меньше этой скорости по причине ограничения скорости передачи либо перегрузки линии связи.)

Службы, не предоставляемые транспортными протоколами Интернета

Мы с вами отметили четыре критерия для служб транспортного протокола: надежная передача данных, пропускная способность, время доставки и безопасность. Что из этого могут предложить протоколы TCP и UDP? Мы уже отметили, что TCP гарантирует надежную доставку данных. Мы также уже знаем, что добавление протокола SSL позволяет обеспечить безопасность данных при отправке через TCP. Но в нашем кратком описании протоколов TCP и UDP отсутствуют упоминания о гарантированном времени доставки и о пропускной способности — транспортные протоколы сегодняшнего Интернета это обеспечить *не* могут. Следует ли из этого, что приложения, чувствительные к скорости передачи, такие как IP-телефония, не могут работать в Интернете? Конечно же, нет — такие приложения уже многие годы с успехом используют Интернет, поскольку были они разработаны таким образом, чтобы максимально учесть отсутствие гарантий временной доставки данных. Подробнее мы изучим некоторые особенности построения этих приложений в главе 7. Тем не менее возможности «умного» дизайна приложений ограничены, например, когда задержки сети становятся очень большими либо пропускная способность каналов связи достаточно мала. Одним словом, можно сделать вывод, что сегодняшний Интернет, как правило, обеспечивает удовлетворительное обслуживание приложе-

ний, чувствительных ко времени доставки, но не может гарантировать их нормальную работу в исключительных условиях.

Приложение	Протокол прикладного уровня	Базовый транспортный протокол
Электронная почта	SMTP[554]	TCP
Удаленный терминальный доступ	Telnet[425]	TCP
Всемирная паутина	HTTP[483]	TCP
Передача файлов	FTP[427]	TCP
Потоковый мультимедийный контент	HTTP (например, YouTube)	TCP
IP-телефония	SIP[502], RTP[518] или проприетарное (например, Skype)	UDP или TCP

Рис. 2.5. Популярные Интернет-приложения и используемые ими протоколы прикладного и транспортного уровней

На рис. 2.5 показаны транспортные протоколы, используемые некоторыми популярными Интернет-приложениями. Мы видим, что электронная почта, удаленный терминальный доступ, Всемирная паутина и приложения передачи файлов — все они используют протокол TCP. Это связано с тем, что он обеспечивает надежную передачу данных, гарантируя, что все данные в конечном итоге доберутся до места назначения. Поскольку приложения IP-телефонии (например, Skype) допускают некоторые потери данных, но требуют для эффективной своей работы определенное значение минимальной скорости передачи, разработчики этих приложений обычно предпочитают запускать их через протокол UDP, избегая тем самым механизма управления перегрузкой и дополнительных затрат на гарантированную доставку пакетов, присущих протоколу TCP. Но так как многие брандмауэры (большинство их типов) настроены на блокирование UDP-трафика, то приложения IP-телефонии очень часто разрабатываются с возможностью использования протокола TCP в качестве резервного для случаев, когда соединиться по протоколу UDP не удается.

2.1.5. Протоколы прикладного уровня

Мы только что узнали, что сетевые процессы взаимодействуют друг с другом, отправляя сообщения в сокеты. Но как построены эти сообщения? Что означают различные поля в сообщениях? Когда именно про-

цессы отправляют эти сообщения? Все эти вопросы ведут нас в область протоколов прикладного уровня, которые определяют как прикладные процессы, запущенные на различных конечных системах, передают друг другу сообщения. В частности, протокол прикладного уровня определяет:

- Типы сообщений, которыми обмениваются процессы, например сообщения запроса или сообщения ответа
- Синтаксис различных типов сообщений, то есть состав полей сообщения и их порядок
- Семантика полей, то есть значение информации, содержащейся в каждом из них
- Набор правил для определения времени и порядка передачи сообщений и получения ответов

Некоторые из протоколов прикладного уровня описаны в документах RFC, поэтому являются общедоступными. Например, вы можете найти в RFC описание протокола прикладного уровня HTTP (HyperText Transfer Protocol, протокол передачи гипертекста[483]). Если разработчик, браузера, например, следует правилам документов RFC относительно протокола HTTP, то, соответственно, браузер способен загружать веб-страницы с любого веб-сервера, который также соответствует правилам RFC для HTTP. Многие протоколы являются проприетарными (частными) и поэтому недоступны для общего пользования. Например Skype использует проприетарный протокол прикладного уровня.

Очень важно понимать различие между сетевыми приложениями и протоколами прикладного уровня. Протокол прикладного уровня — это только одна из частей сетевого приложения (хотя и очень важная часть!). Рассмотрим пару примеров. Всемирная паутина — это клиент-серверное приложение, позволяющее пользователям получать документы с веб-сервера по запросу. Веб-приложение состоит из множества компонентов, включающих стандарты для форматов документа (то есть, HTML), веб-браузеров (например, Firefox или Microsoft Internet Explorer), веб-серверов (например, Apache и Microsoft-серверов) и протокола прикладного уровня. Протокол прикладного уровня для Всемирной паутины — HTTP — определяет формат и последовательность сообщений, которыми обмениваются браузер и веб-сервер. Таким образом, HTTP — это только одна часть (хотя и важная) веб-приложения. То же самое можно сказать про приложение электронной почты, которое также включает множество компонентов: почтовые серверы, со-

держащие почтовые ящики пользователей; почтовые клиенты (такие, как Microsoft Outlook), которые позволяют пользователям читать и создавать сообщения; набор стандартов для определения структуры сообщения электронной почты; наконец, протоколы прикладного уровня, определяющие порядок прохождения сообщений между серверами и почтовыми клиентами и порядок интерпретации для содержимого заголовков этих сообщений. Основной протокол прикладного уровня для электронной почты — это SMTP (Simple Mail Transfer Protocol)[554]. Таким образом, SMTP, являющийся основным протоколом электронной почты, также является хотя и важным, но одним из приложений электронной почты.

2.1.6. Сетевые приложения, рассматриваемые в данной книге

Интернет-приложения, как проприетарные, так и свободного пользования, разрабатываются ежедневно. Мы здесь не будем рассматривать все их разнообразие, а сфокусируемся на самых важных, на наш взгляд. В этой главе мы обсудим пять приложений: Всемирную паутину, передачу файлов, электронную почту, службу каталогов и одноранговые (пиринговые или P2P) приложения. В первую очередь рассмотрим Всемирную паутину, причем не только из-за того, что это наиболее популярный вид взаимодействия на сегодняшний день, но и потому, что используемый ею протокол HTTP достаточно прост для изучения. После рассмотрения Всемирной паутины вкратце изучим FTP, являющийся в некотором смысле противоположностью HTTP. Затем перейдем к рассмотрению электронной почты — ключевого приложения Интернета. Электронная почта сложнее, чем Всемирная паутина, в том смысле, что она предполагает использование не одного, а нескольких протоколов прикладного уровня. Затем рассмотрим систему DNS, предоставляющую службы каталогов для Интернета. Большинство пользователей взаимодействуют со службой DNS не напрямую, а через свои приложения (включая Всемирную паутину, передачу файлов и электронную почту). DNS прекрасно демонстрирует, как часть функциональности сетевого ядра (трансляция сетевого имени в сетевой адрес) может быть реализована на прикладном уровне в Интернете. Наконец, в завершение главы мы изучим несколько одноранговых приложений, сосредоточившись на приложениях общего файлового доступа и службах распределенного поиска. В главе 7 мы рассмотрим мультимедиа-приложения, включая потоковое видео и передачу голоса по IP-сетям.

2.2. Всемирная паутина и HTTP

До начала 90-х годов Интернет преимущественно использовался учеными, исследователями и студентами университетов для обеспечения удаленного доступа, передачи файлов с локальных хостов на удаленные и обратно, получения и отправки новостей, а также для работы с электронной почтой. Несмотря на то, что эти приложения были весьма полезны (какими и остаются по сей день), об Интернете мало кто знал за рамками исследовательских и ученых сообществ. В начале 1990-х на сцене появилось главное приложение Интернета — это Всемирная паутина (веб)[49]. Всемирная паутина стала первым Интернет-приложением, на которое обратила внимание общественность. Оно серьезно повлияло и продолжает влиять на то, как люди взаимодействуют между собой внутри своего рабочего окружения и за его пределами, и позволило поднять Интернет на новый уровень — в результате он стал, в сущности, единой крупной сетью данных.

Одна из возможных причин, почему Всемирная паутина привлекла пользователей — это то, что принцип ее функционирования — работа *по запросу*. Пользователи получают то, что они хотят и когда им это требуется, в отличие от традиционных радио- и телевещания, которые предоставляют информацию тогда, когда поставщик делает ее доступной для аудитории. Вдобавок к этому Всемирная паутина привлекательна для пользователей многими другими своими свойствами. На сегодняшний день сделать информацию доступной по сети — невероятно легкая вещь, доступная каждому: каждый пользователь может стать издателем, не применяя особых усилий. Гиперссылки и поисковые системы помогают нам не потеряться в океане веб-сайтов. Веб-графика, формы, Java-апплеты и другие «навороты» позволяют нам взаимодействовать с веб-страницами и сайтами, а веб-серверы используются в качестве платформы для построения многих популярнейших приложений, включая YouTube, Gmail и Facebook.

2.2.1. Обзор протокола HTTP

HTTP (протокол передачи гипертекста) — это протокол прикладного уровня для Всемирной паутины, составляющий ее самое сердце. Он определен в документах RFC 1945[460] и RFC 2616[483]. HTTP реализуется в двух частях приложений: клиентской и серверной. Клиентская и серверная части программ, исполняемые на различных конечных системах,

общаются друг с другом, обмениваясь сообщениями HTTP. Протокол HTTP определяет структуру этих сообщений и порядок обмена между клиентом и сервером. Перед тем как детально объяснить HTTP, сделаем краткий обзор терминологии Всемирной паутины.

Веб-страница (также называемая документом) состоит из объектов. **Объект** — это простой файл, имеющий уникальный URL-адрес, например файл формата HTML, изображение в формате JPEG, Java-апплет или видеоклип. Большинство веб-страниц состоит из базового HTML-файла, который содержит ссылки на несколько объектов. Например, если веб-страница содержит HTML-текст и пять рисунков в формате JPEG, это значит, что веб-страница содержит шесть объектов: базовый файл плюс пять изображений. Базовый файл содержит в себе ссылки на другие объекты, в виде их URL-адресов, находящихся в тексте базового файла. Каждый URL-адрес состоит из двух частей: имени сервера, содержащего объект и пути до этого объекта. Например, URL-адрес

`http://www.someSchool.edu/someDepartment/picture.gif`

содержит адрес хоста **`www.someSchool.edu`** и имя пути **`/someDepartment/picture.gif`**. Мы будем использовать слова *браузер* и *клиент* как взаимозаменяемые, так как веб-браузеры (такие как Internet Explorer или Firefox) реализуют клиентскую сторону протокола HTTP. Веб-серверы, реализующие серверную сторону HTTP, содержат веб-объекты, на которые указывают ссылки. Популярными сегодня веб-серверами являются Apache и Microsoft Internet Information Server.

Протокол HTTP определяет порядок того, как веб-клиенты запрашивают веб-страницы с веб-сервера и как сервер передает эти страницы клиентам. Подробнее взаимодействие между клиентом и сервером мы рассмотрим позже, но основная идея продемонстрирована на рис. 2.6. Когда пользователь запрашивает веб-страницу (например, щелкает по гиперссылке), браузер отправляет HTTP-запрос объектов этой страницы серверу. Сервер получает запрос и отвечает сообщением HTTP, которое содержит объекты.

HTTP использует TCP в качестве базового транспортного протокола. Сначала HTTP-клиент инициирует TCP-соединение с сервером. Когда соединение установлено, процессы браузера и сервера получают доступ к TCP через свои сокеты. На клиентской стороне, как мы уже знаем, сокет — это дверь между клиентским процессом и TCP-соединением; со стороны сервера — между серверным процессом и TCP-соединением. Клиент отправляет HTTP-запрос, получая HTTP-ответ через свой со-

кет. Аналогично HTTP-сервер принимает запрос и отсылает ответное сообщение, используя свой сокет. Как только клиент отправил сообщение в сокет, оно попадает в руки протокола TCP. Из раздела 2.1 нам известно, что TCP обеспечивает службу надежной доставки. Это означает, что каждый HTTP-запрос, посланный клиентским процессом, обязательно прибудет к серверу, и наоборот, каждый HTTP-ответ, посланный серверным процессом, обязательно будет получен клиентом. Тут мы видим основное преимущество иерархической (многоуровневой) архитектуры — протоколу HTTP не нужно заботиться о потерянных данных или о том, как восстанавливаются эти данные, переданные по сети. Это все работа протокола TCP и протоколов нижележащих уровней стека.

Рис. 2.6. Процедура запросов и ответов HTTP

Отметим важный момент, что сервер отправляет запрошенные файлы клиенту без сохранения какой-либо информации о нем. Если конкретный клиент запрашивает один и тот же объект дважды за определенный промежуток времени, сервер не сообщает, что такой объект только что был запрошен. Вместо этого он пересылает объект, как будто бы он уже забыл, что делал это только что. Так как HTTP-сервер не обрабатывает информацию о клиенте, говорят, что HTTP — это **протокол без сохранения состояния**. Отметим также, что Всемирная паутина использует клиент-серверную архитектуру, описанную в разделе 2.1. Веб-сервер всегда находится в режиме онлайн, имеет фиксированный IP-адрес и обслуживает запросы от потенциальных миллионов различных браузеров.

2.2.2. Непостоянные и постоянные соединения

В большинстве Интернет-приложений клиент и сервер взаимодействуют друг с другом продолжительный период времени, в течение которого клиент делает серию запросов, а сервер отвечает на каждый из них. Запросы в серии, в зависимости от типа приложения и от того, как оно используется, могут происходить один за другим или с некоторыми интервалами, как периодическими, так и произвольными. Когда взаимодействие клиента с сервером происходит через TCP, разработчик приложения должен принять важное решение — отправлять каждую пару запрос-ответ через *отдельное* либо через *одно и то же* TCP-соединение? В первом случае говорят, что используются **непостоянные соединения**; во втором — **постоянные соединения**. Чтобы это лучше понять, давайте рассмотрим преимущества и недостатки постоянных соединений в контексте определенного приложения, а именно с применением протокола HTTP, который способен работать как с разными типами соединения. Несмотря на то, что HTTP использует по умолчанию постоянное соединение, HTTP-клиенты и серверы могут быть сконфигурированы таким образом, чтобы переключить свою работу в режим непостоянных соединений.

Непостоянные HTTP-соединения

Давайте рассмотрим этапы передачи веб-страницы от сервера к клиенту в случае непостоянного соединения. Предположим, что страница состоит из базового HTML-файла и десяти изображений в формате JPEG и все эти 11 объектов находятся на одном сервере. Предположим, что URL-адрес для базового HTML-файла имеет вид

`http://www.someSchool.edu/someDepartment/home.index`

Вот что происходит:

1. HTTP-клиент инициирует TCP-соединение с сервером **`www.someSchool.edu`** по порту 80, являющемуся портом по умолчанию для протокола HTTP. Этому TCP-соединению выделяются сокеты на клиентской и северной стороне.
2. HTTP-клиент отправляет запрос серверу через свой сокет. Запрос включает путь к базовому файлу **`/someDepartment/home.index`**. (HTTP-сообщения более детально рассмотрим ниже)
3. Процесс HTTP-сервера получает запрос через свой сокет, извлекает объект **`/someDepartment/home.index`** из своего места хране-

ния (оперативной памяти или диска), помещает объект в ответное HTTP-сообщение и отправляет клиенту через свой сокет.

4. Процесс HTTP-сервера дает команду протоколу TCP закрыть соединение (на самом деле, TCP-соединение не разрывается до тех пор, пока сервер не получит информацию об успешном получении ответа клиентом).

5. HTTP-клиент получает ответ от сервера, и TCP-соединение разрывается. Сообщение указывает, что полученный объект — это HTML-файл. Клиент извлекает файл из сообщения, обрабатывает его и находит ссылки на 10 объектов (файлов в формате JPEG).

6. Шаги с первого по четвертый повторяются для каждого из десяти JPEG-объектов.

Когда браузер получает веб-страницу, он отображает ее на экране. Два различных браузера, кстати, могут интерпретировать веб-страницу по-разному. Спецификации протокола HTTP (RFC 1945[460] и RFC 2616[483]) определяют только протокол взаимодействия между программой клиента и программой сервера, но ничего не говорят о том, как веб-страница должна интерпретироваться клиентом.

Описанные выше шаги демонстрируют использование непостоянных соединений HTTP, при которых TCP-соединение закрывается после того, как сервер получает объект. Таким образом, каждое TCP-соединение передает ровно одно сообщение-запрос и одно сообщение-ответ. В нашем примере, когда пользователь запрашивает веб-страницу, устанавливаются 11 TCP-соединений.

Здесь мы намеренно не уточняем, получил ли клиент эти 10 JPEG-файлов через 10 последовательно создаваемых TCP-соединений, либо некоторые приняты через параллельное TCP-соединение. На самом деле большинство современных браузеров в режиме по умолчанию открывают от пяти до десяти параллельных TCP-соединений, и каждое из них обрабатывает одну транзакцию из запроса и ответа, а степень этого параллелизма может быть сконфигурирована пользователем. Если тот пожелает, число параллельных соединений можно установить равным единице, и в этом случае 10 соединений будут устанавливаться последовательно. Как мы увидим в следующей главе, использование параллельных соединений сокращает время ответа сервера.

Перед тем как продолжить, давайте произведем некоторые вычисления для оценки интервала времени, проходящего с отправки клиентом

запроса базового HTML-файла до момента, когда весь файл получен клиентом. Для этого определим **время оборота (round-trip time, RTT)**, которое есть не что иное, как время, требуемое пакету малого размера для передачи от клиента к серверу и обратно (еще его называют **временем двусторонней задержки**). Время оборота включает в себя задержку распространения, задержку ожидания в промежуточных маршрутизаторах или коммутаторах и задержку на обработку пакета. (Их мы обсудили с вами в разделе 1.4.) Рассмотрим, что происходит, когда пользователь нажимает на гиперссылку. Как показано на рис. 2.7, это приводит к тому, что браузер инициирует TCP-соединение с веб-сервером; оно включает «тройное рукопожатие» — клиент отправляет серверу небольшой TCP-сегмент, сервер подтверждает и отправляет в ответ также небольшой TCP-сегмент, и, наконец, клиент еще раз отправляет сегмент с подтверждением серверу.

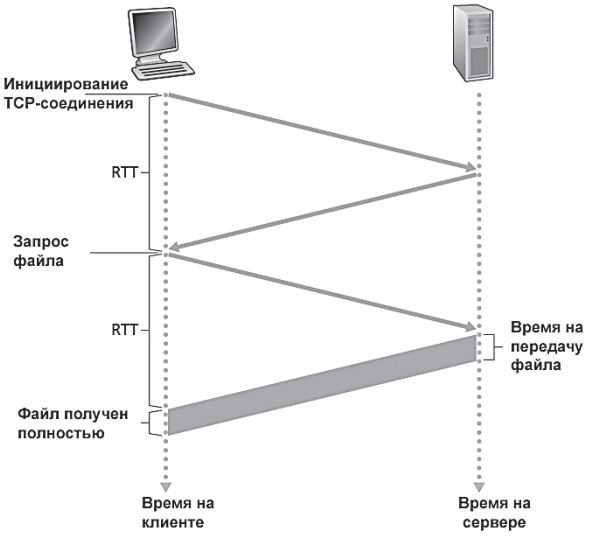


Рис. 2.7. Расчет времени, необходимого на запрос и получение HTML-файла

Первые две части тройного рукопожатия занимают одно время оборота или 1 RTT. После этого вместе с третьей частью клиент отправляет HTTP-запрос в TCP-соединение. Как только запрос прибывает на сервер, тот отправляет в ответ запрошенный HTML-файл. Данная пара запрос — ответ занимает еще одно время оборота (1 RTT). Таким образом, общее время ответа приблизительно равно удвоенному времени оборота плюс время на передачу HTML-файла.

Постоянное HTTP-соединение

Непостоянные соединения имеют некоторые недостатки. Во-первых, для *каждого запрашиваемого объекта* должно устанавливаться и обслуживаться новое соединение.

Для каждого из этих соединений протокол TCP должен выделить буфер, а также сохранить несколько переменных как на клиенте, так и на сервере. Это бывает затруднительно для веб-сервера, учитывая, что он может обслуживать сотни различных клиентов одновременно. Во-вторых, как мы уже сказали, передача каждого объекта вызывает задержку, равную двум значениям времени оборота: 1 RTT для установления TCP-соединения и еще 1 RTT — для отправки запроса и получения ответа, а это все дополнительное время задержки.

В случае с постоянным соединением сервер после отправки ответа клиенту оставляет TCP-соединение открытым. Через одно и то же соединение можно отправить последовательность запросов и ответов между одним и тем же клиентом и сервером. В частности, одно постоянное TCP-соединение позволяет передать всю веб-страницу (в примере выше это базовый HTML-файл и десять изображений).

Более того, через одно постоянное соединение можно отправить одному и тому же клиенту много веб-страниц, размещенных на том же сервере. Эти запросы объектов могут быть сделаны один за другим, без ожидания ответов на обрабатываемый запрос (так называемая *конвейеризация*). Обычно HTTP-сервер закрывает соединение, когда оно не используется в течение определенного времени (настраиваемый интервал тайм-аута). Когда сервер получает последовательные запросы, он отправляет объекты также один за другим. По умолчанию HTTP использует постоянное соединение с конвейеризацией. Мы численно сравним производительность постоянного и непостоянного соединений в упражнениях глав 2 и 3. Также можете посмотреть соответствующие публикации[203, 368].

2.2.3. Формат HTTP-сообщения

В документах по спецификации протокола HTTP (RFC 1945[460] и RFC 2616[483]) содержится описание форматов HTTP-сообщений. В протоколе HTTP существуют два типа сообщений: сообщение-запрос и сообщение-ответ. Их мы обсудим ниже.

Сообщение-запрос протокола HTTP

Ниже представлено типичное сообщение-запрос протокола HTTP:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

Если внимательно посмотреть на это сообщение, можно многое из него почерпнуть. Во-первых, мы видим, что оно представлено в обычном текстовом формате ASCII, так что любой человек может его прочитать. Во-вторых, мы видим, что сообщение состоит из пяти строк, каждая из которых заканчивается символами возврата каретки и перевода строки, а последняя — двумя парами этих символов. Сообщение-запрос в общем случае может содержать количество строк от одной и более, но необязательно пять, как в нашем примере. Первая строка HTTP-сообщения называется **строкой запроса**; следующие строки называются **строками заголовка**. В строке запроса содержатся три поля: поле метода, поле URL и поле версии протокола HTTP. Поле метода может принимать несколько различных значений, включая `GET`, `POST`, `HEAD`, `PUT` и `DELETE`. Большинство сообщений-запросов протокола HTTP используют метод `GET`. Он применяется, когда браузер запрашивает объект, идентифицирующийся полем URL. В нашем случае браузер запрашивает объект `/somedir/page.html`. Поле версии протокола говорит само за себя; в нашем примере версия `HTTP 1.1`.

Теперь давайте посмотрим на строки заголовка в нашем примере. Строка заголовка `Host: www.someschool.edu` указывает адрес хоста, на котором размещается объект. Вы, возможно, подумаете, что данная строка не является необходимой, так как соединение с данным хостом по этому адресу уже установлено, но, как мы увидим в разделе 2.2.5, эта информация требуется для кэширующего прокси-сервера. Включение строки заголовка `Connection:close` означает: браузер сообщает серверу, что он не собирается работать с постоянным соединением, и его нужно разорвать после отправки запрашиваемого объекта. Строка `User-agent:` указывает на агент пользователя, то есть, на тип браузера, который совершает запрос к серверу. В нашем случае агент пользователя Mozilla/5.0 — это браузер Firefox. Данная информация очень полезна, так как сервер на самом деле может отсылать различным типам браузеров разные версии одного и того же объекта (причем

обе версии задаются одним URL-адресом). Наконец, строка заголовка `Accept-language: fr` указывает, что пользователь предпочитает получить версию объекта на французском языке, если, конечно, такая версия существует на сервере; в противном случае сервер будет отсылать версию по умолчанию. Заголовок `Accept-language:` является одним из многих заголовков согласования, доступных в протоколе HTTP.

После нашего примера давайте рассмотрим общий формат сообщения-запроса, представленный на рис. 2.8. Как мы видим, общий формат соответствует нашему примеру выше. Однако вы можете заметить, что после строк заголовка (при дополнительных символах возврата каретки и перевода строки) присутствует «тело сообщения». Данное тело сообщения является пустым при использовании метода `GET` и содержит информацию при использовании метода `POST`. Чаще всего метод `POST` применяется, когда пользователь заполняет формы, например, вводит слова поиска в поисковой системе. Используя сообщение `POST`, пользователь запрашивает веб-страницу сервера, но ее содержимое зависит от того, что пользователь ввел в поля формы. Если значением поля метода является `POST`, то тело сообщения содержит информацию, которую пользователь ввел в поля формы.

Рис. 2.8. Общий формат сообщения-запроса HTTP

Заметим, что запросы, генерируемые с помощью форм, необязательно используют метод `POST`. Очень часто в HTML-формах применяется и метод `GET`, а данные, введенные в поля формы, подставля-

ются в URL-адрес запрашиваемой страницы. Например, если форма использует метод GET и содержит два поля, в которые вводятся значения monkeys и bananas, то URL-адрес примет следующий вид: **www.somesite.com/animalsearch?monkeys&bananas**. В ваших ежедневных путешествиях во Всемирной паутине вы, очевидно, встречались с URL-адресами такого типа.

Метод HEAD аналогичен методу GET. Когда сервер получает запрос с помощью метода HEAD, он отправляет ответное HTTP-сообщение, но не пересылает в нем запрашиваемый объект. Разработчики приложений обычно используют этот метод для целей отладки. Метод PUT часто применяется совместно с инструментами веб-публикаций. Он позволяет пользователю загружать объект по указанному адресу на конкретный веб-сервер. Он также используется приложениями, которые требуют загрузки объектов на веб-серверы. Метод DELETE позволяет пользователю либо приложению удалять объект на веб-сервере.

Сообщение-ответ протокола HTTP

Ниже мы приводим типичное ответное сообщение, используемое протоколом HTTP. Это может быть, например, ответом на сообщение-запрос, которое мы изучили в предыдущем разделе.

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 09 Aug 2011 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html
(данные данные данные данные данные...)
```

Давайте внимательно посмотрим на это сообщение. Оно состоит из трех частей: первоначальной **строки состояния**, **шести строк заголовка** и **тела сообщения**. Тело сообщения — это ключевая его часть, она содержит сам запрашиваемый объект (представлен здесь строкой данные данные данные данные данные...). Строка состояния содержит три поля: поле версии протокола, код состояния и соответствующее сообщение состояния (фраза состояния). В нашем примере строка состояния указывает, что сервер использует версию HTTP/1.1 и что все OK (то есть сервер нашел запрашиваемый объект и начал его передачу).

Теперь посмотрим на строки заголовка. Сервер использует строку `Connection: close`, чтобы сообщить клиенту, что он собирается закрыть TCP-соединение после отправки сообщения. Строка `Date:` указывает время и дату создания ответа сервером. Обратите внимание, что это не время создания или изменения объекта, а время, когда сервер извлекает объект из своей файловой системы, вставляет его в ответное сообщение и отправляет клиенту. Строка заголовка `Server:` означает, что сообщение было генерировано веб-сервером `Apache`; она аналогична строке заголовка `User-agent:` сообщения-запроса. Время последнего изменения объекта указывает строка состояния `Last-Modified:`, которая, как мы вскоре узнаем более подробно, является наиболее критичной для кэширования объекта, как для локальных клиентов, так и в сетевых кэширующих серверах (также известных как прокси-серверы). Строка заголовка `Content-Length:` показывает число байт в пересылаемом объекте. Строка `Content-Type:` информирует о том, что объект в теле сообщения является текстом в формате HTML. (На тип объекта указывает не расширение файла, а именно строка заголовка `Content-Type:`.)

Теперь давайте посмотрим на показанный на рис. 2.9 общий формат сообщения-ответа, частным случаем которого является рассмотренный выше пример. Добавим несколько слов относительно кодов состояния и связанных с ними фраз, которые указывают на результат запроса. Наиболее используемые коды состояния включают:

- `200 OK`: запрос выполнен успешно, и информация возвращена в ответном сообщении.
- `301 Moved Permanently`: запрошенный объект перемещен; новый URL-адрес указывается в заголовке ответного сообщения, содержащем строку `Location:`. Программное обеспечение клиента будет автоматически перенаправляться на этот новый URL-адрес.
- `400 Bad Request`: это общепринятый код ошибки, указывающий, что запрос не может быть распознан сервером.
- `404 Not Found`: запрошенный документ на сервере не существует.
- `505 HTTP Version Not Supported`: запрошенная версия HTTP-протокола не поддерживается сервером.

Хотите посмотреть на реальное сообщение-ответ протокола HTTP? Очень рекомендуем, и это достаточно просто! Сначала нужно соединиться с помощью команды Telnet с вашим любимым веб-сервером, по-

том набрать однострочное сообщение с запросом какого-нибудь объекта, который размещен на этом сервере.

Рис. 2.9. Общий формат сообщения-ответа протокола HTTP

Например, наберите в командной строке следующее:

```
telnet cis.poly.edu 80
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

(Дважды нажмите клавишу **Enter** после того, как наберете последнюю строку.) В результате откроется TCP-соединение с хостом `cis.poly.edu` по порту 80 и ему отправится сообщение-запрос. Вы должны увидеть ответное сообщение, которое включает базовый файл формата HTML, расположенный на домашней странице профессора Росса. Если вы хотите посмотреть просто строки сообщения HTTP без получения самого объекта, замените `GET` на `HEAD`. Затем попробуйте заменить значение `/~ross/` на `/~banana/` и посмотрите, какое сообщение вы получите в результате.

В данном разделе мы обсудили с вами несколько строк заголовка, которые могут применяться внутри запросов и ответов протокола HTTP. На самом деле в спецификации протокола определено очень много строк заголовков, используемых веб-серверами, браузерами и кэширующими серверами. Мы рассмотрели только небольшое количество

из этой огромной массы. Еще несколько из них мы изучим, когда будем обсуждать кэширование в разделе 2.2.5. Очень полезное обсуждение протокола HTTP, включающее информацию о заголовках и кодах состояния дается в публикации Кришнамурты[297].

Каким же образом решается, какие строки заголовка включать браузеру в сообщение-запрос, а веб-серверу — в ответное сообщение? На самом деле, строки заголовка, которые будет генерировать браузер, зависят от нескольких факторов: от типа браузера и его версии (например, браузер версии HTTP/1.0 не будет генерировать ни одной строки версии 1.1), от пользовательской конфигурации (например, используемого языка), а также от версии объекта — кэшированной, но, возможно, устаревшей, или последней, актуальной версии. Поведение веб-серверов аналогично: существуют различные продукты, версии и конфигурации, многообразие которых оказывает влияние на то, какие строки заголовка будут включаться в сообщение-ответ сервера.

2.2.4. Взаимодействие пользователя и сервера: cookie-файлы

Мы упомянули выше, что HTTP-сервер не сохраняет состояние соединения. Это упрощает разработку серверного программного обеспечения и позволяет создавать высокопроизводительные веб-серверы, способные обрабатывать тысячи одновременных TCP-соединений. Однако очень часто возможность идентифицировать пользователей очень даже желательна либо по причине того, что серверу нужно ограничить пользовательские права доступа, либо чтобы предоставлять набор услуг в зависимости от идентификации пользователя. Для этих целей в протоколе HTTP используются объекты *cookie* (*«Куки»*). Механизм cookie, определенный в документе RFC 6265[562], позволяет веб-сайтам отслеживать состояние пользовательского соединения. В наши дни подавляющее большинство коммерческих веб-сайтов используют данный механизм.

Как мы видим на рис. 2.10, технология cookie включает четыре основных компонента: (1) строка заголовка в ответном HTTP-сообщении сервера; (2) строка заголовка в HTTP-запросе клиента; (3) cookie-файл, хранящийся на конечной системе пользователя и управляемый его браузером; (4) база данных на стороне веб-сервера. Давайте, используя рис. 2.10, посмотрим, как работает этот механизм. Предположим, Сьюзен, которая выходит в Интернет со своего домашнего компьюте-

ра, используя браузер Internet Explorer, в первый раз посещает сервер **Amazon.com**. Предположим также, что она уже посещала ранее сайт eBay. Когда запрос приходит на веб-сервер Amazon, он создает уникальный идентификационный номер, а также запись в своей базе данных, которая индексируется этим идентификационным номером. Затем веб-сервер Amazon посылает ответ браузеру Сьюзен, включающий в HTTP-сообщение заголовок `Set-cookie:`, и в нем содержится соответствующий идентификационный номер. Например, строка заголовка может иметь вид: `Set-cookie: 1678`

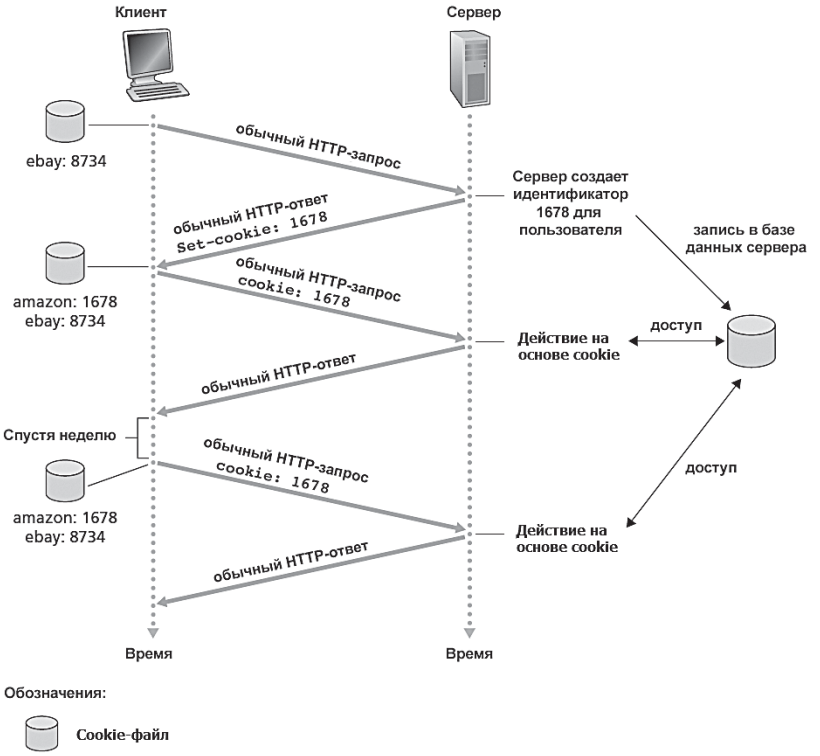


Рис. 2.10. Использование механизма cookie для сохранения состояния пользовательского сеанса

Когда браузер Сьюзен получает ответное HTTP-сообщение, он видит заголовок `Set-cookie:` и добавляет строку в специальный cookie-файл, которым управляет. Данная строка включает имя сервера и идентификационный номер из заголовка `Set-cookie:`. Заметим, что cookie-файл уже содержит запись для веб-сайта eBay, так как Сьюзен

в прошлом уже посещала данный сайт. По мере того, как Сьюзен продолжает работать с сайтом сервера Amazon, каждый раз, когда она запрашивает веб-страницу, ее браузер обращается к своему cookie-файлу, извлекает идентификационный номер этого сайта и помещает строку cookie-заголовка, включающую идентификационный номер, в HTTP-запрос. В каждом таком HTTP-запросе к серверу Amazon содержится строка заголовка:

```
Cookie: 1678
```

С помощью такого инструмента сервер способен отслеживать активность Сьюзен на сайте Amazon. Хотя веб-сайту и не нужно знать имя Сьюзен, ему точно известно, какие страницы посетил пользователь 1678, в каком порядке и сколько раз! Amazon использует механизм cookie, чтобы организовать, например, карту покупок — он хранит информацию о товарах, выбранных Сьюзен в течение пользовательского сеанса браузера, и она может оплатить их все вместе в самом конце сеанса.

Если Сьюзен возвратится на сайт Amazon, скажем, через неделю, то ее браузер будет продолжать добавлять строку заголовка `Cookie: 1678` в сообщение-запрос. Веб-сайт Amazon станет рекомендовать Сьюзен товары, основываясь на тех веб-страницах, которые она посещала в прошлом. Если вдобавок Сьюзен зарегистрируется — передав имя, адрес электронной почты, информацию кредитной карты — Amazon может включить эти сведения в свою базу данных, связав имя Сьюзен с ее идентификационным номером (и со всеми страницами, которые она посетила на сайте в прошлом!) Это дает возможность сайту Amazon и многим другим интернет-магазинам предлагать совершать покупки в один клик — когда Сьюзен выберет товар в следующий свой визит, ей не нужно будет заново вводить имя, номер карты или адрес.

Таким образом, мы с вами видим, что механизм cookie может быть использован, чтобы идентифицировать пользователя. При первом посещении сайта пользователь вводит некоторую информацию о своей идентификации (например, имя), а в последующих сеансах браузер передает cookie-заголовок на сервер, тем самым идентифицируя пользователя на сервере. Можно сказать, что механизм cookie создает как бы дополнительный сеансовый уровень поверх протокола HTTP, который не сохраняет информацию о состоянии соединения. Например, когда пользователь заходит в приложение электронной почты через веб-интерфейс (допустим, используя сервис Hotmail), браузер отсыла-

ет серверу информацию cookie, позволяющую ему идентифицировать пользователя в течение сеанса работы приложения.

Несмотря на то, что механизм cookie очень часто упрощает процесс Интернет-покупок для пользователей, с другой стороны, тут возникает проблема нарушения конфиденциальности информации. Как мы только что видели, использование комбинации объектов cookie и предоставленных пользователем учетных данных способно привести к тому, что веб-сайт может получить достаточно информации о пользователе и потенциально передать эту информацию третьей стороне. Очень полезное обсуждение положительных и отрицательных сторон механизма cookie вы можете найти на веб-сайте Cookie Central[113].

2.2.5. Веб-кэширование

Веб-кэш, также называемый **прокси-сервером** — это элемент сети, который обрабатывает HTTP-запрос в дополнение к «настоящему» веб-серверу. Для этого на прокси-сервере имеется собственное дисковое хранилище, куда помещаются копии недавно запрошенных объектов. Как показано на рис. 2.11, браузер пользователя может быть настроен таким образом, чтобы все HTTP-запросы сначала направлялись в веб-кэш. В качестве примера предположим, что браузер запрашивает объект `http://www.someschool.edu/campus.gif`. Вот что происходит в этом случае:

1. Браузер устанавливает TCP-соединение с прокси-сервером и отправляет ему HTTP-запрос объекта.
2. Прокси-сервер проверяет, есть ли копия запрошенного объекта, хранящаяся локально. Если да, то этот объект возвращается в сообщении-ответе браузеру клиента.
3. Если прокси-сервер не нашел необходимый объект, то он открывает TCP-соединение с веб-сервером, то есть с `www.someschool.edu`. Затем отправляется HTTP-запрос объекта в соединение между прокси-сервером и веб-сервером. После получения запроса веб-сервер отправляет объект прокси-серверу внутри HTTP-ответа.
4. При получении объекта прокси-сервер сохраняет его копию, а также отправляет ее браузеру клиента вместе с HTTP-ответом (через существующее TCP-соединение между клиентом и прокси-сервером).

Отметим, что веб-кэш является и сервером, и клиентом одновременно: когда он получает запросы от браузера и отправляет ему ответы, он

выступает в роли сервера, когда он обменивается сообщениями с веб-сервером, то играет роль клиента.

Рис. 2.11. Запросы клиентов через прокси-сервер

Обычно прокси-серверы устанавливают Интернет-провайдеры. Например, какой-нибудь университет может установить прокси-сервер и настроить браузеры всех факультетов таким образом, чтобы запросы в Интернет проходили через него.

Технология веб-кэширования распространена в Интернете по двум причинам: во-первых, она позволяет уменьшить время ответа на запрос клиента, особенно если полоса пропускания между клиентом и веб-сервером намного меньше, чем между клиентом и прокси-сервером. Обычно между клиентом и прокси-сервером устанавливается высокоскоростное соединение, и поэтому прокси-сервер способен доставить объект клиенту очень быстро. Вторая причина, как мы вскоре продемонстрируем на примере, заключается в том, что прокси-сервер может уменьшить трафик в сети доступа организации, а это позволяет снизить расходы и положительно сказывается на производительности приложений, использующих сеть.

Давайте рассмотрим пример для лучшего понимания того, какую же выгоду дает использование прокси-серверов. На рис. 2.12 показаны две сети — сеть организации и публичный Интернет. Сеть организации представляет собой высокоскоростную ЛВС. Маршрутизатор в организации и маршрутизатор в Интернете соединены между собой каналом

связи со скоростью 15 Мбит/с. Веб-серверы, находящиеся в Интернете, размещены по всем миру. Размер среднего объекта будем считать равным 1 Мбит, а среднюю скорость запросов от браузеров организации к веб-серверам — составляющей 15 запросов в секунду. Допустим, что размер HTTP-запроса очень мал и не создает трафика ни в сетях, ни в канале связи между маршрутизаторами.

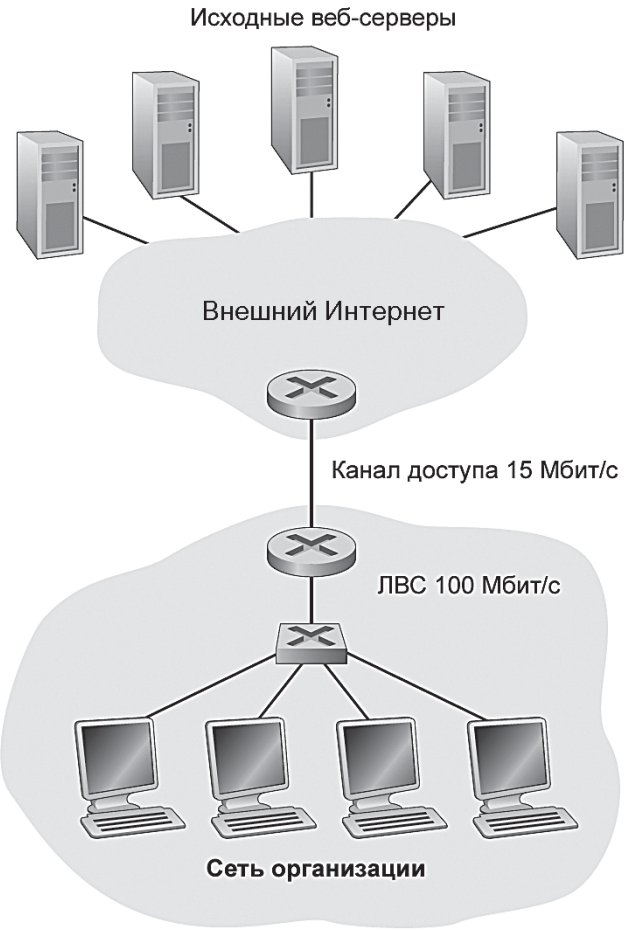


Рис. 2.12. Узкое место — канал связи между сетью организации и Интернетом

Также предположим, что среднее время, прошедшее с момента, когда маршрутизатор со стороны Интернета перенаправляет HTTP-запрос (внутри IP-дейтаграммы) до получения ответа (обычно в составе нескольких дейтаграмм), равно двум секундам. Будем называть это вре-

мя «задержкой Интернета». Общее время ответа — то есть время от момента запроса объектом браузера до получения им объекта — это сумма задержки ЛВС, задержки доступа (то есть задержки сигнала между двумя маршрутизаторами) и задержки Интернета. Произведем расчет для оценки этой задержки. Интенсивность трафика в сети ЛВС (см. раздел 1.4.2) равна

$$(15 \text{ запросов/с}) \times (1 \text{ Мбит/запрос}) / (100 \text{ Мбит/с}) = 0{,}15$$

в то время как интенсивность трафика в канале доступа (между маршрутизаторами) равна

$$(15 \text{ запросов/с}) \times (1 \text{ Мбит/запрос}) / (15 \text{ Мбит/с}) = 1$$

Величина интенсивности трафика в ЛВС, равная 0,15, обычно приводит к задержкам порядка 10 мс; следовательно, задержками ЛВС можно пренебречь. Однако, как обсуждалось в разделе 1.4.2, по мере того, как интенсивность трафика достигает единицы (как в случае с сетью доступа на рис. 2.12), задержка становится очень большой и возрастает бесконечно. Таким образом, среднее время ответа на запрос может быть порядка нескольких минут, если не больше, что, очевидно, неприемлемо для пользователей организации. Необходимо какое-то решение.

Одним из путей решения может быть увеличение скорости доступа с 15 Мбит/с, скажем, до 100 Мбит/с. Это понизит интенсивность трафика до 0,15, и в этом случае общее время ответа будет примерно равно 2 с, то есть задержке Интернета. Но данный вариант решения предполагает, что сеть организации должна быть модернизирована, а это дополнительные расходы.

Теперь рассмотрим альтернативное решение, в котором вместо модернизации сети используется прокси-сервер организации. Данное решение продемонстрировано на рис. 2.13. Доля запросов, обслуживаемых прокси-сервером (коэффициент попадания в кэш), обычно варьируется от 0,2 до 0,7. Предположим, что веб-кэш обеспечивает для организации 40% обрабатываемых запросов. Так как клиенты и прокси-сервер соединены одной и той же высокоскоростной ЛВС, 40% запросов будут удовлетворены прокси-сервером немедленно, в течение 10 мс. Тем не менее оставшиеся 60% запросов должны быть обработаны оригинальными веб-серверами. Но при оставшихся 60% запросов, проходящих через сеть доступа, интенсивность трафика в ней уменьшается с 1 до 0,6. Обычно интенсивность, меньшая, чем 0,8, на соединениях с пропускной способностью 15 Мбит/с соответствует небольшим задержкам порядка

10 мс. Такая задержка сравнима с двухсекундной задержкой Интернета. Средняя задержка в этом случае получается

$$0{,}4 \times (0{,}01\text{с}) + 0.6 \times (2{,}01\text{с})$$

что немногим больше, чем 1,2 с. Таким образом, второй вариант решения обеспечивает еще более низкое время ответа, чем первый, в то же время не требуя модернизировать сеть организации. Конечно же, нужно будет покупать и устанавливать прокси-сервер, но эти расходы относительно невысоки — многие прокси-серверы используют бесплатное программное обеспечение, которое работает на недорогих персональных компьютерах.

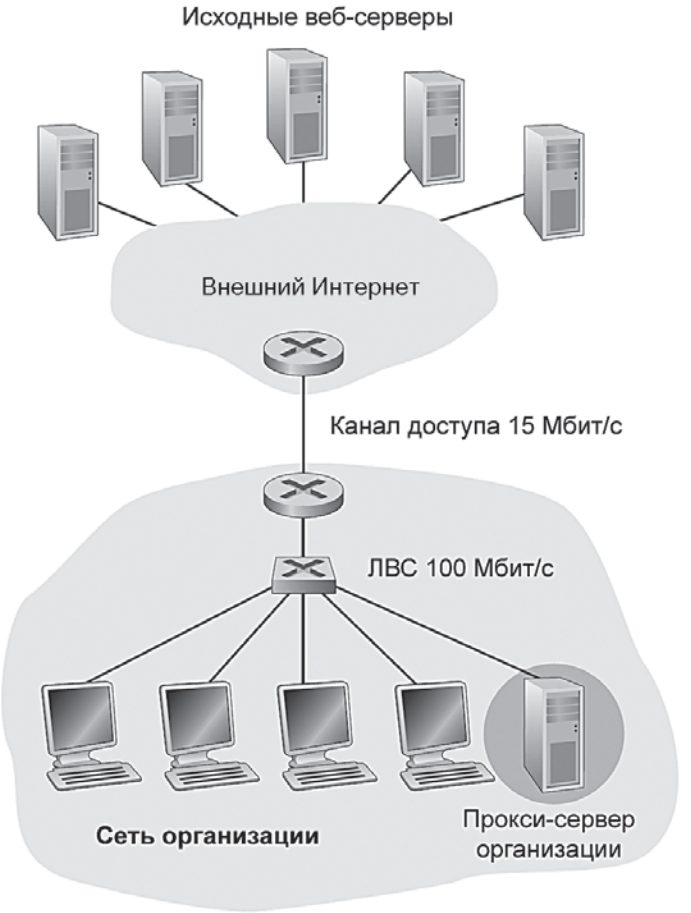


Рис. 2.13 Добавление прокси-сервера в сеть организации

С появлением **сетей доставки контента (Content Distribution Networks, CDNs)** прокси-серверы стали играть важную роль в Интернете. Компании-владельцы сетей CDN инсталлируют территориально распределенные прокси-серверы в Интернете, тем самым локализуя объем трафика. Существуют как разделяемые (такие как Akamai и Limelight), так и выделенные CDN-сети (такие как Google и Microsoft). Мы обсудим сети CDN более подробно в главе 7.

2.2.6. Метод `GET` с условием

Несмотря на то, что кэширование уменьшает время ответа пользовательскому браузеру, добавляется новая проблема — копия объекта, находящегося в кэше, может быть устаревшей, то есть с тех пор, как она была кэширована клиентом, объект, размещенный на веб-сервере, мог уже измениться. Но HTTP-протокол, к счастью, имеет механизм, позволяющий прокси-серверу проверять актуальность объектов. Для этого применяется так называемый **метод GET с условием** (условный GET-запрос). В этом случае HTTP-запрос использует, во-первых, метод `GET`, а во-вторых, добавляет строку заголовка `If-Modified-Since:`.

Чтобы проиллюстрировать работу условного метода `GET`, давайте рассмотрим пример. Сначала прокси-сервер от имени браузера отправляет сообщение-запрос серверу:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

Затем веб-сервер отправляет ответное сообщение с запрошенным объектом прокси-серверу:

```
HTTP/1.1 200 OK
Date: Sat, 8 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 7 Sep 2011 09:23:24
Content-Type: image/gif
(данные данные данные данные данные...)
```

Прокси-сервер перенаправляет объект браузеру, но, кроме того, сохраняет у себя в кэше его локальную копию. При этом важно отметить, что вместе с объектом сохраняется дата его последнего изменения. Далее, допустим, спустя неделю, браузер запрашивает опять тот же самый

объект через прокси-сервер. Так как объект на веб-сервере мог за прошедшую неделю измениться, прокси-сервер производит проверку актуальности его версии, вставляя в запрос условный оператор GET. Это выглядит следующим образом:

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 7 Sep 2011 09:23:24
```

Заметим, что значение строки заголовка If-modified-since: в точности совпадает со значением строки Last-Modified:, которая была отправлена сервером неделю назад.

Данный метод GET просит сервер переслать объект заново только в том случае, если он был изменен с момента, указанного в строке. Предположим, что объект не изменился с момента времени 7 Sep 2011 09:23:24. В этом случае ответ веб-сервера будет выглядеть следующим образом:

```
HTTP/1.1 304 Not Modified
Date: Sat, 15 Oct 2011 15:39:29
Server: Apache/1.3.0 (Unix)
(пустое тело сообщения)
```

Мы видим, что в ответ на условный метод GET веб-сервер отправляет сообщение, только не включает в него запрашиваемый объект. Включение запрашиваемого объекта в ответное сообщение было бы только пустой тратой пропускной способности и увеличило бы ответ сервера, особенно если размер объекта достаточно большой. Также обратите внимание в ответном сообщении на строку состояния 304 Not Modified, которая указывает прокси-серверу, что он может работать с тем объектом, который хранится у него в кэше, и смело отправлять его браузеру клиента.

На этом мы завершаем обсуждение протокола HTTP (протокола прикладного уровня), первого из протоколов Интернета, рассмотренного нами подробно. Мы познакомились с форматом HTTP-сообщений и с действиями, предпринимаемыми клиентом и веб-сервером при отправке и получении сообщений. Мы также коснулись инфраструктуры веб-приложений, включая прокси-серверы, механизмы cookie, базы данных на сервере — всего, что определенно связано с работой HTTP-протокола.

2.3. Передача файлов по протоколу FTP

Типичный FTP-сеанс представляет собой передачу файлов пользователем, сидящим за одним хостом (локальным), на другой хост (удаленный) или обратно. Для того чтобы пользователь имел доступ к удаленной учетной записи, он должен ввести идентификатор и пароль, после чего будет способен передавать файлы из локальной системы в удаленную и обратно. Как показано на рис. 2.14, пользователь взаимодействует с протоколом FTP через пользовательский агент FTP. Сначала пользователь предоставляет имя удаленного хоста, заставляя клиентский процесс FTP на локальном устройстве установить TCP-соединение с процессом FTP-сервера на удаленном хосте. Затем вводит идентификатор и пароль, которые отправляются через TCP-соединение в составе FTP-команд. Как только сервер авторизует пользователя, тот может копировать один или несколько файлов, хранящихся на локальной системе, в удаленную систему (или наоборот).

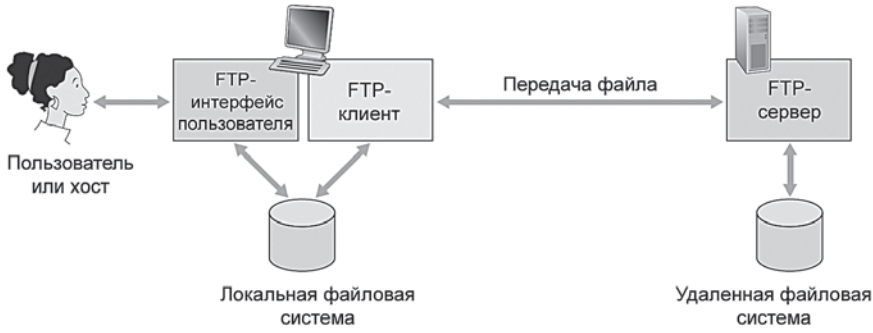


Рис. 2.14 Протокол FTP передает файлы между локальной и удаленной файловыми системами

HTTP и FTP являются протоколами передачи данных и имеют много общего, например, они оба работают поверх протокола TCP. Однако эти два протокола прикладного уровня имеют и очень существенные различия: протокол FTP использует для передачи файла два параллельных TCP-соединения: **управляющее соединение** и **соединение данных**. Управляющее соединение используется для отправки контрольной информации между двумя хостами — такой, как идентификатор пользователя, пароль, команды для смены каталога, а также команды передачи и получения файлов. Соединение данных используется для передачи самого файла. Говорят, что FTP отправляет управляющую информацию

вне полосы (out-of-band). Протокол HTTP, как вы помните, отсылает строки заголовков запроса и ответа в одном и том же TCP-соединении вместе с передаваемым файлом. Поэтому говорят, что протокол HTTP отправляет управляющую информацию **внутри полосы** (in-band). В следующем разделе мы с вами увидим, что основной протокол электронной почты SMTP также передает управляющую информацию внутри полосы. Соединение данных и управляющее соединение отображены на рис. 2.15.

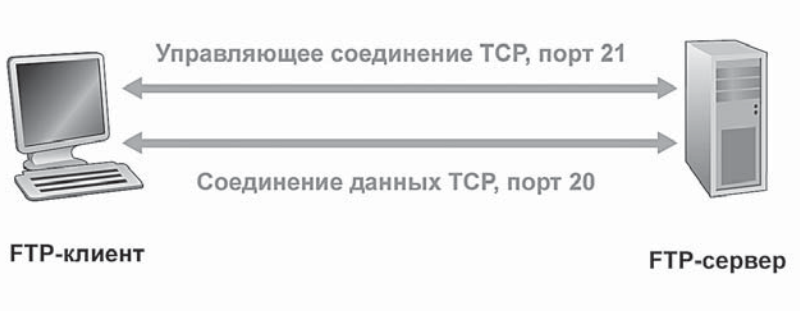


Рис. 2.15 Управляющее соединение и соединение данных

Когда пользователь начинает FTP-сеанс с удаленным хостом, клиентская сторона протокола FTP (пользователь) сначала инициирует управляющее TCP-соединение с серверной стороной (удаленным хостом) на серверном порту 21. Клиент FTP отправляет идентификатор и пароль пользователя через это управляющее соединение. Также через него отправляются команды на изменение удаленного каталога. Когда серверная сторона получает команду на передачу файла через управляющее соединение (либо получить, либо отправить файл с удаленного хоста), сервер инициирует TCP-соединение данных с клиентской стороной. FTP отправляет ровно один файл через это соединение данных и затем закрывает его. Если в течение этого же сеанса пользователь запрашивает передачу другого файла, то для него FTP открывает еще одно соединение данных. Таким образом, при передаче файлов управляющее соединение остается открытым в течение всего пользовательского сеанса, а для передачи каждого файла внутри этого сеанса создается новое соединение данных (то есть соединение данных в FTP является непостоянным).

В течение пользовательского сеанса FTP-сервер должен хранить состояние этого сеанса, а именно серверу нужно как-то связывать управ-

ляющее соединение с конкретной пользовательской учетной записью, сервер должен отслеживать текущий каталог пользователя по мере того, как он пробегает по дереву каталогов. Такое отслеживание состояния каждого пользовательского сеанса значительно ограничивает общее число одновременно обслуживаемых сеансов FTP. Напомним, что HTTP, наоборот, является протоколом, не сохраняющим состояние сеанса пользователя.

2.3.1. Команды и ответы протокола FTP

Мы завершим этот раздел кратким обсуждением некоторых из наиболее распространенных FTP-команд и ответов. Команды, посылаемые клиентом серверу, и ответы сервера клиенту передаются по управляющему соединению в виде семиразрядных символов формата ASCII, так что команды FTP, как и HTTP-команды, свободно можно прочитать. Для разделения команд используется пара символов: возврат каретки и перевод строки. Каждая команда состоит из четырех символов ASCII в верхнем регистре и необязательных параметров. Некоторые из наиболее распространенных команд вы видите ниже:

- `USER username`: используется для передачи идентификатора пользователя серверу
- `PASS password`: используется для передачи пользовательского пароля серверу
- `LIST`: используется для запроса у сервера списка всех файлов, находящихся в текущем удаленном каталоге. Список файлов передается через соединение данных (новое и непостоянное), а не через управляющее TCP-соединение
- `RETR filename`: используется для вызова (то есть, получения) файла из текущего каталога на удаленном хосте. Данная команда приводит к тому, что удаленный хост инициирует соединение данных и отправляет запрашиваемый файл через это соединение.
- `STOR filename`: используется для сохранения (то есть отправки) файла в текущий каталог удаленного хоста

Обычно одной команде, введенной пользователем, соответствует одна FTP-команда, отправленная через управляющее соединение. На каждую команду сервер дает ответ клиенту. Ответы представляют собой трехзначные числа, сопровождаемые необязательным сообщением.

Структура такого ответа похожа на структуру строки состояния ответного HTTP-сообщения с кодом состояния и фразой. Некоторые распространенные ответы FTP-сервера представлены ниже:

- `331 Username OK, password required`
- `125 Data connection already open; transfer starting`
- `425 Can't open data connection`
- `452 Error writing file`

Тем читателям, кому интересно поподробнее изучить FTP-команды и ответы сервера, рекомендуем обратиться к документу RFC 959.

2.4. Электронная почта в Интернете

Электронная почта, появившись на заре развития Интернета и уже тогда завоевав широкую популярность, и по сей день остается одним из важнейших и широко используемых приложений, став за эти годы более совершенным и мощным инструментом в руках пользователей[596].

Как и обычная почта, почта электронная представляет собой асинхронное средство коммуникации — люди отправляют и читают сообщения, когда им удобно, не заботясь о том, чтобы скоординировать это время с расписанием других людей. Но, в отличие от обычной, электронная почта — средство гораздо более быстрое, менее затратное и максимально простое в доставке. Современная электронная почта предлагает мощные инструменты, включая прикрепление вложений, гиперссылок, текста в HTML-формате, а также встроенные фото и видео.

В данном разделе мы изучим протоколы прикладного уровня, лежащие в основе электронной почты Интернета. Перед тем как углубляться в обсуждение этих протоколов, давайте взглянем на общую картину, представляющую систему электронной почты и ее основные составляющие.

На рис. 2.16 представлена почтовая система Интернета. Мы видим из этой диаграммы, что она включает три основных компонента: **пользовательский агент**, **почтовые серверы** и **простой протокол передачи почты** (Simple Mail Transfer Protocol, **SMTP**). Сейчас мы опишем каждый из этих компонентов с точки зрения отправителя, Алисы, которая направляет сообщение электронной почты получателю — Бобу. Пользовательские агенты позволяют создавать и прочитывать сообщения, отвечать на них, перенаправлять и сохранять. Примерами таких аген-

тов электронной почты являются Microsoft Outlook и Apple Mail. Когда Алиса закончила составлять письмо, ее пользовательский агент отправляет это сообщение на ее почтовый сервер, где оно помещается в очередь исходящих сообщений. Когда Боб собирается прочитать письмо, его пользовательский агент извлекает сообщение из его почтового ящика, находящегося на почтовом сервере Боба.

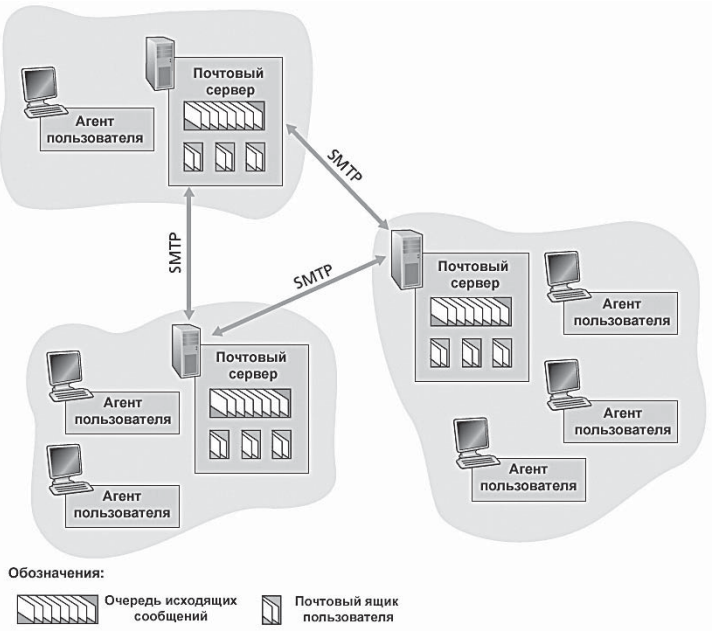


Рис. 2.16 Система электронной почты Интернета

Почтовые серверы образуют ядро инфраструктуры электронной почты. Каждый получатель (такой, как Боб) имеет свой **почтовый ящик**, размещенный на одном из почтовых серверов. Почтовый ящик Боба обслуживает сообщения, которые направлены Бобу. Обычное сообщение, таким образом, начинает свой путь в пользовательском агенте отправителя, доходит до почтового сервера отправителя и направляется к почтовому серверу получателя, где помещается в его почтовый ящик.

ИСТОРИЯ

Веб-почта

В декабре 1995 года, спустя несколько лет после того, как люди узнали о Всемирной паутине, Джек Смит (Jack Smith) и Сабир Батиа

(Sabeer Bhatia) обратились в известную венчурную компанию Draper Fisher Jurvetson с предложением разработать бесплатную систему электронной почты с доступом через веб-интерфейс. Их идея состояла в том, чтобы предоставить любому желающему бесплатную учетную запись электронной почты и сделать ее доступной через веб-интерфейс. В обмен на 15% капитала будущей компании Draper Fisher Jurvetson профинансировали Смита и Батиа, которые основали компанию Hotmail. Имея трех штатных сотрудников и 14 человек, работающих на временной основе, компания сумела разработать и запустить почтовую службу к июлю 1996 года. Спустя месяц после запуска служба Hotmail уже насчитывала 100 000 подписчиков, а в декабре 1997 года их число увеличилось до 12 000 000. Вскоре после этого ее приобрела компания Microsoft. Как сообщается, сумма сделки была приблизительно равна 400 миллионам долларов. Успех Hotmail часто связывают с тем, что она имела преимущество «первопроходца» и разработала свой собственный «вирусный маркетинг» (возможно, некоторые из читающих эту книгу студентов окажутся в числе новых предпринимателей, которым предстоит в будущем запустить инновационные Интернет-службы с помощью этой методики).

Веб-почта продолжает развиваться, ее инструменты становятся из года в год все мощнее и совершеннее. Одна из наиболее популярных служб на сегодняшний день — это Gmail от компании Google, предлагающая пользователям гигабайты бесплатного дискового пространства, расширенные возможности для фильтрации спама и обнаружения вирусов, шифрование почтовых сообщений (с использованием SSL), услуги по сбору почты от сторонних почтовых служб, а также удобный интерфейс с поисковой системой. Последние годы также становятся популярными асинхронный обмен сообщениями в социальных сетях, таких, как, например, Facebook.

Когда Боб пытается получить доступ к сообщениям, находящимся в его почтовом ящике, почтовый сервер проводит аутентификацию (используя имя пользователя и пароль). Почтовый сервер Алисы должен как-то обрабатывать ошибки почтового сервера Боба. Если сервер Алисы не может доставить почту на сервер Боба, то он сохраняет сообщение в **очереди сообщений** и пытается отправить его позже. Повторные попытки отправки обычно делаются каждые 30 минут в зависимости от настроек сервера. Если попытки неуспешны в течение нескольких дней, сервер удаляет сообщение и уведомляет об этом отправителя (Алису).

SMTP является ключевым протоколом прикладного уровня для электронной почты в Интернете. Он использует службу надежной до-

ставки данных, предоставляемую протоколом TCP для передачи сообщений от почтового сервера отправителя к почтовому серверу получателя. Как и большинство протоколов прикладного уровня, SMTP имеет две стороны: клиентскую, исполняемую на сервере отправителя, и серверную — на сервере получателя. Обе части протокола работают на каждом почтовом сервере. То есть, когда сервер отправляет почту другим почтовым серверам, он выполняет функции SMTP-клиента, а когда получает почту от других серверов, выступает в роли SMTP-сервера.

2.4.1. Протокол SMTP

Протокол SMTP, определенный в документе RFC 5321, составляет самое сердце электронной почты Интернета. Как указывалось выше, SMTP передает сообщение от почтовых серверов отправителей к серверам получателей. Протокол SMTP намного старше, чем HTTP. (Самый первый документ RFC, относящийся к протоколу SMTP, датируется 1982 годом.) Хотя протокол SMTP имеет много прекрасных достоинств, что подтверждается широким его использованием в Интернете, тем не менее в нем присутствуют некоторые минусы, унаследованные из былых времен. Например, он ограничивает тело любого сообщения почты семиразрядным форматом ASCII. Это имело смысл в начале 1980-х годов, когда объем передаваемых данных был невелик, и никто не отправлял по почте большие вложения или аудио- и видеофайлы. Но в сегодняшнюю мультимедийную эпоху такое ограничение достаточно ощутимо — оно требует, чтобы двоичные мультимедийные данные были преобразованы в формат ASCII перед отправкой через SMTP, а после передачи через SMTP декодированы обратно в бинарный формат. Как мы помним из раздела 2.2, протокол HTTP не требует такого преобразования для мультимедийных данных перед их передачей.

Давайте рассмотрим работу протокола SMTP на простом примере. Предположим, Алиса хочет отправить Бобу простое текстовое сообщение формата ASCII.

1. Алиса запускает свой пользовательский агент электронной почты, указывает ему электронный адрес Боба (например, `bob@someschool.edu`), создает сообщение и дает команду пользовательскому агенту его отправить.
2. Пользовательский агент Алисы отправляет сообщение на ее почтовый сервер, где оно помещается в очередь.

3. Клиентская часть протокола SMTP, запущенная на почтовом сервере Алисы, видит сообщение в очереди и открывает TCP-соединение с SMTP сервером, запущенном на почтовом сервере Боба.
4. После первоначального SMTP-рукопожатия клиент SMTP отправляет сообщение Алисы в установленное TCP-соединение.
5. На почтовом сервере Боба серверная часть SMTP принимает сообщение, которое затем помещается в почтовый ящик Боба.
6. Боб запускает свой пользовательский почтовый агент, чтобы прочитать сообщение в удобное ему время.

Данная схема проиллюстрирована на рис. 2.17.

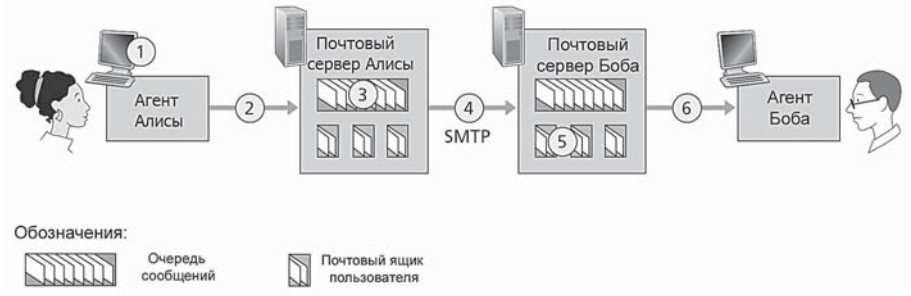


Рис. 2.17. Отправка сообщения от Алисы к Бобу

Очень важно отметить, что протокол SMTP не использует каких-либо промежуточных серверов для отправки почты, даже когда два почтовых сервера располагаются на противоположных концах мира. Если сервер Алисы, например, находится в Гонконге, а сервер Боба — в Сент-Луисе, TCP-соединение между ними — это непосредственное соединение между серверами в Гонконге и Сент-Луисе, и если почтовый сервер Боба не работает, то сообщение остается в почтовом сервере Алисы и ожидает новых попыток, а не помещается в какой-нибудь промежуточный почтовый сервер.

Теперь давайте подробнее рассмотрим, как SMTP передает сообщение от отправляющего сервера к принимающему. Мы увидим, что протокол SMTP имеет много общего с протоколами, используемыми при непосредственном взаимодействии друг с другом двух человек. Во-первых, клиент SMTP (запущенный на почтовом сервере отправителя) устанавливает соединение с сервером SMTP (запущенном на почтовом сервере получателя) на порту 25. Если сервер недоступен, клиент пытается

соединиться еще раз. Как только соединение устанавливается, клиент и сервер обмениваются рукопожатиями прикладного уровня — так же, как люди обычно приветствуют друг друга перед обменом информацией, SMTP-клиенты и серверы представляются друг другу перед передачей данных. В течение этой фазы рукопожатия клиент SMTP указывает почтовый адрес отправителя (того, кто генерирует сообщение) и почтовый адрес получателя. Как только клиент и сервер SMTP представились друг другу, клиент отправляет сообщение. SMTP может полагаться на службу TCP, которая надежно и без ошибок доставит данные на сервер. Затем клиент повторяет этот процесс в рамках того же самого TCP-соединения, если у него есть еще одно сообщение для отправки серверу; в противном случае он дает команду протоколу TCP закрыть соединение.

Давайте взглянем на пример расшифровки сообщений, которыми обмениваются SMTP-клиент (К) и SMTP-сервер (С). Имя хоста клиента `crepes.fr`, а в качестве сервера выступает `hamburger.edu`. Текстовые строки, предваренные символом `К:` — это сообщения, которые клиент направил в свой TCP-сокет, а строки, начинающиеся с `С:` — сообщения, отправляемые в TCP-сокет сервером. Как только TCP-соединение устанавливается, мы получаем следующее:

```
C: 220 hamburger.edu
K: HELO crepes.fr
C: 250 Hello crepes.fr, pleased to meet you
K: MAIL FROM: <alice@crepes.fr>
C: 250 alice@crepes.fr ... Sender ok
K: RCPT TO: <bob@hamburger.edu>
C: 250 bob@hamburger.edu ... Recipient ok
K: DATA
C: 354 Enter mail, end with "." on a line by itself
K: Вы любите кетчуп?
K: Как насчет анчоусов?
K: .
C: 250 Message accepted for delivery
K: QUIT
C: 221 hamburger.edu closing connection
```

В этом примере клиент отправляет сообщение («`Вы любите кетчуп? Как насчет анчоусов?`») с почтового сервера `crepes.fr` на

почтовый сервер hamburger.edu. В составе диалога клиент дает пять команд: HELO (сокращенно от HELLO), MAIL FROM, RCPT TO, DATA и QUIT. Названия команд говорят сами за себя. Клиент также отправляет строку, состоящую из единственной точки, которая указывает серверу на конец сообщения (в формате ASCII каждое сообщение заканчивается символами CRLF.CRLF, где CR и LF означают возврат каретки и перевод строки, соответственно). На каждую команду сервер выдает ответ, сопровождающийся кодом ответа и некоторым (необязательным) англоязычным объяснением. Отметим здесь, что SMTP использует постоянные соединения: если сервер-источник пытается отправить несколько сообщений одному и тому же серверу-получателю, он может направить их все через одно TCP соединение. В таком случае, каждое сообщение клиент будет начинать новой строкой MAIL FROM: crepes.fr и заканчивать отдельной точкой, а команду QUIT будет выдавать только после того, как все сообщения отправлены.

Очень рекомендуем вам попробовать прямой диалог с SMTP-сервером, используя команду Telnet. Чтобы сделать это, наберите в командной строке

```
telnet serverName 25
```

где serverName — это имя локального почтового сервера. Набрав такую команду, вы просто устанавливаете TCP-соединение между вашим хостом и почтовым сервером. После ввода этой строки вы должны сразу же получить ответ от сервера с кодом 220. После этого дайте несколько SMTP-команд: HELO, MAIL FROM, RCPT TO, DATA,

CRLF.CRLF и QUIT. Рекомендуем также выполнить упражнение 3 в конце главы, в котором вы создадите простой пользовательский агент, реализующий клиентскую часть SMTP. Это позволит вам отправлять сообщения электронной почты произвольному получателю через локальный почтовый сервер.

2.4.2. Сравнение с протоколом HTTP

Давайте произведем краткое сравнение протоколов SMTP и HTTP. Оба протокола используются для передачи файлов с одного хоста на другой: HTTP передает файлы (или объекты) от веб-сервера веб-клиенту (обычно с помощью браузера); SMTP передает файлы (то есть сообщения электронной почты) от одного почтового сервера к другому. При передаче файлов оба протокола используют постоянное соедине-

ние. В этом они похожи. Однако, существует и значительная разница между ними: во-первых, протокол HTTP является **протоколом получения (pull-протокол)** — пользователи применяют его, чтобы получить информацию с сервера, которую туда кто-то загрузил, в любой удобный им момент. В частности, TCP-соединение инициируется компьютером, который хочет получить файл. С другой стороны, протокол SMTP является **протоколом отправки (push-протокол)** — сервер-источник отправляет файл почтовому серверу-приемнику. В этом случае TCP-соединение инициируется компьютером, который хочет отправить файл.

Второе отличие, о котором мы уже раньше упоминали, состоит в том, что протокол SMTP требует, чтобы любое сообщение, включенное в тело сообщения электронной почты, имело семиразрядный формат ASCII. Если в сообщении содержатся другие символы (например, символы французского языка с диакритическими знаками) или двоичные данные (такие, как файл изображения), то оно должно быть перекодировано в семиразрядный формат ASCII. Протокол HTTP не накладывает таких ограничений.

Третье важное отличие заключается в том, как обрабатываются документы, содержащие текст и изображения (возможно и другие мультимедийные файлы). Как мы знаем из раздела 2.2, HTTP инкапсулирует каждый объект в свое собственное HTTP-сообщение, в то время как протокол SMTP помещает все объекты сообщения в одно.

2.4.3. Форматы почтового сообщения

Когда Алиса отправляет обычную почтовую открытку Бобу, она пишет на конверте адрес Боба, свой собственный адрес, дату и так далее — сюда она может включить любую сопроводительную информацию, которую посчитает нужным добавить. Аналогично при отправлении электронной почты от одного адресата к другому такая же сопроводительная информация содержится в заголовке, предваряющем тело самого сообщения. Эта информация обычно содержится в наборе строк заголовка, которая определяется документом RFC 5322. Заголовочные строки и тело сообщения разделяются пустой строкой (то есть возвратом каретки и переводом строки). В документе RFC 5322 указан точный формат для строк заголовка электронной почты, а также их смысловое значение. Как и в случае с HTTP, каждая строка заголовка содержит обычный читаемый текст, состоящий из ключевого слова, после которо-

го следует двоеточие и значение. Некоторые ключевые слова обязательные, а некоторые — нет. Каждый заголовок должен иметь строку `From:` и строку `To:`, может включать строку `Subject:` и другие необязательные строки. Важно отметить, что эти строки заголовка *отличаются* от SMTP-команд, изученных в разделе 2.4.1 (даже несмотря на то, что они содержат такие же слова, как From и To). Команды, которые рассматривались в том разделе, составляли часть рукопожатия протокола SMTP; строки заголовка в этом случае являются частью самого почтового сообщения.

Типичный заголовок сообщения выглядит следующим образом:

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Поиск смысла жизни.
```

После заголовка стоит пустая строка, затем тело сообщения в формате ASCII. Попробуйте использовать Telnet для отправки почтовому серверу сообщения, содержащего некоторые строки заголовка, включая строку `Subject:`. Чтобы это сделать, выполните команду `telnet serverName 25`, как уже обсуждалось в разделе 2.4.1.

2.4.4. Протоколы доступа к электронной почте

Когда SMTP доставляет сообщение с почтового сервера Алисы на почтовый сервер Боба, оно помещается в почтовый ящик Боба. Здесь мы предполагаем, что Боб читает свою почту, входя на серверный хост и затем исполняя почтовую программу на этом хосте. До начала 90-х годов это было типичным способом получения сообщений. Но сегодня для доступа к электронной почте используется клиент-серверная архитектура. Типичный пользователь читает сообщения электронной почты с помощью почтового клиента, исполняемого на его конечной системе, например на офисном персональном компьютере, на ноутбуке или смартфоне. Пользуясь почтовым клиентом на своих устройствах, пользователь получает широкий набор доступных ему функций, чтобы просмотреть прикрепленные мультимедийные вложения.

Если получатель (Боб) запускает пользовательский агент на своем персональном компьютере, было бы естественным и почтовый сервер разместить на этом же компьютере. При таком подходе почтовый сервер Алисы должен напрямую общаться с компьютером Боба. Но здесь су-

ществует одна проблема. Вспомним, что почтовый сервер управляет почтовыми ящиками и работает с клиентской и серверной частями SMTP. Если бы почтовый сервер Боба размещался на его локальном персональном компьютере, то Боб должен был бы держать этот компьютер постоянно включенным и соединенным с Интернетом, чтобы получать новые сообщения электронной почты, которые могут поступить в любое время. Это очень неудобно, поэтому типичный пользователь запускает почтовый агент на локальном устройстве и осуществляет доступ к своему почтовому ящику, хранящемуся на разделяемом почтовом сервере, который всегда доступен. Обычно этот сервер обслуживает ящики многих пользователей, и очень часто его предоставляет компания, являющаяся Интернет-провайдером.

Теперь рассмотрим путь сообщения электронной почты, отправленного от Алисы Бобу. Мы уже узнали, что в какой-то точке этого пути оно должно быть помещено в почтовый сервер Боба. Это можно сделать, если пользовательский агент Алисы отправляет сообщение напрямую почтовому серверу Боба, и делается это с помощью SMTP, который для того и предназначен, чтобы «проталкивать» сообщения от одного хоста к другому. Однако в реальности типичный пользовательский агент отправителя не общается напрямую с почтовым сервером получателя. Вместо этого, как показано на рис. 2.18, почтовый агент Алисы использует протоколы SMTP для отправки сообщения на ее почтовый сервер, а затем почтовый сервер Алисы использует SMTP (уже как клиент) для ретрансляции (перенаправления) сообщения почтовому серверу Боба. Зачем нужна двухшаговая процедура? Основная причина в том, что без перенаправления через почтовый сервер Алисы, ее пользовательский агент в случае, если почтовый сервер-получатель недоступен, не знает, что делать с сообщением. А если Алиса поместит сообщение на собственный почтовый сервер, он, в свою очередь, начнет периодически отправлять сообщение серверу Боба, например, каждые 30 минут до тех пор, пока тот не заработает (уж если и сервер Алисы не работает, то она хотя бы может обратиться с жалобой к своему администратору вычислительной сети или провайдеру). Документы RFC для протокола SMTP определяют, какие команды и каким образом используются для перенаправления почтовых сообщений через множественные SMTP-серверы.

Осталось одно недостающее звено в этой головоломке! Каким образом получатель, такой как Боб, имеющий пользовательский почтовый агент на своем локальном компьютере, будет получать свои сообщения, которые размещены на почтовом сервере его Интернет-провайдера? За-

метим, что пользовательский агент Боба не может использовать SMTP, чтобы получить эти сообщения, так как SMTP — это push-протокол (протокол отправки, а не получения). Ключом к этой задаче будет специальный протокол доступа к почте, который передает сообщения с почтового сервера Боба на его локальное устройство. Существует несколько популярных протоколов доступа к почте, включая **POP3 (Post Office Protocol, протокол почтового отделения, версия 3)**, **IMAP (Internet Mail Access Protocol**, **протокол доступа к почте через Интернет)** и HTTP.

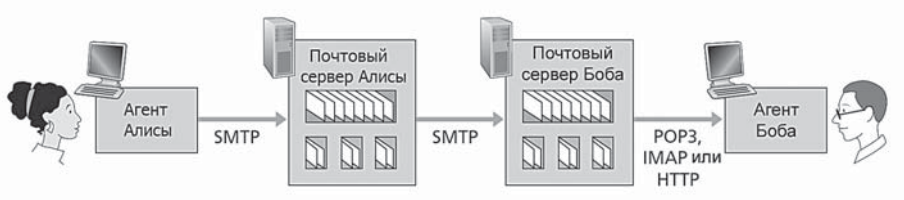


Рис. 2.18. Протоколы электронной почты и их взаимодействующие компоненты

На рис. 2.18 представлены протоколы, используемые для работы с почтой в Интернете: SMTP применяется для передачи от почтового сервера отправителя к почтовому серверу получателя, а также для передачи сообщения от пользовательского агента отправителя к его почтовому серверу. С помощью протоколов доступа к электронной почте таких, как POP3, происходит передача сообщений от почтовых серверов получателя к их пользовательским почтовым агентам.

Протокол POP3

Протокол доступа к электронной почте POP3, описанный в RFC 1939[459], является достаточно простым. Его простота определяет относительно скромную функциональность. POP3 начинает работать, когда агент пользователя (клиент) открывает TCP-соединение с почтовым сервером по порту 110. После установления соединения работа протокола POP3 проходит три состояния: авторизация, транзакция и обновление. Во время первой фазы работы протокола (авторизации) агент пользователя отправляет учетные данные (имя и пароль) для аутентификации его на сервере. Во второй фазе (транзакции) пользовательский агент получает сообщение, а также может помечать сообщения для удаления, удалять эти пометки, запрашивать почтовую статистику. Фаза обновления наступает после того, как клиент дал команду `quit`, завер-

шая сеанс POP3. В этот момент почтовый сервер удаляет помеченные сообщения и закрывает соединение.

На этапе транзакции агент пользователя отправляет команды, а сервер выдает ответ на каждую Существует два возможных ответа: `+OK` (иногда сопровождаемый данными от сервера клиенту) используется, чтобы указать, что предыдущая команда была успешной; `-ERR` используется, если предыдущая команда выполнена с ошибкой.

На этапе авторизации существует две основных команды: `user <username>` и `pass <password>`. Для демонстрации этих команд запустите Telnet, чтобы соединиться напрямую с POP3-сервером, используя порт 110, и наберите эти команды. Предположим, что `mailServer` — это имя вашего почтового сервера. У вас выйдет что-то похожее на следующее:

```
telnet mailServer 110
+OK POP3 server ready
user bob
+OK
pass hungry
+OK user successfully logged on
```

Если вы ошибетесь в наборе команды, сервер POP3 выдаст ответ с сообщением — `ERR`.

Теперь посмотрим на этап транзакции. Пользовательский агент, применяющий протокол POP3, может быть сконфигурирован пользователем в двух режимах: «загрузить и удалить» или «загрузить и сохранить». Последовательность команд, выдаваемых пользовательским агентом, зависит от того, в каком режиме агент работает. В режиме «загрузить и удалить» будут выдаваться команды `list`, `retr` и `dele`. В качестве примера предположим, что у пользователя в его почтовом ящике содержатся два сообщения. В диалоге, представленном ниже, строки, начинающиеся с `K:` — это команды со стороны клиента, а строки, начинающиеся с `C:` — это ответ почтового сервера. Транзакция будет выглядеть следующим образом:

```
K: list
C: 1 498
C: 2 912
```

```
C: .
K: retr 1
C: (бла бла...
C: ..................
C: ..........бла)
C: .
K: dele 1
K: retr 2
C: (бла бла...
C: ..................
C: ..........бла)
C: .
K: dele 2
K: quit
C: +OK POP3 server signing off
```

Сначала пользовательский агент запрашивает почтовый сервер вывести список с размером всех хранящихся в ящике сообщений, затем агент читает и удаляет каждое сообщение с сервера. Отметим, что после фазы авторизации пользовательский агент использует только 4 команды: `list`, `retr`, `dele` и `quit`, синтаксис которых определен документом RFC 1939. После обработки команды `quit` POP3-сервер переходит в фазу обновления и удаляет сообщения 1 и 2 из почтового ящика пользователя.

Проблема заключается в том, что Боб как получатель почты, возможно, захочет иметь доступ к своим почтовым сообщениям из нескольких мест: с офисного компьютера, с домашнего, а также, возможно, с мобильного устройства. Режим «загрузить и удалить» ограничивает в этом Боба: если Боб, например, прочитает сообщение на своем рабочем компьютере, он не сможет его заново прочитать со своего мобильного устройства, либо позже вечером с домашнего компьютера. В режиме «загрузить и сохранить» агент пользователя оставляет сообщение на почтовом сервере после его загрузки. В таком случае Боб может читать эти сообщения с разных устройств сколько угодно раз и в удобное ему время.

Во время POP3-сеанса между пользовательским агентом и почтовым сервером POP3-сервер обрабатывает некоторую информацию:

в частности, он отслеживает, какие сообщения пользователь пометил для удаления. Однако состояние POP3-сеанса сервер не сохраняет, и это значительно упрощает реализацию POP3-серверов.

Протокол IMAP

После того как Боб загрузил свои сообщения на локальный компьютер, с помощью доступа по протоколу POP3 он может создавать почтовые папки и перемещать в них загруженные сообщения. Затем он может удалять их, разносить сообщения по папкам, осуществлять поиск (по имени получателя или по теме), но размещение писем в папке на локальной машине влечет проблемы для мобильного пользователя, который предпочитает работать с иерархической структурой папок на удаленном сервере и иметь доступ к нему с любого компьютера. Такое невозможно осуществить с помощью протокола POP3.

Чтобы решить эту и некоторые другие проблемы, был разработан протокол IMAP, описанный в документе RFC 3501[517]. Он, как и POP3, является протоколом доступа. Его функциональность более широкая, чем у POP3, а, значит, более сложная, поэтому и реализация клиентских и серверных частей тоже значительно сложнее, чем у POP3.

IMAP-сервер связывает каждое сообщение с определенным каталогом. Когда сообщение впервые прибывает на сервер, оно привязывается к каталогу INBOX получателя (входящие сообщения). После этого получатель может перемещать сообщения в новые, созданные пользователем каталоги, читать, удалять и так далее. Протокол IMAP обеспечивает пользователям команды, позволяющие создавать каталоги, перемещать сообщения из одного в другой, а также производить поиск в удаленных каталогах по определенным критериям. В отличие от POP3, IMAP-сервер обрабатывает информацию о состоянии пользователя в течение IMAP-сеанса, например имена папок и привязку сообщений к каталогам.

Еще одна важная функция протокола IMAP заключается в том, что он предоставляет команды, позволяющие пользовательскому агенту получать части сообщений, например, только заголовок, либо одну из частей MIME-сообщения. Такая функциональность очень полезна, например, при низкоскоростном соединении (при использовании модема, например) между агентом пользователя и его почтовым сервером. В этом случае, пользователь может загружать только заголовки, избегая долгой загрузки больших сообщений, содержащих аудио- или видеоданные.

Электронная почта через веб-интерфейс

Все больше пользователей в сегодняшние дни отправляют или получают электронную почту, используя веб-браузеры. Впервые доступ к электронной почте через веб был представлен компанией Hotmail в середине 1990-х годов. Сегодня эту услугу предлагают такие компании, как Google, Yahoo! и другие крупные провайдеры. В этом случае пользовательским агентом является обычный веб-браузер, и пользователь взаимодействует со своим удаленным почтовым ящиком через протокол HTTP. Когда получатель, такой как Боб, хочет получить доступ к сообщению в своем почтовом ящике, то оно с его почтового сервера отправляется в браузер Боба с помощью протокола HTTP. Протоколы POP3 и IMAP при этом не используются. То же самое происходит, когда отправитель, такой как Алиса, хочет послать сообщение: оно передается от ее браузера к ее почтовому серверу через HTTP, а не через SMTP. При этом, однако, SMTP все же используется, когда почтовый сервер Алисы отправляет или получает сообщение с других почтовых серверов.

2.5. DNS — служба каталогов Интернета

Чтобы идентифицировать определенного человека, пользуются разными способами. Например, мы можем идентифицировать себя именем и фамилией, номерами социального страхования, а также номером водительского удостоверения. Одни методы идентификации могут быть более предпочтительными по сравнению с другими. Например, компьютеры налогового управления США (Internal Revenue Service, IRS) используют не имена людей, а номера социального страхования фиксированной длины. С другой стороны, обычные люди предпочитают использовать запоминающиеся идентификаторы (представьте, как нелепо будет выглядеть фраза: «Привет, мое имя 132-67-9875. А это мой муж, 178-87-1146»).

Так же, как и людей, хосты в Интернете можно идентифицировать несколькими способами. Один из способов идентификации — по **имени хоста**. Такие имена наглядны и нравятся людям, например, `cnn.com`, `www.yahoo.com`, `gaia.cs.umass.edu` или `cis.poly.edu`. Однако, имена хостов, хотя и предоставляют некоторую информацию, но очень небольшую (имя `www.eurecom.fr`, оканчивающееся кодом страны `.fr`, сообщает нам, что данный хост, вероятно, находится во Франции и не более того). Кроме того, имена хостов могут содержать разной дли-

ны буквенно-цифровые символы, что способно привести к трудностям при обработке их маршрутизаторами. По этим причинам хосты идентифицируются еще и так называемыми **IP-адресами**.

Детальнее мы обсудим IP-адреса в главе 4, но здесь уместно сказать о них несколько слов. IP-адрес состоит из 4 байт и имеет строго иерархическую структуру. Выглядит он таким образом: `121.7.106.83`, где каждая точка отделяет один байт, представленный десятичной записью от 0 до 255. IP-адрес является иерархическим, и когда мы читаем его слева направо, мы получаем все более определенную информацию о том, в каком месте Интернета располагается хост (то есть внутри какой подсети иерархической структуры сетей). Точно так же, когда мы смотрим на обычный почтовый адрес, мы получаем подробную информацию о расположении, которое этот адрес описывает.

2.5.1. Службы, предоставляемые DNS

Мы только что видели, что существует два способа идентификации хостов — по имени и по IP-адресу. Люди, например, используют мнемонические имена, которые легко запомнить, в то время как для маршрутизаторов предпочтительнее структурированные, имеющие фиксированную длину адреса. Чтобы увязать эти предпочтения, нужна какая-то служба, которая бы транслировала имена хостов в IP-адреса. Эту задачу выполняет **система доменных имен Интернета** (Domain Name System, **DNS**). DNS — это, во-первых, распределенная база данных, реализованная с помощью иерархии **DNS-серверов**, а во-вторых — протокол прикладного уровня, позволяющий хостам обращаться к этой базе данных. Часто в качестве DNS-серверов используются компьютеры с операционной системой Unix и с программным обеспечением BIND (Berkeley Internet Name Domain)[53]. Протокол DNS работает поверх UDP и использует порт 53.

DNS используется другими протоколами прикладного уровня — включая HTTP, SMTP и FTP — для транслирования переданных пользователем имен хостов в IP-адреса. В качестве примера рассмотрим, что происходит, когда браузер (то есть клиент HTTP), запущенный на хосте некоторого пользователя, запрашивает URL-адрес `www.someschool.edu/index.html`. Для того чтобы пользовательский хост был способен отправить HTTP-запрос веб-серверу `www.someschool.edu`, он должен сначала получить его IP-адрес. Это происходит следующим образом:

1. На этом хосте пользователя запускается клиентская часть приложения DNS.
2. Браузер извлекает имя сервера `www.someschool.edu` из URL-адреса и передает его клиентской части приложения DNS.
3. DNS-клиент отправляет запрос, содержащий это имя, серверу DNS.
4. DNS-клиент получает ответ от сервера, содержащий IP-адрес, который соответствует имени запрашиваемого веб-сервера.
5. После получения IP-адреса от клиента DNS браузер может инициировать TCP-соединение по порту 80 с процессом HTTP-сервера, находящегося по указанном адресу.

Мы видим из этого примера, что DNS вызывает дополнительные задержки для использующих его приложений (иногда очень значительные). К счастью, как мы обсудим ниже, желаемый IP-адрес очень часто уже содержится в кэше ближайшего DNS-сервера, и это уменьшает как средние задержки, так и общий сетевой трафик, вызванный DNS-запросами.

Кроме трансляции имен хостов в адреса, DNS предоставляет еще некоторые важные службы:

- **Назначение псевдонимов хостам.** Хосты с довольно сложными именами могут иметь один или несколько псевдонимов. Например, хост `relay1.west-coast.enterprise` может иметь псевдонимы, скажем, `enterprise.com` и `www.enterprise.com`. В этом случае имя `relay1.west-coast.enterprise` называют **каноническим именем хоста**. В отличие от канонических имен псевдонимы обычно более лаконичны. Приложения могут запрашивать DNS-сервер, чтобы получить не только IP-адреса, но и каноническое имя того или иного сервера по его псевдониму.

ПРИНЦИПЫ В ДЕЙСТВИИ

DNS: Важные сетевые функции с помощью клиент-серверной модели

Подобно протоколам HTTP, FTP и SMTP, DNS является протоколом прикладного уровня, так как, во-первых, он работает между взаимодействующими конечными системами, используя клиент-серверную модель, а, во-вторых, основывается на нижележащем транспортном протоколе, чтобы передавать DNS-сообщения между этими конеч-

ными системами. Однако, с другой стороны, роль DNS отличается от веб-приложений, приложений передачи файлов или электронной почты. С DNS, в отличие от всех них, пользователь не взаимодействует напрямую. DNS обеспечивает одну из ключевых функций Интернета, а именно трансляцию имен хостов в их IP-адреса для пользовательских приложений или другого программного обеспечения Интернета. Мы отмечали в разделе 1.2, что в архитектуре Интернета наиболее сложна периферия сети. Система DNS, которая реализует важный процесс преобразования имени в адрес, используя клиенты и серверы, размещенные на границе сети — это еще один пример такой клиент-серверной философии.

- **Назначение псевдонимов почтовому серверу.** По вполне понятным причинам крайне желательно, чтобы адреса электронной почты были наглядными. Например, если у Боба есть учетная запись на сервере Hotmail, то его адрес может выглядеть достаточно просто: `bob@hotmail.com`. Однако имя хоста почтового сервера Hotmail может быть достаточно сложным, не совсем удобным для чтения (например, когда каноническим именем хоста является что-то вроде `relay1.west-coast.hotmail.com`). Поэтому, почтовые приложения также могут использовать псевдонимы почтовых серверов для запросов DNS. В действительности записи MX (см. ниже) позволяют сделать так, чтобы почтовый сервер и веб-сервер какой-либо компании имели идентичные имена хостов (псевдонимы); например, веб-сервер и почтовый сервер могут оба называться `enterprise.com`.
- **Распределение нагрузки.** Система DNS может использоваться также для того, чтобы осуществлять распределение нагрузки между дублирующими серверами, например реплицированными веб-серверами. Достаточно нагруженные сайты, такие как, например, `cnn.com`, реплицируются на несколько серверов, запущенных каждый на своей конечной системе и имеющих различные IP-адреса. Для дублированных (реплицированных) веб-серверов *набор* IP-адресов привязан к одному каноническому имени хоста. Этот набор содержится в базе данных сервера DNS. Когда клиент делает запрос DNS-серверу по имени, привязанному к набору адресов, сервер в ответ выдает весь набор IP-адресов, но с каждым ответом делает ротацию их порядка. Так как клиент обычно отправляет HTTP-запрос к первому из списка адресов, то такая DNS-ротация помогает распределять трафик среди реплицированных серверов. Такая же DNS-ротация может использоваться и для почтовых серверов, которые тоже иногда имеют

одинаковые псевдонимы. Компании-поставщики контента, такие как, например, Akamai, могут использовать DNS более изощренными способами[134] для предоставления распределенного веб-контента (см. главу 7).

Протокол DNS, описанный в документах RFC 1034 и RFC 1035, обновлен в нескольких дополнительных документах RFC. На самом деле DNS — это сложная система, и мы затрагиваем здесь только ключевые аспекты ее работы. Заинтересованные читатели могут обратиться к документам RFC, к книге Альбица[4], а также к старым источникам, в которых прекрасно описана работа DNS[346, 347].

2.5.2. Как работает DNS

Давайте представим краткий обзор того, как же работает DNS. Рассмотрим трансляции имени хоста в IP-адрес.

Предположим, что некоторое приложение (веб-браузер или почтовый клиент), запущенное на пользовательском хосте, требует трансляции имени хоста в IP-адрес. Приложение запускает клиентскую часть DNS, указывая имя хоста, которое нужно преобразовать. (На многих Unix-компьютерах для осуществления такой трансляции приложения вызывают команду `gethostbyname()`.) После этого DNS пользовательского хоста отправляет сообщение-запрос в сеть. Все DNS-запросы и ответы пересылаются внутри UDP-дейтаграмм на порт 53. После некоторой задержки, варьируемой от миллисекунд до секунд, DNS пользовательского хоста получает ответное сообщение. Полученное сообщение с информацией по соответствию имени и адреса затем передается приложению. То есть, с точки зрения запущенного на пользовательском хосте приложения, DNS — это некоторый черный ящик, предоставляющий простую службу трансляции. Но в действительности этот черный ящик достаточно сложен и состоит из большого набора DNS-серверов, распределенных по всему миру, а также протокола прикладного уровня, который определяет порядок взаимодействия DNS-серверов с запрашивающими хостами.

Самой простейшей схемой построения для службы DNS мог бы быть единственный DNS-сервер, содержащий все соответствия адресов и имен. С помощью такого централизованного сервера клиенты просто бы направляли все запросы в одно место, и DNS-сервер отвечал бы им напрямую. Несмотря на то что такая простота очень привлекательна,

в сегодняшнем Интернете с его огромным (и растущим) количеством хостов это сделать фактически невозможно. При такой централизованной организации службы могут возникать следующие проблемы:

- **Единая точка отказа.** Если DNS-сервер вышел из строя, это нарушает работу всего Интернета.
- **Объем трафика.** Единственный DNS-сервер должен обрабатывать все DNS-запросы (все HTTP-запросы и сообщения электронной почты, сгенерированные сотнями миллионов хостов).
- **Удаленность централизованной базы данных.** Единственный DNS-сервер не может располагаться близко ко всем клиентам одновременно. Если мы расположим DNS-сервер в Нью-Йорке, то запросы, например, из Австралии будут идти с другого конца планеты и окажутся, естественно, очень медленными, что приведет к значительным задержкам.
- **Обслуживание.** Единственный DNS-сервер должен будет хранить записи для всех хостов Интернета. Это приведет к необходимости содержать огромную базу данных, которой при этом придется часто модифицироваться, чтобы учитывать каждый появляющийся новый хост.

И наконец, централизованная база данных, расположенная на единственном DNS-сервере, *не может быть масштабирована*. Таким образом, в силу всех этих причин служба DNS строится как распределенный сервис, и она является прекрасным примером того, как распределенная база данных может быть реализована в Интернете.

Распределенная иерархическая база данных

Для того чтобы иметь возможность масштабирования, DNS служба использует огромный набор серверов, которые организованы в иерархическую структуру и распределены по всему миру. Не существует единственного DNS-сервера, содержащего таблицы соответствия для всех хостов Интернета. Вместо этого информация о соответствии адресов именам распределена по DNS-серверам. В первом приближении существует три класса серверов DNS — корневые DNS-серверы, DNS-серверы верхнего уровня и авторитетные (ответственные за свою зону) DNS-серверы. Их иерархическая структура показана на рис. 2.19.

Чтобы понять, как эти три класса серверов взаимодействуют, предположим, что DNS-клиент пытается определить IP-адрес для интернет-

хоста www.amazon.com. При этом должно происходить следующее: сначала клиент делает запрос одному из корневых серверов, который возвращает IP-адреса серверов верхнего уровня домена com. После этого клиент обращается к одному из этих серверов, который, в свою очередь, возвращает IP-адрес авторитетного сервера для amazon.com. Наконец, клиент обращается к одному из авторитетных серверов домена amazon.com, а тот возвращает IP-адрес хоста www.amazon.com.

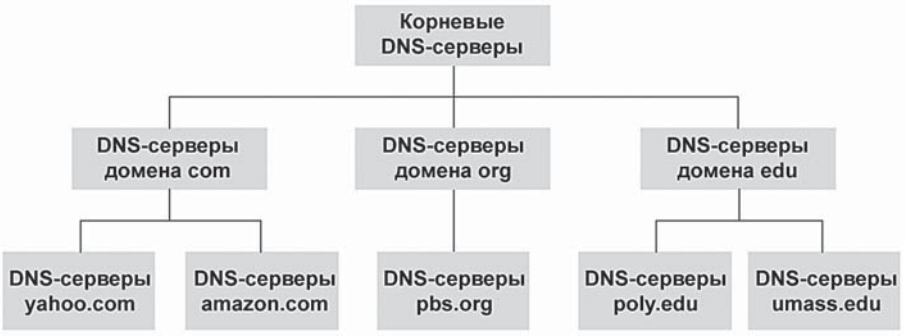


Рис. 2.19. Часть иерархической структуры DNS-серверов

Вскоре мы подробнее изучим процесс поисковых запросов DNS, а пока давайте посмотрим на эти три класса DNS-серверов:

- **Корневые DNS-серверы.** В Интернете существует 13 корневых DNS-серверов (обозначающиеся латинскими буквами от A до M), и большинство из них размещено в Северной Америке. Карта корневых DNS-серверов на 2012 год представлена на рис. 2.20; список текущих корневых серверов можно на сайте Root servers[570]. Хотя мы ссылаемся на каждый из этих 13 серверов, как на отдельный, за каждым из них по соображениям безопасности и надежности стоят несколько реплицированных серверов. На момент осени 2011 года их число составляло 247.
- **DNS-серверы верхнего уровня.** Эти серверы отвечают за домены верхнего уровня, такие как com, org, net, edu, а также gov и национальные домены верхнего уровня, такие как uk, fr, ca, jp, ru и т.д. Серверы домена com обслуживаются компанией Verisign Global Registry Services. Серверы домена edu — компанией Educause. Полный список доменов верхнего уровня можно посмотреть в базе данных IANA[224].
- **Авторитетные DNS-серверы.** Каждая организация, имеющая публично доступные хосты (веб-серверы или почтовые серверы)

в Интернете, должна предоставить также доступные DNS-записи, которые сопоставляют имена этих хостов с IP-адресами и которые находятся на авторитетном DNS-сервере. Организация может хранить эти записи либо на своем авторитетном DNS-сервере, либо на авторитетном DNS-сервере своего Интернет-провайдера. Очень многие университеты и крупные компании реализуют и обслуживают свои собственные первичные и вторичные (резервные) DNS-серверы.

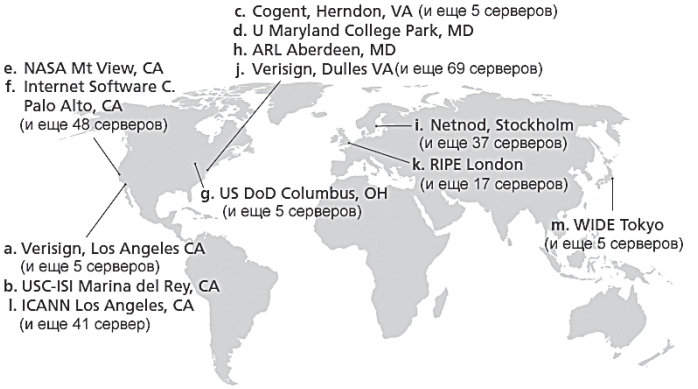


Рис. 2.20. Корневые DNS-серверы на 2012 год (имя, организация, расположение)

Корневые, DNS-серверы верхнего уровня и авторитетные серверы — все принадлежат иерархии DNS-серверов, как показано на рис. 2.19. Но существует еще один важный тип DNS-серверов, называемый **локальным DNS-сервером**. Он не обязательно принадлежит иерархии серверов, но, тем не менее, его роль в структуре DNS очень важна. Каждый Интернет-провайдер — верхнего или нижнего уровня — имеет локальный DNS-сервер (так же называемый сервером имен по умолчанию). При подключении любого хоста к Интернету провайдер предоставляет владельцу хоста IP-адреса одного или нескольких своих локальных DNS-серверов (обычно посредством службы DHCP, обсуждаемой в главе 4). Вы можете легко определить IP-адрес вашего локального DNS-сервера, просто проанализировав состояние сетевого соединения в Windows или Unix. Локальный DNS-сервер для хоста — это обычно ближайший к нему сервер. Он может даже находиться в одной локальной сети с хостом. Для сетей доступа он обычно находится не более чем через несколько маршрутизаторов от пользовательского хоста. Когда хост производит DNS-запрос, этот запрос, в первую очередь, передается

на локальный DNS-сервер, который действует как прокси-сервер, перенаправляя запрос в иерархию DNS-серверов. Далее мы обсудим это детально.

Давайте рассмотрим простой пример. Предположим, что хост `cis.poly.edu` желает получить адрес хоста `gaia.cs.umass.edu`. Также предположим, что для первого хоста DNS-сервер называется `dns.poly.edu`, а авторитетный DNS-сервер для `gaia.cs.umass.edu` называется `dns.umass.edu`. Как показано на рис. 2.21, сначала хост `cis.poly.edu` отправляет сообщение с DNS-запросом к своему локальному DNS-серверу, а именно `dns.poly.edu`. В данном запросе содержится имя хоста, которое нужно преобразовать в адрес, а именно `gaia.cs.umass.edu`. Локальный DNS-сервер перенаправляет сообщение-запрос на корневой DNS-сервер. Корневой DNS-сервер извлекает из записи суффикс .edu и возвращает локальному DNS-серверу список IP-адресов для серверов верхнего уровня, ответственных за домен `edu`. После этого локальный DNS-сервер перенаправляет запрос к одному из этих серверов верхнего уровня. Сервер верхнего уровня берет суффикс `umass.edu` и отвечает сообщением, содержащим IP-адрес авторитетного DNS-сервера Массачусетского университета, а именно `dns.umass.edu`. Наконец, локальный DNS-сервер отсылает запрос непосредственно к `dns.umass.edu`, который возвращает ответ с IP-адресом нужного хоста `gaia.cs.umass.edu`. Заметим, что в данном примере для того, чтобы получить таблицу соответствия для одного имени хоста, отправляются 8 DNS-сообщений: 4 сообщения-запроса и 4 ответных! Вскоре мы увидим, каким образом такой трафик запросов можно уменьшить с помощью DNS-кэширования.

В нашем предыдущем примере предполагалось, что DNS-сервер верхнего уровня хранит информацию об авторитетном DNS-сервере для имени целевого хоста. В общем случае, это не всегда верно. Обычно, DNS-сервер верхнего уровня может иметь информацию только о промежуточном DNS-сервере, которому, в свою очередь, уже известен авторитетный DNS-сервер для имени целевого хоста. Например, предположим опять, что в Университете Массачусетса есть DNS-сервер, называемый `dns.umass.edu`. Еще предположим, что в каждом из отделений университета есть свой DNS-сервер, являющийся авторитетным для всех хостов своего отделения. В таком случае, когда промежуточный DNS-сервер `dns.umass.edu` получает запрос для хоста с именем, оканчивающемся на `cs.umass.edu`, он возвращает серверу `dns.poly.edu` IP-адрес сервера `dns.cs.umass.edu`, который является авторитет-

ным для всех имен хостов, оканчивающихся на `cs.umass.edu`. Затем локальный сервер `dns.poly.edu` отправляет запрос авторитетному DNS-серверу, который возвращает желаемую таблицу соответствия локальному DNS-серверу, а тот передает ее запрашивающему хосту.

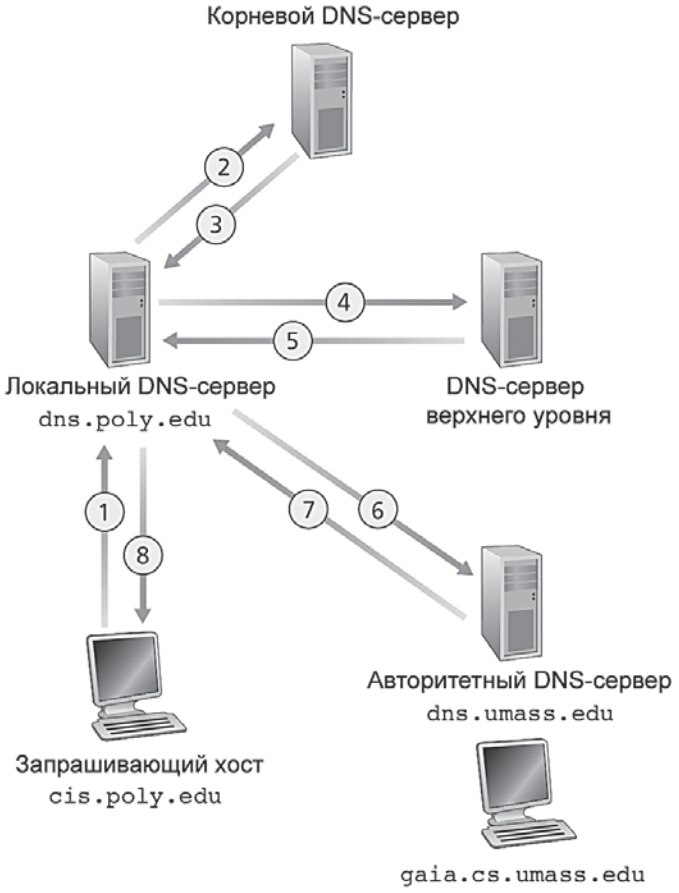


Рис. 2.21. Взаимодействие различных DNS-серверов

В этом случае будет отправлено не 8, а уже 10 DNS-сообщений! Пример, показанный на рис. 2.21, использует как **рекурсивный**, так и **итеративный запросы**. Запрос, отправленный с `cis.poly.edu` к серверу `dns.poly.edu`, является рекурсивным, так как `dns.poly.edu` получает информацию по поручению `cis.poly.edu`. Остальные три запроса являются итеративными, так как все три ответа непосредственно возвращаются к тому, кто запрашивал, а именно к `dns.poly.edu`.

Теоретически любой DNS-запрос может быть либо итеративным, либо рекурсивным. Например, рис. 2.22 показывает цепочку, в которой все запросы рекурсивные. На практике обычные запросы используют схемы, представленные на рис. 2.21: запрос от хоста к локальному DNS-серверу является рекурсивным, а все остальные запросы — итеративными.

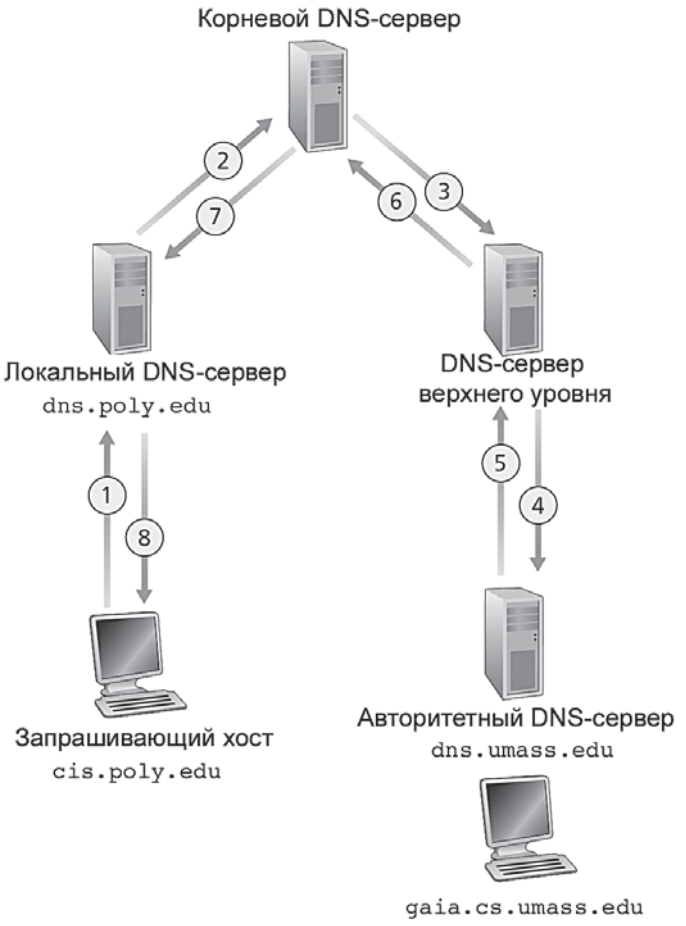


Рис. 2.22. Рекурсивные запросы DNS

DNS-кэширование

Мы не обсудили очень важную часть системы DNS — **DNS-кэширование**. DNS-кэширование, на самом деле, очень широко используется, чтобы улучшить производительность и уменьшить число DNS-запро-

сов, гуляющих по сети. Принцип кэширования достаточно прост. На каком-то из этапов цепочки запросов, когда DNS-сервер получает ответ (содержащий, например, таблицу соответствий имен IP-адресам), он может сохранить эту таблицу в своей локальной памяти (кэше). Например, для рис. 2.21 каждый раз, когда локальный DNS-сервер `dns.poly.edu` получает ответ от какого-либо другого DNS-сервера, он может кэшировать эту информацию. Если DNS-серверу поступает еще один запрос для того же самого имени хоста, DNS-сервер может предоставить желаемый IP-адрес, хранящийся у него в кэше, даже если он не является авторитетным сервером для данного имени хоста. Так как имена хостов и IP-адреса необязательно бывают постоянными, DNS-серверы обычно сбрасывают информацию в своем кэше через какой-то период времени (в некоторых случаях это составляет два дня).

В качестве примера предположим, что хост `apricot.poly.edu` запрашивает у сервера `dns.poly.edu` IP-адрес хоста `cnn.com`. Затем, спустя несколько часов, еще один хост Политехнического университета, скажем, `kiwi.poly.edu` запрашивает у `dns.poly.edu` то же самое имя хоста. В этом случае DNS-сервер может немедленно возвратить IP-адрес хоста `cnn.com` из своего кэша, не обращаясь к другим DNS-серверам. DNS-серверы способны также хранить в своем кэше IP-адреса DNS-серверов верхнего уровня, поэтому они могут очень часто выступать в качестве корневых DNS-серверов в цепочке запросов.

2.5.3. Записи и сообщения DNS

Серверы DNS образуют распределенную базу данных, где хранятся **ресурсные записи**, включая те, которые содержат таблицы соответствия имен IP-адресам. Каждый DNS-ответ несет в себе одну или более ресурсных записей. Далее мы рассмотрим кратко ресурсные записи DNS; более подробную информацию по ним можно найти в книге Альбица[4] или в документах RFC 1034[430] и RFC 1035[431].

Ресурсная запись представляет собой набор из четырех полей:

```
(Имя, Значение, Тип, Время жизни)
```

Время жизни определяет, когда ресурсная запись должна быть удалена из кэша. В примерах ниже мы будем игнорировать поле `Время жизни`. Содержание полей `Имя` и `Значение` зависит от поля `Тип`:

- Если Тип равен A, то в поле Имя содержится имя хоста, а в поле Значение — IP-адрес этого хоста. То есть запись типа A предоставляет стандартную таблицу преобразования имени в IP-адрес, например: (relay1.bar.foo.com, 145.37.93.126, A). Это ресурсная запись типа A.
- Если Тип равен NS, то поле Имя является значением домена (например, foo.com), а поле Значение содержит имя хоста авторитетного DNS-сервера, который отвечает за получение IP-адресов соответствующего домена. Такой тип записи используется, чтобы направлять DNS-запросы дальше по цепочке. Например, запись вида (foo.com, dns.foo.com, NS) является записью типа NS.
- Если Тип равен CNAME, то поле Значение содержит каноническое имя для псевдонима, который находится в поле Имя. Такая запись обеспечивает запросы канонического имени для определенного имени хоста. Пример такой записи: (foo.com, relay1.bar.foo.com, CNAME).
- Если Тип равен MX, то в поле Значение содержится каноническое имя почтового сервера, имеющего псевдоним, находящийся в поле Имя. Пример такой записи: (foo.com, mail.bar.foo.com, MX). Такие записи позволяют почтовым серверам иметь простые псевдонимы. Отметим, что, с помощью записи MX, компании могут использовать одинаковые псевдонимы для почтовых серверов и для веб-серверов. Чтобы получить каноническое имя почтового сервера DNS-клиент должен будет запрашивать запись MX, а если требуется каноническое имя другого сервера — запись CNAME.

Если DNS-сервер является авторитетным для какого-то определенного имени хоста, то он содержит запись типа A для этого имени хоста (он может содержать запись типа A в своем кэше, даже если не является авторитетным). Если сервер не является авторитетным для имени хоста, тогда он будет содержать запись типа NS для домена, который включает это имя хоста; а также запись типа A, в которой находится IP-адрес DNS-сервера, содержащегося в поле Значение записи типа NS. В качестве примера предположим, что DNS-сервер верхнего уровня домена edu не является авторитетным для хоста gaia.cs.umass.edu. В таком случае этот сервер будет содержать запись для домена, который включает хост gaia.cs.umass.edu, например, (umass.edu, dns.umass.edu, NS). Он также будет иметь у себя запись типа A, которая содержит соответствие адреса DNS-сервера dns.umass.edu IP-адресу, например, (dns.umass.edu, 128.119.40.111, A).

DNS сообщения

Существуют только два типа DNS-сообщений — DNS-запрос и DNS-ответ. Мы уже упоминали эти понятия ранее в разделе. Оба типа имеют одинаковый формат, представленный на рис. 2.23. Значения различных полей DNS-сообщений следующие:

- Первые 12 байт являются *секцией заголовка* и содержат несколько полей. В первом из этих полей находится 16-разрядное число, идентифицирующее запрос. Этот идентификатор копируется в ответное сообщение, позволяя клиенту сопоставить полученный отклик с отправленным запросом. В поле «флаги» находится несколько флагов. Однобитный флаг «запрос/ответ» указывает, является ли сообщение запросом (0) или ответом (1). Однобитный флаг «авторитетный ответ» устанавливается в ответном сообщении, если DNS сервер является авторитетным для запрашиваемого хоста. Еще один однобитный флаг «требуется рекурсия» устанавливается, когда клиент (хост или другой DNS-сервер) требует от сервера обработать этот запрос самому (рекурсивный запрос). Однобитный флаг «рекурсия возможна» устанавливается в 1 в отклике DNS-сервера в случае, если он поддерживает рекурсию. В заголовок также включены 4 числовых поля, указывающих количество секций данных четырех различных типов, которые идут после заголовка.
- *Секция запросов* содержит информацию о выполняемом запросе, которая включает, во-первых, поле имени, содержащее запрашиваемое имя хоста, и, во-вторых, поле типа, которое указывает тип запроса, например, адрес, связанный с именем (тип A) или имя почтового сервера (тип MX).
- В ответе DNS-сервера *секция откликов* содержит ресурсные записи для запрашиваемых имен. Напомним, что каждая ресурсная запись содержит тип (например, A, NS, CNAME и MX), значение и время жизни. DNS-ответ способен возвращать несколько ресурсных записей, так как имя хоста может быть привязано не к одному IP-адресу, например, для реплицированных веб-серверов, обсуждаемых ранее.
- Секция *серверов имен* содержит записи других авторитетных серверов.
- Секция *дополнительной информации* включает другие полезные записи. Например, в отклике на запрос MX содержится ресурсная запись, представляющая каноническое имя хоста почтового сервера. В секции дополнительной информации будет содержаться запись

типа A, в которой находится IP-адрес для канонического имени почтового сервера.

Рис. 2.23. Формат сообщений DNS

Каким же образом вы можете отправить DNS-запрос непосредственно с вашего рабочего компьютера какому-либо DNS серверу? Это легко сделать, используя **программу *nslookup***, доступную как на Windows, так и на UNIX-платформах. Например, на хосте с операционной системой Windows откройте командную строку и просто наберите *nslookup*. После этого вы можете отправить DNS-запрос любому DNS-серверу (корневому, авторитетному или серверу верхнего уровня). После получения ответа nslookup отобразит включенные в него записи (в читаемом формате). В качестве альтернативы вы можете использовать какой-либо вебсервер, позволяющий удаленно выполнять процедуру nslookup (вы можете найти его в поисковых машинах). Лабораторная Wireshark в конце этой главы поможет вам в более подробном изучении службы DNS.

Добавление записей в базу данных DNS

Выше мы обсудили, каким образом вызываются записи из базы данных DNS. Вам, должно быть, интересно, как они в эту базу попадают. Давайте посмотрим, как это делается, на примере. Предположим, вы создали новую компанию, которая называется Network Utopia. Первым делом вы захотите зарегистрировать доменное имя `networkutopia.com` в компании-регистраторе. Регистратор — это коммерческая организация, которая проверяет уникальность доменных имен, добавляет их в базу данных DNS и берет за свои услуги определенную плату. До

1999 года единственным регистратором для доменов com, net и org была компания Network Solutions. В настоящие дни множество компаний-регистраторов конкурируют между собой. Они должны получать аккредитацию в *корпорации по управлению доменными именами и IP-адресами* (*Internet Corporation for Assigned Names and Numbers, ICANN*). Полный список аккредитованных регистраторов можно найти на сайте **www.internic.net**.

Когда вы регистрируете домен `networkutopia.com` у какого-то регистратора, вам необходимо предоставить ему имена и IP-адреса ваших первичного и вторичного DNS-серверов. Предположим, что эти адреса и имена таковы: `dns1.networkutopia.com`, `dns2.networkutopia.com`, `212.212.212.1` и `212.212.212.2`. Регистратор должен убедиться, что для этих двух авторитетных DNS-серверов записи типа NS и типа A добавлены в DNS-серверы верхнего уровня домена com. Например, для первичного авторитетного сервера `networkutopia.com` регистратор добавит следующие две ресурсные записи в систему DNS:

```
(networkutopia.com, dns1.networkutopia.com, NS)

(dns1.networkutopia.com, 212.212.212.1, A)
```

Вы также должны убедиться, что в ваши авторитетные DNS-серверы добавлены ресурсные записи для вашего веб-сервера `www.networkutopia.com` (запись типа A) и для вашего почтового сервера `mail.networkutopia.com` (запись типа MX). До недавнего времени содержимое каждого из DNS-серверов настраивалось статически с помощью конфигурационного файла, создаваемого администратором вычислительной сети. Не так давно появилась опция в протоколе DNS для обновления базы данных через DNS-сообщения. Документы RFC 2136[467] и RFC 3007[489] описывают приемы динамических обновлений DNS.

О БЕЗОПАСНОСТИ

Уязвимости DNS

Мы узнали, что служба DNS является одним из важнейших компонентов инфраструктуры Интернета. Многие важные службы — в том числе веб-приложения и электронная почта — просто не в состоянии функционировать без DNS. Поэтому возникает естественный вопрос, может ли DNS подвергнуться нападению? Является ли служба DNS беспомощной мишенью, ожидающей, когда выведут из строя ее и вместе с ней большинство интернет-приложений?

Первый тип атаки, который приходит на ум, это DDoS-атака на сервер DNS с переполнением полосы пропускания канала (см. раздел 1.6). Например, злоумышленник может попытаться отправить каждому корневому DNS-серверу огромный поток пакетов — настолько значительный, что большинство законных DNS-запросов никогда не будут обработаны. Такая масштабная DDoS-атака на корневые серверы DNS, в действительности, имела место 21 октября 2002 года, когда злоумышленники использовали ботнет для отправки огромного потока ICMP-сообщений каждому из 13 корневых серверов DNS. (Сообщения ICMP обсуждаются в главе 4. На данный момент достаточно знать, что ICMP-пакеты — это специальные виды IP-дейтаграмм.) К счастью, это крупномасштабное нападение привело к минимальным последствиям и мало или вообще никак не отразилось на работе пользователей Интернета. Нападавшие смогли направить большой поток пакетов на корневые DNS-сервера, но многие из них были защищены пакетными фильтрами, настроенными на блокировку ICMP-трафика. В результате они успешно перенесли атаку и продолжили функционировать в нормальном режиме. Помогло и то, что большинство локальных серверов DNS кэшируют IP-адреса серверов верхнего уровня и заменяют собой корневые серверы при запросах.

Потенциально более эффективной DDoS-атакой против DNS была бы отправка потока DNS-запросов к серверам верхнего уровня, например всем обрабатывающим домен .com, так как, во-первых, DNS-запросы сложнее фильтровать, а во-вторых, для локальных серверов DNS обрабатывать запросы за серверы верхнего уровня сложнее, чем за корневые. Но и в этом случае кэширование в локальных серверах DNS значительно бы уменьшило тяжесть последствий.

Служба DNS может быть атакована и другими способами. При атаке «человек посередине» (MITM-атака) злоумышленник перехватывает запросы от хостов и возвращает фиктивные ответы. В такой разновидности атаки, как «отравление DNS», злоумышленник посылает поддельные ответы DNS-серверу, заставляя сервер принять фиктивные записи в свой кэш. Любой из этих двух видов атак может быть использован для перенаправления ничего не подозревающего пользователя на веб-сайт злоумышленника. Эти атаки, однако, достаточно трудно осуществить, так как они требуют перехвата пакетов или «удушения» (снижения пропускной способности) серверов[603].

Другой важной разновидностью является не атака на службу DNS как таковая, а использование инфраструктуры DNS для проведения DDoS-атаки против целевого хоста (например, почтового сервера

вашего учебного заведения). При этом атакующий передает DNS-запросы на многие авторитетные серверы DNS, и в каждом запросе подставляет адрес целевого хоста в качестве поддельного источника запроса. DNS-серверы затем отправляют свои ответы непосредственно на целевой хост. Если запросы построить таким образом, чтобы размер ответа был гораздо больше (в байтах) размера самого запроса (так называемое усиление DNS), то злоумышленник может потенциально подавить хост-жертву с помощью трафика от DNS-северов без необходимости генерировать свой собственный. Такие атаки с использованием «DNS-отражения» имели некоторый ограниченный успех[345].

Можно сделать вывод, что система DNS продемонстрировала себя на удивление устойчивым к атакам сервисом. На сегодняшний день не было совершено ни одной атаки, которая бы парализовала работу этой службы.

Атаки с использованием «DNS-отражения» можно рассматривать лишь как угрозу определенной конфигурации DNS-серверов.

После того как все эти шаги будут завершены, пользователи Интернета смогут посещать ваш веб-сайт и отправлять электронную почту сотрудникам вашей компании. Давайте закончим обсуждение DNS, проверив, что это действительно так. Эта проверка также поможет закрепить наши знания о DNS. Предположим, Алиса, находясь в Австралии, хочет посмотреть во Всемирной паутине страницу **www.networkutopia.com**. Как уже говорилось ранее, ее хост сначала отправит DNS-запрос ее локальному серверу DNS. Локальный сервер DNS, в свою очередь, свяжется с сервером верхнего уровня домена com. (Локальный сервер DNS обратится к корневому DNS-серверу, если в своем кэше не обнаружит адрес сервера верхнего уровня.) Этот сервер верхнего уровня содержит ресурсные записи типов NS и A, перечисленные выше, поскольку регистратор добавил эти записи во все серверы верхнего уровня домена com. Он отправляет их в ответном сообщении локальному DNS-серверу Алисы. После этого локальный сервер DNS отправляет сообщение серверу с адресом `212.212.212.1`, запрашивая запись типа A, соответствующую имени **www.networkutopia.com**. Эта запись предоставляет IP-адрес нужного веб-сервера, скажем, `212.212.71.4`, который локальный сервер DNS передает обратно хосту Алисы. Теперь браузер Алисы может инициировать TCP-соединение с хостом `212.212.71.4` и отправлять запрос HTTP через это соединение. Вот так! Когда кто-то путешествует по сети, кроме видимых глазу событий в его браузере, за кадром происходит еще много интересных вещей!

2.6. Одноранговые приложения

Приложения, описанные в этой главе до сего момента — в том числе Всемирная паутина, электронная почта и DNS — все используют клиент-серверную архитектуру, опираясь значительным образом на всегда доступные сервера. Напомним из раздела 2.1.1, что в случае с одноранговой архитектурой (P2P) серверы либо мало используются, либо их совсем нет. Вместо этого пары хостов, называемых пирами (одноранговыми узлами), общаются друг с другом напрямую, причем подключаются они не обязательно постоянно, а периодически, а их владельцы — не провайдеры услуг, а обычные пользователи ноутбуков и настольных компьютеров.

В этом разделе мы рассмотрим два различных типа приложений, которые особенно хорошо подходят для построения одноранговых систем. Первый из них — это файловый обмен (раздача файлов), когда приложение раздает файл от одного источника к большому числу узлов. Это приложение прекрасно демонстрирует самомасштабируемость одноранговой архитектуры. В качестве конкретного примера мы расскажем о популярной системе BitTorrent. Вторым типом является распределенная по большому количеству узлов база данных. Для этого мы рассмотрим понятие распределенных хеш-таблиц (Distributed Hash Table, DHT).

2.6.1. Одноранговый файлообмен

Мы начинаем наше знакомство с одноранговыми системами с рассмотрения самой распространенной задачи, а именно раздачи большого файла с одного сервера большому числу хостов (так называемых пиров). В качестве раздаваемого файла может выступать новая версия операционной системы Linux, программная заплатка для существующей операционной системы или приложения, музыкальный файл в формате MP3 или видеофайл формата MPEG. При клиент-серверном обмене файлами сервер должен отправить копию файла каждому из клиентов, что увеличивает нагрузку и потребляет значительную часть серверного трафика. В случае с одноранговым обменом каждый узел может перераспределить любую часть файла, которую он уже получил, между любыми другими узлами (пирами), помогая тем самым серверу в выполнении раздачи файла. По состоянию на 2012 год наиболее популярным протоколом одноранговой обмена файлами является BitTorrent, разра-

ботанный Брэмом Коэном (Bram Cohen). В настоящее время существует много различных независимых клиентов, работающих по протоколу BitTorrent, так же как есть ряд браузерных клиентов, которые используют протокол HTTP. В этой части мы сначала рассмотрим самомасштабируемость одноранговой архитектуры в контексте файлового обмена. Затем мы опишем BitTorrent более подробно, выделяя наиболее важные характеристики и особенности данного протокола.

Масштабируемость одноранговой архитектуры

Для сравнения клиент-серверной и одноранговой архитектур и демонстрации самомасштабируемости второй рассмотрим простую количественную модель раздачи файла фиксированному числу узлов для обоих типов архитектуры. Как показано на рис. 2.24, сервер и пиры имеют доступ к Интернету.

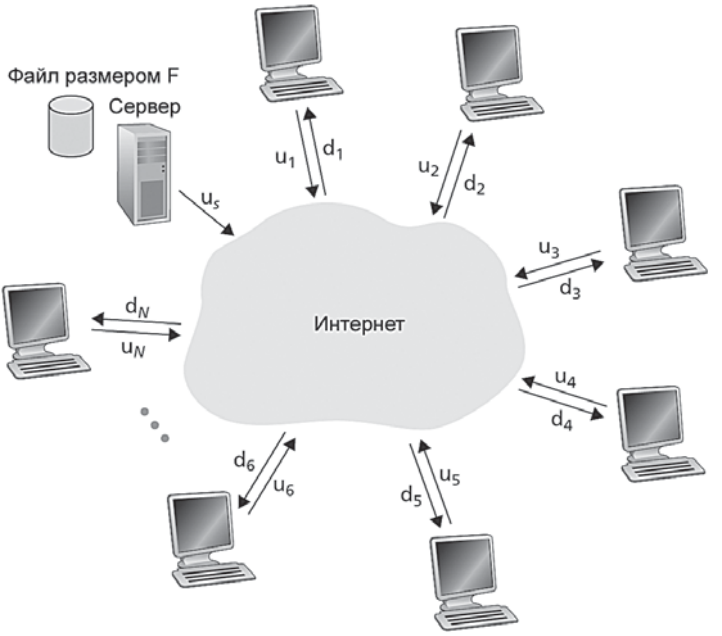


Рис. 2.24. Пример раздачи файла

Обозначим скорость исходящего канала доступа сервера через u_s, скорость исходящего канала доступа i-го пира как u_i, а скорость загрузки для i-го пира — d_i. Также обозначим через F размер раздаваемого файла

(в битах), а через N — число пиров, желающих получить копию файла. **Временем раздачи** будет являться время, которое требуется для того, чтобы получить копию файла на всех N узлах. В нашем анализе времени раздачи для обеих архитектур мы делаем упрощающее (вообще говоря, верное[15]) предположение, что ядро Интернета имеет достаточную пропускную способность и все узкие места связаны только с сетями доступа. Мы также предполагаем, что ресурсы сервера и клиентов не задействованы ни в каких других сетевых приложениях, так что вся их пропускная способность полностью отдана на раздачу этого файла.

Давайте сначала определим величину времени раздачи для клиент-серверной архитектуры. Обозначим ее через D_{cs}. В этом случае ни один из пиров не участвует в раздаче файлов. Отметим следующие моменты:

- Сервер должен передавать одну копию файла каждому из N пиров. Таким образом, сервер будет передавать NF бит. Поскольку скорость, с которой сервер отдает данные, равна u_s, время на раздачу файла должно быть как минимум NF/u_s.
- Обозначим через $d_{\min}$ скорость загрузки самого медленного пира, то есть $d_{\min} = \min\{d_1,d_p,...d_n\}$. Пир с минимальной скоростью загрузки не может получить все F бит файла меньше, чем за $F/d_{\min}$ секунд. Таким образом, минимальное время раздачи будет как минимум $F/d_{\min}$.

Учитывая два вышесказанных замечания, мы получаем:

$$D_{CS} \geq \max\left\{\frac{NF}{u_S}, \frac{F}{d_{\min}}\right\}$$

Данная формула описывает нижнюю границу времени раздачи для клиент-серверной архитектуры. В домашних заданиях вас попросят показать, каким образом сервер может организовать передачу данных так, чтобы фактически достичь нижней границы. Поэтому давайте обозначим ее как фактическое время раздачи файла, то есть

$$D_{CS} = \max\left\{\frac{NF}{u_S}, \frac{F}{d_{\min}}\right\} \qquad (2.1)$$

Из уравнения 2.1 мы видим, что при достаточно больших значениях N время раздачи для клиент-серверной архитектуры будет равно NF/u_s. Таким образом, время раздачи увеличивается пропорционально количеству пиров N. Так, например, если число пиров за неделю возрастает тысячекратно от тысячи до миллиона, то время, требуемое на раздачу файла всем пирам, возрастает в тысячу раз.

Теперь давайте проведем аналогичный анализ для одноранговой архитектуры, где каждый пир помогает серверу в раздаче файла. В частности, когда он получает некоторый файл данных, то может использовать возможности своего исходящего канала для раздачи данных другим пирам. Расчет времени раздачи в этом случае немного сложнее, чем для клиент-серверной архитектуры, так как оно зависит от того, каким образом каждый пир раздает части файла всем остальным. Тем не менее простое выражение для минимального времени раздачи можно получить[301]. Сначала отметим следующие моменты:

- В начале раздачи файл хранится только на сервере. Чтобы этот файл был передан всем пирам, сервер должен отправить все биты файла в линию связи по крайней мере 1 раз. Таким образом, минимальное время раздачи будет не менее чем F/u_s (в отличие от клиент-серверной схемы бит, отправленный сервером, не нужно будет отправлять еще раз, поскольку узлы сами могут распределить его между собой).
- Как и в клиент-серверном случае, пир с самой низкой скоростью загрузки не может получить все F бит раздаваемого файла менее чем за $F/d_{\min}$ секунд, что и будет минимальным значением времени раздачи.
- Наконец, отметим, что общая скорость выгрузки всей системы в целом равна сумме скоростей выгрузки сервера и всех пиров, то есть $u_{\text{всего}} = u_s + u_1 + \ldots u_N$. Система должна доставить (выгрузить) F бит каждому из N пиров, таким образом, выгрузив всего NF бит. Это не может быть сделано со скоростью большей, чем $u_{\text{всего}}$. Таким образом, минимальное время раздачи также не может быть более чем $NF/(u_s + u_1 + \ldots u_N)$.

Объединяя эти три наблюдения вместе, получаем значение для времени минимальной раздачи в одноранговой архитектуре, обозначенное D_{P2P}.

$$D_{\text{P2P}} \geq \max \left\{ \frac{F}{u_S}, \frac{F}{d_{\min}}, \frac{NF}{u_S + \sum_{i=1}^{N} u_i} \right\} \qquad (2.2)$$

Уравнение 2.2 определяет нижнюю границу времени минимальной раздачи для одноранговой архитектуры. Получается, если представить, что каждый пир может раздавать бит сразу же, как только получит его, то схема раздачи будет на самом деле достигать этой нижней границы[301]. (В домашнем задании мы рассмотрим этот особый случай.) В реальности же, когда раздаются сегменты файла, а не отдельные биты, уравнение 2.2

будет являться хорошим приближением реального времени минимальной раздачи. Таким образом, примем нижнюю границу, описанную уравнением 2.2 как действительное минимальное время раздачи, то есть

$$D_{\text{P2P}} = \max\left\{\frac{F}{u_s}, \frac{F}{d_{\min}}, \frac{NF}{u_s + \sum_{i=1}^{N} u_i}\right\} \qquad (2.3)$$

На рис. 2.25 представлено сравнение минимальной скорости раздачи для клиент-серверной и одноранговой архитектур, учитывая предположение, что все пиры имеют одинаковую скорость отдачи, равную u. Здесь мы установили значения $F/u = 1$ час, $u_s = 10u$ и $d_{\min} >= u_s$. Таким образом, пир может передать полный файл за 1 час, скорость передачи сервера в 10 раз больше, чем у пира, и (для упрощения) по предположению скорость загрузок у каждого из узлов достаточно высока. Из рис. 2.25 мы видим, что минимальное время раздачи для клиент-серверной архитектуры возрастает линейно и до бесконечности при возрастании количества пиров. Однако для одноранговой архитектуры минимальное время раздачи не только всегда меньше, чем для клиент-серверной, но его значение не превышает одного часа для *любого* числа N пиров. Таким образом, приложения в одноранговой архитектуре могут быть самомасштабируемыми. Такая масштабируемость — это прямое следствие того, что пиры при такой схеме являются одновременно и получателями данных, и их отправителями.

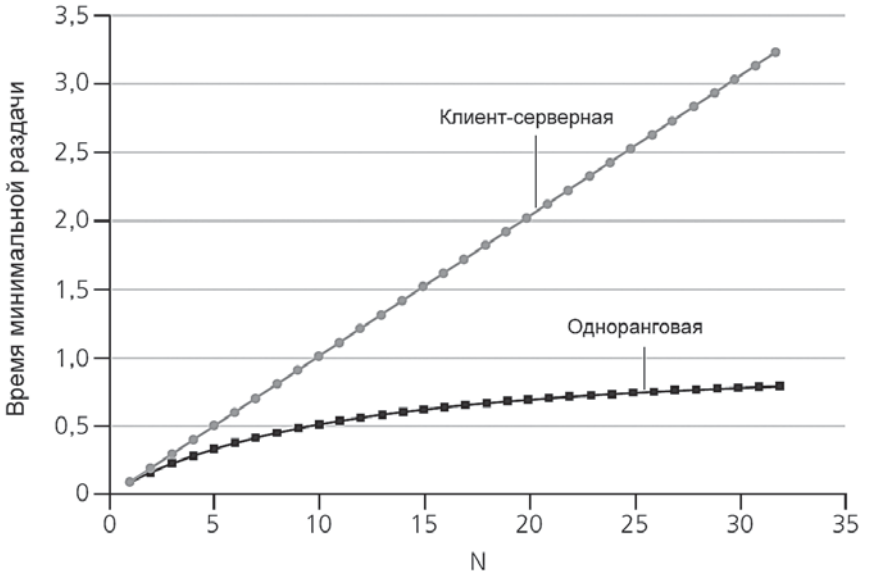


Рис. 2.25. Время раздачи для одноранговой и клиент-серверной архитектур

BitTorrent

BitTorrent является популярным одноранговым протоколом для файлового обмена[87]. В лексиконе данного протокола набор всех пиров, участвующих в раздаче конкретного файла, называется *торрентом*. Пиры в торренте обмениваются друг с другом «*кусками*» (*сегментами*) файла равной длины. Типичный размер сегмента равен 256 кбит. Когда пир впервые присоединяется к торренту, он не имеет сегментов. С течением времени он собирает их больше и больше. Во время загрузки сегментов он может одновременно выгружать их, отдавая другим узлам. После того как пир загрузил файл полностью, он может (эгоистично) покинуть торрент или (бескорыстно) остаться и продолжить раздачу сегментов другим участникам. Кроме того, каждый из участников может покинуть торрент в любой момент времени, загрузив только некоторую часть сегментов, а также впоследствии опять присоединиться к раздаче в любое удобное для него время.

Давайте поближе познакомимся с работой BitTorrent. Так как это довольно сложная система, мы опишем только основные ее механизмы, опуская детали. Каждый торрент имеет инфраструктурный узел, называемый *трекером*. Когда пир присоединяется к торренту, он регистрируется на трекере и периодически информирует его о своем присутствии в раздаче. Трекер отслеживает пиров-участников торрента. Конкретный торрент в определенный момент времени может содержать меньше десяти, а может, более тысячи пиров, участвующих в раздаче.

Как показано на рис. 2.26, когда Алиса в качестве нового участника присоединяется к торренту, трекер случайным образом выбирает последовательность пиров (скажем, 50 штук) из набора участвующих в раздаче и отправляет их IP-адреса Алисе. Обладая списком этих пиров, Алиса пытается установить параллельные TCP-соединения со всеми членами этого списка. Будем называть всех пиров, с которыми Алиса успешно установила TCP-соединения, «соседними» (на рис. 2.26 показано, у Алисы в этой раздаче три соседних пира; обычно их бывает больше). Спустя определенное время некоторые из них могут покинуть торрент, а другие (не входящие в первоначальное число 50) — попытаться установить TCP-соединение с Алисой. Таким образом, соседство узлов будет со временем изменяться.

В конкретный момент времени каждый пир имеет у себя определенный набор сегментов раздаваемого файла, а остальные пиры — другие

наборы сегментов. Алиса периодически запрашивает у своих соседей (через TCP-соединения) список сегментов, которыми они владеют. Если у Алисы L различных соседей по раздаче, то она получит L списков сегментов. Обладая этой информацией, Алиса будет запрашивать (опять же через TCP-соединения) сегменты, которых у нее еще нет.

Таким образом, в любой конкретный момент времени Алиса будет хранить определенный набор сегментов и в то же время знать, какие сегменты находятся у ее соседей. Зная это, Алиса должна будет принять два важных решения. Во-первых, какие сегменты она должна запросить в первую очередь у своих соседей? А во-вторых, кому из ее соседей должна она отправить запрашиваемые сегменты? Чтобы решить, какие сегменты запрашивать, Алиса использует метод, называемый **«сначала редкие»**. Смысл состоит в том, чтобы определить, какие среди тех сегментов, которых у нее нет, являются наиболее редкими у ее соседей (то есть те, которые меньше всего повторяются в соседних пирах), и затем запросить именно их. С помощью такого механизма редкие сегменты начинают раздаваться быстрее, и это приводит к тому, что число копий каждого сегмента в торренте приблизительно уравнивается.

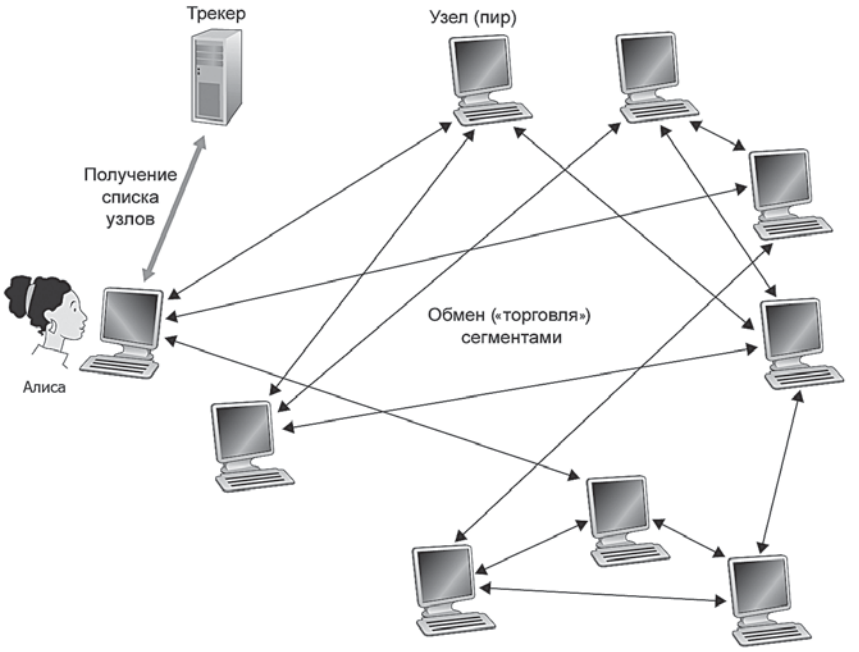


Рис. 2.26. Раздача файлов с помощью BitTorrent

Чтобы определить, кому какие запросы отправить, протокол BitTorrent использует умный «алгоритм торговли». Основная идея его заключается в том, что Алиса отдает предпочтение соседним пирам, которые в настоящий момент поставляют ей данные *на самой высокой скорости*. Конкретнее, для каждого из своих соседей Алиса постоянно измеряет скорость получения от них данных и определяет четырех пиров, которые снабжают ее на самой высокой скорости. Она отвечает им взаимностью и отправляет сегменты этим четырем пирам. Каждые 10 секунд эти скорости пересчитываются, и, возможно, четверка лидирующих узлов меняется. На жаргоне BitTorrent говорят, что эти четыре пира являются **разблокированными**. Важно отметить, что каждые 30 секунд Алиса случайным образом выбирает дополнительного соседа и отправляет ему сегменты файла. Допустим, таким случайным методом был выбран хост Боба. На жаргоне BitTorrent это означает, что Боб **оптимистически разблокирован**. Поскольку Алиса отправляет данные Бобу, она может стать для него одним из четырех лучших поставщиков, и в этом случае Боб тоже начнет отправлять сегменты на хост Алисы. Если скорость передачи достаточно высока, Боб, в свою очередь, тоже может войти в четверку лидеров по поставке сегментов для Алисы. Другими словами, каждые 30 секунд Алиса случайным образом выбирает нового торгового партнера и инициирует с ним торговлю. Если два узла удовлетворены торговлей между собой, они поставят друг друга в свои топсписки и будут продолжать обмениваться данными, пока один из них не найдет лучшего партнера. В конце концов, это приводит к тому, что два узла, способных доставлять данные на совместимых скоростях, как правило, находят друг друга, а способ случайного выбора соседа позволяет подключать к раздаче новый узел и давать ему возможность получать сегменты. Все другие соседние пиры за пределами этой пятерки (четыре топ-узла и один случайный) блокируются (или «удушаются»), то есть они не получают никаких сегментов от Алисы. Протокол BitTorrent насчитывает ряд интересных механизмов, которые здесь не обсуждаются, включая работу с «кусками» (мини-сегментами), конвейеризацию, случайный выбор, режим завершения (endgame) и режим пренебрежения (anti-snubbing)[112].

Описанный выше механизм «торга» по принципу «ты мне, я тебе»[112] стимулирует участников поддерживать раздачу, а не оставаться простыми фрирайдерами (качающими на халяву и не отдающими сегменты другим)[583]. И хотя уже показано, что такую стимулирующую схему можно успешно обходить[322, 326, 393], тем не менее система BitTorrent широко и успешно функционирует, и миллионы узлов по всему миру одновре-

менно обмениваются файлами с другими, являясь участниками сотен тысяч торрентов.

Протокол BitTorrent явился прародителем еще нескольких других одноранговых приложений, таких как PPLive и ppstream (работающих с живым потоком данных).

Интересные варианты реализаций протокола BitTorrent предлагаются в публикациях Гуо[194] и Пятека[393].

2.6.2. Распределенные хеш-таблицы

В этом разделе мы с вами рассмотрим, как можно реализовать простую базу данных в одноранговой сети. Начнем с описания централизованной версии такой простой базы данных, которая будет содержать обычные пары *ключ, значение*. Например, ключи могут являться номерами социального страхования, а значения могут быть соответствующими именами людей; в данном случае пример пары *ключ, значение* — это (156-45-7081, Johnny Wu). Как вариант, ключи могут представлять собой имена контента (фильмов, альбомов, программного обеспечения), а значения — IP-адреса, где хранится данный контент. В данном случае пример пары *ключ, значение* — это (Led Zeppelin IV, 128.17.123.38). Мы делаем запрос к базе данных с помощью ключа. Если в базе данных одна или более пар *ключ, значение* соответствуют запросу, база данных возвращает эти значения. Так, например, если в базе хранятся номера социального страхования и соответствующие им имена людей, мы можем запросить определенный номер, и база возвратит имя человека, которому он принадлежит. В случае если база хранит имена содержимого и соответствующие им IP-адреса, мы запрашиваем имя, а база возвращает IP-адрес, хранящий указанное содержимое, соответствующее этому имени.

Построение такой базы данных — достаточно простая задача, если использовать клиент-серверную архитектуру, при которой все пары *ключ, значение* хранятся на центральном сервере. Но в этом разделе мы рассмотрим, как построить одноранговую версию базы данных, которая будет хранить пары *ключ, значение* на миллионах пиров. Каждый узел в такой системе будет хранить только небольшую часть всех пар *ключ, значение*. Мы разрешим каждому узлу делать запрос к распределенной базе данных с конкретным ключом, после чего та определяет узлы, которые содержат соответствующую пару *ключ, значение* и возвращает эту

пару запрашивающему узлу. Любому узлу также разрешается добавлять новую пару *ключ, значение* в базу данных. Такая распределенная база данных называется **распределенной хеш-таблицей** (distributed hash table, **DHT**).

Перед тем как описать процесс создания распределенной хеш-таблицы, давайте сначала рассмотрим конкретный пример работы DHT в контексте одноранговой файлового обмена. В этом случае, ключом является имя содержимого, а значением — IP-адрес пира, который хранит копию этого содержимого. Так, например, если Боб и Чарли оба имеют по копии последнего дистрибутива Linux, то база данных DHT будет содержать следующие две пары *ключ, значение*: (Linux, IP_{Bob}) и (Linux, $IP_{Charlie}$). В частности, так как DHT распределена по многим пирам, один из них, скажем Дэйв, будет отвечать за ключ «Linux» и хранить у себя соответствующие пары *ключ, значение*. Теперь предположим, что Алиса хочет получить копию Linux. Естественно, ей сначала нужно знать, кто из пиров имеет данную копию дистрибутива перед тем, как она начнет его загрузку. Для этого она запрашивает распределенную хеш-таблицу с помощью ключа «Linux». Хеш-таблица определяет, что Дэйв отвечает за ключ «Linux», и обращается к Дэйву, получает от него пары *ключ, значение* (Linux, IP_{Bob}) и (Linux, $IP_{Charlie}$), и передает их Алисе, после чего она может загружать нужный ей дистрибутив с любого из двух IP — Боба или Чарли.

Вернемся теперь к обобщенной задаче построения распределенной хеш-таблицы для общего случая пар *ключ, значение*. Одним из наивных подходов к построению такой хеш-таблицы было бы случайным образом разбросать пары *ключ, значение* по всем пирам и дать возможность каждому из них обрабатывать список IP-адресов всех узлов, являющихся участниками этой базы данных. При таком построении запрашивающий пир отправляет свой запрос всем участникам, и пиры, которые хранят пары *ключ, значение*, удовлетворяющие ключу запроса, дают ответ. Такой подход абсолютно не масштабируемый, так как требует, чтобы каждый пир не только знал обо всех других (а их, возможно, миллионы!), но еще и делал бы запрос им *всем*.

Теперь опишем элегантный подход к разработке DHT. С этой целью сначала давайте назначим идентификатор каждому пиру — целое число в диапазоне $[0,2^n - 1]$ для некоторого фиксированного n. Заметим, что каждый такой идентификатор может быть выражен n-разрядным представлением. Для каждого ключа потребуем, чтобы он был целым числом в том же диапазоне. Проницательные читатели, конечно, заметили,

что в примере, описанном немного раньше (социальный номер и номер контента) ключи не являются целыми. Чтобы создать целые значения из этих ключей, мы будем использовать хеш-функцию, которая ставит в соответствие каждому ключу, например номеру социального страхования, целое число в диапазоне $[0,2^n - 1]$. Хеш-функция может двум различным входным данным поставить в соответствие одинаковое выходное (одно и то же целое число), но такая вероятность исключительно мала (те, кто не знаком с хеш-функцией, могут обратиться к детальному ее обсуждению в главе 8). Предполагается, что хеш-функция доступна всем пирам в системе, следовательно, когда мы говорим о «ключе», мы имеем в виду хеш оригинального ключа. Поэтому, например, если оригинальный ключ — это «Led Zeppelin IV», то ключ, используемый в распределенной хеш-таблице, будет целым числом, которое равно хешу «Led Zeppelin IV». Вот почему слово «хеш» используется в термине «распределенная хеш-функция».

Обратимся теперь к проблеме хранения пар *ключ, значение* в распределенной хеш-таблице. Основным вопросом здесь является определение правила для присваивания ключей пирам. Учитывая, что каждый пир имеет целочисленный идентификатор и каждый ключ является целым числом в том же диапазоне, естественным подходом было бы присвоить каждую пару *ключ, значение* пиру, имеющему *ближайший* идентификатор к ключу. Для реализации такого подхода, нам нужно определить, что мы будем подразумевать под словом «ближайший», ведь здесь возможны различные варианты. Для удобства давайте определим, что ближайшим будет являться *ближайший следующий* ключ (назовем его *последователем*). Для этого давайте рассмотрим конкретный пример. Предположим, $n = 4$, то есть идентификаторы пиров и ключей находятся в диапазоне [0,15]. Допустим, что есть восемь пиров в системе и их идентификаторы 1, 3, 4, 5, 8, 10, 12 и 15. Предположим также, что мы хотим хранить пару (11, Johnny Wu) в одном из этих восьми пиров. Но в каком из них? Используя нашу договоренность по поводу «ближайшего», так как пир с номером 12 — ближайший последователь для ключа 11, мы сохраняем пару (11, Johnny Wu) в узле 12. Если ключ точно равен одному из идентификаторов, то мы сохраняем пару *ключ, значение* в узле с этим идентификатором; а если ключ больше чем все идентификаторы, мы используем правило сохранения в узле с наименьшим идентификатором.

Предположим теперь, что Алиса хочет добавить пару *ключ, значение* в распределенную хеш-таблицу. Это достаточно просто: сначала она определяет узел, чей идентификатор ближе всех к ключу; затем отправ-

ляет сообщение этому узлу, давая ему команду сохранить пару *ключ, значение*. Но каким образом Алиса определит узел, идентификатор которого является ближайшим к ключу? Если бы она отслеживала все узлы в системе (их идентификаторы и соответствующие IP-адреса), она бы могла определить искомый ближайший узел. Но такой подход требует, чтобы *каждый* из узлов отслеживал *все* другие узлы таблицы — что совсем не подходит для большой системы с миллионами таких узлов.

Циркулярные распределенные хеш-таблицы

Для решения проблемы масштабирования давайте рассмотрим организацию пиров по кругу, когда каждый из них отслеживает только своего предшественника и своего последователя. Пример такой циркулярной таблицы показан на рис. 2.27(а). В этом случае $n = 4$, и у нас те же самые 8 пиров из предыдущего примера. Каждый пир в такой схеме заботится только об информации о своих непосредственных соседях, а именно предшественнике и последователе в круговой диаграмме. Например, пир 5 хранит IP-адреса и идентификаторы пиров 8 и 4, но не обязательно что-то знает о каком-либо другом узле, который может содержаться в распределенной хеш-таблице. Такое круговое размещение пиров — это особый случай **оверлейной (наложенной) сети**. В таких оверлейных сетях узлы формируют логическую сеть, которая расположена как бы поверх компьютерной сети, называемой в таком случае базовой и содержащей физические соединения, маршрутизаторы и хосты. Каналами связи в оверлейных сетях являются не физические, а виртуальные каналы между парами узлов. В оверлейной сети, представленной на рис. 2.27а, находится 8 пиров и 8 оверлейных (виртуальных) соединений; на рис. 2.27б кроме 8 пиров мы видим 16 оверлейных каналов связи. Одно оверлейное соединение обычно использует несколько физических каналов связи и маршрутизаторов базовой сети.

Используя циркулярную оверлейную сеть на рис. 2.27а, предположим теперь, что пир 3 пытается определить, кто в таблице отвечает за ключ 11. Пир 3 создает сообщение типа «Кто отвечает за ключ11?» и отправляет его по часовой стрелке по кругу. Всякий раз, когда какой-либо пир получает это сообщение, он, зная идентификаторы своих предшественников и последователей, может определить, кто из них (либо он сам) должен быть ответственным за запрашиваемый ключ. Если пир не отвечает за ключ, он просто отправляет сообщение дальше, своему последователю. Так, например, когда пир 4 получает сообщение-запрос о ключе 11, он определяет, что не отвечает за ключ (потому что его после-

дователь ближе к этому ключу), поэтому он просто пропускает сообщение дальше, передавая его по кругу пиру 5. Этот процесс продолжается до тех пор, пока сообщение не прибудет к пиру 12, который определяет, что он является ближайшим для ключа 11. В этот момент пир 12 может отправить сообщение обратно запрашивающему пиру 3, указывая, что он (пир 12) отвечает за ключ 11.

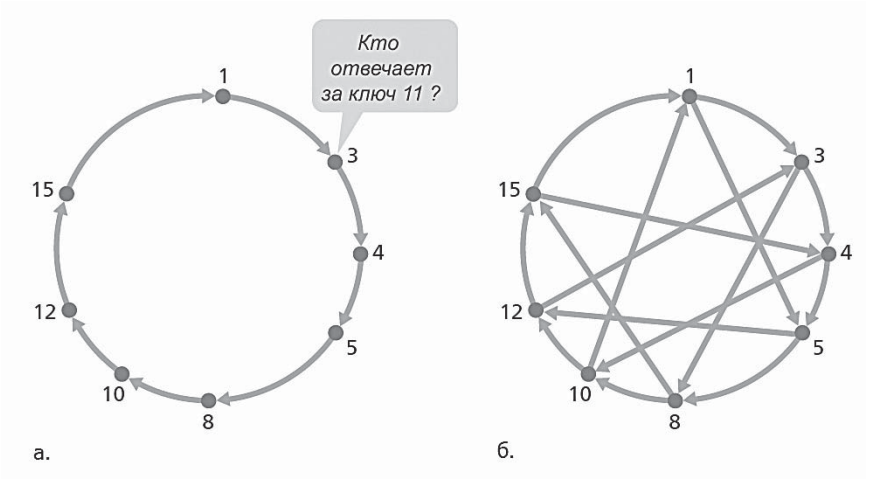


Рис. 2.27 (а). Циркулярная распределенная хеш-таблица. Пир 3 пытается определить, кто отвечает за ключ 11. **(б)**. Циркулярная таблица со ссылками

Циркулярные распределенные хеш-таблицы предлагают очень элегантное решение для уменьшения количества информации, которой должен управлять каждый узел. В частности, узлу необходимо заботиться только о двух своих соседях — непосредственном предшественнике и непосредственном последователе. Но это решение рождает также и новую проблему: чтобы найти узел, ответственный за ключ (в худшем случае), все N узлов хеш-таблицы будут вынуждены перенаправлять сообщения по кругу, и в среднем будет отправлено $N/2$ сообщений.

Таким образом, при построении распределенных хеш-таблиц необходимо найти компромисс между числом соседей, которых должен отслеживать каждый пир, и количеством сообщений, которые требуется отправить, чтобы разрешить один запрос. С одной стороны, если каждый пир отслеживает всех своих соседей, то отправляется только один запрос, но пир должен хранить информацию об N узлах. С другой стороны, с использованием циркулярных таблиц каждый пир хранит

информацию только о двух соседях, но на каждый запрос генерируется в среднем $N/2$ сообщений. К счастью, есть способ оптимизировать таблицы таким образом, что число соседей каждого пира, так же как и число сообщений на запрос, будет оставаться в приемлемых пределах. Например, использовать циркулярную оверлейную сеть в качестве базы с добавлением «ссылок» таким образом, что каждый пир отслеживает не только своих непосредственных предшественника и последователя, но также и сведения об относительно небольшом числе связанных ссылками пиров, распределенных по таблице. Пример такой циркулярной хеш-таблицы со ссылками показан на рис. 2.27б. Ссылки используются для ускорения маршрутизации сообщений запроса, в частности, когда пир получает сообщение с запросом ключа, он перенаправляет сообщение соседнему пиру (своему последователю, либо одному из соседей по ссылкам), который находится ближе к ключу. Так, на рис. 2.27б, когда пир 4 получает запрос ключа 11, он определяет, что ближайшим к ключу среди его соседей является сосед по ссылке 10 и перенаправляет сообщение к нему. Очевидно, что такой механизм ссылок может значительно уменьшить число сообщений, используемых для обработки запроса.

Следующим естественным вопросом будет: «Сколько соседей по ссылке должен иметь каждый пир, и куда эти ссылки должны указывать?» Значительное внимание этой теме было уделено исследовательским сообществом[35, 24]. Было доказано, что распределенные хеш-таблицы могут быть построены таким образом, что число соседей у пира, а также число сообщений на один запрос будет составлять $O(\log N)$, где N — это число пиров. Такая схема построения является удовлетворительным компромиссом между крайними случаями использования сеточной и циркулярно-оверлейной топологии.

Отток пиров

В одноранговых системах пиры могут появляться и исчезать без предупреждения. Таким образом, при построении распределенных хеш-таблиц необходимо позаботиться о механизме реагирования на *отток пиров*. Для более глубокого понимания того, как это можно сделать, давайте еще раз обратимся к циркулярной хеш-таблице на рис. 2.27а. Для обработки оттока пиров нам нужно потребовать, чтобы каждый из них отслеживал (то есть знал IP-адрес) своего первого и второго последователя; например, пир 4 теперь должен отслеживать не только пир 5, но и 8. Еще одно дополнительное требование — это периодическая про-

верка того, что два этих последователя «живы» (например, отправка им время от времени эхо-сообщения и проверка ответа). Давайте рассмотрим, как же обрабатывается хеш-таблица, когда пир внезапно исчезает. Предположим, например, на рис. 2.27а пир 5 выходит из сети. В таком случае два его предшественника (4 и 3) знают, что пир 5 исчез, так как больше не отвечает на эхо-сообщения. Следовательно, пиры 3 и 4 должны обновить свою информацию о состоянии последователей. Пир 4 обновляет свое состояние следующим образом:

1. Пир 4 заменяет своего первого последователя (пир 5) вторым последователем (пир 8).
2. Пир 4 запрашивает у своего нового первого последователя (у пира 8) идентификатор и IP-адрес того непосредственного последователя (это пир 10) и делает его (пир 10) своим вторым последователем.

В домашних заданиях вас попросят определить, каким образом пир 3 обновляет информацию о своих оверлейных маршрутах.

Вкратце узнав, что же происходит, когда пир покидает сеть, давайте рассмотрим теперь, что произойдет, когда он захочет присоединиться к распределенной хеш-таблице. Допустим, например, что пир с идентификатором 13 хочет присоединиться к таблице, и в момент присоединения ему известно только то, что в таблице существует пир с номером 1, и больше ничего. Первым делом пир 13 будет отправлять пиру 1 сообщение, гласящее «кто является предшественником и последователем пира 13?» Это сообщение перенаправляется через таблицу, пока не достигнет пира 12, который понимает, что он является предшественником для номера 13, а соседний к нему пир 15 становится последователем для новичка. Затем пир 12 отправляет эту информацию пиру 13. Теперь пир 13 может присоединиться к таблице, сделав пир 15 своим последователем и указав пиру 12, что он должен сменить своего непосредственного последователя на 13.

Распределенные хеш-таблицы нашли широкое применение на практике. Например, BitTorrent использует Kademlia DHT для создания распределенного трекера. В протоколе BitTorrent ключом является идентификатор торрента, а значением — IP-адреса всех пиров, участвующих в торренте[154, 364]. Таким образом, запросив хеш-таблицу с помощью идентификатора торрента, вновь прибывший пир может определить того, кто отвечает за идентификатор (то есть за отслеживание пиров в торренте). После его нахождения прибывающий пир может запросить у него список других участников торрента.

2.7. Программирование сокетов: создание сетевых приложений

Теперь, когда вы познакомились с некоторым количеством сетевых приложений, давайте рассмотрим, как же эти приложения в действительности создаются. Вспомним из раздела 2.1, что типичное сетевое приложение состоит из двух частей — клиентской и серверной программ, работающих на двух различных конечных системах. Когда запускаются эти две программы, создаются два процесса: клиентский и серверный, которые взаимодействуют друг с другом, производя чтение и запись в сокеты. Таким образом, при создании сетевого приложения основная задача разработчика — написать коды как для клиентской, так и для серверной частей.

Существуют два типа сетевых приложений. Для первого из них функционирование описывается стандартами выбранного протокола, такими как RFC или другие документы: такое приложение иногда называют «открытым», так как правила, определяющие операции в нем, известны всем. В такой реализации клиентская и серверная программы должны соответствовать правилам, продиктованным документами RFC. Например, клиентская программа может быть реализацией клиентской части протокола FTP, определенного в RFC 959 и о котором мы говорили в разделе 2.3; аналогично, серверная программа может быть частью реализации протокола FTP сервера, также описанного в документе RFC 959. Если один разработчик пишет код для клиентской программы, а другой для серверной, и оба внимательно следуют правилам RFC, то эти две программы будут взаимодействовать без проблем. И действительно — многие из сегодняшних сетевых приложений предполагают взаимодействие между клиентскими и серверными программами, которые созданы независимыми разработчиками — например, браузер Firefox, который взаимодействует с веб-сервером Apache, или клиент BitTorrent, взаимодействующий с трекером BitTorrent.

Другой тип сетевых приложений представляют проприетарные (закрытые) приложения. В этом случае клиентские и серверные программы используют протокол прикладного уровня, который открыто не опубликован ни в RFC, ни в каких-либо других документах. Разработчик (либо команда разработчиков) создает клиентские и серверные программы и имеет полный контроль над всем, что происходит в коде. Но так как в кодах программы не реализован открытый протокол, другие независимые разработчики не смогут написать код, который бы взаимодействовал с этим приложением.

В этом разделе мы изучим ключевые понятия клиент-серверной разработки и рассмотрим пример программы, реализующей довольно простое клиент-серверное приложение. Во время фазы разработки одно из первых решений, которое должен принять создатель — определить, каким образом будет работать его приложение: через TCP или через UDP. Вспомним, что TCP является протоколом с установлением соединения и обеспечивает надежную доставку данных по каналу между двумя конечными системами. UDP является протоколом без установления соединения и отправляет независимый пакет данных от одной системы к другой без каких-либо гарантий его доставки. Напомним также, что когда клиентская или серверная программа реализует протокол, описанный в RFC, она должна использовать «хорошо известные» номера портов, связанные с протоколом. Наоборот, те, кто разрабатывает проприетарные приложения, стараются избегать использования известных номеров портов (номера портов кратко обсуждались в разделе 2.1, более подробно о них — в главе 3).

Мы будем рассматривать программирование сокетов с использованием протоколов UDP и TCP на примере простых приложений, написанных на языке Python. Можно было написать коды и на Java, C или C++, но мы выбрали Python в основном из-за того, что он очень четко выражает ключевые концепции сокетов. Количество строк на Python будет меньше, и их легко объяснить даже новичкам в программировании. Но не нужно пугаться, если вы не знакомы с языком Python — для вас это не составит труда, если вы имеете кое-какой опыт программирования на Java, C или C++.

Если вам интересно посмотреть клиент-серверные программы на Java, вы можете загрузить примеры для этой главы (и соответствующие лабораторные работы) на Java с веб-сайта издательства. Для читателей, интересующихся клиент-серверным программированием на C, есть несколько полезных и доступных ссылок[142, 618, 174]; наши примеры ниже, представленные на языке Python, имеют много общего с программами на C.

2.7.1. Программирование сокетов с использованием UDP

В этом разделе мы напишем простую клиент-серверную программу, которая использует протокол UDP; в следующем — аналогичную программу, но с использованием TCP.

Вспомним из раздела 2.1, что процессы, запущенные на различных устройствах, взаимодействуют друг с другом с помощью отправки со-

общений в сокеты. Мы говорили, что каждый процесс похож на дом, а сокет процесса — аналог двери. Приложение работает с одной стороны двери — в доме; протокол транспортного уровня находится по другую сторону двери — во внешнем мире. Разработчик приложения контролирует все, что находится со стороны прикладного уровня, однако со стороны транспортного уровня он мало на что влияет.

Теперь посмотрим детально на взаимодействие между двумя обменивающимися процессами, которые используют UDP-сокеты. Перед тем как отправляющий процесс сможет протолкнуть пакет данных через свой сокет, он должен сначала прикрепить к нему адрес назначения. После того, как пакет проходит через сокет отправителя, Интернет использует этот адрес назначения, чтобы направить пакет сокету принимающего процесса. Когда пакет прибывает в сокет получателя, принимающий процесс извлекает его через сокет, проверяет содержимое пакета и предпринимает соответствующие действия.

Вы можете, очевидно, спросить, что же входит в адрес назначения, который прикрепляется к пакету? Как нетрудно догадаться, одной из его частей является IP-адрес хоста назначения. Это позволит маршрутизаторам в Интернете направить пакет по сети к требуемому адресату. Но так как на хосте могут быть запущены несколько процессов сетевых приложений, причем каждый может использовать один или более сокетов, то необходимо идентифицировать определенный сокет на хосте назначения. При создании сокета ему присваивается идентификатор, называемый **номером порта**. Поэтому адрес назначения включает также и номер порта сокета. Подытожив, скажем, что отправляющий процесс прикрепляет к пакету адрес назначения, который состоит из IP-адреса хоста-приемника и номера порта принимающего сокета. Более того, как мы вскоре увидим, адрес отправителя, состоящий из IP-адреса хоста-источника и номера порта исходящего сокета, также прикрепляются к пакету. Однако это обычно делается *не* в коде UDP-приложения, а автоматически операционной системой отправляющего хоста.

Для демонстрации приемов программирования сокетов с использованием UDP и TCP мы будем использовать простое клиент-серверное приложение, суть которого заключается в следующем:

1. Клиент читает строку символов (данные) с клавиатуры и отправляет данные серверу.
2. Сервер получает данные и преобразует символы в верхний регистр.

3. Сервер отправляет измененные данные клиенту.
4. Клиент получает измененные данные и отображает строку на своем экране.

Рисунок 2.28 отображает основные действия клиента и сервера с сокетами, взаимодействующими через транспортную службу протокола UDP.

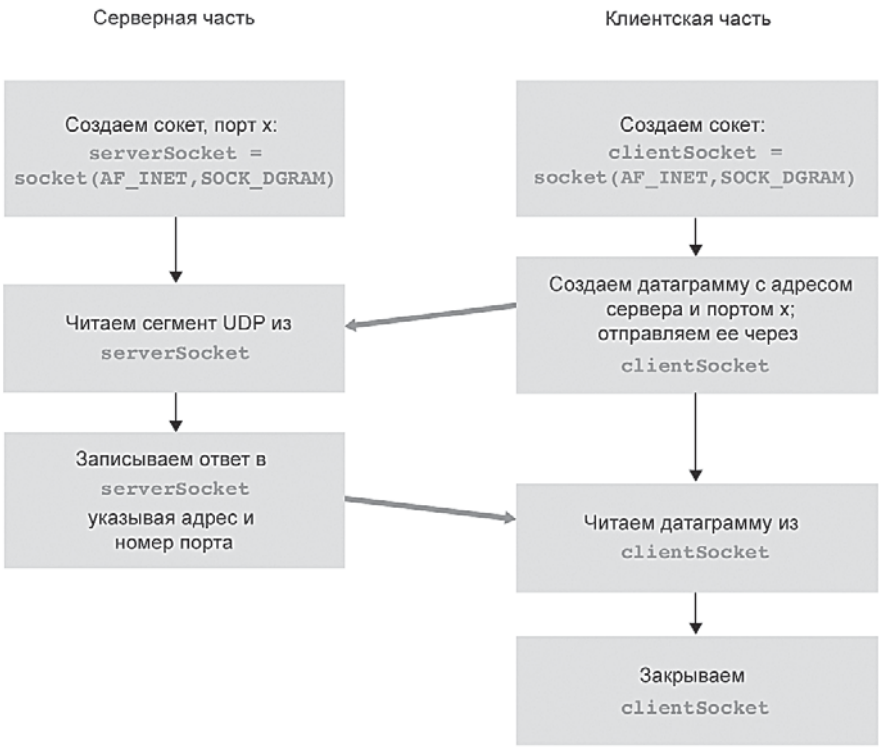


Рис. 2.28. Клиент-серверное приложение, использующее протокол UDP

Давайте посмотрим на пару клиент-серверных программ, реализующих это простое приложение, используя протокол UDP. Мы сделаем подробный построчный анализ каждой программы. Начнем с UDP-клиента, который будет отправлять простое сообщение прикладного уровня серверу. Для того чтобы сервер был способен получать и отвечать на сообщения клиента, он должен быть «готов» — то есть запущен в качестве процесса перед тем, как клиент отправит свое сообщение.

Клиентская программа называется UDPClient.py, а серверная — UDPServer.py. На самом деле «правильный код» будет включать гораз-

до больше строк, например, для обработки ошибок, но мы, чтобы подчеркнуть ключевые моменты, намеренно сделали его как можно короче. Для нашего приложения мы взяли произвольный номер порта сервера 12 000.

UDPClient.py

Ниже представлен код для клиентской части приложения:

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName,
serverPort))
modifiedMessage, serverAddress = clientSocket.
recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

Теперь посмотрим на различные строки кода этой программы.

```
from socket import *
```

Модуль `socket` является базовым для всех сетевых взаимодействий, описываемых на языке Python. Включив эту строку, мы сможем создавать сокеты внутри нашей программы.

```
serverName = 'hostname'
serverPort = 12000
```

В первой строке мы присваиваем переменной `serverName` строковое значение `'hostname'`. Здесь мы должны поставить строку, содержащую либо IP-адрес сервера (например, «128.138.32.126»), либо имя хоста сервера (например, «cis.poly.edu»). При использовании имени хоста произойдет автоматическое преобразование его в IP-адрес. Во второй строке мы устанавливаем значение целочисленной переменной `serverPort` равным `12000`.

```
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
```

В этой строке мы создаем сокет клиента, называя его clientSocket. Первый параметр определяет семейство адресов; в частности, AF_INET указывает, что мы будем использовать протокол IPv4 (пока об этом не думайте — мы обсудим IPv4 в главе 4). Второй параметр указывает на тип сокета — SOCK_DGRAM, что означает, что это UDP-сокет (а не TCP). Отметим, что мы не задаем номер порта клиентского сокета при его создании — позволяем это сделать за нас операционной системе. Теперь, когда создана «дверь» клиентского процесса, мы можем создавать сообщения и отправлять их через нее.

```
message = raw_input('Input lowercase sentence:')
```

raw_input() — это встроенная в Python функция. При ее выполнении пользователю на клиентской стороне предлагается подсказка со словами «Input lowercase sentence» («Введите предложение в нижнем регистре»). После этого пользователь может использовать клавиатуру для ввода строки, которая будет занесена в переменную message. Теперь, когда у нас есть сокет и сообщение, мы можем отправлять это сообщение через сокет хосту назначения.

```
clientSocket.sendto(message,(serverName, serverPort))
```

В этой строке с помощью метода sendto() к сообщению добавляется адрес назначения (serverName, serverPort), и весь результирующий пакет отправляет в сокет процесса — clientSocket. (Как указывалось ранее, адрес источника также добавляется к пакету, хотя это делается автоматически, а не явно через строки кода.) На этом отправка сообщения от клиента серверу через сокет UDP закончена. Как видим, это достаточно просто! После отправки пакета клиент ожидает получения данных с сервера.

```
modifiedMessage,serverAddress=clientSocket.recvfrom(2048)
```

С помощью этой строки данные пакета, прибывающего из Интернета на сокет клиента, помещаются в переменную modifiedMessage, а адрес источника пакетов — в переменную serverAddress. Последняя переменная содержит как IP-адрес, так и номер порта сервера. В действительности программе UDPClient не нужна эта информация, так как она уже знает адрес сервера с самого начала, но, тем не менее, данная строка в программе присутствует. Метод recvfrom() генерирует входной буфер размером 2048 байт, который используется для различных целей.

```
print modifiedMessage
```

Данная строка выводит измененное сообщение на дисплей пользователя. Это первоначальная строка, в которой все символы стали заглавными.

```
clientSocket.close()
```

Здесь мы закрываем сокет, и процесс завершается.

UDPServer.py

Теперь рассмотрим программу серверной части приложения:

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
message, clientAddress = serverSocket.recvfrom(2048)
modifiedMessage = message.upper()
serverSocket.sendto(modifiedMessage, clientAddress)
```

Заметим, что начало программы UDPServer очень похоже на UDPClient. Она также импортирует модуль `socket`, также устанавливает значение целочисленной переменной `serverPort` в `12000`, также создает сокет типа `SOCK_DGRAM` (UDP). Первая, значительно отличающаяся строка, это:

```
serverSocket.bind(('', serverPort))
```

Данная строка связывает (то есть назначает) порт под номером 12 000 с сокетом сервера. Таким образом, в программе UDPServer строки кода (написанные разработчиком приложения) явным образом присваивают номер порта сокету. Связывание порта означает, что теперь, если кто-то отправляет пакет на порт 12 000 нашего сервера, то этот пакет будет направлен в данный сокет. Дальше программа UDPServer переходит в бесконечный цикл `while`, который позволяет получать и обрабатывать пакеты от клиентов в неограниченном количестве.

```
message, clientAddress = serverSocket.recvfrom(2048)
```

Приведенная строка очень похожа на то, что мы уже видели в UDPClient. Когда пакет прибывает на сокет сервера, данные пакета помещаются в переменную message, а данные источника пакетов — в переменную `clientAddress`.

Переменная `clientAddress` содержит IP-адрес и номер порта клиента. Эта информация будет использоваться программой, так как в ней передается адрес возврата и сервер теперь знает, куда направлять свой ответ.

```
modifiedMessage = message.upper()
```

Данная строка является основной в нашем простом приложении. Здесь мы берем введенную клиентом строку и, используя метод `upper()`, меняем ее символы на заглавные.

```
serverSocket.sendto(modifiedMessage, clientAddress)
```

Данная последняя строка кода присоединяет адрес клиента (IP-адрес и номер порта) к измененному сообщению и отправляет результирующий пакет в сокет сервера (как уже упоминалось, адрес сервера также присоединяется к пакету, но это делается не в коде, а автоматически). Затем Интернет доставляет пакет на этот клиентский адрес.

Сервер после отправки пакета остается в бесконечном цикле, ожидая прибытия другого UDP-пакета (от любого клиента, запущенного на произвольном хосте).

Чтобы протестировать ваши части програм, просто запустите UDPClient.py на одном хосте и UDPServer.py — на другом. Не забудьте включить правильное имя или IP-адрес сервера в клиентскую часть.

Затем вы запускаете UDPServer.py на серверном хосте, что создает процесс на сервере, который будет ожидать контакта клиента. После этого вы запускаете UDPClient.py на клиенте, тем самым создаете процесс на клиентской машине. Наконец, вы просто вводите предложение, завершая его возвратом каретки.

Вы можете разработать свое собственное клиент-серверное приложение UDP, просто модифицировав клиентскую или серверную части программы. Например, чтобы вместо конвертирования всех символов в заглавные, сервер подсчитывал число букв *«я»* в строке, или чтобы после получения измененного сообщения пользователь мог продолжать отправлять предложения на сервер.

2.7.2. Программирование сокетов с использованием протокола TCP

В отличие от UDP, протокол TCP является протоколом с установлением соединения. Это означает, что клиент и сервер, перед тем как отправлять данные друг другу, сначала обмениваются рукопожатиями и устанавливают TCP-соединение. Одна сторона этого соединения связана с клиентским сокетом, а другая — с серверным. При создании TCP-соединения с ним связывается адрес клиентского сокета (IP-адрес и номер порта) и адрес серверного сокета (IP-адрес и номер порта). Когда одна сторона пытается отправить данные другой стороне с помощью установленного TCP-соединения, она просто передает их в свой сокет. В этом заключается отличие от UDP, в котором сервер вначале должен прикрепить адрес назначения к пакету и лишь затем отправить пакет в сокет.

Теперь рассмотрим подробнее, как взаимодействуют клиентская и серверная программы при работе через TCP. Задача клиента заключается в инициировании контакта с сервером. Для того чтобы реагировать на начальный контакт клиента, сервер должен быть «готов». Готовность подразумевает под собой две вещи: во-первых, как и в случае с протоколом UDP, TCP-сервер должен быть запущен как процесс, перед тем как клиент попытается инициировать контакт; во-вторых, у серверной программы должна существовать специальная «дверь», а именно специальный сокет, который принимает начальный контакт от клиентского процесса, запущенного на произвольном хосте. Используя нашу аналогию с домом и дверью, мы иногда будем называть начальный контакт клиента как «стук во входную дверь».

Когда серверный процесс запущен, клиентский процесс может инициировать TCP-соединение с сервером. Это осуществляется в клиентской программе с помощью создания TCP-сокета. При этом клиент указывает адрес входного сокета сервера, а именно IP-адрес серверного хоста и номер порта сокета. После создания своего сокета клиент инициирует тройное рукопожатие и устанавливает TCP-соединение с сервером. Данное рукопожатие полностью невидимо для клиентской и серверной программ и происходит на транспортном уровне.

Во время тройного рукопожатия клиентский процесс стучится во входную дверь серверного процесса. Когда сервер «слышит» стук, он открывает новую дверь, а точнее говоря, создает *новый* сокет, который предназначен этому конкретному стучащемуся клиенту. В нашем примере ниже входная дверь — это TCP-сокет, который мы назвали

`serverSocket`; вновь создаваемый сокет, предназначенный для клиента, который организует соединение, мы назвали `connectionSocket`. Те, кто рассматривает TCP-сокеты впервые, иногда путает понятия *входного сокета* (который является начальной точкой контакта для всех клиентов, ожидающих связи с сервером) и вновь создаваемого *сокета соединения* серверной стороны, который последовательно создается для связи с каждым клиентом.

С точки зрения приложений, клиентский сокет и серверный сокет соединения связаны напрямую. Как показано на рис. 2.29, клиентский процесс может отправлять произвольное количество байт в свой сокет и протокол TCP гарантирует, что серверный процесс получит (через сокет соединения) каждый отправленный байт в определенном порядке. Таким образом, TCP предоставляет надежную службу доставки между клиентским и серверным процессами. Так же как люди могут ходить через дверь туда и обратно, клиентский процесс может не только отправлять данные, но и принимать их через свой сокет. Аналогично, серверный процесс не только принимает, но и отправляет данные через свой сокет соединения.

Рис. 2.29. Два сокета серверного процесса

Мы будем использовать то же самое простое клиент-серверное приложение для демонстрации приемов программирования сокетов с использованием TCP: клиент отправляет на сервер одну строку данных,

сервер делает преобразование символов строки в заглавные и отправляет строку обратно клиенту. На рис. 2.30 представлена основная схема взаимодействия клиента и сервера, связанная с сокетами, через транспортную службу протокола TCP.

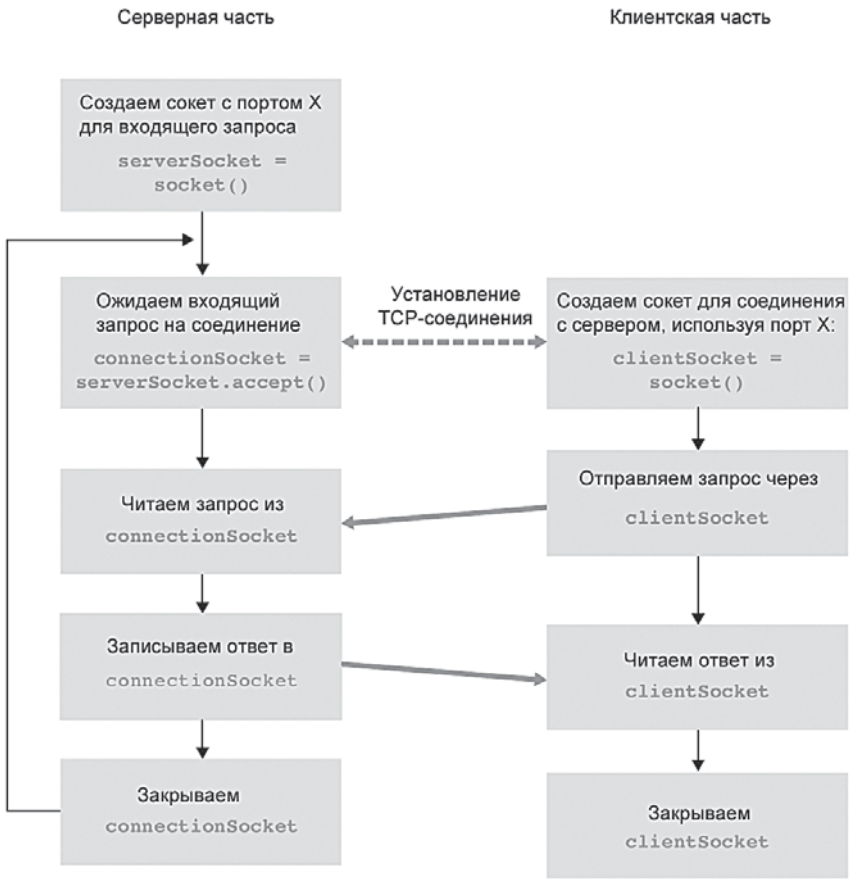


Рис. 2.30. Клиент-серверное приложение, использующее протокол TCP

TCPClient.py

Ниже представлен код клиентской части приложения:

```
from socket import *
serverName = 'servername'
serverPort = 12000
```

```
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

Теперь давайте рассмотрим некоторые строки кода, которые значительно отличаются от реализации в случае с протоколом UDP. Первая отличающаяся строка — это создание клиентского сокета.

```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

В данной строке создается клиентский сокет, называемый `clientSocket`. Первый параметр, опять же, указывает нам, что мы используем сетевой протокол IPv4. Второй параметр определяет тип сокета (`SOCK_STREAM`), другими словами, это и есть TCP-сокет (а не UDP). Заметим, что мы не указываем номер порта клиентского сокета при его создании, а предоставляем сделать это за нас операционной системе. Следующая строка кода, которая очень отличается от того, что мы видели в UDPClient, это:

```
clientSocket.connect((serverName,serverPort))
```

Вспомним, что перед тем как клиент и сервер смогут посылать данные друг другу, используя TCP-сокет, между ними должно быть установлено TCP-соединение. Строка выше как раз инициирует соединение между клиентом и сервером. Параметром метода `connect()` является адрес серверной части соединения. После исполнения этой строки кода осуществляется тройное рукопожатие, и между клиентом и сервером устанавливается TCP-соединение.

```
sentence = raw_input('Input lowercase sentence:')
```

Как и в программе UDPClient, здесь программа получает предложение, введенное пользователем. В строковую переменную sentence записываются все символы, вводимые пользователем до тех пор, пока он не введет символ возврата каретки. Следующая строка также сильно отличается от того, что мы видели в программе UDPClient:

```
clientSocket.send(sentence)
```

Данная строка отправляет строковую переменную sentence через клиентский сокет в TCP-соединение. Заметим, что программа явно *не* создает пакет и не прикрепляет адрес назначения к нему, как это было в случае с UDP-сокетами.

Вместо этого она просто сбрасывает данные строки `sentence` в TCP-соединение. После этого клиент ожидает получения данных от сервера.

```
modifiedSentence = clientSocket.recv(2048)
```

Прибывающие с сервера символы помещаются в строковую переменную `modifiedSentence`. Они продолжают накапливаться в переменной до тех пор, пока в строке не будет обнаружен символ возврата каретки. После печати строки с заглавными символами мы закрываем клиентский сокет:

```
clientSocket.close()
```

Последняя строка закрывает сокет, а, следовательно, закрывает TCP-соединение между клиентом и сервером.

На самом деле она приводит к тому, что TCP-сообщение с клиента отправляется на сервер (см. раздел 3.5)

TCPServer.py

Теперь посмотрим на серверную часть программы:

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
  connectionSocket, addr = serverSocket.accept()
  sentence = connectionSocket.recv(1024)
  capitalizedSentence = sentence.upper()
  connectionSocket.send(capitalizedSentence)
  connectionSocket.close()
```

Опять же, рассмотрим строки, которые значительно отличаются от представленных в программах UDPServer и TCPClient. Так же, как и в программе TCPClient, сервер создает TCP-сокет с помощью строки:

```
serverSocket=socket(AF_INET,SOCK_STREAM)
```

Затем мы связываем номер порта сервера (переменная `serverPort`) с нашим сокетом:

```
serverSocket.bind(('',serverPort))
```

Но в случае с TCP переменная `serverSocket` будет являться нашим входным сокетом. После подготовки входной «двери» мы будем ожидать клиентов, стучащихся в нее:

```
serverSocket.listen(1)
```

Эта строка означает, что сервер «слушает» запросы от клиентов по TCP-соединению. Параметр в скобках определяет максимальное число поставленных в очередь соединений.

```
connectionSocket, addr = serverSocket.accept()
```

Когда клиент стучится в дверь, программа запускает для серверного сокета метод `accept()`, и тот создает новый сокет на сервере — сокет соединения, который предназначен для этого конкретного клиента. Затем клиент и сокет завершают рукопожатие, создавая тем самым TCP-соединение между `clientSocket` и `connectionSocket`, после установления которого клиент и сервер могут обмениваться данными друг с другом, причем данные с одной стороны к другой доставляются с гарантией, а также гарантируется порядок их доставки.

```
connectionSocket.close()
```

После отправки измененной строки клиенту мы закрываем сокет соединения. Но так как серверный сокет остается открытым, любой другой клиент может постучаться в дверь и отправить серверу новую строку для изменения.

На этом мы завершаем обсуждение программирования сокетов с использованием TCP. Вам предлагается запустить две программы на двух различных хостах, а также попробовать их немного изменить по своему усмотрению. Сравните пару программ для UDP с парой для TCP, по-

смотрите, чем они отличаются. Вам также предстоит выполнить немало заданий по программированию сокетов, описанных в конце книги. В конце концов, мы надеемся, что когда-нибудь после успешного написания как этих, так и более сложных программ для работы с сокетами, вы разработаете свое собственное приложение, которое станет популярным, сделает вас богатыми и знаменитыми, и, возможно, вы вспомните авторов этой книги!

2.8. Заключение

В этой главе мы изучили теоретические и практические аспекты сетевых приложений. Мы познакомились с распространенной клиент-серверной архитектурой, используемой многими Интернет-приложениями, и увидели ее в действии — каким образом она применяется в протоколах HTTP, FTP, SMTP, POP3 и DNS. Эти важные протоколы прикладного уровня и связанные с ними приложения (Всемирная паутина, передача файлов, электронная почта и DNS) мы рассмотрели чуть подробнее. Мы познакомились с одноранговой архитектурой, популярность которой все больше возрастает, и посмотрели, как она используется многими приложениями. Мы изучили, каким образом можно применять сокеты для построения сетевых приложений и как с ними работать в транспортных службах с установлением (TCP) и без установления соединения (UDP). На этом первый шаг в нашем путешествии по уровневой сетевой архитектуре завершен!

В самом начале данной книги, в разделе 1.1, мы давали довольно общее и расплывчатое понятие протокола: «формат и порядок сообщений, которыми обмениваются два или более взаимодействующих объектов, а также действия, предпринимаемые при передаче и/или приеме сообщения либо при возникновении другого события». Материал данной главы и особенно наше подробное знакомство с протоколами HTTP, FTP, SMTP, POP3 и DNS значительно добавили сути в это определение. Изучение протоколов прикладного уровня дало нам возможность более интуитивно почувствовать, что же такое протокол и понять, что он является ключевым понятием в сетевых технологиях.

В разделе 2.1 мы описали модели обслуживания, которые протоколы TCP и UDP предлагают работающим с ними приложениям. Мы

еще ближе познакомились с этими моделями обслуживания, когда разрабатывали простые приложения, которые работают через TCP и UDP, в разделе 2.7. Однако мы мало упоминали о том, каким же образом протоколы TCP и UDP обеспечивают данное обслуживание. Например, мы знаем, что TCP предлагает надежную службу доставки данных, но как он это осуществляет, рассказано не было. В следующей же главе мы внимательно рассмотрим не только вопрос «*что?*», но и также вопросы «*как?*» и «*почему?*», относящиеся к транспортным протоколам.

Вооруженные знаниями о структуре Интернет-приложений и о протоколах прикладного уровня, мы теперь готовы двигаться дальше по стеку протоколов и перейти к изучению транспортного уровня в следующей главе.

Глава 3
ТРАНСПОРТНЫЙ УРОВЕНЬ

Транспортный уровень, расположенный между прикладным и сетевым уровнями, представляет собой центральную часть сетевой архитектуры. Он играет важную роль в предоставлении коммуникационных служб непосредственно для прикладных процессов, запущенных на разных хостах.

В этой главе мы используем педагогический подход, позволяющий разграничить обсуждение принципов транспортного уровня как таковых и реализации этих принципов в существующих протоколах; как и прежде, особое внимание будет уделено протоколам Интернета, в частности протоколам транспортного уровня TCP и UDP.

Для начала мы обсудим взаимодействие транспортного и сетевого уровней, что послужит основой для изучения первой важной функции транспортного уровня: он связывает службы доставки сетевого уровня, работающие между двумя конечными системами — с одной стороны, и службы доставки, действующими между двумя процессами прикладного уровня, запущенными на конечных системах — с другой. Мы продемонстрируем этот функционал при изучении UDP — транспортного протокола Интернета, работающего без установления логического соединения.

Далее мы вернемся к теории и рассмотрим одну из наиболее важных проблем компьютерных сетей: каким образом два объекта могут надежно взаимодействовать в среде, допускающей потерю и повреждение данных. Мы опишем несколько постепенно усложняющихся (и от этого становящихся более реалистичными) ситуаций, отражающих данную проблему, и познакомимся с технологиями, направленными на ее решение. Затем мы покажем, как эти принципы реализованы в Интернет-протоколе TCP, работающем с установлением логического соединения.

Затем мы рассмотрим вторую фундаментальную задачу сетей — управление скоростью передачи объектов транспортного уровня во избежание перегрузок сети или в целях их восстановления после сбоев,

возникающих из-за перегрузок. Мы обсудим причины и последствия перегрузок, а также приемы, которые обычно задействуются чтобы справиться с ними. Полностью разобравшись с вопросами управления перегрузками мы изучим, как это делается в протоколе TCP.

3.1. Введение и службы транспортного уровня

В двух предыдущих главах мы уже упоминали о роли транспортного уровня и предоставляемых им служб. Теперь освежим наши знания о транспортном уровне.

Протокол транспортного уровня обеспечивает **логическое соединение** между прикладными процессами, выполняющимися на разных хостах. *Логическое соединение* с точки зрения приложений выглядит как канал, непосредственно соединяющий процессы, даже если хосты находятся в разных уголках планеты и реальная связь между ними осуществляется с помощью длинной цепи маршрутизаторов и разнообразных линий связи. Процессы прикладного уровня задействуют логическое соединение, предоставляемое транспортным уровнем, для обмена информацией, без учета деталей физической инфраструктуры, используемой для передачи этих сообщений. На рис. 3.1 проиллюстрирована модель логического соединения.

Как показано на рис. 3.1, протокол транспортного уровня поддерживают конечные системы, но не сетевые маршрутизаторы. На стороне отправителя транспортный уровень преобразует сообщения прикладного уровня, которые получает от передающего прикладного процесса, в пакеты транспортного уровня, называемые в контексте интернет-технологий **сегментами** транспортного уровня. Это делается разбиением (при необходимости) сообщений прикладного уровня на фрагменты и добавлением к каждому из них заголовка транспортного уровня. Далее транспортный уровень передает сегмент сетевому уровню отправителя, где сегмент инкапсулируется в пакет сетевого уровня (дейтаграмму) и отсылается. Заметим, что сетевые маршрутизаторы обрабатывают поля дейтаграммы только на сетевом уровне, то есть, они не проверяют поля сегмента транспортного уровня, инкапсулированные в дейтаграмме. На принимающей стороне сетевой уровень извлекает сегмент транспортного уровня из дейтаграммы и передает его вверх транспортному уровню. Далее транспортный уровень обрабатывает

полученный сегмент таким образом, чтобы его данные стали доступны приложению-получателю.

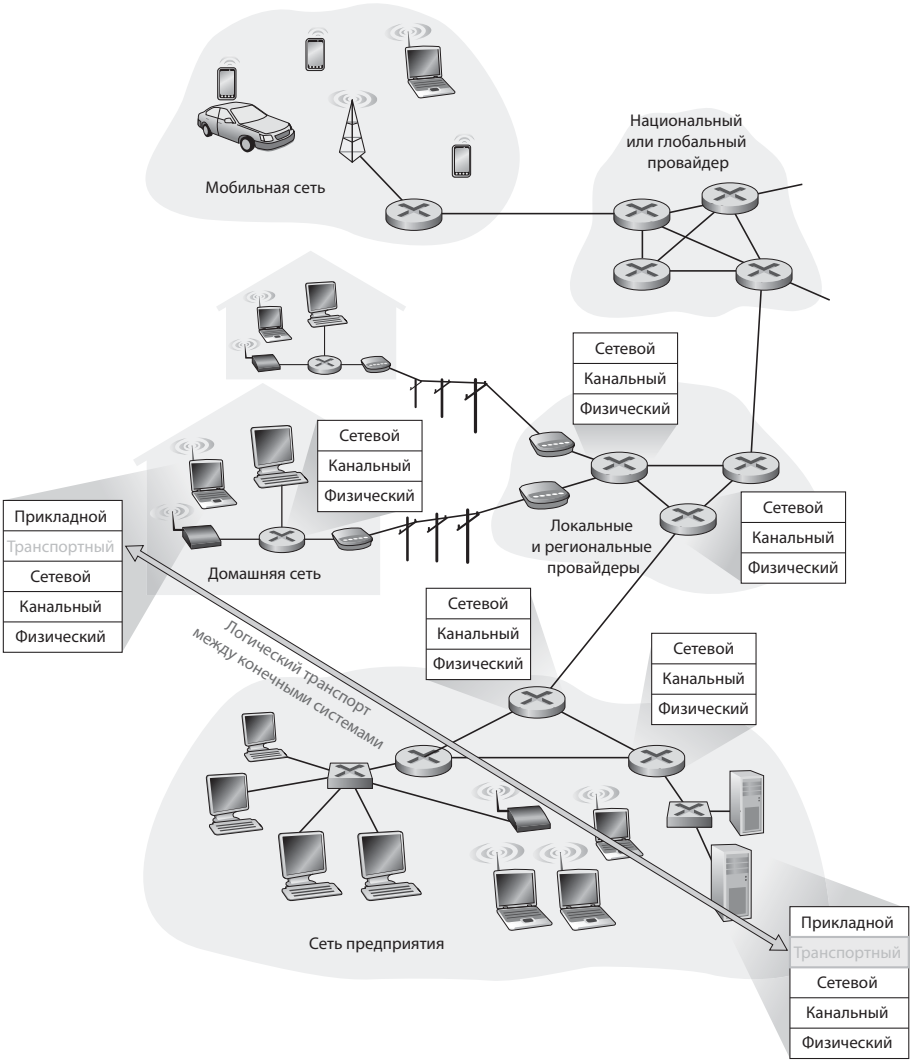


Рис. 3.1. Транспортный уровень обеспечивает логическое, а не физическое соединение между прикладными процессами

Для сетевых приложений обычно доступно несколько протоколов транспортного уровня. Например, в Интернете используются два из них: TCP и UDP. Каждый из этих протоколов предоставляет различный набор служб для работы приложений.

3.1.1. Взаимодействие транспортного и сетевого уровней

Напомним, что транспортный уровень распложен в стеке протоколов сразу над сетевым уровнем. В то время как протокол транспортного уровня обеспечивает логическое соединение между *процессами,* запущенными на различных хостах, протокол сетевого уровня обеспечивает логическое соединение между *хостами*. Это тонкое, но важное различие. Мы поясним его на следующем примере.

Допустим, существует два дома, один на восточном побережье США, другой на западном побережье США, в каждом доме живет двенадцать детей. Дети из семьи на западном побережье — это двоюродные братья и сестры детей из дома на восточном побережье. Дети обоих домов любят писать друг другу письма: каждый ребенок пишет каждому из своих двенадцати родственников каждую неделю, каждое письмо отправляется обычной почтой в отдельном конверте. Таким образом, обе семьи отправляют по 144 письма друг другу в неделю. (Эти дети могли бы сэкономить немало денег, если бы освоили электронную почту.) В каждой семье есть ребенок, ответственный за получение и отправление почты: Анна, в доме на западном побережье и Билл, в доме на восточном побережье. Каждую неделю Анна обходит всех своих братьев и сестер, собирает почту и передает письма почтальону, ежедневно приходящему на дом. Анна также раздает почту своим братьям и сестрам, когда письма приходят в дом на западном побережье. Билл выполняет аналогичную работу в доме на восточном побережье.

В этом примере почтовая служба обеспечивает логическое соединение между двумя домами: она передает почту между домами, а не между детьми. С другой стороны, Анна и Билл обеспечивают логическое соединение между двоюродными братьями и сестрами: собирают и передают им почту.

Отметим, что, с точки зрения братьев и сестер, Анна и Билл *являются* почтовой службой, несмотря на то, что они лишь часть (конечные системы) механизма доставки между конечными точками. Представленный пример семьи служит прекрасной аналогией для взаимодействия транспортного и сетевого уровней:

сообщения приложений = письма в конвертах
процессы = братья и сестры
хосты (или конечные системы) = дома

протокол транспортного уровня = Анна и Билл
протокол сетевого уровня = почтовая служба (включая почтальона)

Продолжая нашу аналогию, заметим, что Анна и Билл выполняют свои обязанности только в собственных домах; они не вовлечены, например, в сортировку писем в любом промежуточном почтовом отделении или их пересылку от одного почтового отделения до другого. Аналогичным образом протокол транспортного уровня действует на конечных системах. На конечной системе транспортный протокол передает сообщение от прикладных процессов в сеть (это и есть сетевой уровень) и обратно, но ничего не сообщает о том, как сообщения перемещаются в сети. На самом деле, как показано на рис. 3.1, промежуточные маршрутизаторы ни выполняют действий, ни распознают какой-либо информации, которою транспортный уровень может добавить в сообщения прикладного уровня.

Продолжая нашу семейную сагу, представим, что Анна или Билл отправляются на отдых, и другая пара кузенов, скажем, Сьюзен и Харви замещают их и выполняют сбор и отправку почты внутри семьи. К сожалению, Сьюзен и Харви не делают этого в точности, как Анна и Билл. Будучи младше, Сьюзен и Харви собирают и отправляют почту реже, а иногда и теряют письма, которые время от времени съедает домашний пес. Таким образом, пара кузенов Сьюзен и Харви не предоставляют того же набора услуг (иначе говоря, такой же модели обслуживания), что Анна и Билл. Аналогичным образом, компьютерным сетям может быть доступно множество транспортных протоколов, и каждый протокол представляет различные модели обслуживания для приложений.

Услуги, которые Анна и Билл могут предоставлять, четко ограничены доступными услугами почтовой службы. Например, если почтовая служба не регламентирует максимальное время доставки почты между двумя домами (например, тремя днями), то Анна и Билл никак не смогут гарантировать ограничение максимальной задержки корреспонденции между любыми парами кузенов. Аналогично, те службы, которые может представлять транспортный протокол, часто ограничены моделью обслуживания протокола нижестоящего сетевого уровня. Если протокол сетевого уровня не гарантирует величину задержки или уровень пропускной способности для сегментов транспортного уровня, отправляемых между хостами, то и протокол транспортного уровня не может гарантировать величину задержки или уровень пропускной способности для сообщений, пересылаемых между процессами сообщений прикладного уровня.

Тем не менее определенные службы *могут* предоставляться протоколом транспортного уровня даже в случаях, когда нижестоящий сетевой протокол не поддерживает соответствующие службы на сетевом уровне. Например, как мы увидим в этой главе, протокол транспортного уровня может предоставлять службу надежной передачи данных для приложения, даже если нижестоящий протокол сетевого уровня ненадежный — то есть даже тогда, когда он допускает потери, искажение и дублирование пакетов. В качестве другого примера, который мы рассмотрим в главе 8 при обсуждении сетевой безопасности, транспортный протокол может использовать шифрование, чтобы обеспечить защиту сообщений приложения от прочтения злоумышленником, даже если сетевой уровень не может гарантировать конфиденциальности сегментов транспортного уровня.

3.1.2. Транспортный уровень в Интернете

Вспомним, что Интернет и в более общем случае сеть TCP/IP использует два отдельных протокола транспортного уровня доступных прикладному уровню. Один из этих протоколов — **UDP** (User Datagram Protocol, Протокол пользовательских дейтаграмм), который предоставляет ненадежную службу передачи данных без установления логического соединения для связи приложений. Второй протокол — это **TCP** (Transmission Control Protocol, протокол управления передачей), обеспечивающий службу надежной передачи данных с установлением соединения для связи между приложениями. При создании сетевого приложения разработчик должен выбрать один из этих двух протоколов. Как мы уже видели в разделе 2.7, разработчики приложений выбирают между протоколами UDP и TCP при создании сокетов.

Для упрощения терминологии в контексте Интернета мы называем пакеты транспортного уровня *сегментами*. Но напоминаем, что в источниках, посвященных Интернету (например, в стандартах RFC), термином «сегмент» именуются пакеты конкретно протокола TCP, а пакеты протокола UDP чаще называются «дейтаграммами». Однако в той же интернет-литературе термин *дейтаграмма* используется и в качестве названия пакетов сетевого уровня! Мы уверены, что в ознакомительной книге о компьютерных сетях (которую вы сейчас читаете), будет логичнее обозначать пакеты протоколов TCP и UDP термином *сегмент**, а пакеты сетевого уровня термином *дейтаграмма*.

* Строго говоря, термин «сегмент» применяется для TCP, где поток данных именно сегментируется (дробится) для передачи его порциями, а сообщения («пакеты») UDP — само по себе цельное и завершенное. — *Примеч. ред.*

Прежде чем приступить к нашему краткому знакомству с протоколами UDP и TCP, стоит сказать несколько слов о сетевом уровне Интернета (мы подробно изучим сетевой уровень в главе 4). Протокол сетевого уровня Интернета называется IP, Internet Protocol (Интернет-протокол). Протокол IP предоставляет логическое соединение между хостами. Модель обслуживания данного протокола — это **ненадежная служба доставки**. Другими словами, IP пытается осуществить успешную доставку сегментов от отправителя до получателя, однако *не дает никаких гарантий*: ни доставки фрагмента, ни сохранения их порядка. Поэтому протокол IP называют **ненадежной службой.** Мы также напоминаем, что каждый хост имеет один адрес сетевого уровня, так называемый IP-адрес. Подробное рассмотрение IP-адресации приведено в главе 4; сейчас необходимо усвоить лишь то, что *каждый хост имеет собственный IP-адрес*.

Взглянув на модель обслуживания протокола IP, давайте теперь резюмируем то, что мы узнали о моделях обслуживания протоколов UDP и TCP. Основной задачей UDP и TCP является обеспечение обмена данными между процессами, выполняющимися на конечных системах, при помощи службы обмена данными между конечными системами, предоставляемой протоколом сетевого уровня. Такое «продолжение» соединения между конечными системами до уровня процессов называется **мультиплексированием** и **демультиплексированием на транспортном уровне** и рассматривается в следующем разделе. Протоколы UDP и TCP также обеспечивают отсутствие искажений данных при передаче, включая в свои заголовки поля обнаружения ошибок. Заметим, что протокол UDP предоставляет минимальный набор служб транспортного уровня: службы обмена данными между процессами и контроля ошибок. Протокол UDP предоставляет только эти службы! В частности, подобно протоколу IP, UDP является ненадежной службой и не гарантирует, что данные, отправленные одним процессом, будут доставлены неповрежденными (и вообще будут доставлены) принимающему процессу. Протокол UDP будет подробно рассмотрен в разделе 3.3.

Протокол TCP, напротив, предоставляет несколько дополнительных служб для приложений. Первая и наиболее важная: служба **надежной передачи данных**. Используя управление потоком, порядковые номера, подтверждения и таймеры (в данной главе мы подробно рассмотрим все перечисленное), протокол TCP гарантирует, что данные будут доставлены от передающего процесса принимающему корректно и в верном порядке. Таким образом, протокол TCP преобразует ненадежную

службу протокола IP между конечными системами в службу надежной передачи данных между процессами. Протокол TCP также предоставляет службу **управления перегрузкой.** По сути, это не столько служба, предоставляемая приложению, сколько служба для всего Интернета. Можно сказать, что управление перегрузкой в протоколе TCP оберегает любое TCP-соединение от заполнения огромными объемами трафика, текущего по каналам между маршрутизаторами и взаимодействующими хостами. Протокол TCP призван предоставить каждому соединению равную долю пропускной способности для обхода перегруженного канала. Для этого применяется регулировка скорости, с которой передающая сторона TCP-соединения может отправлять трафик в сеть. С другой стороны, в протоколе UDP трафик неуправляемый. Приложения, использующие протокол UDP, могут осуществлять отправку на любой скорости, и так долго, сколько потребуется.

Протокол, который предоставляет надежную передачу данных и управление перегрузкой, неизбежно будет сложным. Нам потребуется несколько разделов, чтобы полностью рассмотреть принципы надежной передачи данных и управления перегрузкой, а также дополнительный раздел для изучения службы надежной передачи данных протокола TCP. Эти темы рассмотрены в разделах 3.4–3.8. Придерживаясь нашего подхода к изложению материала, мы сначала будем рассматривать общие задачи, а затем переходить к их практическому решению в протоколе TCP. Аналогично, мы сначала рассмотрим управление перегрузкой в общем, и затем изучим детали реализации этого механизма в протоколе TCP. Но для начала давайте рассмотрим мультиплексирование и демультиплексирование на транспортном уровне.

3.2. Мультиплексирование и демультиплексирование

В этом разделе мы рассмотрим операции мультиплексирования и демультиплексирования на транспортном уровне, «продолжающие» соединение между конечными системами до уровня соединения между процессами. Для того чтобы конкретизировать обсуждение, мы будем рассматривать службу мультиплексирования и демультиплексирования на транспортном уровне в контексте Интернета. Тем не менее эта служба необходима во всех компьютерных сетях.

На принимающем хосте транспортный уровень получает сегменты от нижестоящего сетевого уровня. Транспортный уровень отвечает за

доставку данных этих сегментов соответствующему прикладному процессу, запущенному на хосте. Рассмотрим пример. Предположим, вы работаете за компьютером и загружаете веб-страницу. При этом на компьютере одновременно запущены FTP-сеанс и два Telnet-сеанса. Итак, имеем четыре работающих прикладных сетевых процесса: два процесса Telnet, один процесс FTP и один процесс HTTP. Когда транспортный уровень вашего компьютера получает данные снизу от сетевого уровня, он должен направить полученные данные одному из этих четырех процессов. Теперь давайте узнаем, как это реализовано.

Для начала вспомним из раздела 2.7, что процесс (как часть сетевого приложения) может иметь один или несколько **сокетов**: «дверей», через которые осуществляется обмен данными между процессами в сети. Таким образом, как показано на рис. 3.2, транспортный уровень принимающего хоста на самом деле направляет данные не напрямую к процессу, а лишь в промежуточный сокет. Поскольку в любой момент времени на принимающем хосте может быть более одного сокета, каждый из них имеет уникальный идентификатор. Формат идентификатора, как мы вскоре увидим, зависит от того, к какому протоколу относится сокет — UDP или TCP.

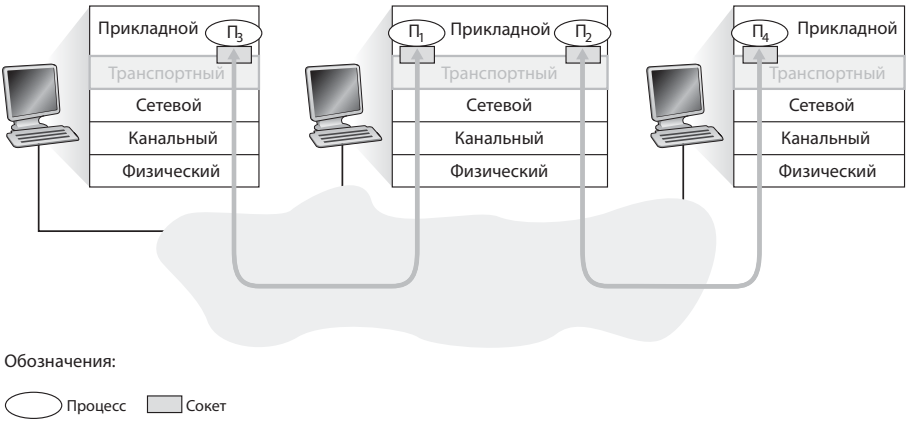


Рис. 3.2. Мультиплексирование и демультиплексирование на транспортном уровне

Теперь рассмотрим, каким образом принимающий хост направляет нужному сокету входящий сегмент транспортного уровня. Для этого каждый сегмент транспортного уровня имеет набор специальных полей. На стороне получателя транспортный уровень проверяет эти поля на соответствие сокету-приемнику, чтобы направить сегмент в нужный

сокет. Эта работа по доставке данных сегмента транспортного уровня нужному, соответствующему сокету называется **демультиплексированием**. Сбор фрагментов данных, поступающих на транспортный уровень хоста-отправителя из различных сокетов, создание сегментов путем присоединения заголовка (который используется при демультиплексировании) к каждому фрагменту и передача сегментов сетевому уровню называется **мультиплексированием**. Заметим, транспортный уровень среднего хоста на рис. 3.2 должен демультиплексировать сегменты, поступающие снизу от сетевого уровня выше к процессам $П_1$ или $П_2$, для чего направляет эти сегменты в соответствующие сокеты процессов. Также транспортный уровень среднего хоста должен собирать исходящие данные этих сокетов, формировать сегменты транспортного уровня и передавать их вниз на сетевой уровень. Хотя мы представили мультиплексирование и демультиплексирование в контексте транспортных протоколов Интернета, важно осознавать, что они имеют значение всякий раз, когда единственный протокол одного уровня (транспортного или любого другого) используется несколькими протоколами на следующем, более высоком уровне.

Чтобы наглядно представить механизм демультиплексирования, вспомним семейную аналогию из предыдущего раздела. Каждый ребенок обладает именем-идентификатором. Билл выполняет операцию демультиплексирования каждый раз, когда получает пачку писем от почтальона, анализируя, кому адресованы письма, а далее собственноручно передавая почту братьям и сестрам. Анна выполняет операцию мультиплексирования, когда она собирает письма у своих братьев и сестер и отдает полученную почту почтальону.

Теперь, когда мы понимаем роль мультиплексирования и демультиплексирования на транспортном уровне, давайте узнаем, как они реализованы на хостах. Из вышесказанного понятно, что для мультиплексирования на транспортном уровне требуются уникальные идентификаторы сокетов и специальные поля в каждом сегменте, указывающие на сокет, которому адресован сегмент. Эти специальные поля, изображенные на рис. 3.3, называются **полем номера порта отправителя** и **полем номера порта получателя**. Сегменты протоколов UDP и TCP имеют и другие поля, которые также будут рассмотрены ниже в данной главе. Номер каждого порта — это 16-разрядное число, принимающее значение из диапазона от 0 до 65535. Номера портов в диапазоне от 0 до 1023 называются **зарезервированными**, то есть они забронированы для использования популярными протоколами прикладного уровня таки-

ми, как протокол HTTP (80 порт) и протокол FTP (21 порт). Список зарезервированных номеров портов приведен в документе RFC 1700, его обновления доступны по адресу **www.iana.org**[499]. Мы должны присваивать номера портов при разработке новых приложений, вроде простого примера в разделе 2.7.

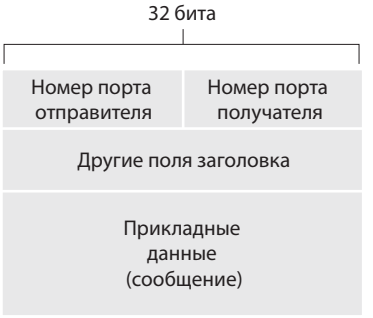


Рис. 3.3. Поля номеров портов отправителя и получателя в сегменте транспортного уровня

Теперь должно быть ясно, каким образом транспортный уровень *может* реализовать службу демультиплексирования. Каждому сокету хоста должен соответствовать номер порта. Когда сегмент поступает на хост, транспортный уровень проверяет номер порта получателя в сегменте и направляет его в соответствующий сокет. Далее данные сегмента передаются через сокет соответствующему процессу. Как мы увидим, подобная схема в основном характерна для протокола UDP. Также мы поймем, что процесс мультиплексирования/демультиплексирования в протоколе TCP еще более тонкий.

Мультиплексирование и демультиплексирование без установления логического соединения

Вспомним из раздела 2.7.1, что программы, написанные на языке Python и запущенные на хосте, могут создавать UDP-сокеты при помощи следующей строки кода:

```
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
```

Когда UDP-сокет создается подобным образом, транспортный уровень автоматически назначает номер порта сокету. Так, в частности, транспортный уровень присваивает сокету номер в диапазоне от 1024

до 65535, который точно не будет использоваться никакими другими UDP-портами хоста. Мы можем назначить номер порта и другим способом: добавим в нашу программу на Python сразу после создания сокета строку, содержащую метод `bind()` для назначения UDP-сокету определенного номера порта, скажем 19157:

```
clientSocket.bind(('', 19157))
```

Если разработчик приложения пишет код, в котором реализует серверную сторону «популярного протокола», тогда он должен назначить соответствующий зарезервированный номер порта. Обычно приложения клиентской стороны позволяют транспортному уровню автоматически (и прозрачно) назначать номер порта, в то время как на серверной стороне приложению присваивается определенный номер порта.

Теперь, когда UDP-сокетам назначены номера портов, мы можем точно описать процедуру мультиплексирования/демультиплексирования для протокола UDP. Предположим, процесс на хосте А с номером UDP-порта 19157 намерен отправить фрагмент данных приложения процессу на хосте Б с номером UDP-порта 46428. Транспортный уровень хоста А создает сегмент, включающий данные приложения, номер порта отправителя (19157), номер порта получателя (46428) и два других значения, которые будут рассмотрены позже, так как не столь важны в настоящем обсуждении. Далее транспортный уровень передает созданный сегмент сетевому уровню. Сформированный сегмент отправляется сетевому уровню, который создает IP-дейтаграмму и «по возможности» доставляет ее хосту Б. Если сегмент поступает на принимающий хост Б, транспортный уровень принимающего хоста проверяет номер порта получателя в сегменте (46428) и доставляет сегмент сокету с номером порта 46428. Заметим, что на хосте Б может быть запущено несколько процессов, каждый с собственным UDP-сокетом и номером порта. Так как UDP-сегменты поступают из сети, хост Б направляет (демультиплексирует) каждый сегмент соответствующему сокету, проверяя номер порта получателя сегмента.

Отметим, что любой UDP-сокет однозначно идентифицируется совокупностью IP-адреса хоста назначения и номера порта. Следовательно, если два UDP-сегмента имеют разные IP-адреса отправителя и/или номера портов отправителя, но одни и те же IP-адрес *получателя* и номер порта *получателя*, то оба сегмента будут направлены одному и тому же процессу получателя, использующему данный сокет.

Неудивительно, если у вас сразу возник вопрос: для чего нужен номер порта отправителя? Как показано на рис. 3.4, в сегменте А-Б номер порта отправителя используется как часть «обратного адреса», когда хост Б должен отправить сегмент обратно хосту А, порт получателя сегмента Б-А принимает свое значение из значения порта отправителя сегмента А-Б. Полный обратный адрес — это IP-адрес хоста А и номер порта отправителя. К примеру, вспомним серверную программу UDP, изученную в разделе 2.7. В программе `UDPServer.py` сервер использует метод `recovfrom()`, чтобы извлечь номер порта клиентской стороны (отправителя) из сегмента, который был получен от клиента; затем программа отправляет новый сегмент клиенту, с принятым номером порта отправителя, служащим в качестве номера порта получателя в этом новом сегменте.

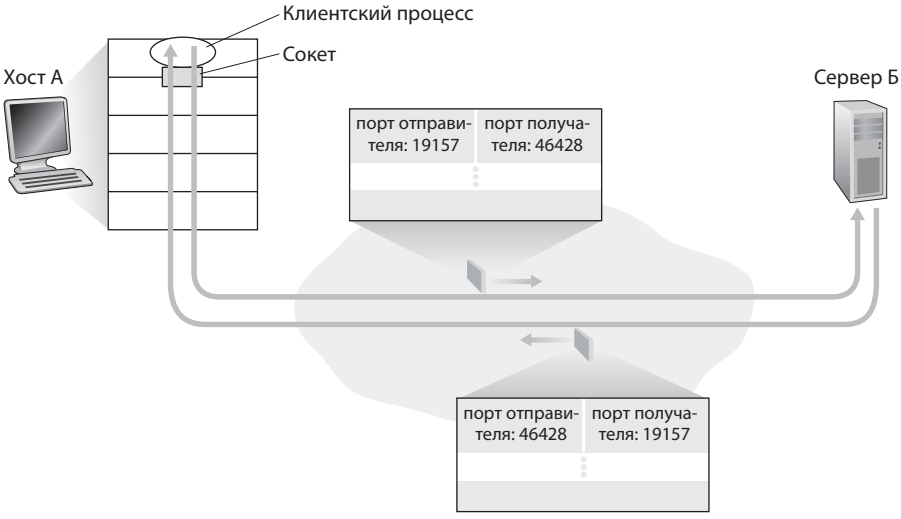


Рис. 3.4. Инверсия номеров портов отправителя и получателя

Мультиплексирование и демультиплексирование с установлением логического соединения

Чтобы усвоить демультиплексирование протокола TCP, мы должны подробнее рассмотреть TCP-сокеты и процесс установления TCP-соединения. Отличие TCP-сокета от UDP-сокета заключается в том, что первый идентифицируется при помощи не двух, а четырех составляющих: IP-адреса отправителя, номера порта отправителя, IP-адреса получателя и номера порта получателя. Таким образом, когда TCP-сегмент

поступает из сети на хост, тот использует все четыре значения, чтобы направить (демультиплексировать) сегмент в соответствующий сокет. В частности, в отличие от протокола UDP, два полученных TCP-сегмента будут иметь различные IP-адреса отправителя или номера портов отправителя, (исключая случай, когда TCP-сегмент содержит первичный запрос на установление соединения) и окажутся направлены в два разных сокета. Далее давайте повторно рассмотрим пример клиент-серверного программирования TCP-сокетов из раздела 2.7.2:

- Серверное приложение TCP имеет «впускающий» сокет, который ожидает запрос на создание соединения от TCP-клиента (см. рис. 2.29) на порт с номером 12000.
- TCP-клиент создает сокет и отправляет сегмент с запросом на создание соединения, для этого используются следующие строки:

```
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,12000))
```

- Запрос на установление логического соединения — это просто TCP-сегмент с портом назначения 12 000 и комбинацией битов в заголовке TCP-сегмента, которая создает соединение (рассматривается в разделе 3.5). Заголовок сегмента также включает номер порта отправителя, который был выбран клиентом.
- Когда операционная система на хосте с запущенным серверным процессом получает входящий сегмент, который содержит запрос соединения с номером порта получателя 12 000, она направляет сегмент серверному процессу, ожидающему его, чтобы принять соединение на порт с номером 12 000. Затем серверный процесс создает новый сокет:

```
connectionSocket, addr = serverSocket.accept()
```

- При обработке сегмента с запросом на установление логического соединения сервер использует четыре параметра: номер порта отправителя сегмента, IP-адрес хоста отправителя, номер порта получателя сегмента и собственный IP-адрес. Вновь созданные сегменты соединения, для которых значения этих четырех полей совпадут со значениями сегмента, устанавливающего соединение, будут направлены (демультиплексированы) в указанный сокет. После того как TCP-соединение установлено, клиент и сервер могут отправлять данные друг другу.

Сервер на хосте может одновременно поддерживать множество TCP-сокетов соединений, каждый из которых связан с процессом и идентифи-

цируется четырьмя приведенными выше параметрами. Когда TCP-сегмент поступает на хост, все четыре поля (IP-адрес отправителя, порт отправителя, IP-адрес получателя, порт получателя) используются для направления (демультиплексирования) сегмента соответствующему сокету.

О БЕЗОПАСНОСТИ

Сканирование портов

Мы увидели, что все серверные процессы открывают порт и ожидают контакта с удаленным клиентом. Некоторые порты зарезервированы для широко известных приложений (например, Всемирная паутина, FTP, DNS, и SMTP-сервер); другие обычно используются популярными приложениями (например, Microsoft 2000 SQL server слушает запросы на порту 1434). Таким образом, если бы мы определили такой порт, открытый на хосте, то мы могли бы присвоить этот порт определенному приложению на хосте. Это очень упрощает работу администраторов вычислительной сети, которые часто интересуются тем, какие сетевые приложения запущены на хостах в их сетях. Но злоумышленники, как правило, также хотят узнать, какие порты открыты на целевом хосте. Если на нем установлено приложение с известной уязвимостью, то этот хост открыт для атаки. Например, SQL server слушающий порт 1434 стал причиной переполнения буфера, позволяющей удаленному пользователю выполнять произвольный код на уязвимом хосте. Эта уязвимость эксплуатировалась червем Slammer[82].

Несложно определить, какое приложение слушает какие порты. Действительно, существует ряд специально предназначенных для этого общедоступных программ, называемых сканерами портов. Возможно, наиболее распространенная из них — это свободно распространяемое приложение nmap, доступное на сайте **nmap.org** и включенное в большинство дистрибутивов Linux. При работе с протоколом TCP программа nmap последовательно сканирует порты, проверяя, для каких из них доступно TCP-соединение. При работе с протоколом UDP программа nmap опять же последовательно сканирует порты, отыскивая UDP-порты, которые отвечают за отправку UDP-сегментов. В каждом случае программа nmap возвращает список открытых, закрытых или недоступных портов. Хост, запустивший программу nmap, может добавить к сканированию любой целевой хост, расположенный *где-либо* в Интернете. Мы вернемся к программе nmap в разделе 3.5.6, когда будем обсуждать управление TCP-соединением.

На рис. 3.5 показана ситуация, в которой хост В инициирует два HTTP-сеанса с сервером Б, а хост А инициирует один HTTP-сеанс

с сервером Б. Хосты А и В, и сервер Б имеют собственные уникальные IP-адреса: А, В, и Б, соответственно. Хост В назначает два разных номера портов отправителя (26145 и 7532) для двух своих HTTP-соединений. Поскольку хост А выбирает номер порта независимо от хоста В, он также может назначить номер порта отправителя 26145 для своего HTTP-соединения. Но это не станет проблемой — сервер Б по-прежнему способен корректно демультиплексировать оба соединения, имеющие одинаковые номера порта отправителя, поскольку каждое из соединений имеет собственный IP-адрес.

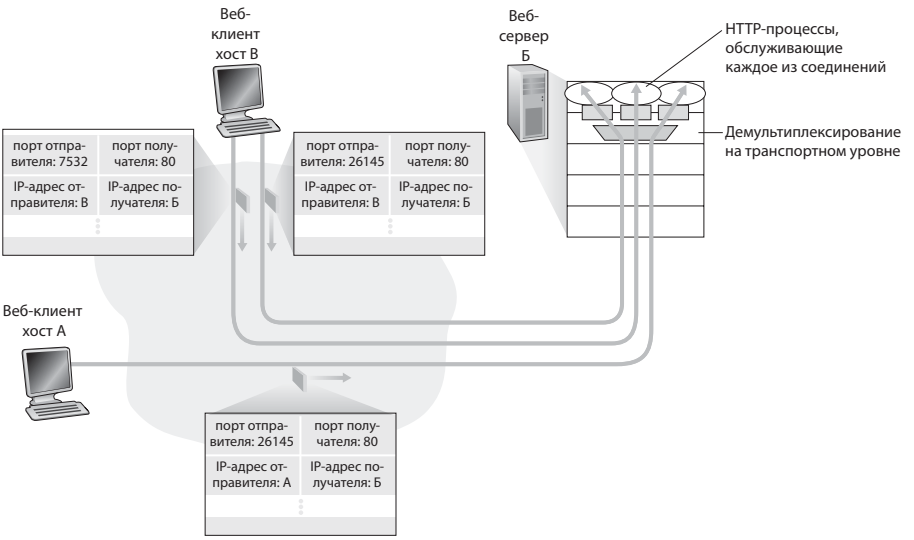


Рис. 3.5. Два клиента, использующие одинаковый порт получателя (80) для соединения с одним веб-сервером приложений

Веб-серверы и протокол TCP

В завершение этой темы мы считаем необходимым дополнительно поговорить о веб-серверах и том, как они используют номера портов. Рассмотрим хост, на 80-м порту которого запущен веб-сервер Apache. Когда клиенты, например браузеры, отправляют сегменты на сервер, *все* сегменты имеют 80-й номер порта получателя. В частности оба начальных создающих соединение сегмента и сегмент, содержащий сообщение HTTP-запроса, будут иметь 80-й порт назначения. Как мы только что описывали, сервер различает сегменты разных клиентов по IP-адресам и номерам портов отправителей.

На рис. 3.5 показан веб-сервер, который порождает новый процесс для каждого соединения. Каждый из этих процессов имеет собственный сокет соединения, через который принимает HTTP-отклики и отправляет HTTP-запросы. Мы уже говорили, что не всегда существует однозначное соответствие между сокетами соединения и процессами. На самом деле сегодня высокопроизводительные сервера часто используют только один процесс и создают для каждого клиентского соединения новый поток с собственным сокетом соединения. Поток можно рассматривать, как подпроцесс внутри основного процесса. При выполнении первого задания по программированию в главе 2, вы создали веб-сервер, который действует именно таким образом. Подобный сервер может одновременно поддерживать множество сокетов соединений для одного процесса, при этом каждый сокет имеет уникальный идентификатор.

Если клиент и сервер используют протокол HTTP с постоянным соединением, то на протяжении всего соединения они обмениваются HTTP-сообщениями, используя один и тот же сокет сервера. В противном случае устанавливается новое TCP-соединение для каждой пары запрос/ответ, которое разрывается после получения ответа. Такое частое открытие и закрытие сокетов способно серьезно повысить нагрузку на веб-сервер (хотя операционные системы могут бороться с этой проблемой). Читателям, которых интересуют вопросы операционных систем, касающиеся постоянных и непостоянных HTTP-соединений, советуем обратиться к публикациям Нильсена[368] и Нахума[362].

Теперь, когда мы обсудили мультиплексирование и демультиплексирование на транспортном уровне, давайте перейдем к обсуждению транспортного протокола UDP, который применяется в Интернете. В следующей части мы увидим, что протокол UDP добавляет еще несколько служб к сетевому протоколу, кроме службы мультиплексирования/демультиплексирования.

3.3. UDP — протокол транспортного уровня без установления соединения

В этом разделе мы подробнее рассмотрим протокол UDP: каким образом он работает и что он делает. Рекомендуем повторить разделы 2.1, где приведен обзор модели обслуживания протокола UDP, и 2.7.1, где обсуждается программирование сокетов для протокола UDP.

Представьте себе, что вам необходимо разработать максимально простой, без лишних функций, протокол транспортного уровня. Как бы вы стали решать эту задачу? Наиболее простым способом является создание такого протокола, который не производит никаких действий с данными. На передающей стороне сообщения приложений неизменными передаются сетевому уровню, а на приемной стороне выполняется отправка сообщений от сетевого уровня прикладному. Ясно, что такой протокол не жизнеспособен: из предыдущего раздела следует, что протоколу как минимум необходимо выполнять операции мультиплексирования и демультиплексирования, обеспечивающие корректный обмен данными между сетевым уровнем и прикладными процессами. Протокол UDP, определенный в стандарте RFC 768[417], выполняет минимум действий, необходимых для протокола транспортного уровня. UDP не добавляет к протоколу IP ничего, за исключением функции мультиплексирования и демультиплексирования. Протокол UDP получает сообщение от процесса приложения, добавляет номера порта отправителя и удаленного порта, необходимые службе мультиплексирования/демультиплексирования, присовокупляет два других небольших поля и передает полученный в результате сегмент сетевому уровню. Сетевой уровень заключает сегмент в дейтаграмму и «по возможности» передает ее хосту назначения. Если сегмент поступает на принимающий хост из сети, протокол UDP использует номер удаленного порта для доставки данных сегмента нужному процессу прикладного уровня. Заметим, что в протоколе UDP отсутствует процедура подтверждения установления соединения на транспортном уровне между отправителем и получателем. По этой причине протокол UDP называют протоколом *без установления логического соединения*.

Примером протокола прикладного уровня, использующего службы протокола UDP, является DNS. Когда DNS-приложение на хосте собирается выполнить запрос, оно создает DNS-запрос и передает его протоколу UDP. Не выполняя рукопожатия с хостом назначения, протокол UDP хоста отправителя добавляет поля заголовка к сообщению и передает получившийся сегмент на сетевой уровень. Сетевой уровень инкапсулирует UDP-сегмент в дейтаграмму и отправляет ее серверу имен. DNS-приложение исходного хоста затем ожидает ответ на этот запрос. Если ответ не получен (возможно, из-за того, что нижестоящий сетевой уровень потерял запрос или отклик), то хост или пытается отправить запрос другому серверу имен, либо информирует вызывающее приложение о том, что получение IP-адреса невозможно.

После приведенных выше рассуждений вполне уместным становится вопрос: есть ли у протокола UDP такие преимущества перед TCP, которые могут заставить разработчика создавать свое приложение с поддержкой UDP, а не TCP? Разве протокол TCP не всегда предпочтительнее, ведь он предоставляет службу надежной передачи данных, в то время как протокол UDP этого не делает? Ответ — нет, поскольку существует много приложений, для которых больше подходит протокол UDP по следующим причинам:

- *Более полный и точный контроль приложения за процессом передачи данных.* Протокол UDP просто упаковывает данные в UDP-сегменты и сразу же отправляет их на сетевой уровень. TCP же, имея механизм управления перегрузками, может снижать темп передачи, если на маршруте образовался затор. Также TCP пытается повторять передачу сегмента, пока не получит подтверждение его приема. Так как приложения реального времени часто не требуют высокой скорости передачи и устойчивы к потере части данных, зато критичны к задержкам, модель обслуживания TCP на практике не очень подходит для них. Как будет обсуждаться ниже, такие приложения могут использовать простой и бесхитростный транспорт UDP, при необходимости реализуя дополнительные функции поверх него, на прикладном уровне.

- *Отсутствует установление соединения.* Как мы увидим позднее, протокол TCP перед началом передачи данных требует тройного рукопожатия. Протокол UDP освобожден от подобной «формальности» и поэтому не вносит дополнительную задержку в процесс передачи. Возможно, именно по этой принципиальной причине DNS-приложения используют протокол UDP, а не протокол TCP — при использовании протокола TCP DNS будет работать значительно медленнее. HTTP-приложения используют протокол TCP, а не UDP, поскольку надежная передача данных более критична для веб-страниц с текстом. Но, как мы кратко обсудили в разделе 2.2, задержка при установлении соединения протокола TCP является важной составляющей задержки при загрузке веб-документов по протоколу HTTP.

- *Не заботится о состоянии соединения.* Протокол TCP поддерживает состояние соединения на конечных системах. Буферы получения и отправки, параметры управления перегрузкой, порядковых номеров и номеров подтверждений — все это , чтобы поддерживать состояние соединения. Мы увидим в разделе 3.5, что эта информация

о состоянии необходима, чтобы реализовать службу надежной передачи данных протокола TCP и предоставить службу управления перегрузкой. Протокол UDP, напротив, не поддерживает состояния соединения и не хранит ни одного из перечисленных параметров. По этой причине сервер, взаимодействующий с конкретным приложением, при работе по протоколу UDP обычно способен поддерживать намного больше клиентов, чем при работе приложения по протоколу TCP.

- *Небольшой заголовок пакета*. TCP сегмент содержит 20 байт заголовка помимо байт данных сегмента, в то время как UDP сегмент использует для заголовка лишь 8 дополнительных байт.

На рис. 3.6 перечислены популярные Интернет-приложения и используемые ими протоколы прикладного и транспортного уровня. Как мы и ожидали, электронная почта, удаленный терминальный доступ, Всемирная паутина и передача файлов выполняются по протоколу TCP. Тем не менее множество важных приложений работают по протоколу UDP. Протокол UDP применяется для обновления RIP таблиц маршрутов (см. раздел 4.6.1). Поскольку обновления RIP отправляются периодически, обычно каждые пять минут, потерянное обновление будет замещено следующим, и такая обработка потерь делает просроченные обновления неактуальными. Протокол UDP также используется для передачи данных сетевого администрирования (SNMP, см. главу 9). В этом случае протокол UDP предпочтительнее протокола TCP, поскольку приложения для администрирования вычислительной сети довольно часто должны работать, когда сеть находится в перегруженном состоянии, в таких ситуациях трудно достичь надежной передачи данных с контролем перегрузки с сохранением качества обслуживания. Как мы упоминали ранее, DNS-приложения работают по протоколу UDP, избегая задержек установления соединения, характерных для протокола TCP.

Как показано на рис. 3.6, оба протокола, и UDP, и TCP, используются сегодня мультимедийными приложениями, такими как Интернет-телефония, видеоконференции в режиме реального времени, и потоковая передача хранимого аудио и видео. В главе 7 мы подробнее рассмотрим эти приложения. Сейчас лишь скажем, что все они выдерживают небольшую потерю данных; иными словами надежная передача не критична для успешного выполнения этих приложений. Более того, приложения, работающие в режиме реального времени, такие, как IP-телефония или видеоконференции, очень плохо реагируют на кон-

троль управления перегрузкой протокола TCP. По этой причине разработчики мультимедийных приложений могут предпочесть протокол UDP протоколу TCP. Однако протокол TCP все больше и больше используется для потоковой передачи мультимедийных данных. Например, согласно результатам конференции ACM Internet Measurement Conference[612], обнаружено, что около 75% вызываемых по требованию потоков и потоков реального времени используют протокол TCP. Поскольку частота потерь пакетов невелика, а в некоторых организациях UDP-трафик блокируется по соображениям безопасности (см. главу 8), все более предпочтительным для передачи мультимедийных данных становится протокол TCP.

Приложение	**Протокол прикладного уровня**	**Нижерасположенный транспортный протокол**
Электронная почта	SMTP	TCP
Удаленный терминальный доступ	Telnet	TCP
Всемирная паутина	HTTP	TCP
Передача файлов	FTP	TCP
Удаленный файловый сервер	NFS	Обычно UDP
Потоковый мультимедийный контент	Обычно проприетарный	UDP или TCP
Интернет-телефония	Обычно проприетарный	UDP или TCP
Сетевое управление	SNMP	Обычно TCP
Протокол маршрутизации	RIP	Обычно TCP
Трансляция имен	DNS	Обычно TCP

Рис. 3.6. Популярные Интернет-приложения и используемые ими протоколы транспортного уровня

Несмотря на широкое применение протокола UDP для мультимедийных приложений, вопрос о его рациональности как минимум остается открытым. Как мы ранее упоминали, протокол UDP не использует службу управления перегрузкой. Но она необходима для защиты сети от перехода в перегруженное состояние, в котором выполняется лишь малая часть необходимой работы. Например, если все пользователи Интернета одновременно станут просматривать потоковое видео с высоким качеством изображения, то процент потерянных вследствие перегрузки пакетов окажется таким большим, что в результате никто ничего не увидит. Кроме того, высокий уровень

потерь связан с нерегулируемой скоростью отправителей протокола UDP, что отличает их от отправителей протокола TCP, которые, как мы видим, *уменьшают* собственную скорость отправки при перегрузке. Таким образом, отсутствие в UDP-соединениях управления перегрузкой может привести к быстрому увеличению количества потерь между UDP отправителями и получателями, и заторам в TCP-сеансах — возможной серьезной проблеме[162]. Многие исследователи предлагают новые механизмы ускорения всех источников, включая источники UDP, для предоставления адаптивного управления перегрузкой[163, 331, 292, 538].

Прежде чем обсуждать структуру UDP-сегмента, оговоримся, что возможность надежной передачи данных для приложений, использующих протокол UDP, *существует*. Для этого механизмы обеспечения надежной передачи (например, рукопожатия и повторных передач, которые мы рассмотрим чуть позже) включают в само приложение. Но это нетривиальная задача, которая может потребовать сложной длительной отладки от разработчика. Тем не менее обеспечение надежной доставки внутри приложения позволяет «попытаться усидеть на двух стульях». То есть процессы приложения могут надежно взаимодействовать независимо от ограничений скорости повторной передачи, накладываемой механизмом управления перегрузкой протокола TCP.

3.3.1. Структура UDP-сегмента

Структура UDP-сегмента, показанная на рис. 3.7, определена в стандарте RFC 768. Данные прикладного уровня размещаются в поле данных UDP-сегмента. Например, для DNS-приложения поле данных содержит или сообщение-запрос, или сообщение-ответ. В случае потокового аудио поле данных заполнено аудио-семплами (элементами аудиоданных). Заголовок UDP-сегмента имеет только четыре поля, каждое из которых состоит из двух байт. Как обсуждалось в предыдущей части, номера портов позволяют хосту-получателю передавать данные приложения соответствующему процессу, запущенному на конечной системе получателе (то есть, предоставляют функцию демультиплексирования). В поле длина указана длина UDP-сегмента (заголовок плюс данные) в байтах. Явное указание значения в поле длины необходимо, поскольку размер поля данных может отличаться для каждого UDP-сегмента. Поле контрольной суммы используется принимающим хостом для проверки на наличие ошибок внутри сегмента. В действи-

тельности, для вычисления контрольной суммы необходимы и некоторые поля IP заголовка в дополнение к UDP-сегменту. Но мы не будем акцентировать на этом внимание сейчас, чтобы понять суть применения контрольной суммы. Вычисление контрольной суммы мы обсудим позже. Основные принципы обнаружения ошибки описаны в разделе 5.2. Поле длина определяет длину UDP-сегмента, включая заголовок, в байтах.

Рис. 3.7. Структура UDP-сегмента

3.3.2. Контрольная сумма UDP

Контрольная сумма UDP предназначена для обнаружения ошибок. Другими словами, контрольная сумма применяется для того, чтобы определить, произошло ли искажение битов UDP-сегмента (например, из-за помех в канале или при хранении на маршрутизаторе) в ходе перемещения от отправителя к получателю. На стороне отправителя протокол UDP формирует первый компонент контрольной суммы как дополнение до единицы суммы всех 16-битных слов сегмента и игнорируя все переполнения. Результат размещается в поле контрольной суммы UDP-сегмента. В этом разделе мы приводим простой пример расчета контрольной суммы. Подробное описание эффективной реализации вычисления контрольной суммы вы можете найти в документе RFC 1071, а примеры вычислений контрольной суммы на реальных данных в публикациях Стоуна[621, 622]. В качестве примера представьте, что у нас есть три следующих 16-битных слова:

0110011001100000
0101010101010101
1000111100001100

Сумма первых двух слов равна:

0110011001100000
0101010101010101
1011101110110101

Сложение полученной суммы с третьим словом дает:

1011101110110101
1000111100001100
0100101011000010

Дополнение до единицы производится заменой всех 0 на 1, и наоборот. Для значения 0100101011000010 это будет значение 1011010100111101, которое и является контрольной суммой. На стороне получателя складываются все четыре слова: три слова сегмента и контрольная сумма. Если ошибок в пакете не было, то сумма, вычисленная на стороне получателя, будет равна 1111111111111111. Если хотя бы один из результирующих битов равен 0, то это значит, что в пакете присутствует ошибка.

Возможно, вызывает удивление, почему протокол UDP в первую очередь проверяет контрольную сумму, если большинство протоколов канального уровня (включая популярный протокол Ethernet) также предоставляют проверку ошибок. Все довольно просто: невозможно гарантировать, что проверка обеспечена на всем пути между источником и получателем; то есть один из каналов, по которым передаются данные, может использовать протокол канального уровня без контроля ошибок. Кроме того, даже если сегмент успешно передан по каналу, не исключено появление ошибок при хранении сегмента в памяти маршрутизатора. Учитывая это, ни надежность канальных соединений, ни обнаружение ошибок в памяти не являются гарантированными, поэтому протокол UDP должен предоставлять средства обнаружения ошибок на транспортном уровне *конечных систем*, если служба передачи данных между конечными системами не предоставляет таких средств. Это пример знаменитого **принципа сквозной связи** в системном дизайне[580], который заключается в необходимости реализации определенной функциональности (в данном случае обнаружение ошибок) на конечных системах: «функции, размещенные на нижних уровнях, могут быть избыточными или малозначимыми в сравнении с расходами на их предоставление на более высоких уровнях».

Поскольку протокол IP может работать в сочетании практически с любым протоколом канального уровня, функция обнаружения оши-

бок на транспортном уровне необходима, чтобы повысить надежность передачи. Хотя протокол UDP выполняет проверку ошибок, он ничего не делает для их исправления. Некоторые реализации протокола UDP просто отклоняют поврежденный сегмент, другие реализации передают его приложению с предупреждением.

На этом мы завершаем обсуждение протокола UDP. Скоро мы увидим, как реализована надежная передача данных приложениям в протоколе TCP, а также и другие дополнительные функции, которых не предоставляет протокол UDP. Приступая к обсуждению протокола TCP, все же было бы полезно отступить на шаг назад и сначала обсудить базовые принципы надежной передачи данных.

3.4. Принципы надежной передачи данных

В этом разделе мы рассмотрим общие принципы надежной передачи данных. Это необходимо, поскольку задача ее реализации решается не только на транспортном, но также на сетевом и прикладном уровнях. Таким образом, вопрос о надежной передаче данных имеет значение для сетей в целом. Если бы кто-нибудь составил список «Топ-10» важных проблем в сетях, эта могла бы его возглавить. Далее мы рассмотрим протокол TCP и покажем, в частности, что протокол TCP использует большинство из принципов, которые мы собираемся описать в этом разделе.

На рис. 3.8 приведена схема надежной передачи данных. Служба абстракции, предоставляемая объектам верхнего уровня, обеспечивает надежный канал, через который могут быть переданы данные. При использовании надежного канала биты передаваемых данных не окажутся повреждены (изменены с 0 на 1 или наоборот) или потеряны, и будут доставлены в том порядке, в котором были отправлены. Это именно та модель обслуживания, которую предоставляет TCP для интернет-приложений, использующих его.

Реализация такой абстракции выполняется в **протоколе надежной передачи данных (reliable data transfer, rdt)**. Задача усложняется, поскольку на самом деле уровень расположенный *ниже* уровня протокола надежной передачи данных может быть ненадежным. Например, TCP — это протокол надежной передачи данных, реализованный над ненадежным протоколом межхостового IP-соединения сетевого уровня. Итак, ниже уровня, предоставляющего надежное соединение между двумя конечными точками, может располагаться единственный физический ка-

нал (как в случае протокола передачи данных канального уровня) или глобальная сеть (как в случае протокола транспортного уровня). Для достижения наших целей мы можем рассматривать этот нижний уровень просто как ненадежный канал между двумя хостами.

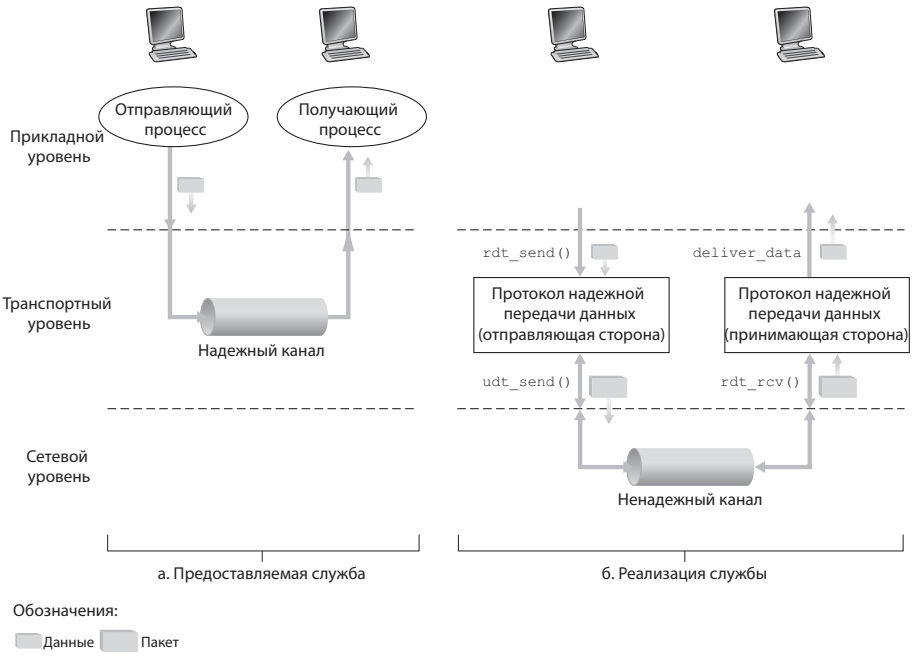


Рис. 3.8. Гарантированная доставка данных: модель обслуживания и ее реализация

В этом разделе мы пошагово разработаем узел-отправитель и узел-получатель, обменивающиеся информацией по протоколу надежной передачи данных. Рассмотрим ряд моделей, в которых структура базового канала будет постепенно усложняться. Например, исследуем, какие механизмы протоколов требуется задействовать в случаях, когда базовый канал может повреждать разряды или терять целые пакеты. Одно из допущений, на которое мы будем опираться ниже, таково: пакеты прибывают на узел-получатель в том же порядке, в котором уходят с узла-отправителя, причем некоторые могут быть потеряны. Таким образом, канал не будет переупорядочивать пакеты. На рис. 3.8 представлены интерфейсы, применяемые в нашем протоколе передачи данных. Узел-отправитель будет активироваться путем вызова метода `rdt_send()`, этот вызов выполняется с вышестоящего уровня. Аббревиатура `rdt` здесь означает «надежная передача данных», а `_send` указывает, что вы-

зов поступает на узел-отправитель. Как только пакет прибудет по каналу на сторону получателя, узел-адресат вызовет метод `rdt_rcv()`. Когда протоколу `rdt` требуется передать данные на уровень выше, эта задача решается путем вызова метода `deliver_data()`. Далее мы будем пользоваться термином «пакет», хотя на транспортном уровне подобная информационная единица называется «сегмент». Дело в том, что учебный материал из этого раздела универсален для всей теории компьютерных сетей, его применение не ограничивается транспортным уровнем Интернета. Поэтому мы полагаем, что здесь уместен более общий термин «пакет».

Здесь мы обсудим только случай **однонаправленной передачи данных**, то есть передачи данных от отправляющей к принимающей стороне. Случай надежной **двунаправленной** (так называемой полнодуплексной) **передачи данных** принципиально не намного сложнее, но более труден для объяснения. Хотя мы обсуждаем только однонаправленную доставку данных, важно отметить, что отправляющая и принимающая стороны нашего протокола будут столь же необходимы для передачи пакетов в *обоих* направлениях, как указано на рис. 3.8. Мы вскоре убедимся и в том, что принимающей и передающей сторонам `rdt` необходимо, кроме данных, обмениваться также различной управляющей информацией. Обе стороны протокола `rdt` и отправляющая, и принимающая отсылают пакеты противоположной стороне при помощи метода `udt_send()`, где `udt` — сокращение от «ненадежная передача данных» (unreliable data transfer).

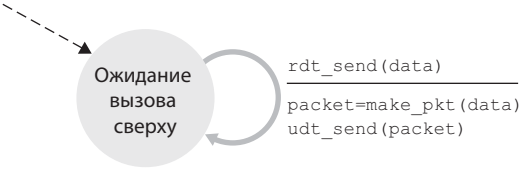
3.4.1. Создание протокола надежной передачи данных

Теперь мы пройдем через создание серии протоколов, в которой каждый следующий будет сложнее предыдущего, постепенно приближаясь к безупречному протоколу надежной передачи данных.

Надежная передача данных по совершенно надежному каналу: `rdt1.0`

Для начала мы рассмотрим простейший случай, в котором нижерасположенный канал тривиален. Схема конечных автоматов **FSM** (finite-state machine) для отправителя и получателя протокола `rdt1.0` приведена на рис. 3.9. FSM-схема на рис. 3.9 (а) определяет действие отправителя, а FSM-схема на рис. 3.9 (б) определяет действие получателя. Важно заметить, что FSM-схемы для отправителя и получателя

различны. Стрелки в FSM-схемах обозначают переходы протокола из одного состояния в другое. Поскольку каждая FSM-схема на рис. 3.9 имеет только одно состояние, то данный переход производится от конца состояния к его началу; вскоре мы увидим более сложные схемы. Событие, вызывающее переход, отражено в надписи над горизонтальной линией перехода, а действие, выполняемое при наступлении события, показано под горизонтальной линией. В случае если не выполняется никаких действий при наступлении события или не наступает никакого события, но выполняется действие, мы будем использовать символ Λ над или под горизонтальной линией, соответственно, чтобы явно обозначить отсутствие события или действия. Начальное состояние FSM-схемы будем обозначать пунктирной стрелкой. Поскольку FSM-схемы на рис. 3.9 имеют только одно состояние каждая, ниже мы рассмотрим и другие FSM-схемы имеющие множество состояний, где определение начального состояния является важным.



а. `rdt1.0`: Отправляющая сторона

б. `rdt1.0`: Принимающая сторона

Рис. 3.9. `rdt1.0`. Протокол для полностью надежного канала

Отправляющая сторона протокола `rdt` просто принимает данные от верхнего уровня, используя событие `rdt_send(data)`, создает содержащий их пакет (действие `make_pkt(data)`) и отправляет его в канал. На практике событие `rdt_send(data)` может быть результатом вызова процедуры (например, `rdt_send()`) верхним уровнем приложений.

На принимающей стороне протокол `rdt` принимает пакет от нижнего канального уровня, используя событие `rdt_rcv(packet)`, извлекает данные из пакета (действие `extract(packet,data)`) и передает их на верхний уровень (действие `deliver_data(data)`). На практике, событие `rdt_rcv(packet)` может быть результатом вызова процедуры (например, `rdt_rcv()`) протоколом нижнего уровня.

В этом простом протоколе не существует различий между блоком данных и пакетом. Также весь поток пакетов от отправителя к получателю по совершенно надежному каналу не требует от принимающей стороны предоставления какого-либо отклика отправителю, поскольку ничего не может пойти неверно! Заметим, мы также предполагаем, что получатель способен принимать данные с той же скоростью, с которой отправитель их посылает. Это позволяет не разрабатывать механизм для снижения скорости передачи по требованию принимающей стороны протокола.

Надежная передача данных по каналу с возможными ошибками бит: `rdt2.0`

Перейдем к более реалистичной модели нижерасположенного канала, который может повреждать биты в пакете. Такие ошибки бит обычно происходят в физических компонентах сети, таких как передача, распространение или буферизация пакета. Мы по-прежнему будем использовать допущение о том, что все переданные пакеты получены (хотя их биты могут быть повреждены) в том порядке, в котором они были отправлены.

Перед тем как начать разработку протокола, обеспечивающего надежную передачу данных по описанному каналу, представим себе следующую аналогию. Предположим, что вы диктуете вашему знакомому длинное сообщение по телефону. Каждый раз после того, как продиктованное предложение принято и записано на бумагу, ваш знакомый говорит: «Да!» Если он вас не понял, то просит повторить предложение еще раз. Этот протокол передачи голосовых сообщений содержит **положительные** («Да!») и **отрицательные** («Повтори это еще раз!») **квитанции**. Квитанции служат для того, чтобы принимающая сторона могла уведомлять передающую о том, содержатся ли искажения в каждом из принятых предложений. В сфере компьютерных сетей протоколы надежной передачи данных, обладающие подобным механизмом многократного повторения передачи, называются протоколами с **автоматическим запросом повторной передачи** (Automatic Repeat reQuest, **ARQ**).

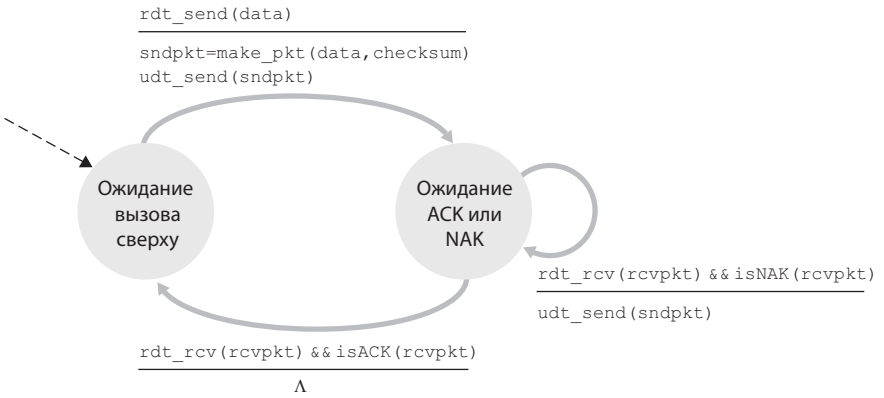
Для разрешения проблем искажения битов в ARQ-протоколах используются три дополнительных механизма:

- *Обнаружение ошибки*. Как следует из названия, данный механизм позволяет определять наличие искаженных битов в принятых данных. В предыдущем разделе было показано, что протокол UDP использует для этой цели значение контрольной суммы. Методы обнаружения и исправления ошибок мы подробно рассмотрим в главе 5; сейчас нам необходимо знать лишь о том, что они основаны на передаче специальных дополнительных битов, не входящих в информационную часть пакета. Эти биты будут помещены в поле контрольной суммы протокола `rdt2.0`.
- *Обратная связь с передающей стороной*. Поскольку приемная и передающая стороны находятся на разных конечных системах, возможно, разделенных тысячами километров, единственный способ уведомить передающую сторону о результате передачи пакетов заключается в организации обратной связи, исходящей от принимающей стороны. Обратная связь, как и в примере с телефонным сообщением, заключается в посылке положительных (ACK) или отрицательных (NAK) квитанций. Минимальная длина квитанции составляет 1 бит, поскольку двух различных значений (0 и 1) достаточно, чтобы указать исход передачи.
- *Повторная передача*. Пакет, при передаче которого были зафиксированы ошибки, подлежит повторной отправке передающей стороной.

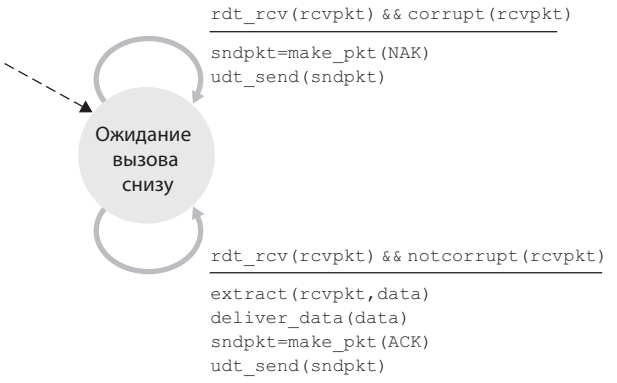
На рис. 3.10 изображены схемы конечных автоматов для протокола `rdt2.0`, осуществляющего обнаружение ошибок, а также передачу положительных и отрицательных квитанций.

Автомат передающей стороны имеет два состояния. Состояние *а* соответствует ожиданию передачи данных от верхнего уровня. При наступлении события `rdt_send(data)` передающая сторона создает пакет `sndpkt`, включающий данные и поле контрольной суммы (например, вычисляемой по аналогии с протоколом UDP, как описано в предыдущем разделе), и отсылает его приемной стороне методом `udt_send(sndpkt)`. Состояние *б* соответствует ожиданию квитанции от приемной стороны. В случае получения квитанции ACK, то есть наступления события `rdt_rcv(rcvpkt) && isACK (rcvpkt)`, передающая сторона переходит в состояние ожидания данных от верхнего уровня. Если квитанция оказывается отрицательной (NAK), происходит повторная передача пакета и ожидание ее результата (квитанции).

Важной особенностью передающей стороны является то, что, находясь в состоянии ожидания квитанции, она не может принимать данные от верхнего уровня; прием новой порции данных возможен только после получения положительной квитанции для текущего пакета. Таким образом, безошибочная передача предыдущего пакета является необходимым условием для начала передачи следующего пакета. Протоколы, функционирующие подобным образом, называют **протоколами с ожиданием подтверждений** (stop-and-wait).



а. `rdt2.0`: сторона отправителя



б. `rtd2.0`: сторона получателя

Рис. 3.10. `rdt2.0` — передача данных, допускающая искажения битов

FSM-схема стороны получателя протокола `rdt2.0` имеет единственное состояние. При поступлении пакета получатель отправляет пакет ACK или NAK, в зависимости от того, был ли поврежден пакет.

На рис. 3.10 обозначения соответствуют событию, при котором пакет был получен, и в нем была найдена ошибка.

Со стороны протокол `rdt2.0` может выглядеть работоспособным, но, к сожалению, он имеет важный недостаток, заключающийся в незащищенности квитанций от возможных искажений! Перед тем как двигаться дальше, обдумайте возможные пути решения этой проблемы. Этот порок, к сожалению, гораздо серьезней, чем кажется на первый взгляд. Для его устранения нам необходимо как минимум включить в квитанцию поле контрольной суммы. Нетривиальным также является решение вопроса о том, каким образом протокол должен действовать при наличии ошибок в квитанциях, поскольку принимающая сторона в этом случае не получает никакой информации о результатах передачи последнего пакета.

Существует три способа обработки поврежденных ACK или NAK:

- Вернемся к примеру с телефонным сообщением и представим себе, как два человека могут решить подобную проблему. Если передающий абонент не расслышал фразу «Да!» или «Повтори это еще раз!», он может обратиться к принимающему абоненту с фразой «Что ты сказал?», являющейся новым типом пакета в протоколе. С другой стороны, как поступить, если и она не будет расслышана? Не понимая, является ли эта фраза новым предложением, принимающий, вероятно, ответит фразой «Что *ты сказал*?»; в свою очередь, она также может быть не расслышана. Очевидно, что описанный способ решения проблемы является весьма громоздким и ненадежным.
- Можно добавить в квитанции некоторое количество контрольных битов, достаточное не только для обнаружения, но и для исправления ошибок. Этот способ решает поставленную проблему в случае, если канал только искажает данные, но не теряет их.
- Можно выполнить обычную повторную передачу пакета, приравняв поврежденные квитанции к отрицательным. Такой подход приводит к **дублированию пакетов**. Основная проблема здесь заключается в том, что принимающая сторона не может определить, положительная или отрицательная квитанция на получение предыдущего пакета была отправлена ей в ответ. Следовательно, она также *не может* определить, является ли последний пакет новым или имела место повторная передача.

Простое решение приведенной задачи, используемое во многих современных протоколах, включая TCP, состоит в добавлении в пакет дан-

ных нового поля, содержащего **порядковый номер** пакета. Порядковый номер формируется передающей стороной, осуществляющей подсчет передаваемых пакетов. Для того чтобы определить, является ли последний принятый пакет новым, принимающей стороне достаточно лишь проанализировать значение порядкового номера. В случае нашего простого протокола роль порядкового номера может играть единственный бит, сохраняющий значение для повторно посылаемого пакета и изменяющий значение на противоположное при передаче нового пакета. Поскольку мы создаем протокол, предполагая, что потеря данных в канале невозможна, включать в квитанции порядковый номер пакета, к которому они относятся, нет необходимости. Передающая сторона всегда получает квитанцию, соответствующую последнему переданному пакету.

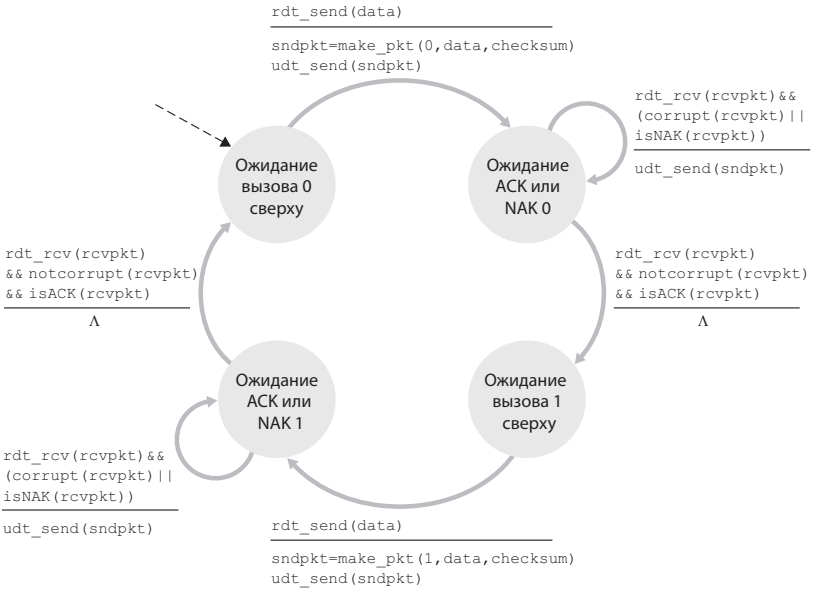


Рис. 3.11. Отправитель протокола `rdt2.1`

На рис. 3.11 и 3.12 приведены схемы конечных автоматов для протокола `rdt2.1`, являющегося исправленной версией `rdt2.0`. Оба автомата теперь имеют вдвое больше состояний по сравнению с протоколом `rdt2.0`. Это объясняется тем, что необходимо различать состояния протокола, соответствующие двум возможным порядковым номерам (0 и 1) принимаемого/передаваемого пакета. Обратите внимание на то, что действия при приеме/передаче пакета с порядковым номером 0 являются зеркальным отображением действий при приеме/передаче паке-

та с порядковым номером 1; единственное различие заключается в обработке порядкового номера.

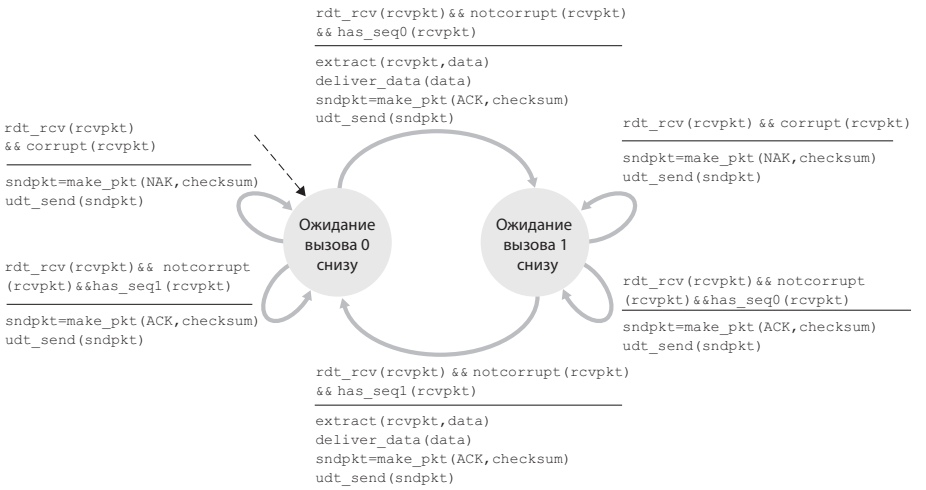


Рис. 3.12. Получатель протокола `rdt2.1`

Протокол `rdt2.1` использует как положительные, так и отрицательные квитанции, направляемые от получателя к отправителю. При поступлении пакета, идущего не по порядку следования, получатель отправляет положительную квитанцию на принятый пакет. Если приходит поврежденный пакет, получатель отправляет на него отрицательную квитанцию. Мы можем добиться эффекта NAK, если перешлем ACK для последнего принятого пакета. Если передающая сторона получит две положительные квитанции для одного и того же пакета (то есть произойдет удвоение положительных квитанций), то это будет указывать на наличие ошибок в пакете, следующем за тем, для которого были получены сдвоенные квитанции. Модель протокола `rdt2.2`, осуществляющего надежную передачу по каналу, допускающему искажения битов без отрицательного рукопожатия, приведена на рис. 3.13 и 3.14. Единственное различие между протоколами `rdt2.1` и `rdt2.2` заключается в том, что на этот раз получатель должен добавлять порядковый номер пакета, на который выслано подтверждение, в ACK-пакет (для этого используется аргумент 0 или 1 в действии `make_pkt()` в FSM-схеме получателя), и отправитель теперь должен проверять порядковый номер пакета, на который получено подтверждение в виде ACK-пакета получателя (что выполняется добавлением аргумента 0 или 1 в действие `isACK()` в FSM-схеме отправителя).

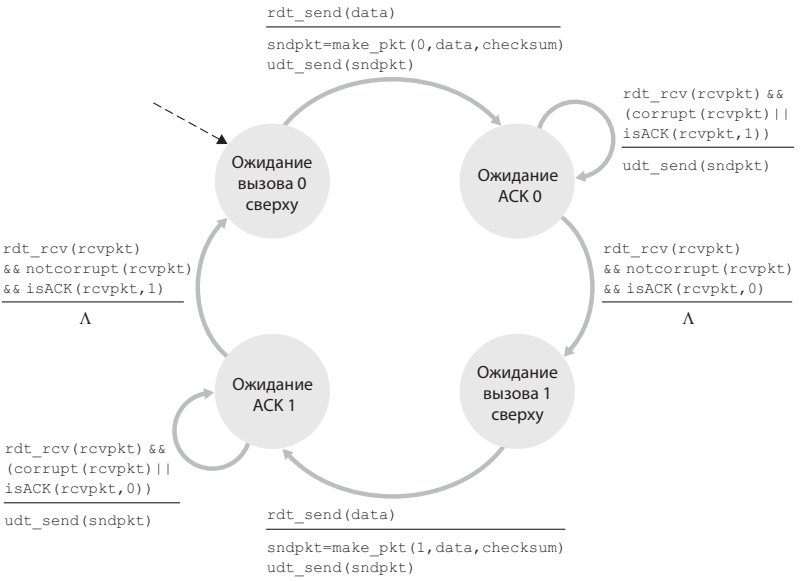


Рис. 3.13. Отправитель протокола `rdt2.2`

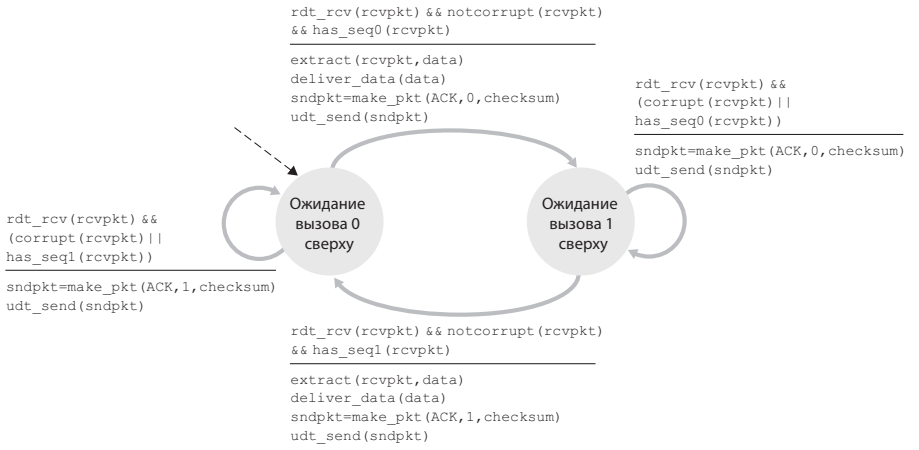


Рис. 3.14. Получатель, работающий по протоколу `rdt2.2`

Надежная передача данных по ненадежному каналу, допускающему искажение и потерю пакетов: `rdt3.0`

Теперь предположим, что нам необходимо обеспечить передачу данных по каналу, в котором возможны не только искажения, но и *потери* пакетов (такая ситуация вполне типична для современных компьютер-

ных сетей, включая Интернет). При разработке протокола нам придется решить две дополнительные задачи: найти способ определить факт потери пакета и указать действия, предпринимаемые в этом случае. Последняя задача решается с помощью контрольных сумм, порядковых номеров, квитанций и повторных посылок — механизмов, реализованных ранее в протоколе `rdt2.2`. Для решения первой задачи нам потребуется применение новых механизмов.

Существует множество подходов к решению проблемы, связанной с потерей пакетов (некоторые будут рассмотрены как дополнительные в упражнениях, приведенных в конце главы). Сейчас нас будут интересовать определение факта потери пакетов, а также методы доставки потерянных пакетов принимающей стороне. Предположим, что при передаче пакета происходит потеря либо его самого, либо квитанции, сгенерированной для этого пакета принимающей стороной. В обоих случаях квитанция не будет получена передающей стороной. Передающая сторона может продолжать ожидание в течение какого-либо промежутка времени, по окончании которого *посчитает* пакет потерянным и выполнит его повторную передачу. Вы должны убедиться самостоятельно, что этот протокол действительно работает.

Возникает вопрос о том, насколько долгим должно быть ожидание. Очевидно, что время ожидания складывается из времени оборота между передающей и принимающей сторонами (возможно, включая промежуточную буферизацию в маршрутизаторах) и времени обработки пакета принимающей стороной. В большинстве компьютерных сетей оценка максимума времени ожидания, особенно с высокой точностью, весьма трудоемка. Кроме того, желательно разрешить проблему передачи потерянного пакета за короткое время, а ожидание в течение максимального времени получения квитанции оказывается слишком долгим. На практике интервал ожидания делают более коротким, исходя из предположения, что вероятность потери пакета весьма велика (хотя и не абсолютна). По истечении этого интервала происходит повторная передача пакета. Поскольку существует ненулевая вероятность получения квитанции для пакета после его повторной передачи, становится возможным **дублирование пакетов**. Эта проблема уже решена нами в протоколе `rdt2.2` с помощью порядковых номеров.

Приведенный механизм является панацеей для передающей стороны: ей не нужно знать о том, потерян ли пакет, потеряна ли его квитанция или ничего не потеряно, но квитанция пришла с задержкой. Во всех трех случаях производится одно и то же действие: повторная

передача пакета. Для контролирования времени в данном механизме используется **таймер отсчета**, который позволяет определить окончание интервала ожидания. Передающей стороне необходимо запускать таймер каждый раз при передаче пакета (как при первой, так и при повторной), обрабатывать прерывания от таймера и останавливать его.

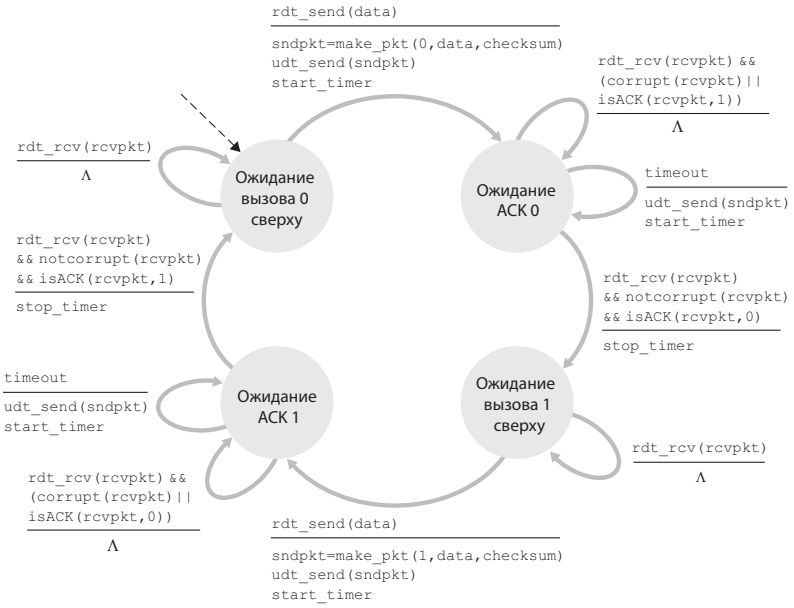


Рис. 3.15. Отправитель протокола `rdt3.0`

На риунке 3.15 изображена FSM-схема отправителя протокола `rdt3.0`, надежно передающего данные по каналу с возможными битовыми ошибками данных или потерей пакетов; в упражнениях вы найдете задание по составлению FSM-схемы получателя протокола `rdt3.0`. Рис. 3.16 показывает, каким образом протокол обрабатывает пакеты, которые не были потеряны и не превысили время ожидания, и каким образом обрабатывается потеря пакетов. На рис. 3.16 временная шкала направлена сверху вниз; заметим, что момент получения пакета будет неизбежно позже момента его отправки, поскольку требуется время на передачу данных, а также возможны задержки из-за перегрузок сети. На рис. 3.16 (б) — (г) скобки на стороне отправителя показывают моменты времени, когда был запущен и остановлен таймер. Некоторые более тонкие подробности данного протокола разобраны в упражнениях в конце этой главы. Так как порядковые номера пакетов равны либо

0, либо 1, то иногда протокол `rdt3.0` называют **протоколом с чередованием битов**.

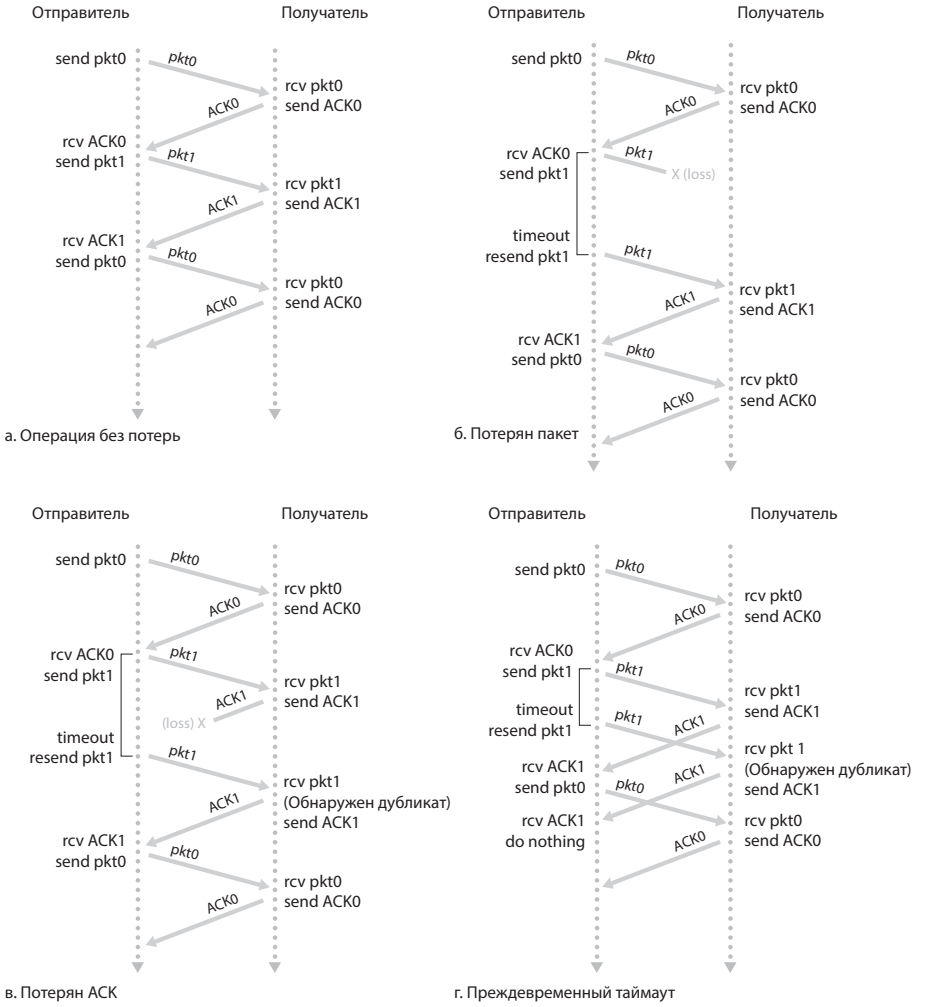


Рис. 3.16. Процедура протокола `rdt3.0`

Итак, мы ознакомились с ключевыми понятиями протоколов надежной передачи данных: контрольными суммами, порядковыми номерами пакетов, таймерами, положительными и отрицательными квитанциями.

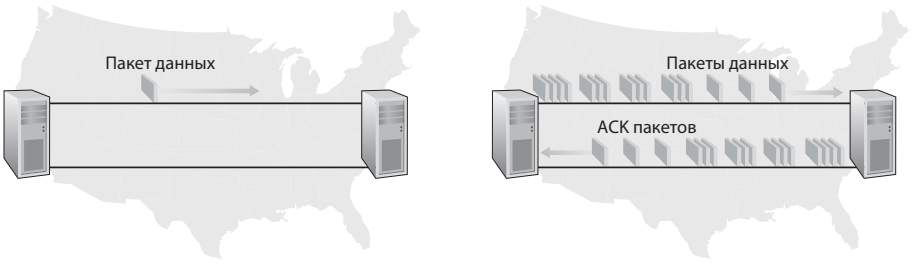
Каждый из элементов выполняет свою ключевую и необходимую роль в протоколе. Теперь у нас есть работающий протокол надежной передачи данных!

3.4.2. Протокол надежной передачи данных с конвейеризацией

Протокол `rdt3.0` является корректным с точки зрения функционирования, однако вряд ли нашлось бы много пользователей, которых бы устроило качество обслуживания этого протокола, особенно в современных высокоскоростных компьютерных сетях. Главной проблемой является то, что он относится к протоколам с ожиданием подтверждений.

Для того чтобы лучше понять последствия ожидания, представим себе следующую ситуацию: один хост расположен на западном побережье США, а другой — на восточном, как показано на рис. 3.17. Время оборота RTT между этими двумя хостами при распространении сигнала со скоростью света приблизительно равно 30 мс. Предположим далее, что хосты соединены каналом со скоростью передачи R, равной 1 Гбит/с (10^9 бит/с). При условии, что размер пакета L, включая заголовок и данные, равен 1000 байт или 8000 бит, время необходимое на доставку пакета по гигабитному каналу равно:

$$d_{\text{передача}} = \frac{L}{R} = \frac{8000 \text{ бит/пакет}}{10^9 \text{ бит/с}} = 8 \text{ мкс}$$



а. Протокол с остановкой и ожиданием в действии

б. Конвейерный протокол в действии

Рис. 3.17. Версия протокола с ожиданием подтверждений и с конвейеризацией

На рис. 3.18а показано, что при использовании нашего протокола с ожиданием подтверждений в случае, если отправитель начинает посылать пакеты в момент времени $t = 0$, то последний бит поступит в канал отправителя в момент времени $t = L/R = 8$ мкс. Далее пакету требуется 15 мс на пересечение страны, и последний бит пакета появится у получателя в момент времени $t = RTT/2 + L/R = 15{,}008$ мс. Для простоты предположим, что пакеты подтверждений крайне малы (настолько, что мы можем пренебречь временем их передачи), и допустим, что получатель может посылать подтверждения сразу, как только будет принят последний бит пакета данных, тогда подтверждение появится у отправителя

в момент времени $t = RTT + L/R = 30{,}008$ мс. В этот момент отправитель может начать передачу следующего сообщения. Таким образом, из 30,008 мс отправитель занят непосредственно отправкой только 0,008 мс. Если мы определим «коэффициент полезного действия» отправителя (или канала) как долю времени, в которую отправитель действительно занят отправкой данных в канале, анализ, приведенный на рис. 3.18а показывает, что протокол с ожиданием подтверждений имеет весьма печальный «КПД» отправителя:

$$d_{\text{отправитель}} = \frac{L/R}{RTT + R/L} = \frac{0{,}008}{30{,}008} = 0{,}00027$$

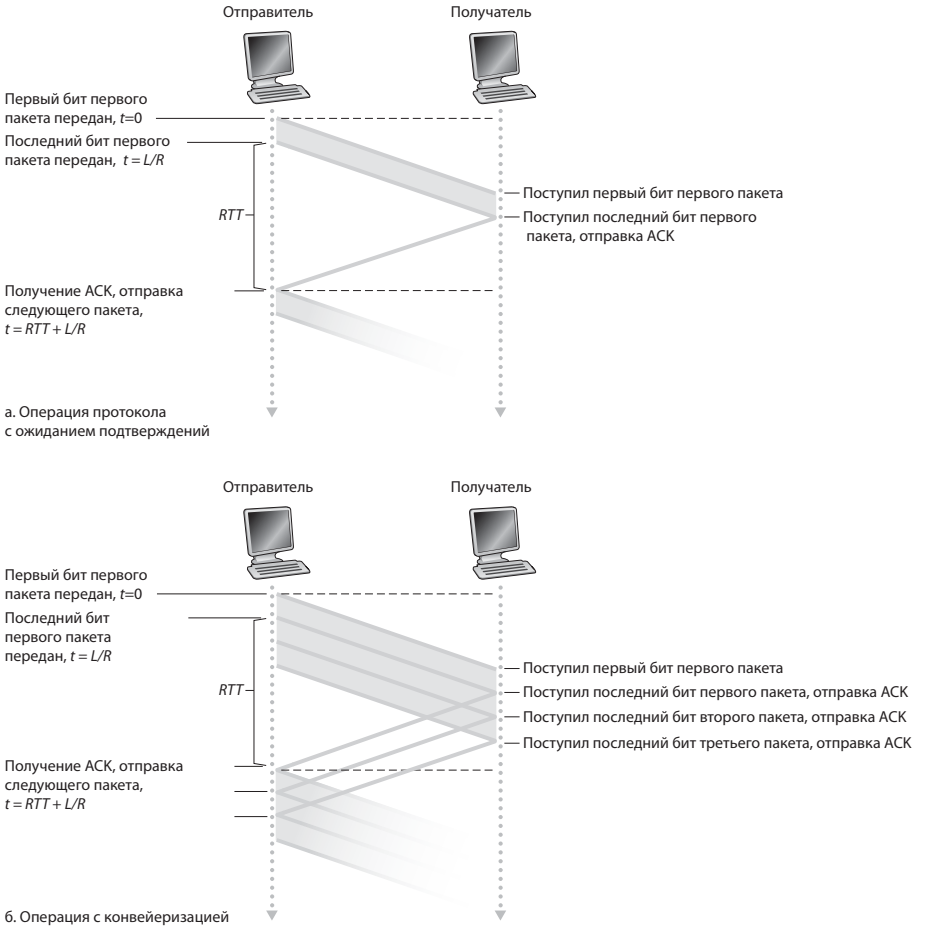


Рис. 3.18. Отправка с ожиданием подтверждений и отправка с конвейеризацией

То есть отправитель был занят только 2,7 сотых от одного процента времени! Кроме того, наши расчеты говорят о том, что передача

данных ведется со скоростью 1000 байт за 30,008 мс, что составляет 267 Кбит/с — величину, в тысячи раз меньшую скорости, обеспечиваемой каналом связи.

Представьте, каким разочарованием для администратора вычислительной сети было бы закупить дорогостоящую линию связи и получить столь низкое качество обслуживания! Этот пример наглядно демонстрирует, каким образом сетевые протоколы могут «тормозить» передачу, обеспечиваемую аппаратно.

Обратите внимание на то, что мы не учли время, затрачиваемое на обработку данных протоколами более низких уровней передающей и принимающей сторон, а также задержки обработки и ожидания, которые могли иметь место при передаче пакетов. Включение этих факторов в модель показало бы еще большую несостоятельность нашего протокола с точки зрения полезности.

Решение конкретно этой описанной проблемы достаточно простое: вместо передачи в стиле протокола с ожиданием подтверждений, отправитель может посылать несколько пакетов без ожидания подтверждений, как показано на рис. 3.17а. Рисунок 3.18б демонстрирует, что передача трех пакетов подряд приводит к трехкратному повышению полезности по сравнению с нашим протоколом. Поскольку пакеты, отправляемые группами, удобно изображать в виде конвейера, данный способ передачи получил название передачи с **конвейеризацией**. Применение конвейеризации в протоколах надежной передачи данных приводит к следующим последствиям:

- Увеличивается диапазон порядковых номеров, поскольку все отсылаемые пакеты (за исключением повторных передач) должны быть однозначно идентифицируемы.
- Появляется необходимость в увеличении буферов на передающей и принимающей сторонах. Так, размер буфера передающей стороны должен быть достаточным для того, чтобы хранить все отправленные пакеты, для которых ожидается получение квитанций. На принимающей стороне может возникнуть необходимость в буферизации успешно принятых пакетов, о чем будет сказано далее.
- Диапазон порядковых номеров и требования к размерам буферов зависят от действий, предпринимаемых протоколом в ответ на искажение, потерю и задержку пакета. В случае конвейеризации существуют два метода исправления ошибок: **возвращение на *N* пакетов назад** и **выборочное повторение**.

3.4.3. Возвращение на *N* пакетов назад (протокол GBN)

В протоколе **Go-Back-N (GBN)** отправитель может одновременно передавать несколько доступных пакетов данных, не ожидая подтверждения, максимальное число неподтвержденных пакетов не должно превышать числа *N*. В этом разделе мы опишем протокол GBN. Настоятельно рекомендуем сначала поэкспериментировать с нашим замечательным апплетом для протокола GBN, доступным на сайте **tinyurl.com/odtpgrk**.

На рис. 3.19 показан диапазон порядковых номеров отправителя протокола GBN. Если мы определим старший порядковый номер неподтвержденного пакета как `base`, и наименьший неиспользуемый порядковый номер `nextseqnum`, то есть порядковый номер следующего отправленного пакета, то можно выделить четыре интервала в диапазоне порядковых номеров. Порядковые номера в интервале `[0,base-1]` соответствуют пакетам, которые уже были переданы и на них получены подтверждения. Интервалу `[base,nexseqnum-1]` соответствуют пакеты, которые уже были отправлены, но на них еще не получено подтверждения. Порядковые номера в интервале `[nexseqnum, base+N-1]` назначаются пакетам, которые могут быть незамедлительно отправлены, как только поступят данные с верхнего уровня. Наконец, порядковые номера большие или равные `base+N` не могут быть использованы, пока текущий неподтвержденный пакет в конвейере (точнее говоря, пакет с порядковым номером `base`) не будет подтвержден.

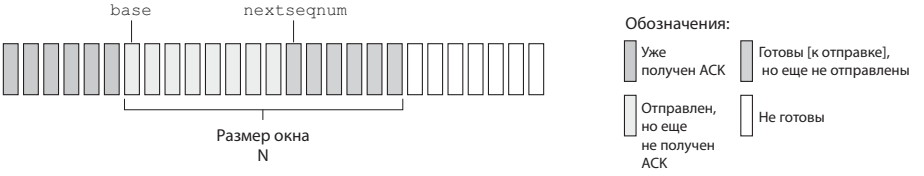


Рис. 3.19. Диапазон порядковых номеров отправителя протокола Go-Back-N

Как понятно по рис. 3.19 диапазон доступных порядковых номеров для отправленных, но еще не подтвержденных пакетов можно представить как «окно» размера *N* в диапазоне порядковых номеров. Во время работы протокола оно скользит вперед по пространству порядковых номеров. По этой причине число *N* означает **размер окна,** и протокол GBN называется **протоколом скользящего окна**. Должно быть вы удивлены, почему мы в первую очередь ограничили числом *N* максимальное количество неотправленных, неподтвержденных пакетов. Почему бы

не оставить их количество неограниченным? В разделе 3.5 мы увидим, что одной из причин является управление потоком. Изучая управление перегрузками в разделе 3.7, мы рассмотрим другую причину.

На практике порядковый номер пакета размещен в поле фиксированного размера в заголовке пакета. Если k — это число бит в поле порядкового номера пакета, тогда диапазон порядковых номеров $[0, 2^k-1]$. Поскольку используется ограниченный диапазон порядковых номеров, все арифметические действия с ними должны выполняться по модулю 2^k, то есть пространство порядковых номеров можно представить в виде кольца размера 2^k, где за порядковым номером 2^k-1 непосредственно следует порядковый номер 0. Напомним, что в протоколе rdt3.0 используется однобитный порядковый номер и диапазон порядковых номеров [0,1]. В нескольких задачах в конце этой главы рассматривается ограниченная последовательность порядковых номеров. В разделе 3.5 мы увидим, что протокол TCP использует 32-битные порядковые номера, которые применяются для подсчета количества байт в потоке, а не пакетов данных.

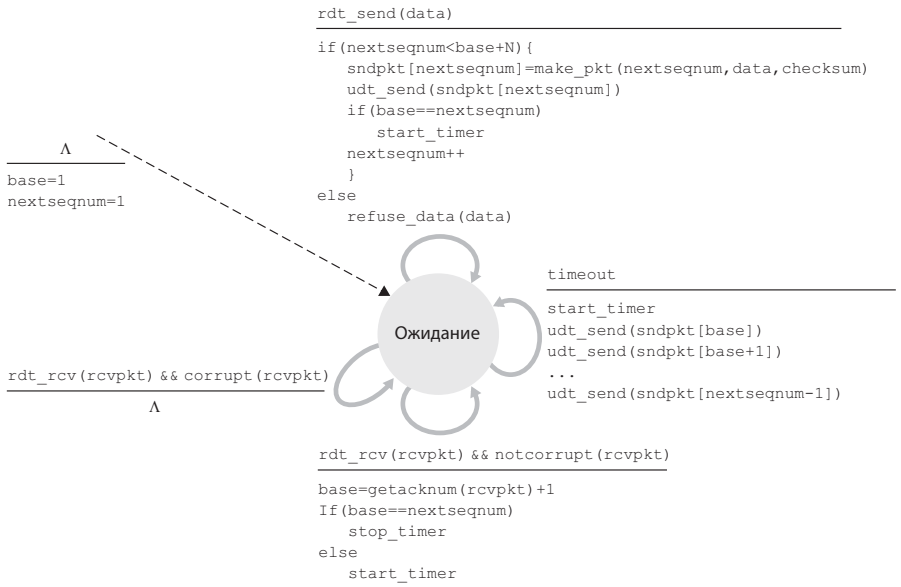


Рис. 3.20. Расширенная FSM-схема для отправителя протокола GBN

На рис. 3.20 и рис. 3.21 приведена расширенная схема конечного автомата для отправляющей и принимающей стороны протокола GBN, использующего только положительные подтверждения. Мы называем ее *расширенной FSM-схемой*, так как мы добавили переменные (анало-

гично переменным языка программирования) `base` и `nextseqnum`, а также операции и условия для этих переменных. Заметим, что расширенная FSM-схема теперь становится похожа на спецификацию языка программирования. Бочман[59] предоставляет замечательное определение дополнительных расширений для техник FSM-схем так же, как и других техник на основе языка программирования для описания протоколов.

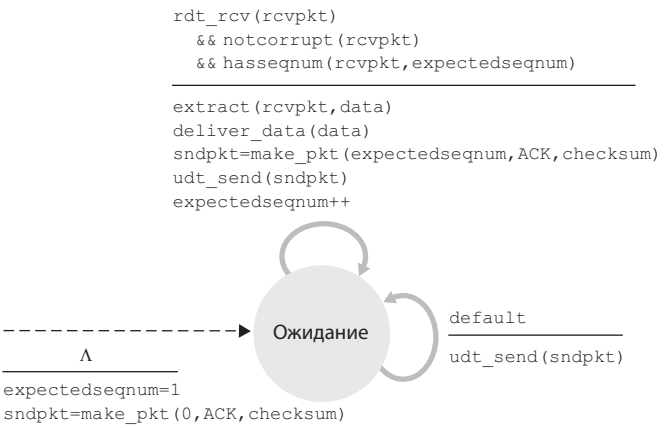


Рис. 3.21. Расширенная FSM-схема для получателя протокола GBN

Отправитель протокола GBN должен поддерживать три типа событий:

- *Вызов протоколом более высокого уровня*. Когда «сверху» вызывается функция `rdt_send()`, передающая сторона сначала проверяет степень заполнения окна (то есть наличие N посланных сообщений, ожидающих получения квитанций). Если окно оказывается незаполненным, новый пакет формируется и передается, а значения переменных обновляются. В противном случае передающая сторона возвращает данные верхнему уровню, и это является неявным указанием, что окно заполнено. Обычно верхний уровень предпринимает повторную попытку передачи данных через некоторое время. В реальном приложении отправитель, скорее всего, либо буферизовал бы данные (вместо немедленной отсылки), либо имел механизм синхронизации (например, семафор или флаг), который позволял бы вышестоящему уровню вызывать `rdt_send()` только при незаполненном окне.
- *Получение подтверждения*. В нашем GBN-протоколе для пакета с порядковым номером n выдается **общая квитанция**, указывающая

на то, что все пакеты с порядковыми номерами, предшествующими *n*, успешно приняты. Мы вернемся к механизму рукопожатия при рассмотрении принимающей стороны GBN-протокола.

- *Истечение интервала ожидания*. Своим названием GBN-протокол обязан механизму реагирования на потери и задержки данных. Как и в случае протокола с ожиданием подтверждения, для определения фактов потерь и задержек пакетов и квитанций GBN-протокол использует таймер. Если интервал ожидания истекает, передающая сторона повторно отправляет все посланные неподтвержденные пакеты. В нашем примере (см. рис. 3.19) передающая сторона использует единственный таймер, который отсчитывает время от момента передачи самого «старого» из пакетов, для которого не получено подтверждение. Если подтверждение получено, но при этом имеются переданные неподтвержденные пакеты, происходит сброс таймера. Отсутствие неподтвержденных пакетов приводит к остановке таймера.

Действия получателя в протоколе GBN также просты. Если пакет с порядковым номером *n* получен верно и в порядке следования, то есть последние данные, отправленные на верхний уровень, поступили из пакета с порядковым номером `n-1`, получатель шлет ACK для `n`-го пакета и передает часть данных пакета верхнему уровню. Во всех остальных случаях получатель не принимает пакет и повторно отправляет ACK для последнего из полученных в верном порядке пакетов. Отметим, что пакеты передаются на верхний уровень по одному и упорядоченно, и если пакет `k` был получен и доставлен, то и все пакеты с порядковым номером меньшим `k` также были доставлены. То есть применение накапливаемого подтверждения — наиболее подходящий вариант для протокола GBN.

В нашем протоколе GBN получатель отклоняет пакеты, пришедшие в неверном порядке. Хотя может показаться глупым и расточительным исключать корректно доставленные, но поступившие в неверном порядке, пакеты, для этого есть определенное обоснование. Напомним, что получатель должен доставить данные на верхний уровень в верном порядке. Теперь предположим, что ожидается пакет `n`, но приходит пакет `n+1`. Так как данные должны быть переданы по порядку, получатель *может* отправить пакет `n+1` в буфер и затем доставить на верхний уровень при получении и доставке пакета `n`. Однако если пакет *n* был потерян, оба пакета, `n` и `n+1`, будут, в конечном счете, переданы повторно, как результат правила повторной передачи отправителя протокола GBN. То есть получатель может просто отклонить пакет `n+1`. Преимущество такого

подхода заключается в простой буферизации на стороне получателя: не нужно использовать буфер для *каждого* следующего, полученного в неверном порядке пакета. То есть в то время, как отправитель должен поддерживать верхнюю и нижнюю границу окна и позиции `nextseqnum` в этом окне, единственная часть информации, которую должен поддерживать получатель — порядковый номер следующего по порядку пакета. Это значение хранится в переменной `expectedseqnum`, как показано на FSM-схеме получателя на рис. 3.21. Конечно, недостатком отбрасывания корректно полученных пакетов является то, что последующая передача этого пакета может быть потеряна или искажена и потребуется еще больше повторных передач.

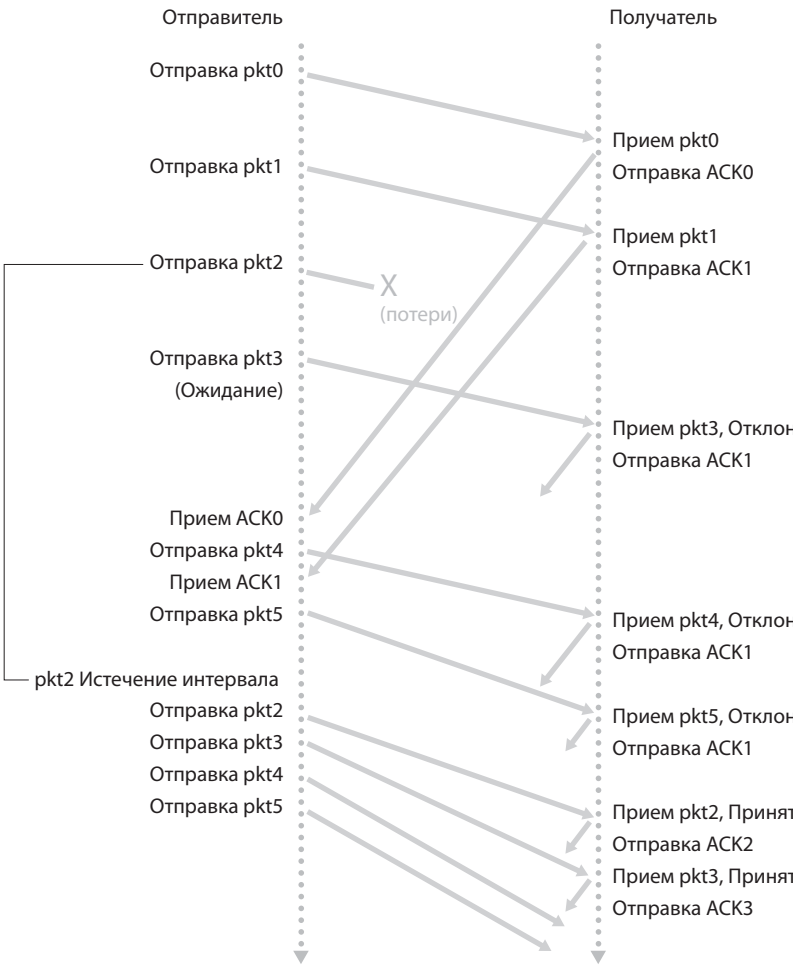


Рис. 3.22. Протокол Go-Back-N в действии

На рис. 3.22 изображены действия протокола GBN в случае, когда размер окна равен четырем пакетам. Из-за такого ограничения окна отправитель посылает пакеты с номерами с 0 по 3, а затем ждет, пока один или более из этих пакетов будут распознаны. Когда любой положительный ACK получен (например, ACK0 или ACK1), окно сдвигается вперед, и отправитель может послать еще один новый пакет (`pkt4` и `pkt5`, соответственно). На принимающей стороне пакет 2 потерян, и поэтому пакеты 3, 4, 5 поступают в неверном порядке и отбрасываются.

Завершая изучение протокола GBN, отметим, что его реализация в стеке будет структурно напоминать расширенную FSM-схему на рис. 3.20. Реализация также будет выполнена в форме различных процедур, действия которых должны происходить в ответ на различные возможные события. В таком **событийно-управляемом программировании** различные процедуры вызываются или другими процедурами стека протокола, или в результате прерывания. Для отправителя такие события могут быть: (1) вызваны объектом верхнего уровня для выполнения `rdt_send()`, (2) таймером прерывания и (3) вызовом `rdt_rcv()` с нижнего уровня при поступлении пакетов. Упражнения по программированию в конце этой главы дадут вам возможность на практике реализовать эти процедуры для модельных, но реалистичных настроек сети.

Мы отметим здесь, что протокол GBN включает в себя почти все методы, перечисленные нами при изучении составляющих надежной доставки данных в разделе 3.5. Они включают в себя использование последовательности чисел, накапливаемые подтверждения, контрольные суммы и операции таймаута и повторной передачи.

3.4.4. Выборочное повторение (протокол SR)

Протокол GBN потенциально позволяет отправителю «заполнить конвейер» пакетами, как показано на рис. 3.17, и таким образом решается проблема эффективного использования канала, которую мы отмечали в протоколах с ожиданием подтверждений. Однако существуют сценарии, в которых сам протокол GBN страдает от проблем производительности. В частности, когда размер окна и произведение пропускной способности на задержку распространения велики, в конвейере может находиться большое количество пакетов. В таком случае ошибка отдельного пакета может вызвать у протокола GBN повторную передачу большого количества пакетов, большинство из которых не требовались. Если вероятность ошибок канала будет увеличиваться, конвейер может

заполниться ненужными повторными передачами. Представьте в нашем сценарии диктовки сообщения, если каждый раз слово искажается, около 1000 слов (например, размер окна 1000 слов) будет повторено. Диктовка станет замедленной.

Как понятно из названия, протоколы выборочного повторения устраняют ненужные повторные передачи данных: отправитель выполняет повторную передачу только тех пакетов, которые пришли получателю с ошибками (то есть потерянные или искаженные пакеты). Такая индивидуальная, при необходимости повторная передача требует от получателя *индивидуального* подтверждения корректно принятых пакетов. Размер окна *N* вновь будет использоваться для ограничения выходящих за границы конвейера неподтвержденных пакетов. Но, в отличие от протокола GBN, в окне уже могут быть подтвержденные пакеты. На рис. 3.23 представлено пространство порядковых номеров отправителя протокола SR. На рис. 3.24 детально показаны различные действия, выполняемые отправителем протокола SR.

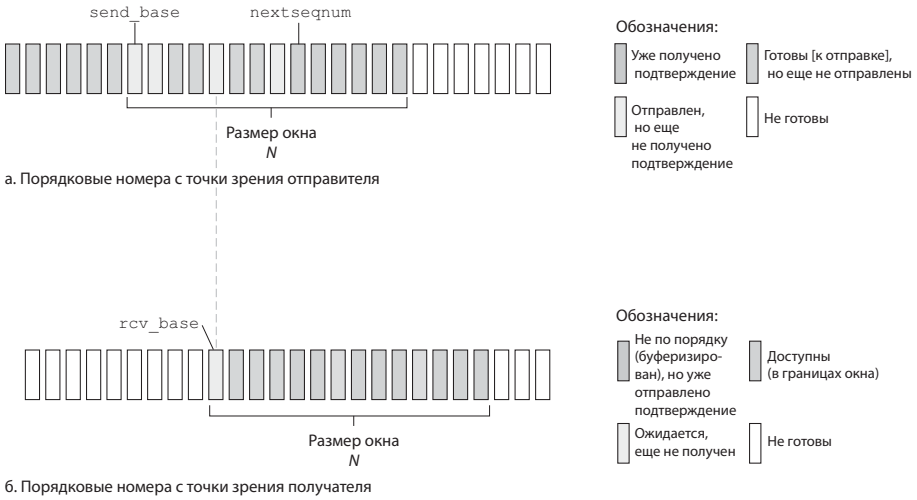


Рис. 3.23. Диапазон порядковых номеров отправителя и получателя протокола выборочного повторения SR

Принимающая сторона выдает квитанцию каждому принятому правильному (не содержащему ошибок) пакету, независимо от того, нарушает он порядок следования или нет. Пакеты, поступившие не по порядку, сохраняются в буфере до того момента, пока все пропущенные (то есть, пакеты с меньшими порядковыми номерами) будут получены, и партия пакетов сможет быть передана на верхний уровень в правиль-

ном порядке. Рис. 3.25 разделяет на элементы различные действия, выполняемые получателем протокола SR. Заметим, что на рис. 3.26 получатель сохраняет в буфер пакеты 3, 4, 5 и доставляет их вместе с пакетом 2 на верхний уровень, когда пакет 2 будет наконец получен.

1. *Данные получены сверху.* При получении данных от верхнего уровня отправитель протокола SR проверяет следующие доступные порядковые номера для пакета. Если порядковый номер попадает в окно отправителя, данные упаковываются в пакет и отправляются; в противном случае буферизируются либо возвращаются на верхний уровень, как в протоколе GBN.

2. *Истечение интервала.* Таймер вновь используется для подтверждения потери пакетов. Но каждый пакет теперь должен иметь собственный логический таймер так, как только один пакет будет передан при истечении интервала ожидания. Всего один системный таймер может использоваться для имитации действия множества логических таймеров[644].

3. *Получен пакет ACK.* Когда получена квитанция, отправитель протокола SR отмечает этот пакет как принятый, если он принадлежит окну. Если порядковый номер пакета равен значению `send_base`, то первая база окна сдвигается вперед до неподтвержденного пакета. Если при сдвиге окна обнаруживаются еще не переданные пакеты, номера которых теперь попадают в окно, то эти пакеты передаются.

Рис. 3.24. Действия и события отправителя протокола SR

1. *Пакет с порядковым номером из диапазона* [`rcv_base, rcv_base+N-1`] *корректно получен.* В этом случае полученные пакеты попадают в окно получателя и подтверждение на этот пакет возвращается отправителю. Если данный пакет ранее не был получен, то он буферизируется. Если он имеет порядковый номер равный базе окна получателя (`rcv_base` на рис. 3.22), то этот пакет и все ранее помещенные в буфер доставляются на верхний уровень. Затем окно получателя сдвигается вперед на количество пакетов доставленных на верхний уровень. Как, например, показано на рис. 3.26. Когда пакет с порядковым номером `rcv_base=2` получен, он и пакеты 3, 4 и 5 могут быть доставлены на верхний уровень.

2. *Пакет с порядковым номером из диапазона* [`rcv_base-N, rcv_base-1`] *корректно получен.* В этом случае положительная квитанция должна быть создана даже в случае, если этот пакет ранее был подтвержден получателем.

3. *Иначе.* Игнорировать пакет.

Рис. 3.25. Действия и события получателя протокола SR

Важно отметить, что на шаге 2 на рис. 3.25 получатель обрабатывает уже принятые пакеты (либо игнорирует их) в определенной последовательности *ниже* текущего базового номера окна. Вы должны убедиться сами, что эти повторные подтверждения действительно необходимы. Рассмотрим пространства порядковых номеров отправителя и получателя на рис. 3.23. Так, если нет положительной квитанции для пакета с номером `send_base`, посланного от получателя к отправителю, то от-

правитель в конце концов повторно передаст пакет с номером `send_base`, даже если ясно (для нас, но не для отправителя), что получатель уже принял этот пакет. Окно отправителя не сдвинется вперед, пока получатель не подтвердит получение этого пакета! Этот пример демонстрирует важный аспект протокола SR (а также многих других протоколов). Отправитель и получатель не всегда имеют одинаковое представление о том, что было и что не было получено. Для протоколов SR это означает, что окна отправителя и получателя не всегда совпадают.

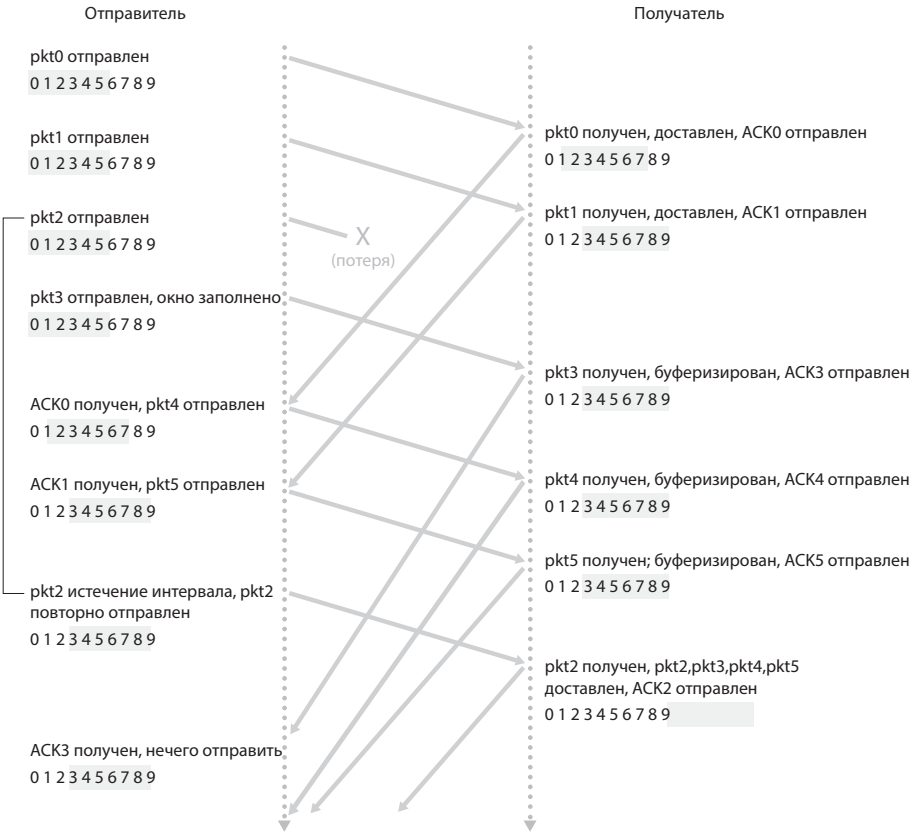


Рис. 3.26. Действия протокола SR

Отсутствие синхронизации между окнами отправителя и получателя имеет важные последствия, когда мы сталкиваемся с ограниченностью диапазона порядковых номеров. Рассмотрим, что могло бы произойти, например, если у нас есть четыре пакета с порядковыми номерами 0, 1, 2, 3, а размер окна равен трем. Предположим, пакеты с 0 по 2 переданы отправителем, корректно получены и подтверждены получателем. В этот

момент окно получателя заполняется четвертым, пятым и шестым пакетами, которые имеют порядковые номера 3, 0 и 1, соответственно. Теперь рассмотрим два сценария. В первом сценарии, показанном на рис. 3.27 (а) ACK пакеты для первых трех пакетов данных потеряны, и отправитель пересылает эти пакеты. Таким образом, получатель далее получает пакет с порядковым номером 0 — копию первого отправленного.

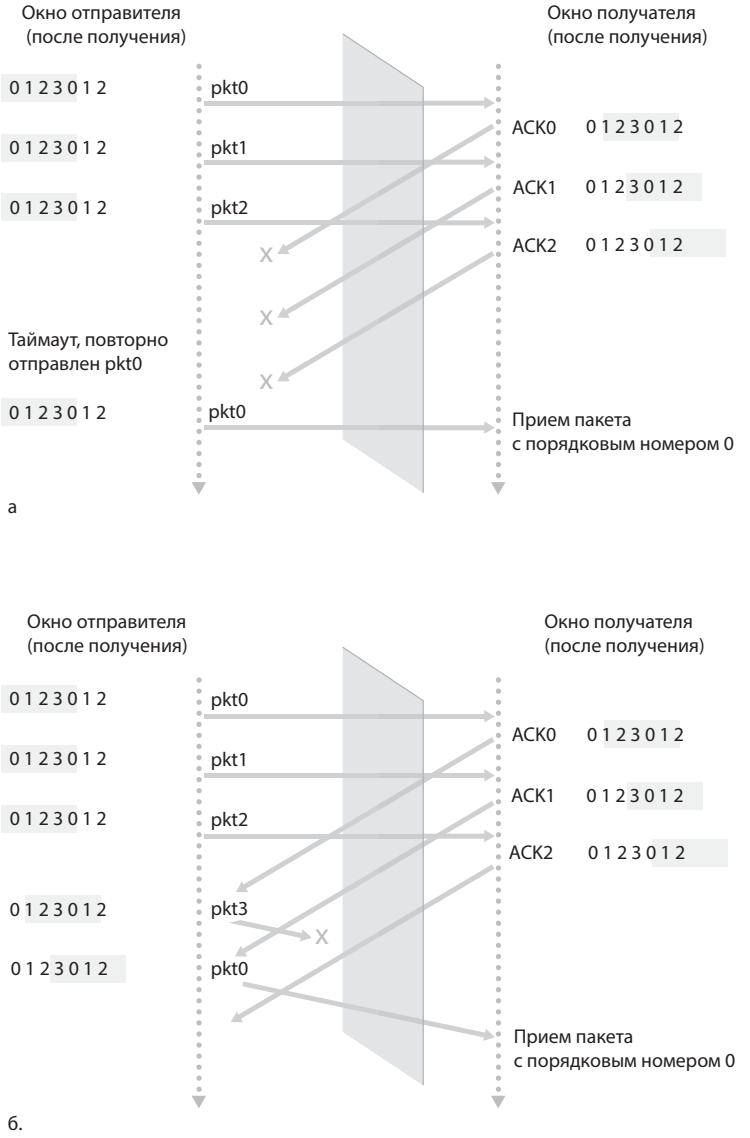


Рис. 3.27. Дилемма получателя протокола SR с очень большими окнами: новый пакет или повторная передача?

Во втором сценарии показанном на рис. 3.27б, квитанции ACK на первые три пакета доставлены верно. Таким образом, отправитель сдвигает окно вперед и отправляет четвертый, пятый и шестой пакеты с порядковыми номерами 3, 0 и 1 соответственно. Пакет с порядковым номером 3 потерян, но пакет с порядковым номером 0, содержащий *новые* данные, получен.

Теперь рассмотрим ту же ситуацию с точки зрения принимающей стороны. Действия, выполняемые передающей стороной, скрыты от нее; принимающая сторона способна лишь следить за последовательностями получаемых пакетов и генерируемых квитанций. Подобная ограниченность приводит к тому, что обе описанные выше ситуации воспринимаются принимающей стороной как *одинаковые*. Она не может отличить исходную передачу первого пакета от повторной. Очевидно, что протокол, размер окна которого на единицу меньше диапазона порядковых номеров, не является работоспособным. Однако насколько малым должен быть размер окна? В одном из упражнений, приведенных в конце этой главы, вам предлагается самостоятельно доказать, что размер окна SR-протокола не должен превосходить половины диапазона порядковых номеров.

На сайте **tinyurl.com/kzt3mhd** вы найдете апплет, который демонстрирует работу протокола SR. Попробуйте провести тот же эксперимент, что и с апплетом для протокола GBN. Совпадут ли ваши ожидания с результатом?

Итак, мы завершаем разговор о протоколах надежной передачи данных. Мы рассмотрели *значительный* объем материала и ознакомились с множеством механизмов, используемых в этих протоколах, — итог накопленным знаниям подводит табл. 3.1. Просмотрев этот раздел с начала, вы сможете увидеть, как различные механизмы постепенно включались в создаваемый протокол передачи данных, делая его все более и более реалистичным и улучшая качество его функционирования.

Завершая обсуждение протоколов надежной доставки данных, отметим еще одно важное допущение. Оно состоит в том, что во время передачи пакетов не может произойти нарушение порядка их следования. В случае, если канал представляет собой физическую линию связи, подобное допущение выглядит естественным, однако при передаче по разветвленной компьютерной сети порядок следования пакетов вполне способен изменяться. Одно из проявлений такого изменения заключа-

ется в том, что пакеты и квитанции с порядковым номером *x* могут появиться на принимающей и передающей сторонах в те моменты времени, когда номер *x* уже не принадлежит текущему окну. Фактически процесс передачи по каналу в этом случае можно представить как буферизацию пакетов и отбрасывание их из буфера в *случайные* моменты времени. Поскольку одни и те же порядковые номера иногда неоднократно используются при передаче, необходимо следить за возможным дублированием пакетов.

Табл. 3.1. Механизмы, обеспечивающие надежную передачу данных и их использование

Механизм	**Применение, комментарий**
Контрольная сумма	Используется для обнаружения битовых ошибок в переданном пакете
Таймер	Отсчет интервала ожидания и указание на его истечение. Последнее означает, что с высокой степенью вероятности пакет или его квитанция потеряны при передаче. В случае, если пакет доставляется с задержкой, но не теряется (преждевременное истечение интервала ожидания), либо происходит потеря квитанции, повторная передача приводит к дублированию пакета на принимающей стороне
Порядковый номер	Используется для порядковой нумерации пакетов данных передаваемых от отправителя к получателю. Разрывы в порядковых номерах полученных пакетов позволяют получателю обнаружить потерю пакета. Одинаковые порядковые номера пакетов означают, что пакеты дублируют друг друга.
Подтверждение	Генерируется принимающей стороной и указывает передающей стороне на то, что соответствующий пакет или группа пакетов успешно приняты. Обычно подтверждение содержит порядковые номера успешно принятых пакетов. В зависимости от протокола различают индивидуальные и групповые подтверждения.
Отрицательное подтверждение	Используется получателем для сообщения отправителю, что пакет был получен некорректно. Отрицательное подтверждение обычно включает в себя порядковый номер пакета, который не был корректно получен.
Окно, конвейеризация	Ограничивают диапазон порядковых номеров, которые могут использоваться для передачи пакетов. Групповая передача и рукопожатие позволяют значительно увеличить пропускную способность протоколов по сравнению с режимом ожидания подтверждений. Как мы увидим, размер окна может быть рассчитан на основе возможностей приема и буферизации принимающей стороны, а также уровня загрузки сети

Механизмы слежения обеспечивают уверенность передающей стороны в том, что при использовании любого из порядковых номеров в сети не будет ни одного пакета с таким же порядковым номером. Обычно

такая уверенность достигается путем введения понятия *максимального времени жизни* пакета. Этот период принимается равным примерно трем минутам для протокола TCP в высокоскоростных сетях[445]. В работе Саншайна[630] описан механизм, полностью решающий проблему, связанную с изменением порядка следования пакетов.

3.5. Протокол TCP: передача с установлением соединения

Теперь, когда мы рассмотрели внутренние принципы надежной доставки данных, давайте вернемся к протоколу TCP — надежному протоколу транспортного уровня Интернета с установлением соединения. Как мы увидим, в протоколе TCP используются многие из методов обеспечения надежной передачи данных, описанных в предыдущем разделе: обнаружение ошибок, повторные передачи пакетов, общие квитанции, таймеры и поля порядковых номеров в заголовках пакетов и квитанциях. Описание TCP содержится в документах RFC 793, RFC 1122, RFC 1323, RFC 2018 и RFC 2581.

3.5.1. TCP-соединение

TCP называется протоколом с **установлением логического соединения** поскольку перед началом обмена данными два процесса осуществляют «рукопожатие» — процедуру, заключающуюся в передаче друг другу специальных сегментов, предназначенных для определения параметров обмена данными. Частью процедуры установления TCP-соединения является инициализация обеими сторонами множества переменных состояния TCP (некоторые из них обсуждались в этом и разделе 3.7), связанных с TCP-соединением.

«TCP-соединение» — это не соединение с TDM или FDM мультиплексированием, как в сетях с коммутацией каналов. Оно также не является виртуальным каналом (см. главу 1), поскольку состояние соединения поддерживается лишь на двух конечных системах. Так как протокол TCP выполняется только на конечных системах, не включая промежуточные элементы сети (маршрутизаторы и переключатели канального уровня), те не поддерживают состояние TCP-соединения. На самом деле промежуточные маршрутизаторы абсолютно не обращают внимания на TCP-соединения; они видят не их, а дейтаграммы.

ИСТОРИЯ

Винтон Серф, Роберт Кан и сеть TCP/IP

В начале 1970-х стали распространяться сети с коммутацией пакетов. В частности, такова была сеть ARPAnet, предшественник Интернета, лишь одна из множества ей подобных. Каждая из этих сетей использовала собственный протокол. Два исследователя, Винтон Серф и Роберт Кан осознали важность межсетевого взаимодействия и изобрели межсетевой протокол, названный TCP/IP (Transmission Control Protocol/Internet Protocol — Протокол управления передачей/Интернет протокол). Хотя Серф и Кан изначально рассматривали протокол как единый объект, позже он был разделен на две части, TCP и IP, функционирующие независимо. Серф и Кан опубликовали статью, посвященную TCP/IP в мае 1974 г. в издании *IEEE Transactions on Communications Technology*[80].

Протокол TCP/IP, основа существования сегодняшней сети Интернет, был разработан до появления персональных компьютеров, рабочих станций, смартфонов и планшетов, до распространения Ethernet, кабельных сетей, DSL, Wi-Fi, и других технологий доступа, а также до Всемирной Паутины, социальных сетей и потокового видео. Серф и Кан видели необходимость в сетевом протоколе, который, с одной стороны, предоставлял бы широкую поддержку еще определяющихся приложений и, с другой стороны, позволял бы взаимодействовать произвольным хостам и протоколам канального уровня.

В 2004-м Серф и Кан получили награду ACM Turing Award, которую сравнивают с Нобелевской премией в области вычислительной техники «За пионерскую работу по проблеме межсетевого обмена, включая разработку и реализацию основных Интернет-протоколов, TCP/IP и за ведущую роль в области компьютерных сетей».

Соединение TCP обеспечивает **дуплексную** передачу файлов. Если оно установлено между процессом А на одном хосте и процессом Б на другом хосте, то данные прикладного уровня могут одновременно передаваться как от процесса А к процессу Б, так и в обратном направлении. Кроме того, TCP-соединение всегда является **двухточечным**, то есть устанавливается между единственной парой отправитель-получатель. Другими словами, при использовании протокола TCP невозможно осуществлять широковещательную передачу данных (см. раздел 4.7), когда они передаются от одного отправителя нескольким получателям за одну операцию. Для протокола TCP два хоста — компания, а три уже толпа!

Теперь рассмотрим, как устанавливается TCP-соединение. Предположим, процесс, запущенный на одном хосте, желает установить

соединение с процессом, выполняющимся на другом хосте. Напомним, что процесс, который инициирует соединение, называется *клиентским*, другой же — *серверным*. Процесс клиентского приложения в первую очередь информирует транспортный уровень клиента о том, что хочет установить соединение с серверным процессом. Клиентская программа на Python из раздела 2.7.2 для этой цели вызывает команду:

```
clientSocket.connect((serverName,serverPort))
```

где `serverName` — это имя сервера, и `serverPort` — процесс на сервере. Протокол TCP на стороне клиента далее переходит к установлению TCP-соединения с протоколом TCP сервера. В конце этой части мы обсуждаем несколько подробнее процедуру установления соединения. Сейчас достаточно знать, что сначала клиент отправляет специальный TCP-сегмент; затем сервер отвечает вторым специальным TCP-сегментом; и, наконец, клиент снова отвечает третьим специальным сегментом. Первые два сегмента не несут никакой полезной нагрузки, то есть не содержат никаких данных прикладного уровня; третий сегмент уже может содержать такие данные. Процедура установления включает отправку трех сегментов, поэтому зачастую ее называют **тройным рукопожатием**.

Когда TCP-соединение установлено, оба прикладных процесса могут отправлять данные друг другу. Давайте рассмотрим процесс передачи данных от клиентского процесса серверному. Процесс клиента передает поток данных через сокет («дверь» процесса), как описано в разделе 2.7. Как только данные прошли через эту дверь, они попадают в распоряжение протокола TCP, запущенного на стороне клиента. Как показано на рис. 3.28, протокол TCP направляет эти данные в **буфер передачи** — один из буферов, создаваемых при установлении соединения. Время от времени TCP будет получать часть данных из буфера и передавать их сетевому уровню. Интересной особенностью спецификации TCP[421] является свобода в выборе моментов для отправки данных, находящихся в буфере. Согласно спецификации, протокол TCP должен «передать эти данные в виде сегментов в любой подходящий для этого момент времени». Максимальный объем данных, который может быть извлечен из буфера и помещен в сегмент, ограничивается **максимальным размером сегмента** (Maximum Segment Size, **MSS**). Обычно MSS устанавливается на основе предварительного измерения длины наибольшего фрагмента канального уровня, который может быть передан текущим хостом (так называемый **максимальный передаваемый блок** (Maximum Transmission Unit, **MTU**)), чтобы быть уверен-

ными, что TCP-сегмент, инкапсулированный в IP-дейтаграмму, вместе с длиной TCP/IP заголовка (обычно 40 байт) полностью будет помещен в фрагмент канального уровня. MTU для протоколов канального уровня Ethernet и PPP равен 1500 байт. Следовательно, значение MSS как правило составляет 1460 байт. Также были предложены подходы для определения MTU для всего маршрута — наибольшего фрагмента канального уровня, который может быть отправлен по всем каналам от начального узла до конечного[440] — и определения MSS на основе значения MTU для всего маршрута. Заметим, что MSS — это максимальное количество данных прикладного уровня в сегменте, а не максимальный размер TCP-сегмента вместе с заголовком. Такая терминология является несколько запутанной, однако распространенной, поэтому мы вынуждены ее придерживаться.

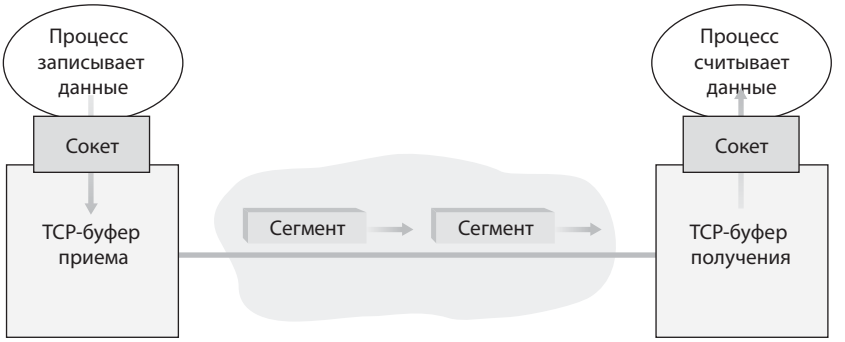


Рис. 3.28. Буферы приема и передачи в протоколе TCP

Протокол TCP составляет пары из фрагментов данных клиента и TCP-заголовков, формируя **TCP-сегменты.** Сегменты передаются вниз на сетевой уровень, где они по отдельности инкапсулируются в IP-дейтаграммы сетевого уровня. Далее IP-дейтаграммы передаются в сеть. Когда протокол TCP получает сегмент с противоположного конца соединения, данные сегмента помещаются в буфер получателя TCP, как показано на рис. 3.28. Приложение считывает поток данных из этого буфера. Каждая сторона соединения имеет собственные буферы приема и передачи. Вы можете воспользоваться апплетом управления потоком, размещенным на сайте **tinyurl.com/csek96t**, чтобы увидеть анимацию, иллюстрирующую принцип работы буферов приема и передачи.

Подведем итог. Мы узнали, что TCP-соединение состоит из буферов, переменных и сокета соединения для обработки на одном хосте и друго-

го набора буферов, переменных и сокета соединения для обработки на другом хосте. При этом для соединения не выделяется никаких буферов или переменных на промежуточных сетевых устройствах (маршрутизаторах, мостах и повторителях).

3.5.2. Структура TCP-сегмента

Кратко рассмотрев TCP-соединение, давайте перейдем к изучению структуры TCP-сегмента. TCP-сегмент состоит из поля данных и полей заголовка. Поле данных содержит фрагмент данных, передаваемых между процессами. Как упоминалось ранее, значение MSS ограничивает максимальный размер для поля данных сегмента. Когда по TCP-соединению отправляется большой файл, например, изображение с веб-страницы, он обычно разбивается на фрагменты размера MSS (за исключением последней части, которая часто получается меньше MSS). Однако интерактивные приложения часто передают фрагменты данных меньшего, чем MSS, размера. Например, приложения удаленного доступа к сети, подобные Telnet, могут передать транспортному уровню 1 байт данных. Поскольку обычно длина заголовка TCP-сегмента составляет 20 байт (на 12 байт больше, чем в UDP), полный размер сегмента в этом случае равен 21 байт.

На рис. 3.29 показана структура TCP-сегмента. Как и в UDP-сегменте, заголовок включает **номера портов отправителя и получателя**, которые используются при мультиплексировании и демультиплексировании данных приложения верхнего уровня. Аналогично заголовку UDP-сегмента, заголовок TCP-сегмента включает **поле контрольной суммы**. Заголовок TCP-сегмента также содержит следующие данные:

- 32-разрядные **поля порядкового номера** и **номера подтверждения**, используемые TCP отправителем и получателем для реализации службы надежной передачи данных, как обсуждается далее.
- 16-разрядное поле **окна приема,** используемое для управления потоком. Мы кратко рассмотрим его применение для указания количества байт, которые получатель готов принять.
- 4 разрядное поле **длины заголовка**, определяющего длину TCP-заголовка в 32-разрядных слова. Длина TCP-заголовка может изменяться в зависимости от наличия дополнительных полей для него. (Обычно дополнительные поля пусты, поэтому стандартная длина заголовка TCP — 20 байт.)

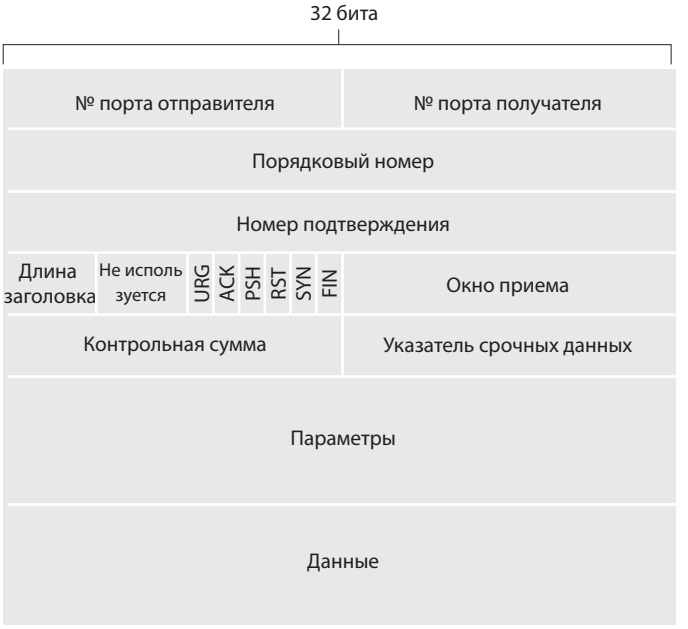


Рис. 3.29. Структура TCP-сегмента

- Необязательное, влияющее на длину заголовка **поле параметров**, которое используется, когда отправитель и получатель договариваются о максимальном размере сегмента (MSS) или при увеличении размера окна в высокоскоростных сетях. Также в этом поле определяется параметр временных меток. См. RFC 854 и RFC 1323 для получения дополнительных сведений.
- **Поле флагов** состоит из 6 бит. **Бит ACK** используется для указания на корректность значения, размещенного в поле подтверждения; то есть, сегмент содержит подтверждение для сегмента, который был успешно получен. Биты **RST**, **SYN**, и **FIN,** используются для установления и завершения соединения, об этом мы поговорим в конце данного раздела. Установка бита **PSH** означает, что получатель должен незамедлительно отправить данные на верхний уровень. Наконец, бит **URG** указывает, что в этом сегменте присутствуют данные, которые объект верхнего уровня стороны отправителя отметил как «срочные». Местонахождение последнего байта в этих срочных данных указывается в 16 битном **поле указателя на срочные данные**. Протокол TCP должен сообщать объекту верхнего уровня стороны получателя о присутствии срочных данных и передавать указатель на конец срочных данных. (На практике биты PSH, URG и указа-

тель на срочные данные не используются. Но для полноты описания мы рассказали об этих полях.)

Порядковые номера и номера подтверждений

Два наиболее важных поля в заголовке TCP-сегмента — это поле порядкового номера пакета данных и поле номера подтверждения. В службе надежной передачи данных протокола TCP эти поля играют очень важную роль. Прежде чем перейти к обсуждению их роли в надежной передаче данных, давайте рассмотрим, что именно помещает протокол TCP в эти поля.

С точки зрения протокола TCP данные представляются как неструктурированный, но упорядоченный поток байт. Применяемые порядковые номера в протоколе TCP указывают *не* на количество переданных сегментов, а на количество переданных байт. **Порядковый номер сегмента** — это номер первого байта в сегменте в потоке байт. Предположим, процесс хоста А хочет отправить поток данных процессу хоста Б по TCP-соединению. Протокол TCP на хосте А явно указывает номер для каждого байта в потоке данных. Допустим, поток данных содержит файл длиной 500 000 байт, а MSS равен 1000 байт, и первому байту потока данных задан номер 0. Как показано на рис. 3.30, протокол TCP создает 500 сегментов для этого потока данных. Первый сегмент получает порядковый номер 0, второй сегмент — 1000, третий сегмент — 2000, и так далее. Каждый порядковый номер помещается в поле порядкового номера в заголовке соответствующего TCP-сегмента.

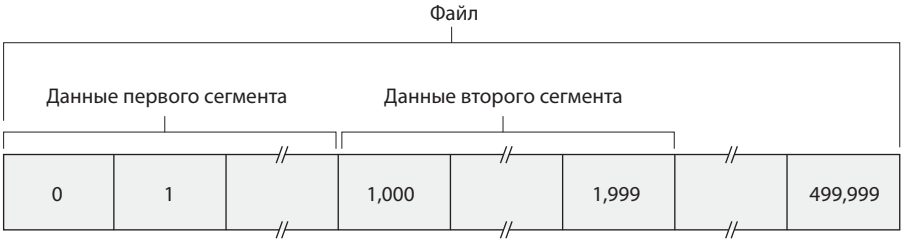


Рис. 3.30. Разделение файла данных на TCP-сегменты

Теперь давайте рассмотрим номера подтверждений. Номера подтверждений устроены несколько хитрее, чем порядковые номера. Напомним, что протокол TCP использует полный дуплекс, поэтому хост А может получать данные от хоста Б в то же время, когда отправляет данные хосту Б, используя то же самое соединение. Каждый из сегмен-

тов, полученных от хоста Б, имеет порядковый номер данных передаваемых от хоста Б к хосту А. *Номер подтверждения хоста А размещаемый в его сегменте — это порядковый номер следующего байта ожидаемого хостом А от хоста Б.* Рассмотрим несколько примеров, чтобы лучше понимать, что происходит. Предположим, что хост А получил все байты, пронумерованные от 0 до 555 от хоста Б и что он собирается отправлять сегмент хосту Б. Хост А ожидает байт 536 и все последующие байты потока данных от хоста Б. Поэтому хост А помещает значение 536 в поле номера подтверждения сегмента, отправляемого хосту Б.

В качестве другого примера, предположим, что хост А получил от хоста Б сегмент, содержащий байты с 0 по 535 и сегмент, содержащий байты с 900 по 1000. По какой-то причине хост А все еще не получил байты с 536 по 899. В этом примере, хост А продолжает ждать байт 536 (и далее) в порядке восстановления потока данных хоста Б. Таким образом, следующий сегмент от хоста А к хосту Б содержит значение 536 в поле номера подтверждения. Поскольку TCP квитирует принятые данные до первого отсутствующего байта, говорят, что он поддерживает **общее квитирование** (рукопожатие).

Последний пример демонстрирует очень важный аспект функционирования протокола TCP. Третий сегмент (содержащий байты 900–1000) принят хостом А раньше, чем второй (содержащий байты 536–899), то есть с нарушением порядка следования данных. Возникает вопрос: как протокол TCP реагирует на нарушение порядка? Оказывается, что спецификация протокола предоставляет программистам, реализующим TCP, полную свободу в разрешении этого вопроса. Тем не менее выделяются два основных подхода: принимающая сторона может либо немедленно проигнорировать сегменты, нарушающие порядок следования данных, либо сохранять принятые сегменты до тех пор, пока недостающие данные не будут получены. Первый подход позволяет упростить программирование, в то время как второй повышает эффективность использования линий связи.

На рис. 3.30 первым порядковым номером является 0, однако на практике стороны протокола TCP выбирают начальный номер случайным образом. Это объясняется необходимостью минимизировать вероятность нахождения в сети сегмента, который сформирован другим (уже закрытым) TCP-соединением между этими же двумя хостами, но который может быть по ошибке отнесен к существующему TCP-соединению. Заметим, что существующее соединение может использовать те же номера портов, что и предыдущее[630].

Telnet: учебный пример на тему порядковых номеров и номеров подтверждений

Протокол Telnet, описанный в документе RFC 854, является популярным протоколом прикладного уровня, применяемым для удаленного доступа. В нем используются службы протокола TCP, и он может выполняться на любой паре хостов. В отличие от большинства приложений передачи данных, обсуждаемых в главе 2, протокол Telnet — это интерактивное приложение. Здесь на примере протокола Telnet мы демонстрируем применение порядковых номеров и номеров подтверждений в протоколе TCP. Мы заметим, что многие пользователи в настоящее время предпочитают протоколу Telnet протокол SSH, поскольку данные, передаваемые в Telnet соединении (включая пароли), не зашифрованы, что делает этот протокол уязвимым к прослушивающим атакам (подробнее в разделе 8.7).

Пусть хост А инициирует Telnet-сеанс с хостом Б. Это означает, что А будет играть роль клиента, а Б — сервера. Каждый символ, введенный пользователем клиентской стороны, передается удаленному хосту; последний отсылает обратно копии символов, которые отображаются на экране пользователя. Это «обратное эхо» позволяет убедиться в том, что вводимые символы успешно принимаются сервером.

Таким образом, введенные символы проходят двойной путь между хостами перед тем, как отобразиться на экране.

Теперь предположим, что пользователь ввел единственный символ «С» и затем отхлебнул кофе. Рассмотрим TCP-сегменты, которые были переданы между клиентом и сервером. Как показано на рис. 3.31, мы предположили, что начальные порядковые номера для клиента и сервера 42 и 79, соответственно. Напомним, что порядковый номер сегмента — это порядковый номер первого байта данных в сегменте.

Таким образом, первый сегмент отравленный клиентом будет иметь порядковый номер 42; первый сегмент, отправленный сервером, будет иметь порядковый номер 79. Напомним, что номер подтверждения — это порядковый номер следующего ожидаемого хостом байта данных. После того, как TCP-соединение установлено, но прежде чем отправлены данные, клиент ожидает байт с порядковым номером 79 и сервер ожидает байт с порядковым номером 42.

Согласно рис. 3.31 было отправлено три сегмента. Первый сегмент, отправленный от клиента серверу, содержал 1 байт ASCII представления

буквы «С» в поле данных. В поле порядкового номера первого сегмента содержится значение 42, как мы только что описали. Также, поскольку клиент еще не получил никаких данных от сервера, в поле номера подтверждения первого сегмента содержится значение 79.

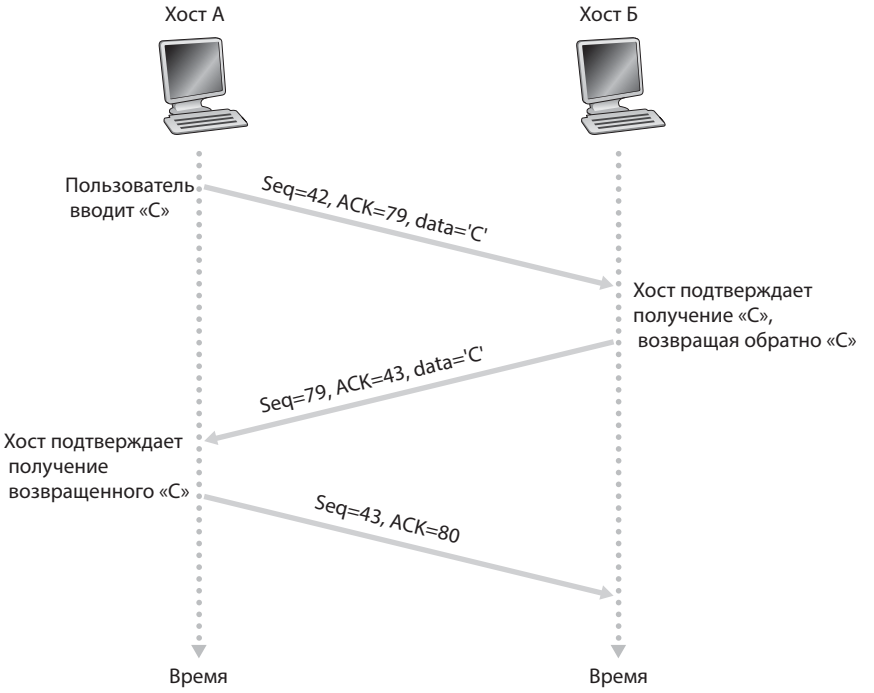


Рис. 3.31. Порядковые номера и номера квитанций для простого Telnet-приложения работающего по протоколу TCP

Второй сегмент был отправлен от сервера клиенту — по двум причинам. Во-первых, это подтверждает получение данных сервером. Помещая значение 43 в поле подтверждения, сервер сообщает клиенту, что он успешно получил все данные, включая 42 байт, и теперь ожидает поступление байта 43. Вторая причина — возвращение обратно символа «С». Т.е. в поле данных второго сегмента хранится ASCII представления символа «С». Этот второй сегмент имеет порядковый номер 79, начальный порядковый номер потока данных передаваемого от сервера клиенту в этом TCP-соединении, так как это самый первый байт данных отправленный сервером. Обратите внимание на то, что квитанции для данных, передаваемых от клиента к серверу, отправляются вместе с данными, передаваемыми от сервера к клиенту. Подобные квитанции называют **вложенными**.

Третий сегмент отправлен от клиента серверу. Его единственная цель — подтвердить данные, которые он получил от сервера. (Напомним, что второй сегмент содержал данные передаваемые от сервера клиенту — букву «C».) Поле данных этого сегмента пустое, то есть в сегменте содержится только подтверждение, которое не было передано с какой-либо частью данных. Поле номера подтверждения содержит 80 потому, что клиент уже получил поток байт до 79 и теперь ожидает байты начиная с 80-го байта. Возможно, вы думаете, что излишне добавлять порядковый номер сегменту, поскольку он не содержит никаких данных. Но, так как порядковый номер является полем заголовка TCP, то сегмент должен иметь какой-то номер.

3.5.3. Время оборота и интервал ожидания

Протокол TCP, подобно протоколу `rdt` из раздела 3.4 использует механизм ожидания/повторной передачи для восстановления потерянных сегментов. Несмотря на концептуальную простоту, при реализации подобного механизма в конкретных протоколах (например, в TCP) приходится учитывать множество нюансов. Например, нужно определить длительность интервала ожидания. Очевидно, что интервал ожидания должен быть больше времени оборота, равного времени получения квитанции передающей стороной, в противном случае неизбежны бесполезные повторные передачи. Однако во сколько раз больше? Как оценить время оборота? Нужно ли связывать таймеры со всеми неподтвержденными сегментами? Все эти вопросы требуют ответов! В данном разделе мы использовали материалы из работы Якобсона[260], посвященной TCP, и текущих рекомендаций IETF по управлению таймерами протокола TCP[563].

Оценка времени оборота

Пожалуй, приступим к изучению оценки времени в протоколе TCP с рассмотрения способа оценки времени оборота между отправителем и получателем. Оценка выполняется описанным ниже образом. Выборочное время оборота (RTT), обозначается `SampleRTT`, для сегмента — это промежуток времени с момента отправки, т. е. передачи его протоколу IP, до прибытия подтверждения о получении сегмента. Вместо того, чтобы измерять значение `SampleRTT` для каждого переданного сегмента, в большинстве реализаций TCP производится только одно измерение SampleRTT за раз. То есть в любой момент времени значение `SampleRTT` будет измеряться только для одного переданного, но еще не

подтвержденного сегмента, тем самым оно обновляется приблизительно один раз за время оборота. Кроме того, протокол TCP никогда не измеряет SampleRTT для переданного повторно сегмента, а выполняет оценку только для передаваемых впервые сегментов[276] (одно из упражнений, приведенных в конце этой главы, предлагает вам объяснить, почему).

Вследствие возможных перегрузок в маршрутизаторах на пути сегментов, а также изменения загрузок конечных систем значение SampleRTT постоянно меняется. Это означает, что в любой момент времени оно может значительно отклониться от типичного. Для получения типичного значения необходимо некоторым способом усреднить величину SampleRTT. Для этого в протоколе TCP вводится величина EstimatedRTT, вычисляемая вместе с каждым новым значением SampleRTT по формуле:

$$\text{EstimatedRTT} = (1 - \alpha) \times \text{EstimatedRTT} + \alpha \times \text{SampleRTT}$$

Подобная запись похожа на оператор языка программирования: новое значение EstimatedRTT получается из предыдущего значения EstimatedRTT и нового значения SampleRTT. Рекомендованное значение $\alpha = 0{,}125$ (или $1/8$)[563], в этом случае формула принимает следующий вид:

$$\text{EstimatedRTT} = 0{,}875 \times \text{EstimatedRTT} + 0{,}125 \times \text{SampleRTT}$$

Заметим, что EstimatedRTT — это взвешенное среднее значений SampleRTT. В упражнениях, приведенных в конце главы, демонстрируется, что при использовании, данной формулы наибольший вес имеют последние измерения значения SampleRTT. Это придает величине EstimatedRTT актуальность, поскольку она в большей степени отражает текущую ситуацию в сети, чем ее предыдущие состояния. В статистике подобные величины называют **экспоненциальным весовым скользящим средним**. Термин «экспоненциальное» означает, что вес каждого значения SampleRTT экспоненциально убывает с появлением новых значений. В упражнениях, приведенных в конце главы, вам будет предложено извлечь экспоненциальный член из формулы EstimatedRTT.

ПРИНЦИПЫ В ДЕЙСТВИИ

Протокол TCP предоставляет надежную передачу данных, используя положительные подтверждения и таймеры; фактически, такой же способ был рассмотрен в разделе 3.4. Протокол TCP подтверждает данные, которые были получены верно, и повторно отправляет сег-

менты в случае, если были потеряны или повреждены сегменты или их квитанции. Текущие версии протокола TCP также имеют неявный механизм NAK, используемый в механизме ускоренной повторной передачи: получение трех дубликатов ACK для данного сегмента обрабатывается, как отрицательная квитанция на следующий за ним сегмент и вызывает его повторную передачу до истечения интервала времени ожидания. Протокол TCP использует порядковые номера, чтобы определять потери или повторения сегментов на стороне получателя. Как и наш протокол надежной передачи данных `rdt3.0`, протокол TCP не способен самостоятельно доподлинно определить, что именно произошло: потеря, повреждение или превышение интервала времени ожидания сегментом или его пакетом ACK. Ответное действие TCP-отправителя всегда будет одинаковым: повторная передача сегмента.

В протоколе TCP также используется конвейеризация, благодаря которой отправитель может одновременно передавать несколько сегментов, не дожидаясь получения пакетов ACK на каждый из них. Мы уже убедились в том, что конвейеризация может существенно увеличить пропускную способность для сеанса при малом отношении размера сегмента к времени оборота. Количество одновременно допустимых просроченных и неподтвержденных сегментов для отправителя определятся механизмами управления потоком и управления перегрузкой протокола TCP. Механизм управления потоком обсуждается в конце этого раздела; управление перегрузкой в TCP рассматривается в разделе 3.7. Сейчас нам достаточно знать об использовании конвейеризации в протоколе TCP.

На рис. 3.32 приведены графики значений `SampleRTT` и `EstimatedRTT` при $\alpha = 1/8$ для TCP соединения между серверами `gaia.cs.umass.edu` (Амхерст, штат Массачусетс) и `fantasia.eurecom.fr` (на юге Франции). Как видно, график `EstimatedRTT` изменяется более гладко по сравнению с графиком `SampleRTT`.

Помимо непосредственно вычисления RTT, полезно знать, как оно изменяется. В документе RFC 6298[563] изменение RTT, `DevRTT`, определяется как отклонение `SampleRTT` от значения `EstimatedRTT`:

$$\texttt{DevRTT} = (1-\beta) \times \texttt{DevRTT} + \beta \times |\, \texttt{SampleRTT} - \texttt{EstimatedRTT} \,|$$

Заметим, что `DevRTT` — это взвешенное скользящее среднее отклонений `SampleRTT` от `EstimatedRTT`. Чем меньше колебания значений `SampleRTT`, тем меньше будет значение `DevRTT`. Рекомендованное значение β составляет 0,25.

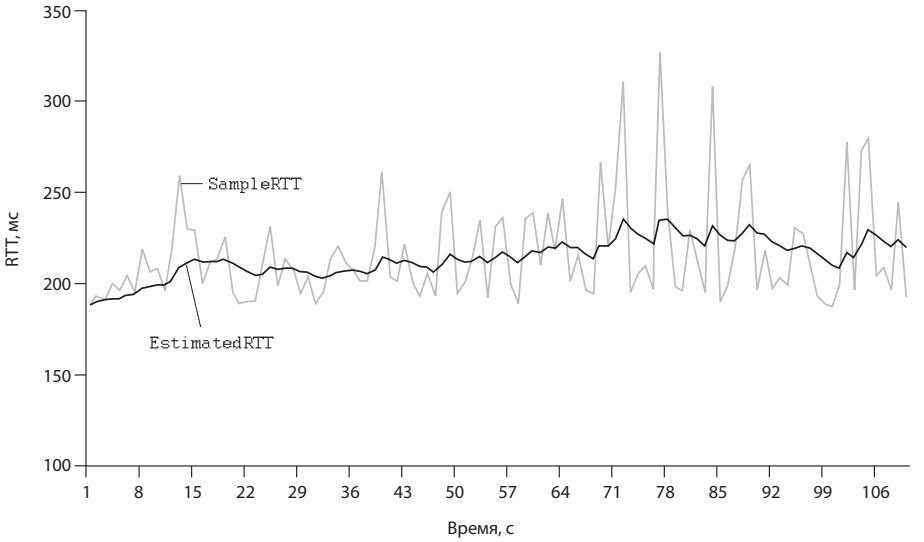


Рис. 3.32. Значения RTT и весовые средние значения RTT

Управление интервалом времени ожидания

Какие из полученных значений `EstimatedRTT` и `DevRTT` должны быть использованы для интервала времени ожидания TCP-соединения? Очевидно, это значение должно превышать или быть равным `EstimatedRTT`, в противном случае будут иметь место лишние повторные передачи. Но если интервал времени ожидания значительно больше значения `EstimatedRTT`, протокол TCP, напротив, не сможет быстро выполнить повторную передачу потерянного сегмента, что приведет к длительной задержке при передаче данных. Поэтому желательно установить интервал времени ожидания, превышающий значение `EstimatedRTT` на некоторую переменную величину. Эта переменная величина должна зависеть от частоты колебаний значений `SampleRTT`: чем больше частота колебаний, тем больше должна быть переменная величина. Вот здесь нам и понадобится значение `DevRTT`. Объединяя все сказанное, мы получаем следующую формулу, позволяющую определить интервал времени ожидания для протокола TCP:

```
TimeoutInterval = EstimatedRTT + 4 × DevRTT
```

Начальное значение `TimeoutInterval` по рекомендации документа RFC 6298[563] равно 1 секунде. При этом в случае превышения интервала времени ожидания, значение `TimeoutInterval` удваивается, чтобы избежать преждевременного таймаута для следующего сегмента,

на который в скором времени поступит подтверждение. Однако после получения сегмента и обновления значения `EstimatedRTT` значение `TimeoutInterval` вновь вычисляется на основе формулы приведенной выше.

3.5.4. Надежная передача данных

Напомним, что служба сетевого уровня Интернета (IP-протокол) ненадежная. Протокол IP не гарантирует доставку дейтаграмм, сохранения их порядка при передаче и целостности данных в них. При использовании службы протокола IP, дейтаграммы могут вызвать переполнение буферов на маршрутизаторах и никогда не достичь своего получателя или поступить в неверном порядке, а их биты могут быть искажены (изменение значения с 0 на 1 и наоборот). Так как сегменты транспортного уровня перемещаются по сети внутри IP-дейтаграмм, то указанные проблемы распространяются и на них.

Протокол TCP реализует собственную **службу надежной передачи данных** над ненадежной службой протокола IP. Служба надежной передачи данных протокола TCP гарантирует, что поток данных, который процесс считывает из своего TCP-буфера получения, не поврежден, не имеет пропусков и дубликатов, и данные в потоке расположены в верном порядке, то есть принимаемый поток байт полностью соответствует потоку байт, который был отправлен с противоположной конечной системы соединения. Предоставление надежной передачи данных протоколом TCP включает в себя принципы, изученные нами в разделе 3.4.

Ранее, создавая собственные протоколы надежной передачи данных, для упрощения мы использовали индивидуальный таймер, связанный с каждым переданным, но еще не подтвержденным сегментом. В теории такой подход весьма хорош, но на практике для управления таймерами могут потребоваться значительные ресурсы. В стандарте RFC 6298[563] содержатся рекомендации по управлению таймерами протокола TCP, предписывающие использовать *единственный* таймер для обработки повторной передачи всех переданных, но еще не подтвержденных сегментов. Протокол TCP, описанный в этом разделе, работает с единственным таймером.

Мы рассмотрим реализацию службы надежной передачи данных протокола TCP в два последовательных этапа. Для начала мы представим крайне упрощенное описание отправителя TCP, который использу-

ет только таймаут времени ожидания для восстановления потерянных сегментов; далее мы усложним описание отправителя, дополнив его дубликатами подтверждений. В последующем обсуждении мы предполагаем, что данные передаются в одном направлении: от хоста А к хосту Б — и что хост А отправляет большой файл.

```
/* Предположим, что отправитель TCP не использует
управление потоком или перегрузкой, а данные,
поступающие от приложения верхнего уровня, меньше,
чем MSS. Передача данных осуществляется только
в одном направлении */
NextSeqNum=InitialSeqNumber
SendBase=InitialSeqNumber
Цикл (бесконечно) {
switch(событие)
событие: данные получены от приложения верхнего
уровня создать сегмент TCP с порядковым номером
NextSeqNum
if (таймер не запущен)
запустить таймер
передать сегмент протоколу IP
NextSeqNum=NextSeqNum+length(данные)
break;
событие: истек интервал времени ожидания повторить
передачу неподтвержденного сегмента с наименьшим
порядковым номером запустить таймер
break;
событие: получен пакет ACK, со значением поля ACK — y
if (y > SendBase) {
SendBase=y
if (есть хотя бы один неподтвержденный)
запустить таймер
}
break;
} /* конец бесконечного цикла */
```

Рис. 3.33. Упрощенный отправитель TCP

На рис. 3.33 представлено крайне упрощенное описание отправителя TCP. Мы видим, что существует три главных события отправителя TCP, относящихся к передаче и повторной передаче: получены данные от приложения верхнего уровня, таймер времени ожидания и получение пакета ACK.

Сразу при наступлении первого главного события, протокол TCP получает данные от приложения, инкапсулирует их в сегмент и передает его протоколу IP. Заметим, что каждый сегмент включает порядковый номер, который является номером первого байта данных сегмента в потоке данных, как было описано в разделе 3.5.2. Также заметим: если таймер еще не был запущен для некоторого другого сегмента, протокол TCP запускает его при передаче сегмента протоколу IP. (Представьте, что таймер связан с самым «старым» неподтвержденным сегментом.) Интервал таймера обозначается, `TimeoutInterval`, и вычисляется на основе значений `EstimatedRTT` и `DevRTT`, как обсуждалось в разделе 3.5.3.

Второе важное событие — это истечение интервала времени ожидания. В этом случае протокол TCP повторно передает сегмент, вызвавший его, и вновь запускает таймер.

Третье важное событие, которое должно обрабатываться TCP-отправителем — это получение пакета ACK от получателя (говоря точнее, получение сегмента, содержащего верное значение в поле ACK). При наступлении этого события, протокол TCP сравнивает значение `y` в поле ACK с переменной `SendBase`. Переменная `SendBase` протокола TCP — это порядковый номер самого раннего неподтвержденного байта (то есть значение `SendBase-1` соответствует порядковому номеру последнего корректного и полученного в правильном порядке байта). Как указывалось ранее, протокол TCP использует общие подтверждения так, что значение `y` является подтверждением получения всех байт до байта с порядковым номером `y`. Если `y>SendBase`, тогда пакет ACK является подтверждением одного или нескольких неподтвержденных ранее сегментов. Далее отправитель обновляет значение переменной `SendBase` и перезапускает таймер, если все еще существуют неподтвержденные сегменты.

Несколько интересных сценариев

Мы только что описали крайне упрощенную версию механизма надежной передачи данных протокола TCP. Но даже в ней существует мно-

жество тонкостей. Чтобы ясно представлять, как работает этот протокол, давайте сейчас разберем несколько простых сценариев. На рис. 3.34 представлен первый сценарий, в котором хост А отправляет один сегмент хосту Б. Допустим, что порядковый номер этого сегмента 92 и он содержит 8 байт данных.

После отправки этого сегмента хост А ожидает от хоста Б сегмент с номером квитанции 100. Несмотря на то, что хост Б получил сегмент от хоста А, квитанция, направленная от хоста Б к хосту А, была потеряна. В связи с чем истекает интервал ожидания, и хост А повторно передает сегмент. Конечно, при получении повторно переданного сегмента хост Б выполняет проверку порядкового номера полученного сегмента и определяет, что полученный сегмент содержит ранее принятые данные. Следовательно, хост Б отклоняет биты данных повторно переданного сегмента.

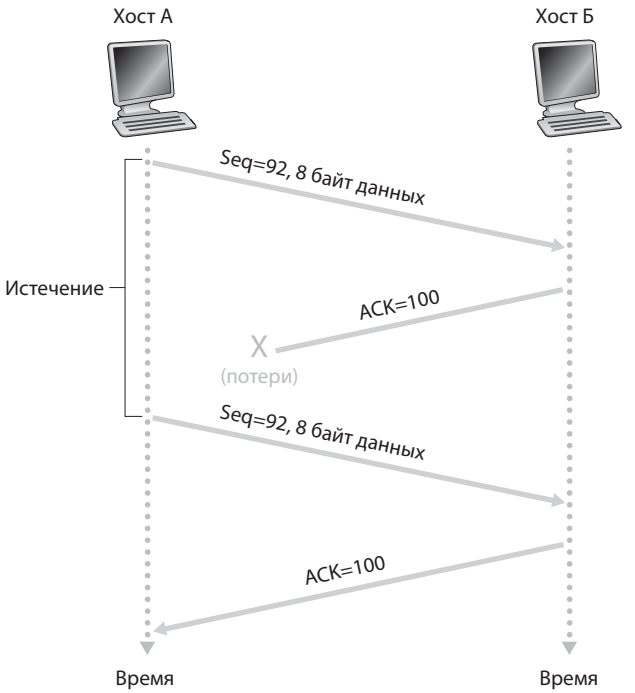


Рис. 3.34. Повторная передача, вызванная потерей квитанции

Во втором сценарии, показанном на рис. 3.35, хост А последовательно отправляет два сегмента. Порядковый номер первого сегмента — 92, сегмент содержит 8 байт данных, второму сегменту присвоен порядко-

вый номер 100 и он содержит 20 байт данных. Допустим, оба сегмента своевременно попали на хост Б, и хост Б отправляет два отдельных подтверждения на эти сегменты. Номер первого подтверждения 100, второго — 120. Теперь предположим, что ни одно из подтверждений не достигло хоста А до истечения интервала ожидания. Когда он истекает, хост А повторно отправляет сегмент с порядковым номером 92 и перезапускает таймер. Поскольку пакет ACK для второго сегмента поступает на хост до окончания следующего интервала времени ожидания, второй сегмент не отправляется повторно.

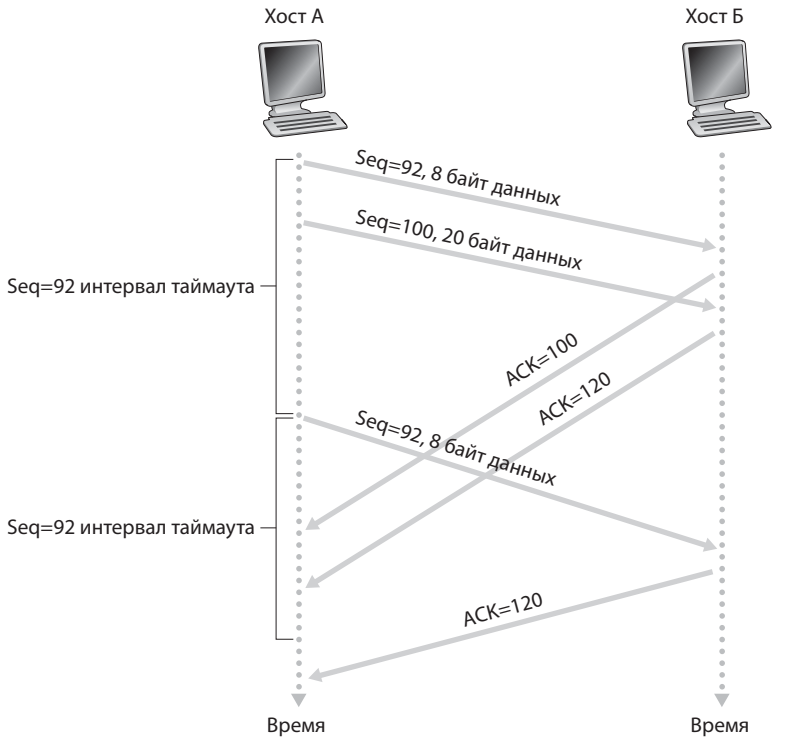


Рис. 3.35. Сегмент с порядковым номером 100 не передается повторно

Допустим, в третьем и последнем сценарии хост А отправляет два сегмента так же, как во втором. Подтверждение на первый сегмент потеряно в сети, но непосредственно перед наступлением таймаута хост А получает пакет ACK с номером подтверждения 120. Хост А, таким образом, «понимает», что хост Б получил *все* данные до 119 байта, поэтому ни один из пакетов не будет отправлен повторно хостом А. Данный сценарий изображен на рис. 3.36.

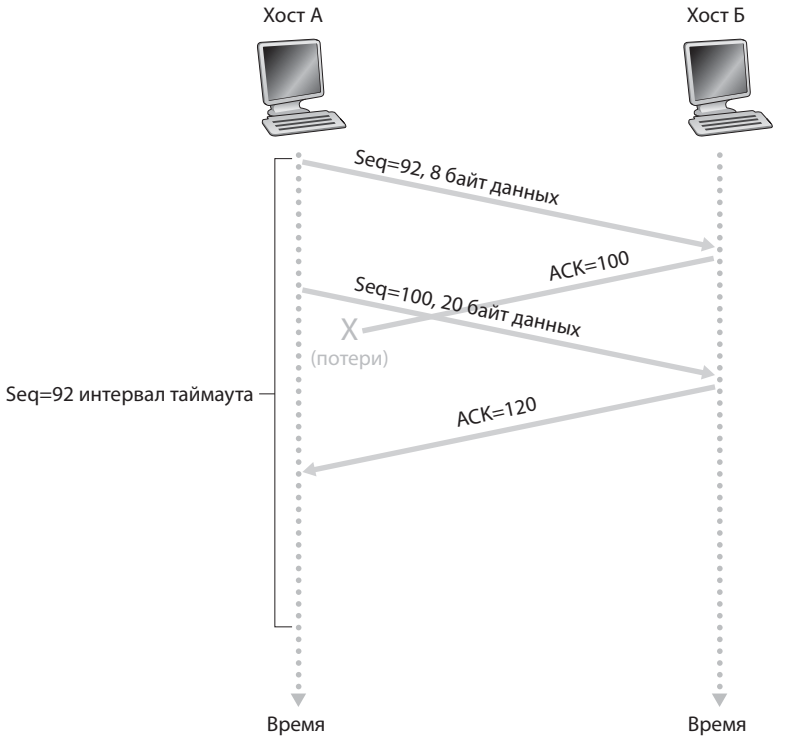


Рис. 3.36. Общая квитанция позволяет избежать повторной передачи первого сегмента

Удвоение интервала

Теперь мы рассмотрим несколько модификаций, которые применяются в большинстве реализаций протокола TCP. Первая модификация относится к изменению интервала времени ожидания, по его истечению. В этой модификации каждый раз при наступлении таймаута протокол TCP повторно передает еще не подтвержденный сегмент с наименьшим порядковым номером, как было описано выше. Но при каждой повторной передаче протокол TCP удваивает значение интервала времени ожидания по сравнению с предыдущим, а не рассчитывает его на основе последних полученных значений `EstimatedRTT` и `DevRTT` (как описывалось в разделе 3.5.3). Например, пусть значение `TimeoutInterval`, связанное с самым ранним еще не подтвержденным сегментом, равно 0,75 секунды при первом истечении интервала времени ожидания. При следующей повторной передаче этого сегмента протокол TCP устанавливает новое значение интервала времени ожидания равное 1,5 с. Если спустя 1,5 с вновь истечет интервал, протокол TCP еще раз повторит

передачу этого сегмента, и в этот раз длина интервала будет равна трем секундам. Таким образом, увеличение интервалов времени ожидания происходит экспоненциально при каждой повторной передаче. Но если таймер запускается по причине одного из двух других событий (получены данные от приложения верхнего уровня или пакет ACK), то значение `TimeoutInterval` определяется на основе последних значений `EstimatedRTT` и `DevRTT`.

Эта модификация представляет собой простую форму механизма управления перегрузкой. Более сложная форма для протокола TCP рассматривается в разделе 3.7. Истечение интервала ожидания чаще всего связано с перегрузками в сети, то есть в очереди одного (или более) маршрутизаторов на пути от отправителя до получателя скапливается много пакетов, вызывая их потерю и/или приводя к длинным задержкам очереди. Если отправитель будет настойчиво продолжать заново отправлять пакеты при наличии перегрузок, то состояние сети может стать еще хуже. Напротив, протокол TCP действует более мягко: отправитель выполняет каждую последующую повторную передачу через все увеличивающийся интервал. При изучении CSMA/CD в главе 5 мы увидим, что подобная идея используется в технологии Ethernet.

Ускоренная повторная передача

Одним из недостатков механизма повторной передачи с интервалами ожидания является то, что они часто оказываются относительно долгими. При потере пакета передающая сторона вынуждена ждать истечения интервала ожидания для того, чтобы осуществить повторную передачу, тем самым увеличивая общую задержку. Здесь на помощь приходит механизм дублирования подтверждений, позволяющий передающей стороне обнаруживать потери пакетов до истечения интервала ожидания. **Дублирующее подтверждение** — это копия положительной квитанции предыдущего сегмента, отправляемая в ответ на получение следующего сегмента, если порядковый номер последнего превышает ожидаемый. Чтобы понять реакцию передающей стороны на появление дублирующих подтверждений, сначала необходимо выяснить причину, побуждающую к использованию такого механизма. В табл. 3.2 приведены основные правила генерации квитанций принимающей стороной TCP[558]. При получении сегмента, порядковый номер которого превышает ожидаемый, протокол регистрирует отсутствие сегмента. Этот разрыв может быть вызван потерей сегмента или нарушением порядка следования сегментов в сети.

Табл. 3.2. Основные правила генерации квитанций принимающей стороной TCP[558]

Событие	Действие TCP на стороне получателя
Получение сегмента с правильным порядковым номером. Для всех данных до ожидаемого порядкового номера уже получены подтверждения.	Задержка отправки пакета ACK. Ожидание 500 мс до получения еще одного следующего по порядку сегмента. Если следующий по порядку сегмент не получен за этот интервал, отправить пакет ACK.
Получение сегмента с правильным порядковым номером. Один из уже полученных сегментов ожидает получения пакета ACK.	Немедленная отправка общей квитанции, подтверждающей получение обоих сегментов
Получение сегмента с большим, чем ожидалось порядковым номером. Обнаружен разрыв.	Немедленная отправка дублирующего подтверждения, показывающего порядковый номер следующего ожидаемого байта (которым является начальный байт промежутка)
Получение сегмента частично или полностью заполняющего разрыв в полученных данных.	Немедленная отправка подтверждения в получении сегмента, начавшего заполнение промежутка в данных

Поскольку протокол TCP не использует отрицательные подтверждения, получатель не может отправить явное отрицательное подтверждение отправителю. Вместо этого он просто повторно подтверждает (то есть генерирует дублирующий пакет ACK) последний полученный в верном порядке байт данных. (Заметим, что табл. 3.2 соответствует случаю, когда получатель не отклоняет пришедшие не по порядку сегменты.)

Поскольку передающая сторона зачастую отсылает несколько сегментов подряд, потеря одного из них вызывает генерацию множества дублирующих подтверждений.

Если на отправку одного и того же сегмента приходит три дублирующих подтверждения, передающая сторона воспринимает это как указание на потерю следующего за ним сегмента (в упражнениях, приведенных в конце главы, затрагивается вопрос о том, почему передающая сторона ожидает три дублирующих подтверждения, а не одно).

Протокол TCP в этом случае осуществляет **ускоренную повторную передачу** пропущенного сегмента, не дожидаясь истечения интервала ожидания[558]. Эта ситуация отражена на рис. 3.37, где потерянный второй сегмент повторно передан до истечения интервала ожидания. Для протокола TCP, использующего механизм ускоренной повторной передачи, следующий фрагмент кода заменяет событие получения пакета ACK на рис. 3.33:

```
событие: получен пакет ACK, со значением поля ACK
равным y
if (y>SendBase) {
     SendBase = y
     if (еще не было подтвержденных сегментов)
       запустить таймер
     }
else {/*дублирующий пакет ACK для уже
подтвержденного сегмента*/
увеличить число дублирующих пакетов ACK полученных
для пакета y
if (количество ACK полученных для y == 3)
/* быстрая повторная передача TCP*/
Повторно передать сегмент с порядковым номером y
}
  break;
```

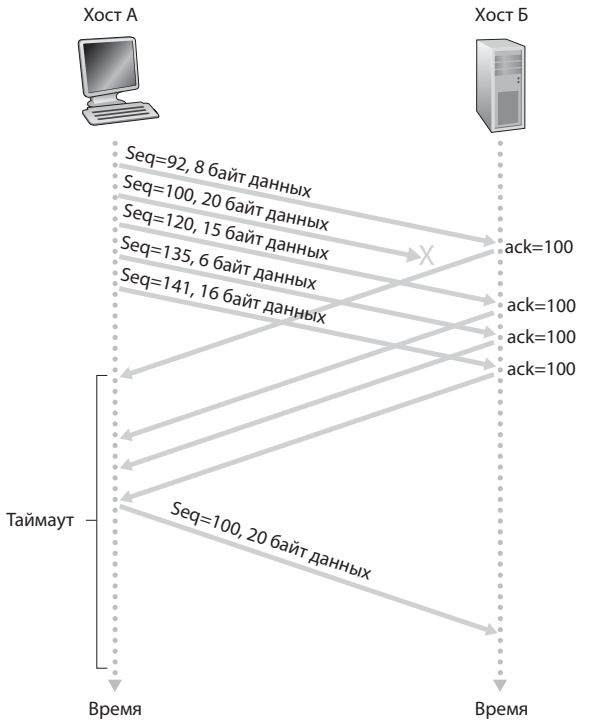


Рис. 3.37. Ускоренная повторная передача: повторная передача пропущенного сегмента до истечения интервала времени ожидания сегмента

Как мы говорили ранее, при реализации механизма интервалов ожидания и повторных передач в протоколах (например, TCP) возникает множество нюансов. Процедуры, приведенные выше, являются результатом пятнадцати лет экспериментирования с TCP-таймерами и в полной мере подтверждают это!

Возвращение на *N* шагов назад или выборочное повторение?

Давайте завершим наше изучение механизма восстановления ошибок для протокола TCP, ответив на следующий вопрос: протокол TCP относится к протоколам GBN или SR? Напомним, что протокол TCP использует общие подтверждения, т. е. полученные в верном порядке и без ошибок сегменты не подтверждаются по отдельности получателем. Следовательно, как показано на рис. 3.33 (смотрите также рис. 3.19), TCP-отправителю необходимо хранить только наименьший порядковый номер переданного, но еще не подтвержденного байта (`SendBase`), и порядковый номер следующего передаваемого байта (`NextSeqNum`). В этом смысле протокол TCP выглядит, как GBN-протокол. Но есть некоторые поразительные отличия между протоколом TCP и GBN-протоколами. В большинстве реализаций протокола TCP корректно полученные, но пришедшие в неверном порядке сегменты отправляются в буфер[617]. Рассмотрим также ситуацию, когда отправитель последовательно отправляет сегменты 1, 2, . . . , N, и все эти сегменты приходят к получателю корректно и без нарушения порядка следования. Кроме того, предположим, что подтверждение для пакета $n < N$ потеряно, но остальные $N - 1$ подтверждений были получены отправителем до того, как наступил таймаут интервала времени ожидания. В этом случае GBN-протокол отправил бы повторно не только пакет n, но и все следующие за ним пакеты $n + 1$, $n + 2$, . . . , N. Протокол TCP, напротив, повторно отправит только один сегмент n. Более того, TCP отправитель не будет выполнять повторную передачу и самого сегмента n, если до истечения интервала времени ожидания для данного сегмента было получено подтверждение сегмента $n + 1$.

Предлагаемая модификация протокола TCP использует, так называемое, **выборочное подтверждение**[463], что позволяет TCP-получателю выборочно подтверждать сегменты, а не отправлять лишь общие подтверждения на корректно полученные и идущие в верном порядке сегменты. В сочетании с выборочной повторной передачей, исключающей повторную отправку уже подтвержденных сегментов, протокол TCP становится похож на SR-протокол. Таким образом, механизм восстановления ошибок протокола TCP отнести к гибриду протоколов GBN и SR.

3.5.5. Управление потоком

Напомним, что оба хоста TCP-соединения выделяют буфер приема на своей стороне. При получении неискаженных и идущих в правильном порядке байт, TCP-соединение помещает данные в этот буфер. Связанный с соединением прикладной процесс будет считывать данные из этого буфера, но не обязательно в момент получения. На самом деле, приложение, для которого предназначены данные, может выполнять другую задачу и не пытаться считывать данные из буфера еще долгое время после получения. При относительно медленном считывании данных принимающим приложением буфер приема может легко оказаться переполнен слишком быстро отправляемым и чрезмерным количеством данных.

Протокол TCP предоставляет своим приложениям **службу управления потоком** для устранения проблемы возможного переполнения приемного буфера отправителем. Таким образом, управление потоком — это служба, регулирующая скорость отправки данных отправителем относительно скорости, с которой их считывает принимающее приложение. Как отмечалось ранее, TCP-отправитель может также замедлить собственную скорость отправки при наличии перегрузок сетевого протокола IP; такой вид управления отправителем называется **управлением перегрузкой**. Данная тема подробно рассматривается в разделах 3.6 и 3.7. Несмотря на то, что действия механизмов управления потоком и управления перегрузкой похожи (замедление отправителя), они, очевидно, используются для совершенно разных целей. К сожалению, многие авторы используют термины, как синонимы, оставляя сообразительному читателю самому разобраться в различиях этих механизмов. Давайте начнем рассмотрение службы управления потоком, предоставляемой протоколом TCP. Чтобы максимально сосредоточиться на механизме управления потоком, мы будем предполагать, что в рассматриваемой реализации TCP-получатель отклоняет сегменты, полученные не по порядку.

TCP обеспечивает управление потоком, с помощью переменной *отправителя*, называемой **окном приема**. Говоря неформально, окно приема применяется для сообщения отправителю о наличии свободного места в буфере получения. Поскольку протокол TCP выполняет дуплексную передачу данных, окно приема поддерживается отправителем, на каждой стороне соединения. Предлагаем рассмотреть окно приема в контексте передачи файла. Предположим, что хост А отправляет большой файл хосту Б по TCP-соединению. Хост Б выделяет бу-

фер получения для соединения; обозначим размер буфера переменной RcvBuffer. Периодически прикладной процесс на хосте Б считывает данные из буфера. Определим следующие переменные:

- LastByteRead: номер последнего байта полученного прикладным процессом хоста Б из буфера;
- LastByteRcvd: номер последнего байта полученного из сети и размещенного в приемном буфере хоста Б.

Поскольку протокол TCP не допускает переполнения выделенного буфера, должно выполняться неравенство:

```
LastByteRcvd – LastByteRead ≤ RcvBuffer
```

Окно приема обозначается rwnd и устанавливается равным свободному пространству в буфере:

```
rwnd = RcvBuffer – [LastByteRcvd – LastByteRead]
```

Значение rwnd является динамическим, поскольку свободное пространство в буфере постоянно меняется. Переменная rwnd изображена на рис. 3.38.

Рис. 3.38. Окно получения (rwnd) и приемный буфер (RcvBuffer)

Как соединение использует переменную rwnd для реализации службы управления потоком? Хост Б сообщает хосту А о доступном пространстве в буфере соединения, размещая текущее значение rwnd в поле окна приема каждого сегмента, направленного хосту А. Первоначально хост Б устанавливает значение rwnd равным значению RcvBuffer. Обратите внимание, что для этого хост Б должен отслеживать несколько переменных, характеризующих соединение.

Хост А в свою очередь отслеживает две переменные: `LastByteSent` и `LastByteAcked`, которые хранят данные о последнем отправленном и последнем подтвержденном байтах. Обратите внимание, что разность двух этих переменных, `LastByteSent − LastByteAcked`, равна количеству неподтвержденных данных, которые хост А отправляет хосту Б. Сохраняя количество неподтвержденных данных меньшим, чем значение `rwnd`, хост А может гарантировать, что буфер получения хоста Б не переполнен. Таким образом, хост А должен на протяжении всего соединения считать, что `LastByteSent − LastByteAcked ≤ rwnd`.

В этой схеме есть одна техническая проблема. Чтобы разобраться, в чем она заключается, предположим, что приемный буфер хоста Б заполнен до отказа и поэтому `rwnd=0`. Далее предположим, что после передачи значения `rwnd=0` хосту А, у хоста Б больше *ничего* нет для отправки хосту А. Подробно рассмотрим, что происходит. При очистке буфера хоста Б прикладным процессом по TCP-соединению хосту А не был отправлен новый сегмент с новым значением `rwnd`, поскольку сегмент передается хосту А по TCP-соединению только, если существуют данные или пакеты ACK для отправки.

Получается, что хост А никогда не получит сообщения об освобождении буфера хоста Б, соответственно, хост А будет заблокирован и больше не сможет отправлять данные! Для решения этой проблемы спецификация TCP требует передачи сегментов, содержащих один байт данных, хостом А, когда окно получателя Б равно 0. Эти сегменты будут подтверждаться получателем. Со временем буфер станет пустым, и подтверждения будут содержать не нулевые значения переменной `rwnd`.

На сайте **tinyurl.com/lyvb45h** представлен Java-апплет, имитирующий действия окна приема TCP.

Обсуждая службу управления потоком протокола TCP, мы кратко отметили, что протокол UDP не поддерживает управление потоком. Для понимания этого вопроса рассмотрим отправку последовательности UDP-сегментов от процесса хоста А процессу хоста Б. В обычной реализации протокола UDP он добавляет сегменты в буфер ограниченного размера, который находится «перед» соответствующим сокетом (т. е. перед «дверью» к процессу). Каждый раз процесс считывает один сегмент целиком. Если процесс не считывает сегменты достаточно быстро, буфер переполнится, и последующие сегменты будут потеряны.

3.5.6. Управление TCP-соединением

В этом подразделе мы рассмотрим вопросы установления и разрыва TCP-соединения. Несмотря на то, что эта тема может показаться не столь увлекательной, как тема надежной передачи данных или контроля потока, она является весьма важной, поскольку процедура установления соединения способна в значительной степени увеличить время ожидания (например, при навигации в Интернете). Более того, большинство из наиболее распространенных сетевых атак, включая невероятно популярную потоковую атаку SYN, используют уязвимость в управлении TCP-соединением. Давайте для начала рассмотрим процедуру установления TCP-соединения. Предположим, процесс, запущенный на одном хосте (клиенте), хочет инициировать соединение с процессом на другом хосте (сервере). Прикладной клиентский процесс первым делом информирует TCP-клиента, что он хочет установить соединение с серверным процессом. Далее протокол TCP на стороне клиента переходит к установлению TCP-соединения с протоколом TCP на стороне сервера следующим образом:

- *Шаг 1.* Протокол TCP на стороне клиента первым отправляет специальный TCP-сегмент протоколу TCP на серверной стороне. Этот специальный сегмент содержит данные прикладного уровня. Но один из флагов в заголовке сегмента (см. рис. 3.29), бит SYN, равен 1. Поэтому первый специальный сегмент называется SYN-сегментом. Кроме того, клиент случайным образом выбирает начальный порядковый номер (`client_isn`) и размещает этот номер в поле порядкового номера начального SYN-сегмента TCP. Этот сегмент инкапсулируется в IP-дейтаграмму и отправляется на сервер. Значительный интерес в настоящее время приобрел поиск надлежащего уровня рандомизации значения `client_isn` для снижения потенциальных угроз безопасности[81].
- *Шаг 2.* Когда IP-дейтаграмма, содержащая SYN-сегмент TCP, поступает на серверный хост (предположим, поступила!), сервер извлекает SYN-сегмент TCP из дейтаграммы, выделяет TCP-буфер и переменные соединения, и отправляет сегмент, уведомляющий клиента об установлении соединения. (В главе 8 мы увидим, что выделение этого буфера и переменных перед завершением третьего шага в трехэтапном процессе установления TCP-соединения делает его уязвимым к атаке типа «отказ в обслуживании», известной как SYN-флудинг.) Этот сегмент также не включает данных прикладного уровня, но в его заголовке содержится три важных элемента. Первый — это бит SYN равный 1. Второй — в поле подтверждения

TCP-сегмента установлено значение `client_isn+1`. Наконец, сервер выбирает собственный начальный порядковый номер (`server_isn`) и помещает выбранное значение в поле порядкового номера в заголовке TCP-сегмента. Этот устанавливающий соединение сегмент на самом деле говорит: «Я получил твой SYN-пакет для начала соединения с тобой, с выбранным тобой начальным порядковым номером равным `client_isn`. Я согласен установить это соединение. Мой начальный порядковый номер `server_isn`.» Устанавливающий соединение сегмент называется **SYNACK-сегментом**.

- *Шаг 3*. После получения SYNACK-сегмента, клиент также выделяет буфер и переменные соединения. Затем клиент отправляет серверу еще один сегмент; который подтверждает устанавливающий соединение сегмент сервера (клиент делает это, помещая значение `server_isn+1` в поле подтверждения в заголовке TCP-сегмента). Бит SYN равен нулю, поскольку соединение уже установлено. На третьем шаге установления TCP-соединения сегмент может содержать прикладные данные, передаваемые от клиента серверу.

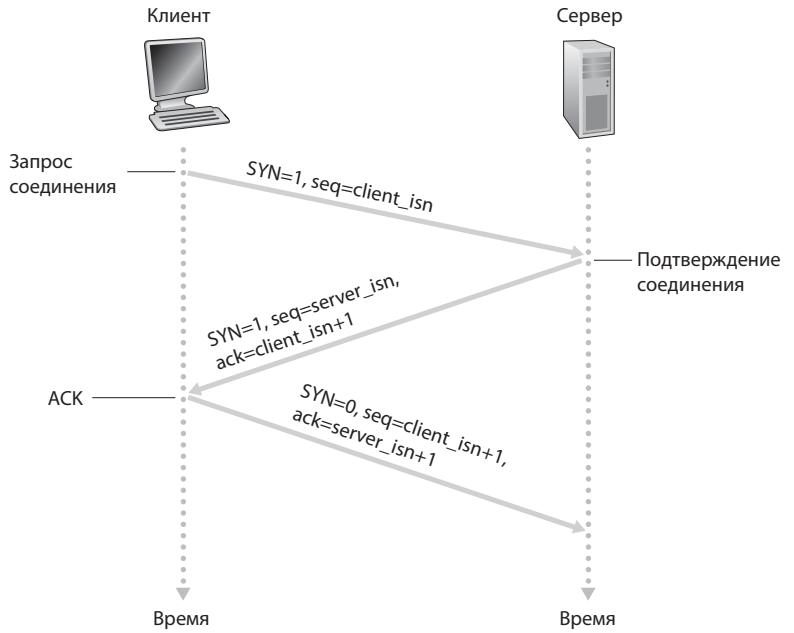


Рис. 3.39. Обмен сегментами в тройном рукопожатии протокола TCP

По завершении трех шагов установления соединения клиент и сервер могут обмениваться данными. Во всех последующих сегментах бит

SYN будет равен 0. Заметим, что для установления соединения между двумя хостами было передано три пакета, как показано на рис. 3.39. По этой причине, процедура установления соединения часто называется **тройным рукопожатием**. Несколько аспектов трехэтапного приветствия TCP рассматриваются в домашних заданиях. (Для чего необходим первоначальный порядковый номер? Почему требуется тройное рукопожатие вместо двойного?) Интересно заметить, что скалолаз и страховщик (тот, кто стоит ниже альпиниста и чья работа — заботиться о страховочном тросе скалолаза) используют идентичный TCP протокол взаимодействия с тройным рукопожатием, чтобы обе стороны были готовы, прежде чем скалолаз начнет восхождение.

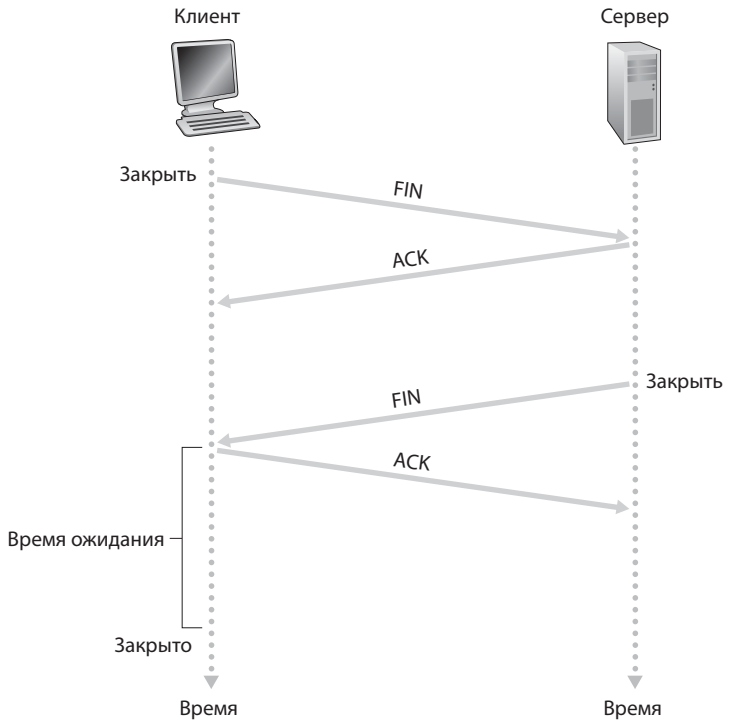


Рис. 3.40. Завершение TCP-соединения

Все хорошие вещи когда-либо заканчиваются, что верно и для TCP-соединения. Любой из двух процессов участвующих в TCP-соединении может его прекратить. Когда соединение завершается, «ресурсы» (буферы и переменные) на хостах очищаются. Предположим, например, клиент решил закрыть соединение, как показано на рис. 3.40. Прикладной процесс клиента вызывает команду закрытия соединения. В этой ситуации

TCP-клиент отправляет специальный TCP-сегмент серверному процессу. Битовый флаг FIN (см. рис. 3.29) в заголовке этого специального сегмента равен 1. При получении этого сегмента сервер отправляет клиенту подтверждающий сегмент. Далее сервер отправляет собственный сегмент закрытия соединения с битом FIN равным 1. И в завершение клиент подтверждает получение закрывающего соединение сегмента сервера. Начиная с этого момента, все ресурсы обоих хостов будут освобождены.

На протяжении всего TCP-соединения оба хоста соединения проходят через серию изменяющихся **TCP-состояний**. На рис. 3.41 изображена обычная последовательность TCP-состояний, через которые проходит *клиент*. TCP-клиент начинает с состояния CLOSED. Приложение на стороне клиента инициирует новое TCP-соединение (создание объекта `Socket` в наших Java и Python примерах в главе 2). В этом случае клиентская сторона TCP отправляет SYN-сегмент серверу. После того, как был отправлен сегмент SYN, клиентская сторона TCP переходит в состояние SYN_SENT. Находясь в состоянии SYN_SENT, клиентская сторона TCP ожидает сегмент от серверной стороны TCP, подтверждающий получение предыдущего сегмента клиента с SYN-битом равным 1. Получив такой сегмент, клиентская сторона TCP переходит в состояние ESTABLISHED. Находясь в состоянии ESTABLISHED, клиентская сторона TCP может отправлять и получать TCP-сегменты, содержащие прикладные (т. е. созданные приложением) данные.

Предположим, что клиентское приложение собирается закрыть соединение. (Отметим, что и сервер может стать инициатором закрытия соединения.) В этом случае клиентская сторона TCP-соединения отправляет TCP-сегмент с битом FIN равным 1 и переходит в состояние FIN_WAIT_1. Находясь в состоянии FIN_WAIT_1, клиентская сторона TCP ожидает TCP-сегмент от сервера с подтверждением. Когда клиентская сторона TCP получает этот сегмент, то она переходит в состояние FIN_WAIT_2. Находясь в состоянии FIN_WAIT_2, клиент ожидает следующий сегмент от сервера с битом FIN равным 1; после получения этого сегмента, клиент TCP-соединения отправляет подтверждение на сегмент сервера и переходит в состояние TIME_WAIT. Состояние TIME_WAIT позволяет клиенту TCP-соединения повторно отправить последнее подтверждение, если этот пакет ACK был потерян. Нахождение в состоянии TIME_WAIT зависит от реализации протокола, но обычно равно 30 секундам, 1 минуте или 2 минутам. После выхода из состояния ожидания TCP-соединение формально считается закрытым, при этом освобождаются все ресурсы клиента, включая номера портов.

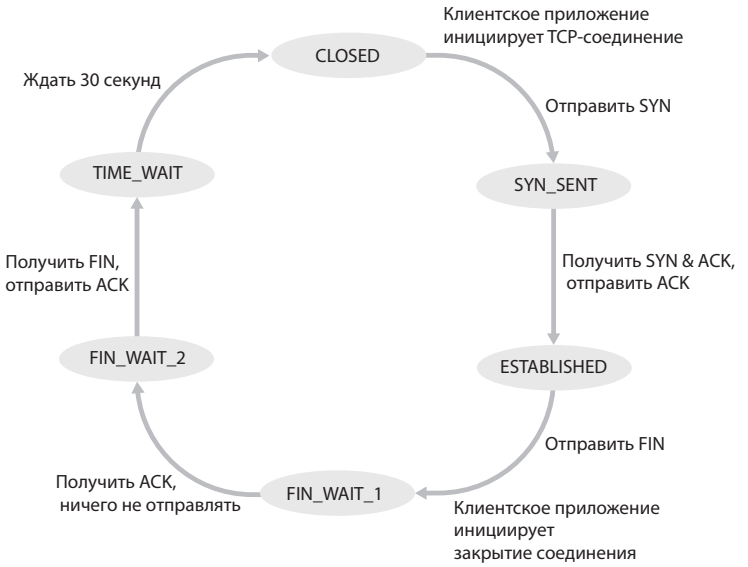


Рис. 3.41. Обычная последовательность TCP-состояний клиента

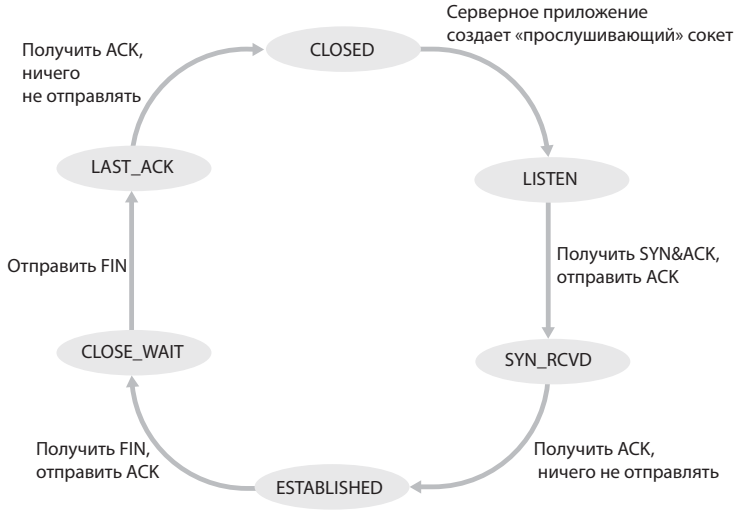


Рис. 3.42. Обычная последовательность состояний TCP проходимых TCP серверной стороны

На рис. 3.42 показана последовательность состояний, через которые обычно проходит серверная сторона TCP-соединения, предполагая, что решение о закрытии соединения принято клиентом. Переходы из одного состояния в другое понятны, и мы не будем останавливаться на их описании. Две приведенные схемы состояний соответствуют только ситуации,

когда открытие и завершение TCP-соединения происходит в штатном режиме. Мы не рассматривали такие нетипичные ситуации, как, например, одновременные попытки закрыть соединение, предпринимаемые обеими сторонами. Если вы хотите узнать об этом и других нюансах TCP, мы советуем вам обратиться к обширной книге Стивенсона[617].

О БЕЗОПАСНОСТИ

Атака SYN flood

Обсуждая тройное рукопожатие TCP-соединения, мы видели, что сервер выделяет и инициализирует переменные соединения и буферы после получения SYN-сегмента. Далее сервер отправляет ответ SYNACK-сегмент и ожидает пакет ACK от клиента. Если клиент не отправляет пакет ACK для завершения третьего шага тройного рукопожатия, в конечном счете (обычно после минуты или более) сервер прерывает наполовину открытое соединение и очищает размещенные ресурсы.

Этот протокол управления TCP-соединением создает условия для классической атаки отказа службы DoS (Denial of Service), известной как **атака SYN-flood**. В этой атаке атакующий или атакующие отправляют большое количество TCP-сегментов SYN, до того как завершится третий этап установления соединения. При таком потоке SYN-сегментов ресурсы соединений сервера истощаются, поскольку они выделяются (но никогда не используются!) для наполовину открытых соединений, что при водит к отказу в обслуживании настоящим клиентам. Такие SYN-flood атаки были в числе первых описанных DoS атак[85]. К счастью, эффективная защита, известная как **SYN cookies**[548], применяется в большинстве распространенных операционных систем. SYN cookies работают следующим образом:

- Когда сервер получает SYN-сегмент, ему неизвестно поступил данный сегмент от настоящего клиента или как часть SYN-flood атаки. Поэтому вместо создания полуоткрытого TCP-соединения для данного SYN-сегмента, сервер инициализирует начальный порядковый номер, представляющий собой сложную функцию (хеш-функцию) от IP-адресов отправителя и получателя и номеров портов SYN-сегмента, иначе говоря, секретный номер известный только серверу. Этот тщательно сработанный порядковый номер и есть так называемый «cookie». Затем сервер отправляет клиенту SYNACK-сегмент с этим специальным значением начального порядкового номера. *Важно, что сервер не запоминает cookie или любую другую значимую информацию, относящуюся к SYN-сегменту.*

- Настоящий клиент возвращает пакет ACK. При его получении сервер должен будет проверить, что пакет ACK соответствует некоторому SYN-сегменту отправленному ранее. Но каким образом это осуществить, если сервер не хранит информации о SYN-сегментах? Как вы могли догадаться, это выполняется с помощью cookie. Напомним, что для настоящего ACK-пакета значение поля подтверждения соответствует начальному порядковому номеру в SYNACK-сегменте (в данном случае это значение cookie) плюс один (см. рис. 3.39). Далее сервер может выполнить некоторую хеш-функцию, используя IP-адреса и номера портов отправителя и получателя SYNACK-сегмента (которые совпадают со значениями из первоначального SYN-сегмента) и секретный номер. Если результат функции плюс один совпадает со значением подтверждения (cookie) клиентского SYNACK-сегмента, сервер устанавливает, что ACK-пакет относится к ранее отправленному SYN-сегменту, а значит, он верный. Далее сервер создает полностью открытое соединение с сокетом.
- С другой стороны, если клиент не возвращает ACK-пакет, то оригинальный SYN-сегмент не может нанести никакого вреда серверу, поскольку сервер еще не выделил никаких ресурсов в ответ на поддельный первоначальный SYN-сегмент.

В нашем предыдущем обсуждении предполагается, что и клиент, и сервер готовы начать соединение, например, что сервер прослушивает порт, на который клиент отправляет свой SYN-сегмент. Давайте рассмотрим, что произойдет, если хост получит TCP-сегмент, чей номер порта или IP-адрес отправителя не будут совпадать с одним из открытых сокетов хоста. Например, пусть хост получил TCP-пакет SYN с портом получателя 80, но хост не принимает соединения на 80-й порт, то есть на нем не запущен веб-сервер. Тогда хост отправит специальный сбрасывающий сегмент отправителю. Этот TCP-сегмент имеет битовый флаг RST (см. раздел 3.5.2) равный 1. Таким образом, когда хост отправляет сбрасывающий сегмент, он сообщает отправителю: «У меня нет сокета для данного сегмента. Пожалуйста, не пересылай сегмент». Когда хост получает UDP-пакет, чей порт назначения не совпадает ни с одним из открытых UDP-сокетов, хост отправляет специальную дейтаграмму ICMP, как обсуждается в главе 4.

Теперь, когда мы достаточно разбираемся в управлении TCP-соединением, давайте вновь обратимся к инструменту сканирования портов nmap и более детально разберемся, как он работает. Чтобы проверить определенный TCP-порт, скажем, порт 6789, на заданном хосте, nmap

отправит этому хосту TCP-сегмент SYN с портом назначения 6789. Существует три возможных варианта развития событий:

- *Хост-отправитель получит TCP-сегмент SYNACK от заданного хоста*. Поскольку это означает, что приложение запущено на TCP-порту 6789 на заданном хосте, nmap вернет «open» (открыт).
- *Хост-источник получит TCP-сегмент RST от заданного хоста*. Это означает, что SYN-сегмент достиг заданного хоста, но на том не запущено приложение с TCP-портом 6789. Но атакующему известно, что сегменты направленные хосту на порт 6789 не блокируются никаким брандмауэром на пути между отправителем и заданным хостом. (Брандмауэры обсуждаются в главе 8.)
- *Хост-источник не получит ничего*. Это скорее всего означает, что SYN-сегмент был блокирован вмешавшимся брандмауэром и никогда не достигнет заданного хоста.

Nmap — мощный инструмент, который может проверить соединение не только с открытыми TCP-портами, но также и с открытыми UDP-портами, для брандмауэров и других конфигураций, и даже для различных приложений и операционных систем. Большинство действий этого инструмента связано с манипулированием управляющими сегментами TCP-соединения[603]. Вы можете загрузить программу nmap с сайта **www.nmap.org**.

На этом наше введение в управление ошибками и потоками протокола TCP завершается. В разделе 3.7 мы вернемся к протоколу TCP и рассмотрим механизм управления перегрузками TCP-соединения немного глубже. Но перед этим мы сделаем шаг назад и рассмотрим управление перегрузкой в широком смысле.

3.6. Принципы управления перегрузкой

В предыдущем разделе мы рассмотрели общие принципы механизма надежной передачи данных в условиях возможных потерь пакетов и реализацию этого механизма в протоколе TCP. Ранее мы упоминали, что на практике такие потери обычно являются результатом переполнения буферов маршрутизаторов при перегрузке сети. Повторная передача пакетов, таким образом, является симптомом наличия перегрузки сети (потери специфических сегментов транспортного уровня), но не устраняет ее причину: слишком много отправителей пытается отправ-

лять большое количество данных. Чтобы справиться с этим, необходим механизм замедления отправителей при возникновении перегрузок сети.

В этой части мы сосредоточимся на задаче управления перегрузкой в общем контексте, пытаясь выяснить: почему она является плохим симптом и как влияет на взаимодействие с приложениями более высокого уровня; изучая различные подходы, которые могут помочь избежать перегрузки, и каким образом на нее реагировать. Изучение общих принципов управления потоком целесообразно, поскольку, как и надежная передача данных, он входит в «топ десять» списка главных проблем сетей. Мы завершим эту часть рассмотрением механизма управления перегрузкой реализованного в службе **ABR** (available bit-rate — **доступная битовая скорость**) сетей **ATM** (asynchronous transfer mode — **режим асинхронной передачи**). Следующий раздел содержит подробное изучение алгоритмов управления перегрузкой в протоколе TCP.

3.6.1. Причины и последствия перегрузки

Мы начнем изучение вопроса с трех ситуаций возникновения перегрузок в сети в порядке возрастания сложности. Для каждой ситуации мы выявим причину и укажем ее негативные последствия (невозможность полного использования ресурсов, ухудшение качества обслуживания). Сейчас мы не станем уделять внимание способам возможного реагирования на перегрузки или их недопущения; нас будет интересовать, что происходит в сети при увеличении частоты передачи пакетов источниками, приводящем к перегрузкам.

Сценарий 1: два отправителя и маршрутизатор с неограниченным буфером

Мы начнем с рассмотрения, наверное, простейшего из возможных сценариев перегрузки: два хоста (А и Б) имеют соединение через один маршрутизатор на пути между отправителем и получателем, как показано на рис. 3.43.

Предположим, что приложение хоста А отправляет данные по соединению (например, передача данных протоколу транспортного уровня с использованием сокета) со средней скоростью $\lambda_{вход.}$ байт/с. Эти данные являются исходными в том плане, что каждый фрагмент данных направляется в сокет только однажды. Нижележащий прото-

кол транспортного уровня является простым. Данные инкапсулируются и отправляются без использования механизмов устранения ошибок (например, повторной передачи) управления потоком или управления перегрузкой. Если игнорировать дополнительные накладные расходы, связанные с добавлением заголовочной информации транспортного и более низкого уровня, скорость, с которой хост А передает трафик маршрутизатору в первом сценарии, равна $\lambda_{вход.}$ байт/с. Хост Б действует в схожей манере, и мы для упрощения предполагаем, что он также отправляет данные со скоростью $\lambda_{вход.}$ байт/с. Пакеты от хоста А и хоста Б передаются через маршрутизатор по общему исходящему каналу с пропускной способностью *R*. Маршрутизатор имеет буферы, которые позволяют ему сохранять входящие пакеты, когда скорость поступления пакетов превышает возможности исходящего канала. В первом сценарии мы предполагаем, что буфер маршрутизатора имеет неограниченный объем.

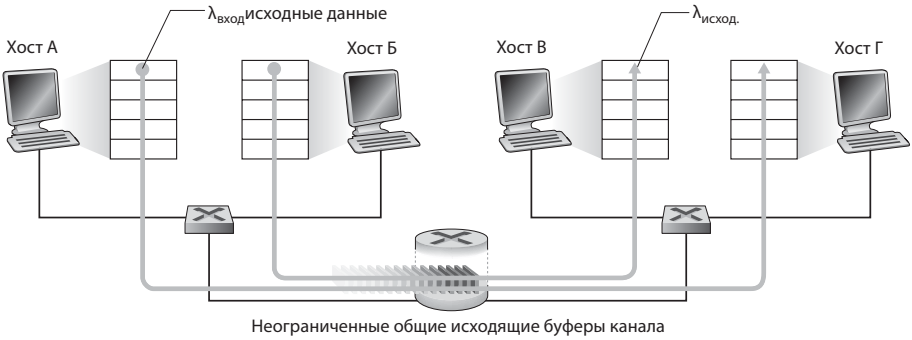


Рис. 3.43. Перегрузка, сценарий 1: два соединения, совместно использующих маршрутизатор с буфером неограниченного объема

Рисунок 3.44 изображает представление соединения хоста А, согласно этому первому сценарию. График слева изображает зависимость производительности соединения (числа байтов в единицу времени, получаемых принимающей стороной) от скорости передачи данных передающей стороной. При скоростях передачи, лежащих в диапазоне от 0 до *R*/2, производительность соединения равна скорости передачи: все данные, посылаемые в сеть передающей стороной, достигают хоста назначения. Если скорость передачи превышает значение *R*/2, производительность соединения остается равной *R*/2. Величина *R*/2 является максимальным значением производительности для данного случая, и какими бы ни были скорости передачи хостов А и Б, ни одному из них не удастся преодолеть этот максимум.

С одной стороны, достижение максимальной производительности каждым соединением можно считать хорошим результатом, поскольку в этом случае ресурсы линии связи используются полностью. Правый график на рис. 3.44, однако, показывает зависимость количества непосредственно близких к пределу возможностей канала. Когда скорость передачи данных достигает $R/2$ (слева), средняя величина задержки все растет и растет. Когда скорость передачи данных превышает $R/2$, среднее число пакетов в очереди маршрутизатора становится неограниченным, и средняя задержка передачи данных между отправителем и получателем приближается к бесконечности (если предположить, что передающие стороны не изменяют скорость передачи в течение бесконечного промежутка времени). Таким образом, оказывается, что высокая производительность негативно воздействует на время передачи данных. Даже в такой чрезвычайно простой и идеализированной ситуации мы столкнулись с вредным воздействием перегрузки в сети на качество обслуживания: *чем ближе скорость передачи данных к пропускной способности линии связи, тем большими становятся задержки ожидания пакетов.*

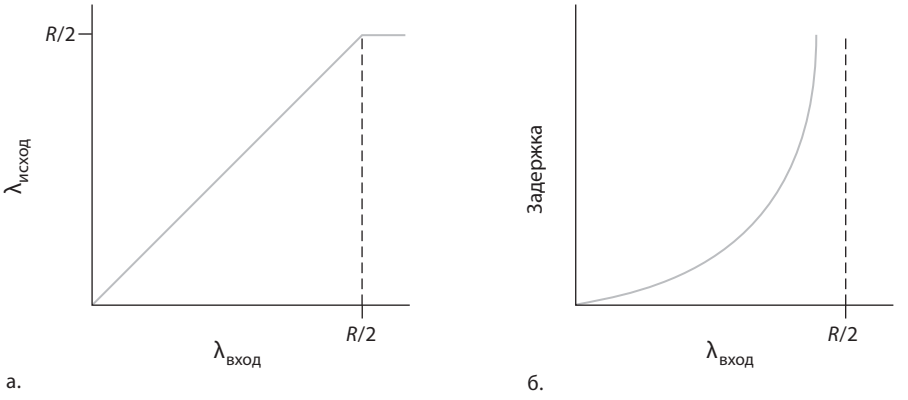


Рис. 3.44. Перегрузка, сценарий 1: пропускная способность канала и задержка передачи данных, как функция от скорости отправки хоста

Сценарий 2: два отправителя и маршрутизатор с буферами ограниченного объема

Давайте внесем в предыдущую ситуацию несколько изменений (см. рис. 3.45). Во-первых, предположим, что теперь объем буфера маршрутизатора ограничен. Последствия такого реалистичного допущения в том, что пакеты будут отбрасываться при поступлении в буфер,

заполненный до отказа. Во-вторых, предположим, что любое соединение надежно. Если пакет, содержащий сегмент транспортного уровня, отбрасывается маршрутизатором, отправитель естественно перенаправит его. Поскольку пакеты могут быть отправлены повторно, мы должны теперь быть более осторожными при использовании термина *скорость отправки*. Конкретно, позвольте нам вновь определить скорость, с которой приложение отправляет оригинальные данные в сокет, как $\lambda_{вход.}$ байт/с. Скорость, с которой транспортный уровень отправляет сегменты (содержащие оригинальные и повторно переданные данные) в сеть будем обозначать $\lambda'_{вход.}$ байт/с. Значение $\lambda'_{вход.}$ иногда называют **ожидаемой нагрузкой** сети.

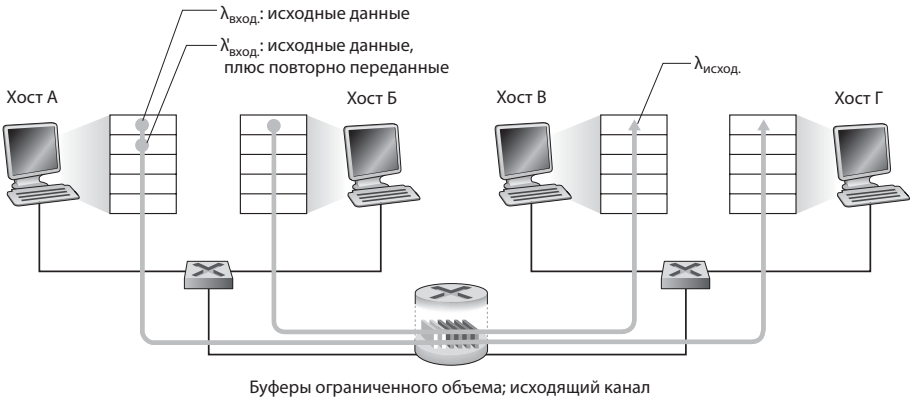


Рис. 3.45. Сценарий 2: два хоста (с возможностью повторной передачи) и маршрутизатор с буферами ограниченного объема

Представление, реализованное в сценарии 2, теперь будет строго зависеть от представления повторной передачи. Во-первых, рассмотрим нереалистичный случай, когда хост А может каким-то образом (волшебным!) определить, свободен или нет, буфер маршрутизатора и поэтому выполняет передачу пакета только, когда буфер свободен. В этом случае не происходят потери, $\lambda_{вход.}$ будет равно $\lambda'_{вход.}$, и пропускная способность соединения будет равна $\lambda_{вход.}$. Этот случай показан на рис. 3.46(а). С точки зрения пропускной способности представление идеальное: все что отправлено — получено. Заметим, что средняя скорость отправки хоста в таком сценарии не может превышать $R/2$, поскольку предполагается, что потеря пакета никогда не произойдет.

Рассмотрим немного более реалистичный случай, когда отправитель выполняет повторную передачу только, если точно известно, что пакет

был потерян. (Опять же, данное предположение несколько натянуто. Но, возможно, что отправляющий хост должен установить значение таймаута достаточно большим, чтобы быть уверенным, что пакет, который еще не был подтвержден, был потерян.) В этом случае соединение должно выглядеть примерно как на рис. 3.46б. Чтобы оценить, что здесь произошло, рассмотрим случай, когда ожидаемая загрузка $\lambda'_{вход.}$ (скорость передачи исходных данных и повторной передачи), равна $R/2$. Согласно рис. 3.46б, производительность в этом случае равна $R/3$. Таким образом, суммарная скорость $0{,}5R$ делится на две составляющие: одна, равная $0{,}333R$, характеризует передачу новых данных, а другая, равная $0{,}166R$, относится к повторным передачам. Здесь мы наблюдаем еще одно негативное влияние перегрузки на качество обслуживания: *отправитель должен выполнять повторную передачу отброшенных в связи с переполнением буфера пакетов*.

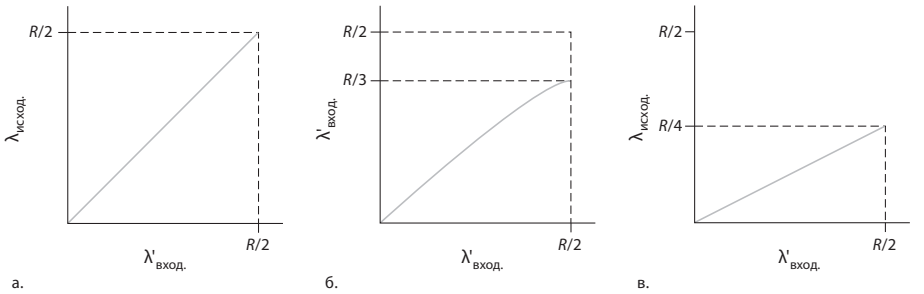


Рис. 3.46. Сценарий 2: производительность с буферами ограниченного объема

Наконец, рассмотрим случай, когда задержки пакета могут вызывать получение квитанции передающей стороной после истечения интервала ожидания (пакет при этом не теряется). В этом случае и оригинальный пакет данных, и повторно переданный могут достичь получателя. Конечно, получателю нужен только один экземпляр этого пакета и он отклонит повторную передачу. В этом случае работа, выполняемая маршрутизатором по перенаправлению повторно переданной копии оригинального пакета, была проделана впустую, так как получатель уже получил исходный пакет. Маршрутизатор мог бы использовать пропускную способность канала с большей пользой, выполняя передачу следующего пакета. *Таким образом, мы видим еще одно следствие перегрузки сети — повторные передачи в сочетании со значительными задержками ожидания приводят к бесполезной трате значительной доли пропускной способности линий связи*. На рис 3.46(в) показана производительность передачи, при которой каждый пакет в среднем предполо-

жительно перенаправлен маршрутизатором дважды. Поэтому значение пропускной способности будет асимптотически равно $R/4$ так, как ожидаемая загрузка предполагается равной $R/2$.

Сценарий 3: четыре отправителя и четыре маршрутизатора с буферами ограниченного объема

В нашем последнем сценарии перегрузки четыре хоста передают пакеты друг другу по соединению через два маршрутизатора каждый, как показано на рис. 3.47. Мы вновь предполагаем, что каждый хост использует механизм интервалов времени ожидания и повторной передачи для реализации службы надежной передачи данных, что все хосты имеют одно и то же значение $\lambda_{вход.}$ и что все каналы маршрутизаторов имеют пропускную способность R байт/с.

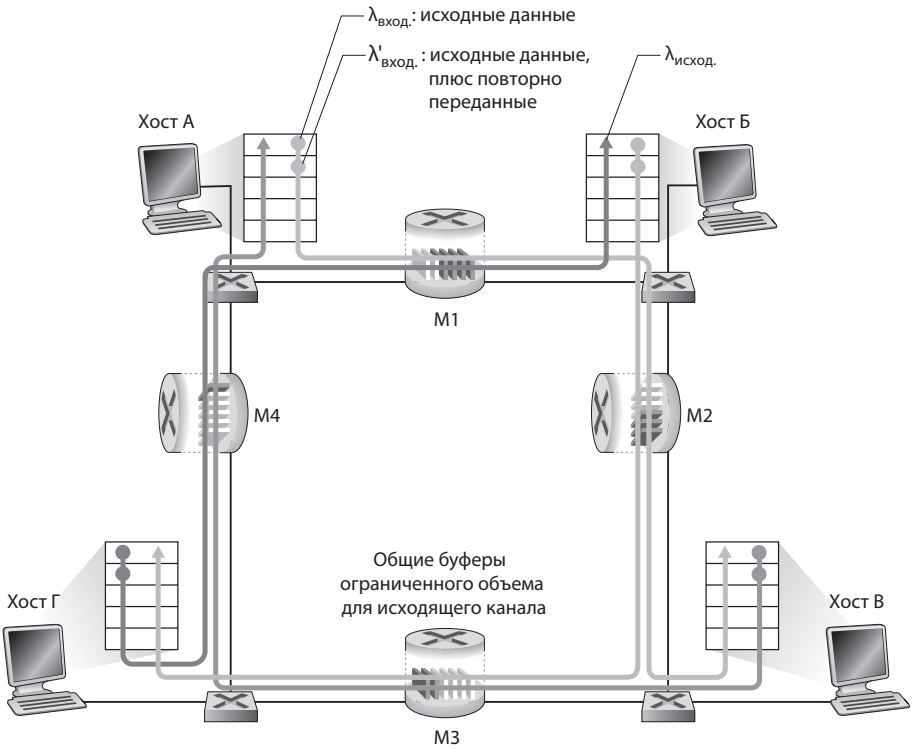


Рис. 3.47. Четыре отправителя, маршрутизаторы с буферами ограниченного объема

Давайте рассмотрим соединение между хостами A и B, проложенное через маршрутизаторы M1 и M2. Соединение A-B использует маршру-

тизатор М1, который также участвует в соединении Г-Б, и маршрутизатор М2 совместно с соединением Б-Г. Для экстремально малых значений $\lambda_{вход.}$ переполнение буфера происходит редко (как в сценариях перегрузки 1 и 2), и пропускная способность приблизительно равна ожидаемой нагрузке. Для несколько больших значений $\lambda_{вход.}$ пропускная способность соответственно увеличивается, поскольку большее количество исходных данных передается в сеть и доставляется получателю, переполнение все еще остается редкой ситуацией. Таким образом, увеличение небольших значений $\lambda_{вход.}$ приводит к увеличению $\lambda_{исход.}$.

Рассмотрев случай предельно низкой нагрузки на сеть, давайте далее перейдем к ситуации, в которой $\lambda_{вход.}$ (и, следовательно, $\lambda'_{вход.}$) принимают экстремально большие значения. Рассмотрим маршрутизатор М2. Трафик соединения А–В поступает на маршрутизатор М2 (после перенаправления с маршрутизатора М1) со скоростью, не превышающей *R*, возможность канала между маршрутизаторами М1 и М2 для любых значений $\lambda_{вход.}$. Если значение $\lambda_{вход.}$ экстремально большое для всех соединений (включая соединение Б–Г), тогда скорость поступления трафика соединения Б–Г на маршрутизатор М2 может намного превышать скорость трафика соединения А–В. Поскольку передаваемые по соединениям А–В и Б–Г данные должны быть получены на маршрутизаторе М2 при ограниченном пространстве буфера, количество трафика соединения А–В, который успешно пройдет через М2 (то есть не будет потерян из-за переполнения буфера) уменьшается при увеличении ожидаемой нагрузки соединения Б–Г. В пределе, когда ожидаемая нагрузка стремится к бесконечности, то есть пустой буфер маршрутизатора М2 немедленно заполняется пакетами соединения Б–Г, производительность соединения А–В стремится к нулю. То есть, иначе говоря, *предполагается, что пропускная способность соединения А–В стремится к нулю в условиях интенсивного трафика.* Эти соображения позволяют представить соотношение ожидаемой нагрузки и пропускной способности, как показано на рис. 3.48.

Причина возможного снижения пропускной способности очевидна и заключается в увеличении количества бесполезной работы сети. В описанном выше сценарии с большим объемом передаваемых данных при отбрасывании пакета вторым маршрутизатором работу по перенаправлению пакета на него, выполняемую первым маршрутизатором, можно считать проделанной «впустую». Производительность сети была бы лучше (точнее не хуже), если бы первый маршрутизатор просто отбрасывал такой пакет и оставался без дела. Кроме того, более выгодно

было бы использовать ресурсы маршрутизатора для передачи других пакетов, вместо повторной передачи потерянного пакета второму маршрутизатору. (Например, при выборе пакета для отправки, возможно, было бы лучше предоставить возможность маршрутизатору выставлять более высокий приоритет для пакетов, которые уже прошли через большее число маршрутизаторов.) *Итак, мы увидели еще одно следствие перегрузки сети: при отбрасывании пакета на одном из маршрутизаторов работа каждого предыдущего маршрутизатора по передаче пакета получателю считается проделанной впустую.*

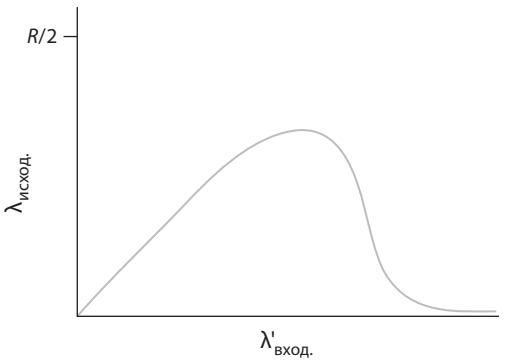


Рис. 3.48. Сценарий 3: производительность с буферами ограниченного объема

3.6.2. Подходы к управлению перегрузкой

В разделе 3.7 мы достаточно подробно рассмотрим особенный подход применяемый для управления перегрузкой в протоколе TCP. В этом разделе мы выделим два основных подхода, которые применяются на практике, рассмотрим определенные сетевые архитектуры и протоколы управления перегрузкой в составе этих подходов.

В самом широком смысле мы можем выделить несколько подходов к управлению перегрузкой, отличающихся степенью помощи сетевого уровня транспортному уровню в управлении перегрузкой:

- *Управление перегрузкой на конечных системах.* В этом подходе сетевой уровень предоставляет *неявную поддержку* транспортному уровню для управления перегрузкой. Даже обнаружение перегрузки в сети должно выполняться исключительно конечными системами путем отслеживания работы сети (например, с фиксацией потерь пакетов или задержки). В разделе 3.7 мы увидим, что протокол TCP

в обязательном порядке использует подход к управлению перегрузкой на конечных системах, поскольку IP-протокол сетевого уровня не предоставляет уведомления о наличии перегрузки в сети. Потеря TCP-сегмента (на что указывает истечение интервала ожидания или получение трех дублирующих подтверждений) используется как индикатор сетевой перегрузки, и протокол TCP, соответственно, уменьшает размер своего окна. Мы также рассмотрим новейшие предложения по управлению перегрузкой в протоколе TCP, в которых увеличение времени оборота воспринимается, как показатель увеличения сетевой перегрузки.

- *Управление перегрузкой с поддержкой на сетевом уровне*. В этом механизме предполагается наличие явной обратной связи, с помощью которой сетевые устройства (маршрутизаторы) информируют конечные системы о нагрузках в сети. В простейшем случае обратная связь может представлять собой передачу бита, свидетельствующего о наличии или отсутствии перегрузки в маршрутизаторе. Подобный подход был реализован в архитектурах IBM SNA[594] и DEC DECnet[262, 405], он также использовался в механизме управления перегрузок ABR сетей ATM, как описано ниже. Также возможна и более сложная форма уведомления. Например, одна из форм управления перегрузкой ABR сетей ATM, которую мы скоро рассмотрим, предоставляет маршрутизатору возможность в явном виде сообщать отправителю о скорости передачи, которую он (маршрутизатор) способен поддерживать в текущем канале. В протоколе XCP[278] используется уведомление, вычисляемое маршрутизатором для каждого отправителя, при этом в заголовке пакета явно указывается, что должен сделать отправитель: понизить или повысить скорость передачи данных.

При сетевом управлении уведомление о состоянии перегрузки, отправляемое отправителю сетью, обычно формируется одним из способов, представленных на рис. 3.49. Явное уведомление может быть отправлено отправителю сетевым маршрутизатором. Такое уведомление обычно принимает форму **пакета-заглушки** (явно сообщающего: «Я перегружен!»). Второй вариант уведомления используется, когда для сообщения о перегрузке маршрутизатор отмечает/обновляет поля в пакете, направляемом от отправителя к получателю. Как только будет принят такой пакет, получатель сообщит отправителю о наличии перегрузки. Заметим, что в последнем случае отправитель узнает о перегрузке только по прошествии интервала времени оборота.

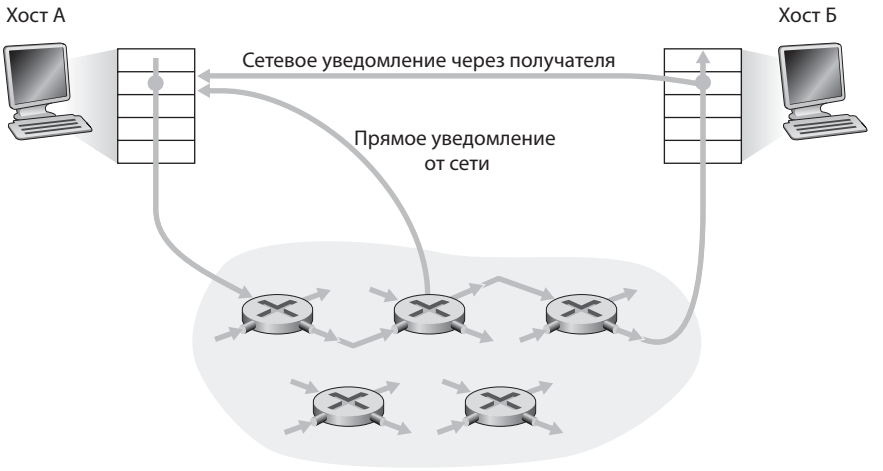


Рис. 3.49. Два возможных уведомления от сети о наличии перегрузки

3.6.3. Пример сетевого управления перегрузкой: служба управления перегрузкой ABR сетей ATM

Мы завершим изучение принципов управления перегрузкой, рассмотрев механизмы ее контроля конечными системами на примере протокола TCP, а сейчас остановимся на службе ABR сетей ATM, в которой реализован другой вид контроля перегрузок с сетевой поддержкой. Важно отметить, что сейчас наша цель заключается не в том, чтобы подробно рассмотреть нюансы архитектуры сети ATM. Нас интересует протокол, в котором управление перегрузками построено прямо противоположным образом по сравнению с протоколом TCP. Ниже мы представим лишь несколько аспектов архитектуры ATM, которые необходимы для понимания управления перегрузкой ABR.

В основе коммутации пакетов, реализуемой в сетях ATM, лежит работа с виртуальными каналами (VC). Как вы помните из главы 1, это означает, что каждый маршрутизатор на пути от исходного к конечному хосту поддерживает информацию о состоянии виртуального канала между ними.

Такое поканальное состояние позволяет коммутатору отслеживать работу отдельных отправителей (например, регистрировать среднюю скорость передачи), а также индивидуально выстраивать управление перегрузками для каждого конкретного отправителя (например, при возникновении перегрузки явно сигнализировать отправителю, что

скорость передачи необходимо уменьшить). Благодаря такому механизму сеть ATM идеально подходит для управления перегрузками с поддержкой на сетевом уровне.

Протокол ABR был разработан как эластичная (адаптивная) служба передачи данных, по действиям напоминающая протокол TCP. При низких нагрузках в сети служба ABR должна была улучшать качество обслуживания, используя свободные ресурсы, а в условиях перегрузки снижать скорости передачи до заранее установленного минимума. Подробное руководство по управлению перегрузкой ATM сети ABR и управления трафиком приведено в работе Джейна[264].

На рис. 3.50 показана схема управления перегрузкой ATM сети ABR. В нашем обсуждении мы адаптировали терминологию (например, использовали термин *коммутатор* вместо *маршрутизатор*, и термин *ячейка* вместо *пакет*). В службе ATM сети ABR ячейки данных передаются от отправителя получателю через последовательность промежуточных коммутаторов. Внедрение между ячеек данных — **ячеек управления ресурсами (RM-ячейки)** может быть использовано для распространения информации о перегрузке сети на коммутаторах и хостах. Когда ячейка управления ресурсами поступает к получателю, она разворачивается и отправляется обратно к отправителю (возможно, при этом получатель изменит содержимое ячейки). Кроме того, коммутаторы могут самостоятельно создавать ячейки управления ресурсами и переадресовывать их непосредственно отправителю. Таким образом, ячейки могут использоваться для предоставления прямого сетевого уведомления и сетевого уведомления через получателя, как показано на рис. 3.50.

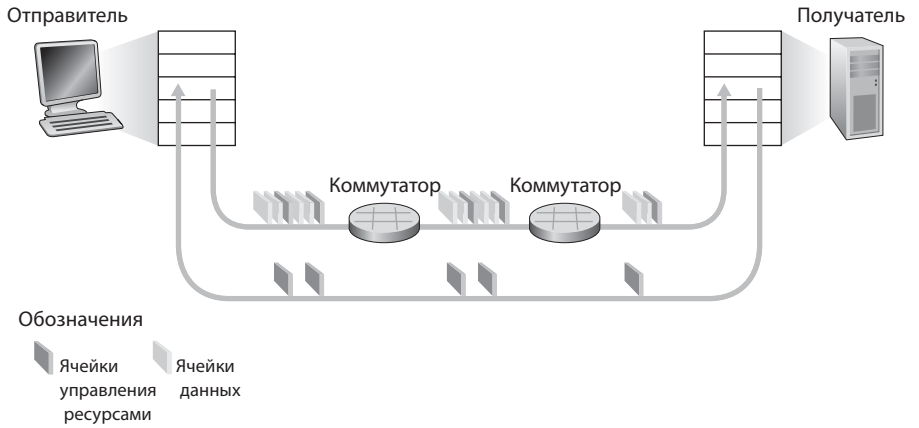


Рис. 3.50. Схема управления перегрузкой для службы ATM сети ABR

Механизм ABR-контроля перегрузок в сетях ATM основан на величине скорости передачи: передающая сторона явно рассчитывает максимальную скорость, с которой может вестись передача, и при необходимости регулирует ее. У ABR есть три возможности оповестить через коммутатор получателя о наличии перегрузок:

- *EFCI-бит.* Каждая *ячейка данных* содержит **бит явного признака ожидаемой перегрузки (explicit forward congestion indication, (EFCI) bit)**. Перегруженный сетевой коммутатор может установить EFCI-бит в ячейке данных равным 1, чтобы указать на перегрузку хосту получателю. Хост получатель должен проверить бит EFCI во всех полученных ячейках данных. Когда ячейка управления ресурсами поступает к получателю, если совсем недавно полученная ячейка данных имеет бит EFCI равный 1, тогда назначение устанавливает бит индикации перегрузки (бит CI) ячейки RM равным 1, и посылает ячейку управления ресурсами обратно отправителю. Используя EFCI в ячейках данных и бит CI в ячейках управления ресурсами, отправитель может тем самым сообщать о перегрузке на сетевом коммутаторе.

- *Биты CI и NI.* Как замечено выше, RM-ячейки передаваемые от отправителя к получателю перемежаются с ячейками данных. Частота вставки ячеек управления ресурсами является настраиваемым параметром, по умолчанию равным одной ячейке управления ресурсами на каждые 32 ячейки данных. Эти ячейки управления ресурсами имеют **бит индикатора перегрузки (congestion indication (CI) bit)** и **бит запрета увеличения (no increase (NI) bit),** которые могут быть установлены перегруженным сетевым коммутатором. Например, коммутатор установит бит NI при передаче ячейки управления ресурсами равным 1 при слабой загруженности, и бит CI равным 1 в условиях сильной перегрузки. Когда хост-получатель получает ячейку управления ресурсами, он пошлет эту ячейку обратно отправителю с нетронутыми битами CI и NI (кроме того, CI может быть установлен равным 1 хостом-получателем в результате работы EFCI механизма описанного выше).

- *Поле ER.* Каждая ячейка RM также содержит 2 байта **поля явной скорости (explicit rate (ER) field)**. Перегруженный коммутатор может понизить значение поля ER, передав ячейку управления ресурсами. Таким образом, полем ER будет установлена минимальная поддерживаемая скорость на всех коммутаторах на пути от отправителя до получателя.

В сетях ATM ABR отправитель регулирует скорость, с которой он может отправлять ячейки, как функция переменных CI, NI, и ER в возвращаемых ячейках управления ресурсами. Правила такой регулировки скорости достаточно сложны и не слишком интересны. Если они вас интересуют, вы можете почитать работу Джейна[264], где подробно рассмотрена эта тема.

3.7. Управление перегрузкой TCP

В этом разделе мы вернемся к изучению протокола TCP. Как мы узнали в разделе 3.5, протокол TCP предоставляет надежный сервис транспортного уровня между двумя процессами, запущенными на разных хостах в сети. Другой ключевой компонент протокола TCP — это механизм управления перегрузкой. Как указано в предыдущей части, протокол TCP должен использовать контроль перегрузки конечными системами, а не сетевой контроль перегрузки, так как IP-протокол сетевого уровня не предоставляет от конечных систем информации, которая сигнализировала бы о перегрузке.

Подход протокола TCP заключается в задании ограничений каждому отправителю на скорость, с которой он отсылает трафик по соединению, как функции, зависящей от загруженности сети. Если TCP-отправитель понимает, что загруженность на пути до удаленного хоста невелика, то он увеличивает скорость отправки; если отправитель определяет, что на пути соединения сеть перегружена, то снижает скорость отправки. Но этот подход порождает три вопроса. Во-первых, каким образом TCP-отправитель ограничивает скорость, с которой он передает данные по соединению? Во-вторых, как TCP-отправитель определяет наличие перегрузки на пути между ним и удаленным хостом? И, в-третьих, какой алгоритм должен использовать отправитель для изменения скорости отправки, как функции от наличия перегрузки между точками соединения?

Для начала давайте рассмотрим, как TCP-отправитель ограничивает скорость, с которой он отправляет трафик по своему соединению. В разделе 3.5 мы видели, что каждая сторона TCP-соединения содержит буфер передачи и приемный буфер, а также несколько переменных (`LastByteRead`, `rwnd` и т.д.) Механизм управления перегрузкой протокола TCP взаимодействует с отправителем, сохраняя путь в дополнительную переменную, **окно перегрузки**. Окно перегрузки обозначается

cnwd и накладывает условие на скорость, с которой TCP-отправитель может отправлять трафик в сеть. В частности, количество неподтвержденных данных отправителя не может превышать минимума cnwd и rwnd, т. е.:

```
LasrByteSent – LastByteAcked <= min {cwnd,rwnd}
```

Чтобы сосредоточиться на управлении перегрузкой (в противоположность управлению потоком), позвольте нам с этого момента предполагать, что буфер TCP-получателя настолько велик, что ограничением окна можно пренебречь; таким образом, количество неподтвержденных данных отправителя ограничено лишь значением cwnd. Кроме того мы будем предполагать, что у отправителя всегда есть данные для пересылки, например, что все сегменты в окне перегрузки уже переданы.

Ограничение объема неподтвержденных данных косвенно влияет на скорость передачи источника. Чтобы понять это, рассмотрим соединения, для которых потери и задержки передачи пакетов незначительны. Тогда приблизительно, в начале каждого RTT, ограничение допускает посылку отправителем cwnd байт данных по соединению; в конце RTT отправитель получает подтверждения на данные. *Таким образом, скорость отправки отправителя приблизительно равна cwnd/RTT байт/с. Регулируя значение cnwd, отправитель может управлять скоростью передачи данных соединения.*

Далее рассмотрим, каким образом TCP-отправитель устанавливает, что присутствует перегрузка на пути между ним и целевым хостом (получателем). Определим «событие потери» для TCP-отправителя, или как истечение интервала ожидания, или как прием трех дублирующих ACK-пакетов от получателя. (В разделе 3.5.4 мы обсуждали истечение интервала ожидания и иллюстрировали это событие на рис. 3.33. Далее мы рассматривали модифицированный вариант с реализацией ускоренной повторной передачи у получателя трех дублирующих ACK-пакетов.) В случае чрезмерной перегрузки один или более маршрутизаторов на всем перегруженном пути вызывают потерю дейтаграммы (содержащей TCP-сегмент). Потерянная дейтаграмма — это результат события потери у отправителя, либо истечение интервала ожидания, либо получение трех дублирующих ACK-пакетов, что воспринимается отправителем, как сигнал перегрузки на пути отправитель-получатель.

Обсудив, как происходит обнаружение ошибки, давайте перейдем к обсуждению более оптимистичной ситуации, когда в сети нет перегруз-

ки, т. е. когда не происходит событие потери. В этом случае подтверждения на предыдущие неподтвержденные сегменты будут получены TCP. Как мы можем увидеть, протокол TCP будет воспринимать поступление этих подтверждений, как сигнал о том, что все хорошо, т. е. сегменты, переданные в сеть, успешно доставлены получателю, и использовать подтверждения для увеличения размера окна перегрузки (и, следовательно, скорости передачи). Отметим: если подтверждение поступило на относительно низкой скорости (например, если конечные хосты пути обладают большой задержкой или используют узкий канал), тогда окно перегрузки будет увеличиваться с такой же малой скоростью. С другой стороны, если подтверждения прибывают с высокой скоростью, тогда окно перегрузки будет расти быстрее. Так как протокол TCP использует подтверждения, как триггер (или часы) для увеличения размера своего окна перегрузки, он называется **автосинхронизируемым**.

При наличии *механизма* настройки `cwnd` для управления скоростью отсылки, остается важный вопрос: *как* TCP-отправитель определит, с какой скоростью необходимо выполнять отправку? Если TCP-отправители корректно и очень быстро отправляют данные, они могут переполнить сеть, что приведет к коллапсу перегрузки, представленному на рис. 3.48. На самом деле версия протокола TCP, которую мы кратко изучили, была разработана в соответствии с наблюдаемым коллапсом перегрузки в сети Интернет[260] на более ранней версии протокола TCP. Но, если TCP-отправители действуют аккуратно и выполняют отправку очень медленно, они могут не полностью использовать ширину канала в сети; то есть, TCP-отправители могли бы быстрее выполнять отправку без перегрузки сети. Как же в таком случае TCP-отправителю определить, с какой скоростью отправлять данные так, чтобы не привести к перегрузке сети, но и использовать всю доступную пропускную способность? Достаточно ли осведомлен TCP-отправитель или же существует подход, используя который TCP-отправители могут регулировать собственную скорость отправки на основе только локальной информации? Протокол TCP решает эти вопросы, используя следующие руководящие принципы:

- *Потеря сегмента означает перегрузку, и, следовательно, скорость TCP-отправителя должна быть снижена при потере сегмента.* Напомним, что в нашей дискуссии в разделе 3.5.4 истечение интервала ожидания или получение четырех квитанций для данного сегмента (один исходный ACK-пакет и три дублирующих ACK-пакета) интерпретируется как явный признак «события потери» сегмента, сле-

дующего за четырежды подтвержденным, после чего срабатывает повторная передача потерянного сегмента. С точки зрения управления перегрузкой, возникает вопрос: как TCP-отправитель должен уменьшить размер своего окна перегрузки, и, следовательно, свою скорость отправки, в ответ на это выявленное событие потери сегмента.

- *Подтвержденный сегмент указывает на то, что сеть доставила сегмент отправителя получателю. И, следовательно, скорость отправителя может быть увеличена, когда поступает ACK-пакет на предыдущий еще не подтвержденный сегмент.* Получение квитанций считается явным указанием того, что все в порядке — сегменты были успешно доставлены от отправителя получателю, и, следовательно, сеть не перегружена. Поэтому размер окна перегрузки может быть увеличен.
- *Зондирование пропускной способности.* На основе подтверждений ACK, указывающих на свободный маршрут от отправителя к получателю, и события потери, указывающие на перегрузку, стратегия TCP по настройке скорости передачи состоит в увеличении скорости передачи после каждого ACK, пока не произойдет событие потери, затем скорость снижается. TCP-отправитель, таким образом, увеличивает скорость передачи, чтобы определить ту, при которой начнется возрастание перегрузки канала, после чего снижает скорость, и затем начинает зондировать вновь, чтобы увидеть, изменится ли перегрузка при возрастании скорости. Поведение TCP-отправителя напоминает поведение ребенка, который просит и получает больше и больше вещей пока, наконец, ему не скажут «Нет!», но отступив назад, он начнет делать запросы вновь вскоре после отказа. Заметьте, что нет явного сообщения о состоянии перегрузки от сети (ACK-пакеты и события потери работают, как сигналы) и что каждый TCP-отправитель действует на основании локальной информации, не синхронизируясь с другими TCP-отправителями.

Приведя этот обзор управления перегрузкой протокола TCP, теперь мы можем рассмотреть детали знаменитого **алгоритма управления перегрузкой протокола TCP**, который впервые был описан в работе Якобсона[260] и стандартизирован в документе RFC 5681[558]. Алгоритм состоит из трех важных этапов: (1) медленный старт, (2) предотвращение перегрузки (также используется термин «устранение затора») и (3) быстрое восстановление. Медленный старт и предотвращение перегрузки — самые значимые компоненты управления перегрузкой протокола TCP. Они различаются между собой тем, как увеличивается размер окна в ответ

на получение ACK-пакета. Вскоре мы увидим, что на этапе медленного старта размер `cwnd` увеличивается быстрее (вопреки своему названию!), чем на этапе предотвращения перегрузки. Этап быстрого восстановления для TCP-отправителей рекомендован, но не обязателен.

Медленный старт

В начале TCP-соединения значение `cwnd` обычно инициализируется малым значением равным 1 MSS[507], и результирующее значение начальной скорости отправки, грубо говоря, равно MSS/RTT. Например, если MSS = 500 байт и RTT = 200 мс, результирующая начальная скорость отправки составляет всего 20 кбит/с. Поскольку доступная ширина канала для TCP-отправителя может быть больше, чем MSS/RTT, то TCP-отправитель заинтересован в быстром подсчете доступной ширины канала. Таким образом, на этапе **медленного старта**, значение `cwnd` начинается с 1 MSS и увеличивается на 1 MSS каждый раз, когда переданный сегмент впервые подтверждается. В примере на рис. 3.51, TCP-отправитель посылает первый сегмент в сеть и ожидает подтверждение.

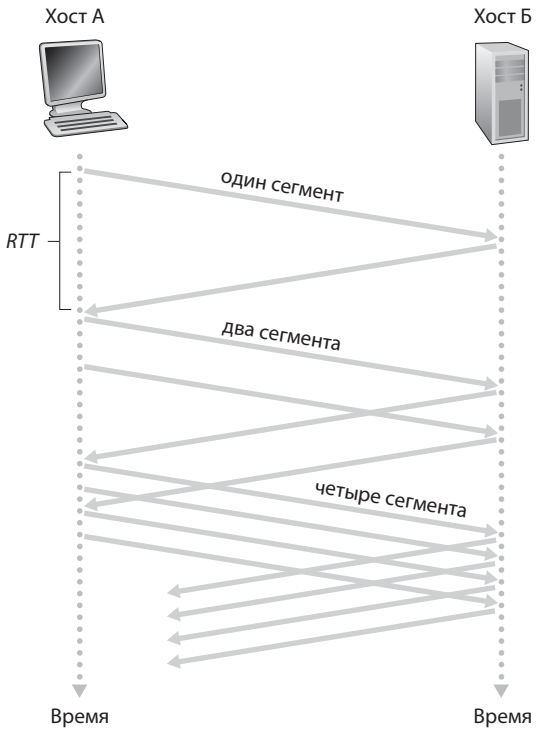


Рис. 3.51. Медленный старт TCP

Когда это подтверждение поступает, TCP-отправитель увеличивает окно перегрузки на один MSS и отправляет два сегмента максимального размера. Далее эти сегменты подтверждаются и отправитель увеличивает окно перегрузки на 1 MSS после каждого подтверждения. Окно перегрузки достигает размера 4 MSS и так далее. Результат этого процесса состоит в удвоении скорости отправки каждый RTT. Таким образом, TCP-соединение начинается с низкой скорости отправки данных, но она увеличивается экспоненциально на протяжении этапа медленного старта.

Но когда должен прекратиться этот экспоненциальный рост? Медленный старт предполагает несколько ответов на этот вопрос. Во-первых, если произошло событие потери (например, перегрузка), на что указывает истечение интервала ожидания, TCP-отправитель устанавливает значение `cwnd` равным 1 и начинает этап медленного старта сначала. Он также устанавливает значение второй переменной состояния `ssthresh` (краткое от “slow start threshold” порог медленного старта) равным `cwnd/2` — половине от значения окна перегрузки в момент, когда перегрузка была обнаружена. Вторая ситуация, при которой может прекратиться этап медленного старта, непосредственно связана со значением `ssthresh`.

Поскольку `ssthresh` — это половина значения `cwnd`, когда перегрузка в последний раз была обнаружена, то может быть немного безрассудно продолжать удваивать `cwnd`, когда оно достигает значения `ssthresh` или превосходит его. Таким образом, когда значение `cwnd` равно `ssthresh`, этап медленного старта прекращается, и протокол TCP переходит в режим предотвращения перегрузки. Как мы увидим, протокол TCP увеличивает `cwnd` более осторожно, находясь в режиме предотвращения перегрузки.

Последний вариант, при котором этап медленного старта может прекратиться — это получение трех дублирующих ACK-пакетов, в этом случае протокол TCP осуществляет ускоренную повторную передачу (см. раздел 3.5.4) и переходит в состояние быстрого восстановления, как обсуждается далее.

Поведение протокола TCP на этапе медленного старта объединено в FSM-схеме управления перегрузкой протокола TCP на рис. 3.52. Алгоритм медленного старта построен на основе работы Якобсона[260]; подход схожий с медленным стартом был независимо разработан и в работе Джейна[261].

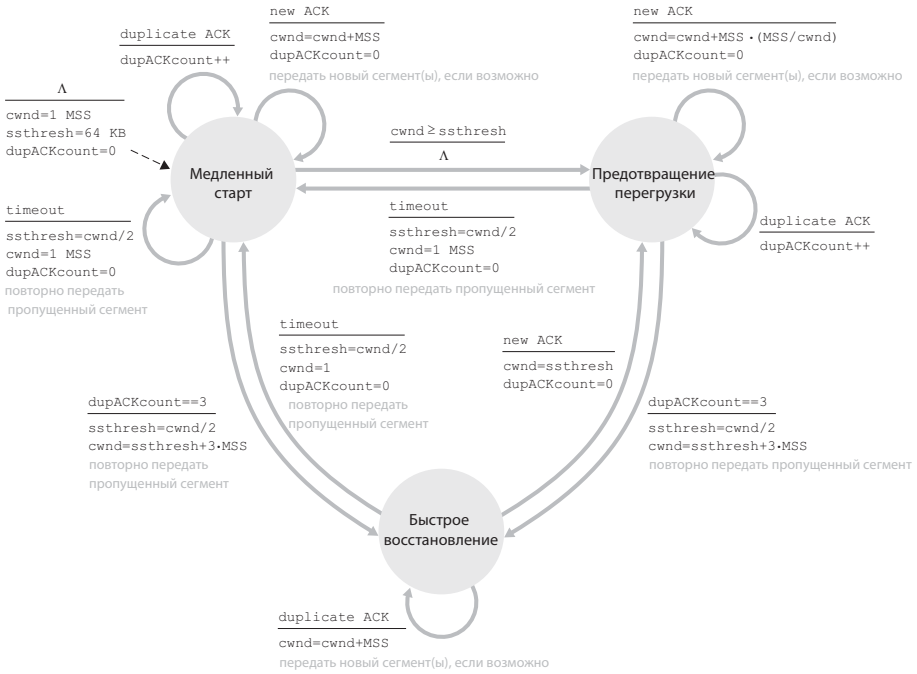


Рис. 3.52. FSM-схема управления перегрузкой протокола TCP

ПРИНЦИПЫ В ДЕЙСТВИИ

Расслоение TCP: оптимизация производительности облачных сервисов

Для облачных сервисов, таких как поисковые системы, электронная почта и социальные сети, желательно обеспечить малое время отклика, в идеале создавая у пользователя впечатление, что сервис выполняется на его собственной машине (включая их смартфоны). Это может быть главной проблемой, поскольку пользователи часто расположены далеко от дата-центров, которые отвечают за обслуживание динамического контента связанного с облачными сервисами. На самом деле, если конечная система расположена далеко от дата-центра, то значение RTT будет большим, что может привести к неудовлетворительному времени отклика, связанному с этапом медленного старта протокола TCP.

Рассмотрим задержку получения ответа на поисковый запрос. Обычно серверу требуется три окна TCP на этапе медленного старта для доставки ответа[384]. Таким образом, время с момента, когда конечная система инициировала TCP-соединение, до времени, когда она получила последний пакет в ответ, составляет примерно 4 RTT (один

RTT для установления TCP-соединения, плюс три RTT для трех окон данных) плюс время обработки в дата-центре. Такие RTT задержки могут привести к заметно замедленной выдаче результатов поиска для многих запросов. Более того, могут присутствовать также и значительные потери пакетов в сетях доступа, приводящие к повторной передаче TCP и еще большим задержкам.

Один из способов смягчения этой проблемы и улучшения восприятия пользователем производительности заключается в том, чтобы (1) развернуть внешние серверы ближе к пользователям и (2) использовать расслоение TCP путем **разделения TCP-соединения** на внешнем сервере. При расслоении TCP клиент устанавливает TCP-соединение с ближайшим внешним сервером, который поддерживает постоянное TCP-соединение с дата-центром с очень большим окном перегрузки TCP[384, 631, 91]. При использовании такого подхода, время отклика примерно равно $4 \times RTT_{FE} + RTT_{BE}$ + время обработки, где RTT_{FE} — время оборота между клиентом и внешним сервером, и RTT_{BE} — время оборота между внешним сервером и дата-центром (внутренним сервером). Если внешний сервер закрыт для клиента, то это время ответа приближается к RTT плюс время обработки, поскольку значение RTT_{FE} ничтожно мало и значение RTT_{BE} приблизительно равно RTT. В итоге, расслоение TCP может уменьшить сетевую задержку, грубо говоря, с $4 \times$ RTT до RTT, значительно повышая субъективную производительность, особенно для пользователей, которые расположены далеко от ближайшего дата-центра. Расслоение TCP также помогает сократить задержку повторной передачи TCP, вызванную потерями в сетях. Сегодня Google и Akamai широко используют свои сервисы CDN в сетях доступа (см. раздел 7.2), чтобы обеспечить расслоение TCP для облачных сервисов, которые они поддерживают[91].

Предотвращение перегрузки

При переходе в состояние предотвращения перегрузки значение `cwnd` приблизительно составляет половину своего значения при перегрузке, обнаруженной в последний раз — перегрузка может быть прямо за углом! Таким образом, вместо удвоения значения `cwnd` каждые `RTT` протокол TCP использует более консервативный подход и увеличивает значение `cwnd` только на один `MSS` каждые `RTT`[558]. Это может быть выполнено несколькими способами. Обычный подход: TCP-отправитель увеличивает значение cwnd на `MSS` байт (`MSS`/`cwnd`) каждый раз, когда приходит подтверждение. Например, если `MSS` равно 1460 байт и `cwnd` равно 14 600 байт, тогда 10 сегментов

будут отправлены за один RTT. Каждый принятый ACK-пакет (предположим, что на сегмент данных приходит один сегмент ACK) увеличивает размер окна перегрузки на 1/10 MSS, и тогда значение окна перегрузки будет увеличено на один MSS после получения всех ACK-пакетов на все 10 сегментов данных.

Но когда предотвращение перегрузки должно привести к линейному увеличению (на 1 MSS в RTT)? Алгоритм предотвращения перегрузки протокола TCP действует аналогично алгоритму медленного старта при наступлении таймаута: значение cwnd устанавливается равным 1 MSS, и значение ssthresh обновляется на половину значения cwnd в случае потери сегмента. Напомним, однако, что событие потери также может быть определено получением трех дублирующих ACK-пакетов. В этом случае сеть подсчитывает доставленные от отправителя получателю сегменты (на что указывает получение дублирующих ACK-пакетов). Поведение протокола TCP для такого типа потерь должно быть менее радикальным, чем при потерях, на которые указывает таймаут. Протокол TCP устанавливает значение cwnd равным половине предыдущего значения (добавив в 3 MSS для лучшего подсчета полученных дублирующих ACK-пакетов) и устанавливает значение ssthresh равным половине значения cwnd при получении трех дублирующих ACK-пакетов. Далее выполняется переход в состояние быстрого восстановления.

Быстрое восстановление

На этапе быстрого восстановления значение cwnd увеличивается на 1 MSS для каждого полученного дублирующего ACK-пакета потерянного сегмента, вызвавшего переход TCP-соединения в состояние быстрого восстановления. В конце концов, когда приходит ACK-пакет для пропущенного сегмента, TCP-соединение переходит в состояние предотвращения перегрузки после уменьшения значения cwnd. Если истек интервал ожидания, быстрое восстановление переходит в состояние медленного старта после выполнения таких же действий, как при медленном старте и предотвращении перегрузки: значение cwnd устанавливается равным 1 MSS, а значение ssthresh — равным половине значения cwnd в случае события потери.

Этап быстрого восстановления — рекомендованный, но не обязательный компонент протокола TCP[558]. Интересно, что ранняя версия протокола TCP, известная как **TCP Tahoe**, безусловно сокращала раз-

мер окна перегрузки на 1 MSS и переходила на этап медленного старта после события потери, на которое указывало и получение трех дубликатов ACK, и наступление таймаута. Новейшая версия протокола TCP, **TCP Reno**, поддерживает этап быстрого восстановления.

На рис. 3.53 показана эволюция окна перегрузки протокола TCP для обеих версий Reno и Tahoe. На этом рисунке, порог окна в начале равен 8 MSS. Для первых восьми этапов передач версии протокола Tahoe и Reno предпринимают одинаковые действия. Окно перегрузки экспоненциально увеличивается на этапе медленного старта и достигает порога на четвертом цикле передачи. Затем окно передачи возрастает линейно до наступления события получения трех дублирующих ACK-пакетов сразу после 8 цикла передачи.

Заметим, что окно перегрузки равно $12 \times MSS$ в момент наступления события потери. Значение ssthresh далее устанавливается равным $0.5 \times cwnd = 6 \times MSS$. В версии протокола TCP Reno, окно перегрузки устанавливается равным $cwnd = 9 \times MSS$ и затем возрастает линейно. В версии протокола TCP Tahoe окно перегрузки устанавливается равным 1 MSS и возрастает экспоненциально, пока не достигнет значения ssthresh, после чего оно возрастает линейно.

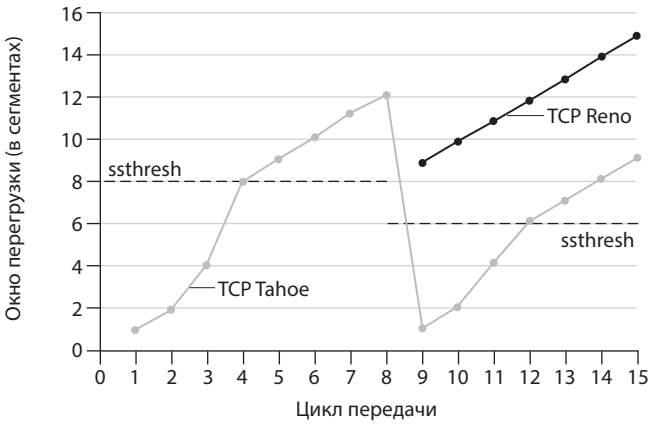


Рис. 3.53. Эволюция окна перегрузки TCP (Tahoe и Reno)

На рис. 3.52 представлена полная FSM-схема алгоритма управления перегрузкой TCP: медленный старт, предотвращение перегрузки и быстрое восстановление. Рисунок также показывает, где передаются новые сегменты или может произойти повторная передача сегментов. Хотя это важное отличие между управлением ошибками/повторной

передачей протокола TCP и управлением перегрузкой протокола TCP, также важно понимать, что эти два аспекта неразрывно связаны между собой.

Управление перегрузкой в протоколе TCP: ретроспектива

После детального изучения этапов медленного старта, предотвращения перегрузки и быстрого восстановления было бы неплохо обернуться назад и рассмотреть общую картину. Не считая начального этапа медленного старта, когда начинается соединение, и предполагая, что о потерях свидетельствует получение трех дублирующих ACK-пакетов, а не истечение интервала ожидания, управление перегрузкой протокола TCP можно разделить на два этапа: линейное (аддитивное) увеличение значения `cwnd` на 1 `MSS` в `RTT` и последующее уменьшение вдвое (мультипликативное уменьшение) значения `cwnd` в случае получения трех дублирующих ACK-пакетов. По этой причине управление перегрузкой протокола TCP часто называют формой управления перегрузкой с **аддитивным ускорением и мультипликативным замедлением** (additive-increase, multiplicative-decrease, **AIMD**). Алгоритм управления перегрузкой AIMD приводит к пилообразному поведению, как показано на рис. 3.54, который также хорошо отражает наше наглядное представление о «зондировании» ширины канала TCP: протокол TCP линейно увеличивает размер своего окна перегрузки (и, следовательно, скорость передачи), пока не происходит получение трех дублирующих ACK-пакетов. После этого протокол уменьшает размер окна перегрузки вдвое, но затем вновь начинает его линейное увеличение, выполняя зондирование, чтобы понять, есть ли еще дополнительная свободная ширина канала.

Как было указано ранее, многие реализации протокола TCP используют алгоритм Reno[379]. Множество вариантов алгоритма Reno было предложено в документах RFC 3782[524] и RFC 2018[463]. Алгоритм протокола TCP Vegas[63, 13] призван предотвратить перегрузку, при сохранении хорошей пропускной способности. Основная идея алгоритма Vegas: (1) обнаружить перегрузку на маршрутизаторе между отправителем и получателем *прежде*, чем произойдет потеря пакета и (2) снижать скорость линейно, если обнаружится угроза потери пакета. Угроза потери пакета предсказывается наблюдением за значением `RTT`. Чем длиннее `RTT` пакетов, тем выше загруженность маршрутизатора. Операционная система Linux поддерживает несколько алгоритмов управления перегрузкой (включая TCP Reno и TCP Vegas), что позволяет администраторам вы-

числительной сети при настройке выбирать оптимальную версию протокола TCP. В качестве версии протокола TCP по умолчанию в версии Linux 2.6.18 была установлена CUBIC[197]. Эта версия протокола TCP разработана для приложений с высокой пропускной способностью. Недавний обзор многочисленных версий протокола TCP приведен в работе Афанасьева[11].

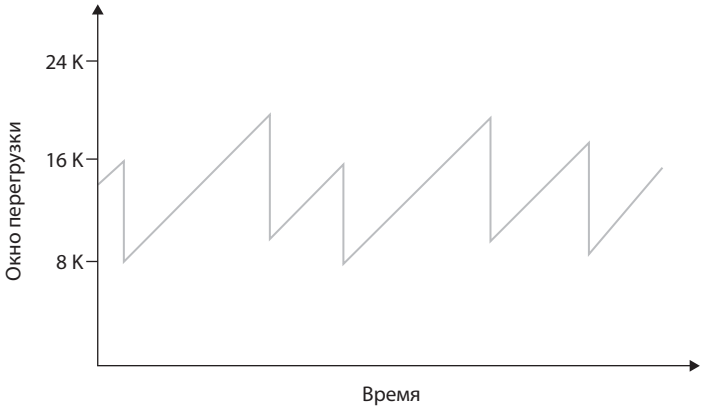


Рис. 3.54. Управление перегрузкой с аддитивным ускорением и мультипликативным замедлением

TCP алгоритм AIMD был разработан на основе огромного количества инженерной проницательности и экспериментов с управлением перегрузкой в действующих сетях. Спустя десять лет теоретические анализы показали, что алгоритм управления перегрузкой протокола TCP работает как распределенный алгоритм асинхронной оптимизации, что приводит сразу к нескольким важным аспектам оптимизации производительности как пользователя, так и сети в целом[281]. С тех пор была наработана обширная теория по управлению перегрузкой[611].

Макроскопическое описание пропускной способности TCP-соединения

Рассмотрим пилообразное поведение протокола TCP, чтобы выяснить, какой может быть средняя пропускная способность (то есть, средняя скорость) TCP-соединения на протяжении длительного времени. В этом анализе мы не станем учитывать этап медленного старта, который наступает после истечения интервала ожидания. (Этот этап обычно очень короткий, по причине экспоненциального роста ско-

рости.) В течение одного времени оборота скорость, с которой протокол TCP отправляет данные, выражается функцией, зависящей от размера окна перегрузки и текущего значения *RTT*. Когда размер окна равен *w* байт и текущее время оборота равно *RTT* секунд, тогда скорость передачи протокола TCP приблизительно равна *W*/*RTT*. Затем протокол TCP зондирует возможную дополнительную ширину канала, увеличивая значение *w* на 1 MSS за каждое *RTT*, пока не наступит событие потери. Обозначим через *W* значение *w* в момент наступления события потери. Предположим, что значения *RTT* и *W* приблизительно постоянны в течение всего времени существования соединения, скорость передачи протокола TCP меняется в пределах от *W*/(2×*RTT*) до *W*/*RTT*.

Эти предположения ведут к крайне упрощенной макроскопической модели стационарного поведения протокола TCP. Сеть отклоняет пакет соединения, когда скорость возрастает до *W*/*RTT*; скорость в таком случае уменьшается вдвое, а затем увеличивается на MSS/*RTT* каждые *RTT*, пока вновь не достигнет значения *W*/*RTT*. Этот процесс повторяется снова и снова. Поскольку пропускная способность TCP-соединения (то есть, скорость) возрастает линейно между двумя значениями экстремума, мы получаем:

$$\text{Средняя скорость соединения} = \frac{0{,}75 \times W}{RTT}$$

Используя эту идеализированную модель стационарной динамики TCP-соединения, мы можем также получить интересное выражение, которое отражает отношение уровня потерь соединения к его доступной пропускной способности[331]. Вывод этого выражения отведен на домашние задания.

Более сложная модель, которая была получена эмпирическим путем по измеренным данным, приведена в работе Падхи[378].

TCP-соединение по каналу с высокой пропускной способностью

Важно понимать, что управление перегрузкой протокола TCP развивалось в течение многих лет и продолжает развиваться до сих пор. Обзор текущих вариантов протокола TCP и обсуждение развития протокола TCP изложены в работах Флойда[164], Афанасьева[11] и документе RFC 5681[558]. И то, что было хорошо, когда основная масса TCP-соединений передавала лишь SMTP-, FTP-, и Telnet-трафик, не обязательно подходит для современного Интернета, базирующегося на протоколе HTTP,

или для будущего Интернета с сервисами, которые пока еще невозможно представить.

Потребность в дальнейшем развитии протокола TCP может быть продемонстрирована на высокоскоростных соединениях, которые необходимы для приложений сетевых и облачных вычислений. Например, рассмотрим TCP-соединение использующее 1500-байтовые сегменты и имеющее *RTT равное 100 мс*, и допустим, мы хотим отправить данные по этому соединению на скорости 10 Гбит/с.

Придерживаясь документа RFC 3649[522], мы отметим, что если используется формула пропускной способности TCP-соединения, приведенная выше, для достижения пропускной способности 10 Гбит/с средний размер окна перегрузки должен составлять 83 333 сегмента. Такое большое количество заставляет подумать о том, что один из этих 83 333 передаваемых сегментов будет потерян. Что произойдет в случае потери? Или предположим другой вариант: какая часть передаваемых сегментов может быть потеряна, чтобы при этом алгоритм управления перегрузкой протокола TCP, определенный на рис. 3.52, продолжил поддерживать скорость 10 Гбит/с? В контрольных вопросах к этой главе вам предложено вывести формулу зависимости пропускной способности TCP-соединения, как функции зависящей от частоты потерь (L), времени оборота (RTT), и максимального размера сегмента (MSS):

$$\text{Средняя пропускная способность соединения} = \frac{1{,}22 \times MSS}{RTT\sqrt{L}}$$

Используя эту формулу, мы можем увидеть, что для достижения пропускной способности 10 Гбит/с с современным алгоритмом управления перегрузкой протокола TCP допустима вероятность потери сегментов не более чем 2×10^{-10} (или эквивалентно, одно событие потери на каждые 5 000 000 000 сегментов), что является очень низкой величиной потерь. Это сподвигло исследователей к созданию новой версии протокола TCP, которая специально рассчитана на такие высокоскоростные среды; подробнее см. в работах Джина[269], Келли[282], Ха[197] и документе RFC 3649[522].

3.7.1. Выравнивание скоростей передачи

Рассмотрим K TCP-соединений, каждое из которых имеет отдельный путь между конечными точками сети, но все они выполняют пере-

дачу по узкому каналу со скоростью передачи *R* бит/с. Под *узким каналом* мы подразумеваем, что средняя пропускная способность соединения равна 1,22×MSS/RTT×2L для каждого соединения, при этом остальные каналы на протяжении всего пути соединения не перегружены и имеют достаточную пропускную способность по сравнению с узким каналом. Предположим, каждое соединение передает огромный файл и в узком канале нет UDP-трафика. Говорят, что механизм контроля перегрузки обеспечивает *выравнивание скоростей,* если средняя скорость передачи каждого соединения приблизительно равна *R/K;* то есть каждое соединение получает равную долю пропускной способности общего канала.

Так как TCP-соединения могут устанавливаться в разные моменты времени и тем самым иметь неравные размеры окон перегрузки, возникает вопрос: обеспечивает ли выравнивание скоростей алгоритм аддитивного увеличения и мультипликативного уменьшения протокола TCP? В работе Чиу[94] представлено красивое и интуитивно понятное объяснение того, почему управление перегрузкой протокола TCP сходится к предоставлению равной доли пропускной способности общего узкого канала между конкурирующими TCP-соединениями.

Рассмотрим простейший случай, когда два TCP-соединения используют общую линию связи с пропускной способностью *R*, как показано на рис. 3.55. Допустим, что два соединения имеют одинаковые значения MSS и RTT (так что, если у них одинаковый размер окна перегрузки, то и пропускная способность будет одинаковой), что они имеют большое количество данных для отправки, и что по этому общему каналу больше не проходит других TCP-соединений или UDP-дейтаграмм. Также будем игнорировать этап медленного старта TCP-соединения и предполагать, что TCP-соединение работает в режиме предотвращения перегрузки (AIMD) все время.

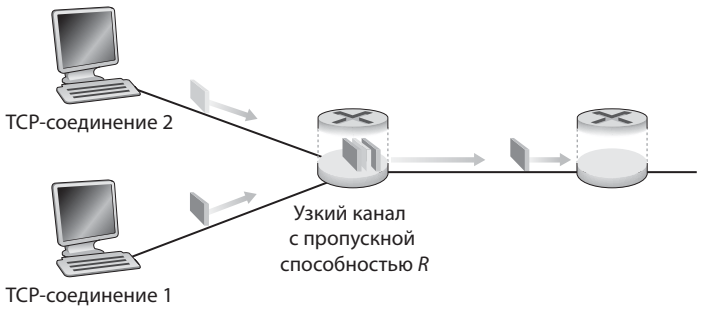


Рис. 3.55. Два TCP-соединения, использующие общий узкий канал

На рис. 3.56 изображен график пропускной способности двух TCP-соединений. Если TCP-соединение равномерно распределяет использование общего канала, то график пропускной способности должен опускаться под углом 45 градусов (равномерное распределение пропускной способности) относительно первоначального. В идеале сумма пропускных способностей двух соединений должна быть равной *R*. (разумеется, нет смысла выравнивать скорости передачи для каждого соединения, если в результате они окажутся нулевыми!) Таким образом, цель алгоритма контроля перегрузки — добиться, чтобы на графике пропускная способность каждого из соединений оказалась как можно ближе к точке пересечения линии равных скоростей с линией максимального использования ресурсов, как на рис. 3.56.

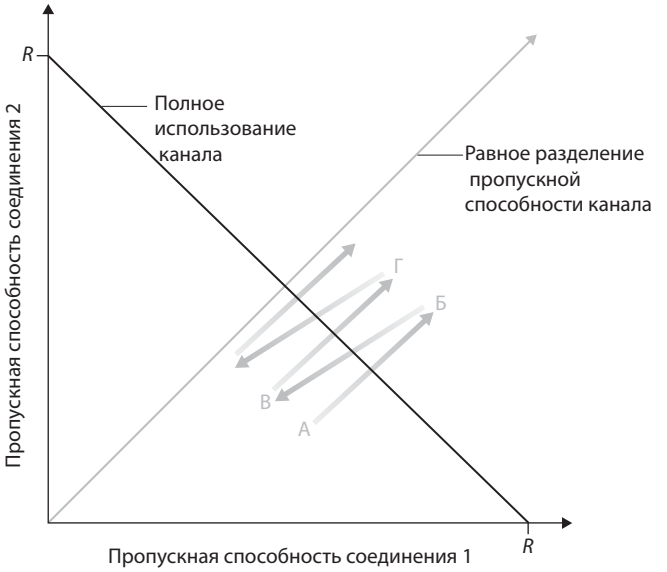


Рис. 3.56. Пропускная способность двух TCP-соединений 1 и 2

Допустим, что размер окна TCP-соединения такой, что в заданный момент времени, соединения 1 и 2 получают пропускную способность, обозначенную точкой *A* на рис. 3.56. Поскольку суммарная скорость передачи двух соединений меньше, чем *R*, то не происходит потерь, и оба соединения увеличивают размер окна на 1 MSS в RTT, что обусловлено работой TCP-алгоритма предотвращения перегрузок. Таким образом, совместная пропускная способность двух соединений изменяется вдоль линии, наклоненной под углом в 45 градусов (равномерное увеличение для каждого соединения) и начинающейся в точке *A*. Со временем совместно

используемая двумя соединениями пропускная способность канала превысит значение *R*, что в конце концов приведет к потерям пакетов. Допустим, что соединения 1 и 2 испытывают потери пакетов, когда используемая ими пропускная способность достигает значения, указанного на графике точкой *Б*. Затем соединения 1 и 2 уменьшают размер окон вдвое. Полученная в результате пропускная способность отражена на графике точкой *В*, в середине вектора начинающегося в точке *Б* и заканчивающегося в оригинале. Поскольку совместно используемая пропускная способность канала становится меньше, чем *R* в точке *В,* оба соединения вновь увеличивают свою пропускную способность вдоль линии, наклоненной под углом 45 градусов и начинающейся в точке *В*. В конечном итоге вновь появятся потери, например, в точке *Г,* и оба соединения вновь вдвое уменьшат размер собственных окон, и так далее. Вы должны самостоятельно убедиться в том, что пропускная способность, используемая двумя соединениями, действительно отклоняется от равномерного использования пропускной способности канала. Кроме того, вы должны самостоятельно убедиться в том, что два соединения будут сходиться к этому поведению независимо от того, в какой точке графика они находятся сейчас! Хотя количество идеализированных допущений выходит за рамки сценария, все еще интуитивно понятно, почему протокол TCP стремится к равномерному использованию пропускной способности соединениями.

Мы предположили, что проблемная линия связи не задействована другими соединениями, кроме TCP, время оборота для всех соединений одинаково и между каждой парой хостов установлено единственное TCP-соединение. На практике подобные допущения не выполняются, и приложения клиент/сервер зачастую используют неравные доли пропускной способности линии связи. В частности, сеансы с меньшими значениями времени оборота способны захватывать большую часть свободных ресурсов, так как быстрее наращивают размеры окна перегрузки по сравнению с другими соединениями[308].

Выравнивание скоростей и протокол UDP

Мы только что увидели, каким образом управление перегрузкой протокола TCP регулирует скорость передачи данных приложений совместно с механизмом окна перегрузки. Многие мультимедийные приложения, такие, как IP-телефония и видеоконференции, зачастую не используют протокол TCP именно по этой причине: они не хотят замедления скорости передачи, даже если сеть перегружена. Напротив, эти приложения используют протокол UDP, в котором нет встроенного механизма управ-

ления перегрузкой. При использовании протокола UDP приложения могут передавать свои аудио- и видеоданные в сеть с постоянной скоростью и возможными потерями пакетов, а не уменьшать собственную скорость для «справедливости» во время перегрузки и не терять пакеты. С точки зрения протокола TCP, мультимедийные приложения, работающие по протоколу UDP, действуют несправедливо: они не координируются с другими соединениями и не регулируют соответствующим образом свою скорость передачи. Поскольку управление перегрузкой протокола TCP снизит эту скорость, если возникнет перегрузка при появлении потерь, что не нужно UDP-отправителям, они могут вытеснять TCP-трафик. Одной из главных задач, стоящих перед современными исследователями в области Интернет-технологий, является создание механизмов контроля перегрузки, ограждающих пропускную способность Интернета от «губительного» воздействия UDP-соединений[162, 163, 292].

Выравнивание скоростей и параллельные TCP-соединения

Даже если механизм, заставляющий UDP-соединения выравнивать пропускные способности линий связи, будет создан, он не решит существующую проблему до конца. Это объясняется тем, что приложениям невозможно запретить использовать параллельные TCP-соединения. К примеру, веб-браузеры часто используют их для передачи объектов веб-страниц. В большинстве браузеров можно настроить количество таких соединений. Когда приложение использует множественные параллельные соединения, оно получает большую долю пропускной способности перегруженного канала. Для примера рассмотрим канал со скоростью передачи данных R, поддерживающий 9 одновременно действующих клиент-серверных приложений, каждое из которых использует единственное TCP-соединение. Если появляется новое приложение, также использующее TCP-соединение, то каждое приложение получит скорость передачи около $R/10$. Но если новое приложение использует 11 параллельных TCP-соединений, то оно вполне справедливо получит более половины общей скорости. Из-за распространенности веб-трафика в Интернете параллельные соединения не редки.

3.8. Заключение

Мы начали эту главу с изучения сервисов, которые протокол транспортного уровня может предоставить сетевым приложениям. В предельном случае, протокол транспортного уровня может быть очень простым

и предлагать службы без излишеств, предоставляя приложениям только функцию мультиплексирования/демультиплексирования для процесса взаимодействия. Интернет-протокол UDP — пример подобного протокола транспортного уровня без излишеств. Как другая крайность, протокол транспортного уровня может предоставлять разнообразные гарантии приложениям, такие как надежная передача данных, гарантированные время отклика и пропускная способность. В любом случае, сервисы транспортного уровня часто ограничены рамками модели обслуживания протоколов более низкого сетевого уровня. Если протокол сетевого уровня не может предоставить гарантии задержки или пропускной способности для сегментов транспортного уровня, то и протокол транспортного уровня не может предоставить таких гарантий для сообщений, пересылаемых между процессами.

В разделе 3.4 мы узнали, что протокол транспортного уровня может предоставлять службу надежной передачи данных, даже если нижерасположенный сетевой уровень ненадежный. Мы увидели, что предоставление службы надежной передачи данных имеет множество нюансов, но задача вполне может быть выполнена совокупностью механизмов подтверждений, таймеров, повторной передачи и порядковых номеров.

Хотя в этой главе мы рассмотрели надежную передачу данных, мы должны помнить, что она может быть предоставлена протоколами канального, сетевого, транспортного или прикладного уровней. Любой из четырех верхних уровней стека протоколов способен реализовать механизмы подтверждений, таймеров, повторных передач и порядковых номеров и предоставлять службу надежной передачи данных для вышестоящего уровня. На самом деле на протяжении многих лет инженеры и ученые использовали отдельно созданные и реализованные протоколы канального, сетевого, транспортного уровней и уровня приложений, которые предоставляют службу надежной передачи данных (хотя многие из этих протоколов бесследно исчезли).

В разделе 3.5 мы подробно рассмотрели протокол TCP — надежный протокол транспортного уровня, ориентированный на интернет-соединения. Мы узнали, что TCP — это сложный протокол, включающий механизмы управления соединением, управления потоком и интервалы времени ожидания, а также службу надежной передачи данных. Фактически протокол TCP сложнее, чем в нашем описании — мы намеренно не обсуждали различные TCP-патчи, исправления и улучшения, которые реализованы в различных версиях TCP. Вся эта сложность, однако, скрыта от сетевых приложений. Если клиент на одном хосте хочет

надежно передать данные серверу на другом хосте, он просто открывает TCP-сокет для сервера и закачивает данные в этот сокет. Клиент-серверное приложение пребывает в блаженном неведении о сложности протокола TCP.

В разделе 3.6 мы изучили управление перегрузкой в целом, а в разделе 3.7 показали, как реализовано управление перегрузкой в протоколе TCP. Мы узнали, что управление перегрузкой важно для благополучия сети. Без управления перегрузкой в сети запросто могут появиться заторы, из-за которых лишь малая часть данных будет передаваться между конечными точками соединения, или вовсе никаких. В разделе 3.7 мы узнали, что протокол TCP реализует механизм управления перегрузкой на конечных точках, который аддитивно увеличивает скорость передачи, когда путь TCP-соединения оценен, как не перегруженный, и мультипликативно уменьшает скорость передачи, при обнаружении потерь. Этот механизм также стремится предоставить каждому из TCP-соединений, проходящих через загруженный канал, равную долю пропускной способности канала. Мы также несколько глубже изучили воздействие установления TCP-соединения и медленного старта на задержку. Мы заметили, что во многих важных сценариях установление соединения и медленный старт значительно влияют на задержку между конечными точками соединения. Еще раз подчеркнем, что поскольку управление перегрузкой протокола TCP развивалось на протяжении многих лет, оно остается областью интенсивных исследований и, скорее всего, будет дорабатываться в ближайшие годы.

Наше обсуждение предназначенных для Интернета транспортных протоколов в этой главе было сфокусировано на протоколах UDP и TCP — двух «рабочих лошадках» транспортного уровня сети Интернет. Однако двадцать лет работы с этой парой протоколов выявили обстоятельства, при которых ни один из них не подходит идеально, что побудило исследователей заняться разработкой дополнительных протоколов транспортного уровня — ряд из них теперь предложены в стандарте IETF.

Datagram Congestion Control Protocol (DCCP)[538] предоставляет UDP-подобный ориентированный на сообщения ненадежный сервис с малыми накладными расходами, но с выбираемыми приложением формами управления перегрузкой, сравнимыми с TCP. Если приложению потребуется ненадежная или относительно надежная передача данных, то она должна обеспечиваться самим приложением, возможно, с приме-

нением механизма, который мы изучали в разделе 3.4. Протокол DCCP предлагается для использования в таких приложениях, как потоковая передача мультимедийных данных (см. главу 7), которые могут найти компромисс между затратами времени и надежной доставкой данных, но хотят быть независимыми от загруженности сети.

Протокол передачи и управления потоком SCTP (Stream Control Transmission Protocol)[547, 504] — надежный протокол, ориентированный на обмен сообщениями, который позволяет нескольким различным потокам прикладного уровня быть мультиплексированными в единое SCTP-соединение (так называемый «многопоточный» подход). С точки зрения надежности, разные потоки используют раздельно обрабатываемые соединения, так что потеря пакета в одном потоке не влияет на доставку данных в другом потоке. Протокол SCTP также позволяет передавать данные по двум исходящим путям, когда хост соединен с двумя или более сетями, с дополнительной возможностью доставки следующих не по порядку данных и с набором других возможностей. Алгоритмы управления потоком и перегрузкой протокола SCTP по существу такие же, как в протоколе TCP.

TCP-Friendly Rate Control (TFRC)[556] — это протокол управления перегрузкой, а не полнофункциональный протокол транспортного уровня. В нем определен механизм управления перегрузкой, который может использоваться в других протоколах транспортного уровня, таких как DCCP (на самом деле один из двух доступных для выбора приложением протоколов в DCCP — это TFRC). Цель протокола TFRC — сгладить пилообразное поведение (см. рис. 3.54) управления перегрузкой протокола TCP, при поддержании на длительном временном интервале скорости, которая бы была «разумно» близка к TCP. Обладая более равномерной скоростью отправки, чем в протоколе TCP, протокол TFRC отлично подходит для мультимедийных приложений таких, как IP-телефония или потоковое вещание, для которых важна равномерная скорость. В основе протокола TFRC лежит уравнение, параметром которого является измеренная частота потерь пакетов[378], — оно определяет, какой должна быть пропускная способность TCP-соединения, если в TCP-сеансе происходят потери данных. Далее вычисленное значение используется протоколом TFRC для определения скорости отправки.

Только будущее определит, получат ли широкое распространение протоколы DCCP, SCTP или TFRC. Пока очевидно, что они предоставляют расширенные возможности по сравнению с протоколами TCP

и UDP, но протоколы TCP и UDP зарекомендовали себя «достаточно хорошо» на протяжении многих лет. Победит ли «лучшее» или «достаточно хорошее», будет зависеть от сложной совокупности технических, социальных и коммерческих соображений.

В главе 1 мы рассказали, что компьютерная сеть может быть разделена на «сетевую периферию» и «сетевое ядро». Сетевая периферия включает все, что происходит на конечных системах. Сейчас мы завершаем наше обсуждение сетевой периферии, полностью рассмотрев прикладной и транспортный уровни. Время исследовать ядро сети! Это путешествие начинается со следующей главы, в которой мы будем изучать сетевой уровень, и продолжается в главе 5, где мы рассмотрим канальный уровень.

Глава 4
СЕТЕВОЙ УРОВЕНЬ

В предыдущей главе говорилось о том, что транспортный уровень обеспечивает передачу данных от отправителя к получателю, благодаря межхостовой коммуникационной службе сетевого уровня. Также выяснилось, что на транспортный уровень не поступает информация о том, как именно реализуется данная функция. Вероятно, вам уже не терпится узнать, каковы внутренние механизмы межхостовой коммуникации, словом, как она действует.

В этой главе вы узнаете, как именно сетевой уровень реализует механизм межхостовой коммуникации. Вы убедитесь, что в отличие от транспортного и прикладного уровней, сетевой затрагивает все хост-системы и маршрутизаторы в сети. Поэтому протокол сетевого уровня является самым сложным и интересным в стеке протоколов.

Сетевой уровень также является одним из сложнейших во всем стеке, поэтому здесь нам предстоит изучить массу фундаментального материала. Изучение начнется с краткого обзора сетевого уровня и предоставляемых на нем служб. Кроме того, из этой главы вы узнаете о двух основных способах структурирования доставки пакетов — работе с дейтаграммами и моделью виртуального канала — и убедитесь в том, насколько важную роль играет адресация при доставке пакетов системе-получателю.

В данной главе будет установлено существенное различие между такими функциями сетевого уровня, как **перенаправление** и **маршрутизация**. Перенаправление заключается в передаче пакета между входами и выходами *одного* маршрутизатора. В процесс же маршрутизации вовлечены *все* маршрутизаторы в сети, чьи коллективные взаимодействия через протоколы маршрутизации определяют пути, по которым пакеты достигают места назначения. Эту разницу важно понимать при изучении данной главы.

Чтобы приобрести глубокие знания о перенаправлении пакетов, вы заглянете «внутрь» маршрутизатора, узнаете о его аппаратной ар-

хитектуре и организации. Затем увидите, как происходит перенаправление пакетов в Интернете, на примере небезызвестного протокола IP (Internet Protocol). Вы изучите адресацию сетевого уровня и познакомитесь с форматом дейтаграмм IPv4, затем перейдете к изучению преобразования сетевых адресов (NAT), фрагментации дейтаграмм, протокола ICMP и IPv6.

Также внимание будет уделено такой функции сетевого уровня, как маршрутизация. Вы узнаете, что задача алгоритма маршрутизации заключается в определении оптимальных путей (маршрутов) доставки пакетов от отправителей к получателям. Сначала мы изучим теорию алгоритмов маршрутизации, подробно остановившись на двух наиболее распространенных классах алгоритмов: дистанционно-векторных и зависящих от состояния канала. Поскольку алгоритмы маршрутизации существенно усложняются по мере роста количества сетевых маршрутизаторов, все больший интерес привлекают системы иерархической маршрутизации. Кроме того, вы сможете перейти от теории к практике, когда изучите протоколы маршрутизации внутридоменных (RIP, OSPF и IS-IS) и междоменных систем Интернета (BGP). В конце главы будет рассказано о широковещательной и групповой маршрутизации.

Таким образом, данная глава делится на три основных части. Первая часть (разделы 4.1 и 4.2) раскрывает функции сетевого уровня. Во второй части (разделы 4.3 и 4.4) говорится о перенаправлении пакетов. Наконец, основная тема третьей части (разделы 4.5 и 4.7) — маршрутизация.

4.1. Введение

На рис. 4.1 схематически изображена простая сеть с двумя хост-системами (X1 и X2) и несколькими маршрутизаторами между ними. Предположим, что система X1 посылает некоторую информацию системе X2, и изучим, какую роль играет сетевой уровень для данных систем и маршрутизаторов. Сетевой уровень получает от транспортного уровня в системе X1 сегменты информационного блока, заключает их в дейтаграммы (т. е. в пакеты сетевого уровня) и посылает их на ближайший маршрутизатор (M1). В пункте назначения (X2) сетевой уровень получает дейтаграммы от ближайшего маршрутизатора (M2), извлекает сегменты данных и доставляет их на транспортный уровень в системе X2. Основная роль маршрутизаторов заключается в обработке дейтаграмм и передаче их от одной конечной системы к другой. Обратите внимание, что маршрути-

заторы на рис. 4.1 изображены с сокращенным стеком протоколов, т. е. не показаны уровни выше сетевого, поскольку маршрутизаторы не считая собственного управления не используют протоколы прикладного и транспортного уровней, о которых рассказывалось в главах 2 и 3.

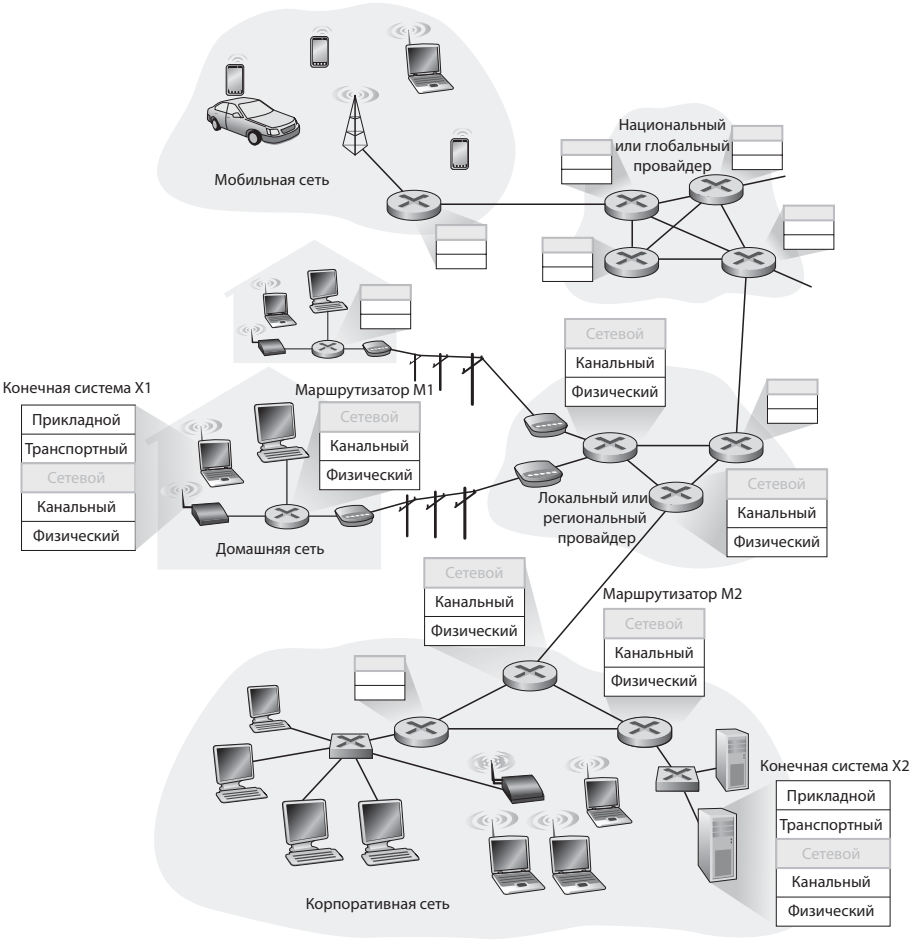


Рис. 4.1. Сетевой уровень

4.1.1. Перенаправление и маршрутизация

Вы уже знаете, что задача сетевого уровня, заключающаяся в перемещении пакетов от отправителя получателю, проста только на первый взгляд. Для ее реализации сетевой уровень выполняет две важных функции:

- *Перенаправление*. Когда маршрутизатор получает пакет из входящего канала, он должен обработать его и передать в соответствующий исходящий. Например, пакет, прибывающий из системы X1 на маршрутизатор M1, должен быть передан на следующий маршрутизатор на пути к системе X2. В разделе 4.3 вы заглянете внутрь маршрутизатора и узнаете, как именно он передает пакет из одного канала в другой.

- *Маршрутизация*. Сетевой уровень должен определить маршрут (путь) перемещения пакетов от отправителя к получателю. Алгоритмы, вычисляющие такие пути, называются **алгоритмами маршрутизации**. Например, именно алгоритм маршрутизации определил бы, по какому пути пройдут пакеты от системы X1 к системе X2.

Авторы, пишущие о сетевом уровне, зачастую употребляют понятия *перенаправление* и *маршрутизация* в качестве синонимов. В данной книге вышеупомянутые термины применяются более точно. Перенаправление производится внутри конкретного маршрутизатора и заключается в передаче пакета от входного интерфейса до соответствующего выходного. Процесс же маршрутизации охватывает всю сеть, определяя путь пакета от точки отправления до места назначения. Если сравнить данную ситуацию с поездкой, описанной в разделе 1.3.1, то можно отметить, что водителю часто встречались перекрестки по пути в конечный пункт. Можно представить перенаправление как проезд через перекресток: водитель въезжает на него, сворачивает на дорогу, по которой он продолжит путь, и покидает перекресток.

Маршрутизация в данном примере представляется как процесс планирования поездки: перед отправкой водитель изучил карту и выбрал один из возможных путей, каждый из которых состоит из множества дорожных сегментов, соединяющихся на перекрестках.

Каждый маршрутизатор имеет **таблицу перенаправления**. Маршрутизатор передает пакет, сверяя значение в поле его заголовка с таблицей перенаправления. Хранящееся в таблице значение указывает, на какой выходной интерфейс пакет должен быть передан. В зависимости от протокола сетевого уровня значение заголовка может содержать адрес получателя пакета или тип соединения, к которому пакет относится. На рис. 4.2 приведен пример. Пакет со значением 0111 в поле заголовка поступает в маршрутизатор. Маршрутизатор ищет его в своей таблице перенаправления и определяет, что выходным интерфейсом для данного пакета является интерфейс 2. Затем маршрутизатор передает внутри

себя этот пакет в интерфейс 2. В разделе 4.3 вы заглянете внутрь маршрутизатора и изучите функцию перенаправления более детально.

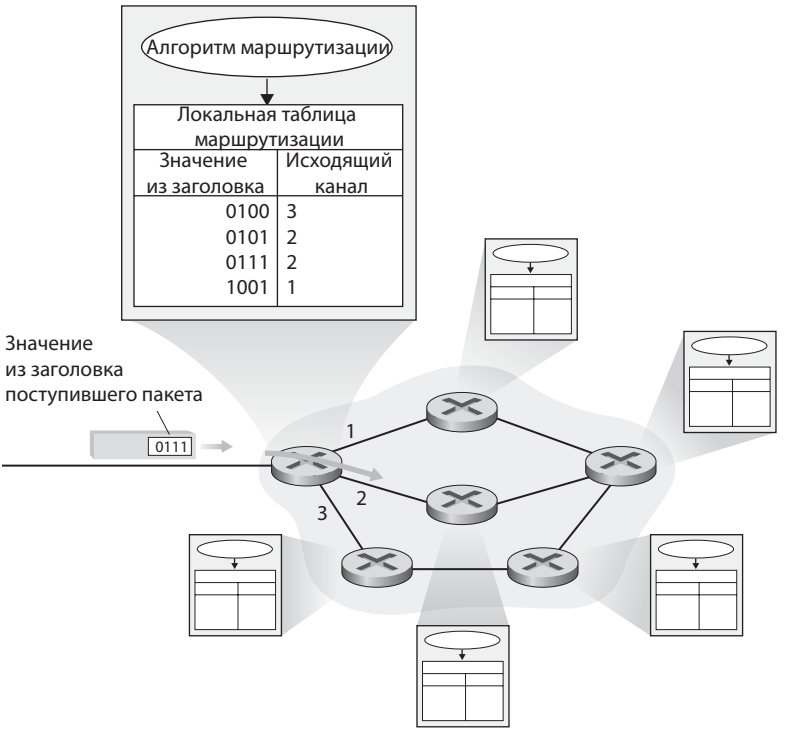


Рис. 4.2. Работа алгоритмов маршрутизации со значениями в таблице маршрутизации

Вы можете задаться вопросом, как настраиваются таблицы маршрутизации в маршрутизаторах. Данная проблема имеет важное значение, а также демонстрирует зависимость процессов маршрутизации и перенаправления. Как показано на рис. 4.2, алгоритм маршрутизации определяет значения, записываемые в таблицы маршрутизаторов. Алгоритм маршрутизации может быть централизованным (т. е. алгоритм выполняется на центральном узле сети, и его результаты раздаются всем маршрутизаторам) или децентрализованным (т. е. на каждом маршрутизаторе запущен собственный распределенный алгоритм маршрутизации). В любом случае маршрутизатор получает сообщения протокола маршрутизации, которые используются для настройки его таблицы маршрутизации. Функциональные различия перенаправления и маршрутизации можно проиллюстрировать гипотетической (хотя и технически возможной) ситуацией, при которой все таблицы напрямую настраивались бы админи-

страторами вычислительной сети, выступающими в роли маршрутизаторов. При таких обстоятельствах полностью отпала бы необходимость в протоколах маршрутизации. Конечно, операторам пришлось бы связываться друг с другом всякий раз при необходимости настройки таблиц маршрутизации, чтобы обеспечить доставку каждого пакета. Кроме того, работа человека чревата ошибками, а сама настройка таблиц скорее всего будет проходить слишком медленно по сравнению со скоростью, достижимой при помощи протокола маршрутизации, и не будет успевать за изменениями в сетевой топологии. К счастью, и функции перенаправления, *и* функции маршрутизации предусмотрены в любых сетях!

Раз уж мы заговорили о терминологии, стоит упомянуть два других термина, которые также часто путают. Словосочетание *пакетный коммутатор* будет применяться для общего обозначения устройств, перенаправляющих пакеты из входного интерфейса в выходной согласно значению в поле заголовка пакета. Функционирование пакетных коммутаторов — **сетевых**, описанных в главе 5, — базируется на значениях в поле информационного блока канального уровня (коммутаторы в главе 5 упоминаются, как устройства канального уровня — уровень 2). Работа других пакетных коммутаторов, называемых **маршрутизаторами**, зависит от значений в поле заголовка сетевого уровня.

Маршрутизаторы являются устройствами сетевого уровня (уровень 3), но также они задействуют и протоколы уровня 2, чтобы обеспечить функционирование собственных служб. Чтобы понять это важное различие между двумя видами коммутаторов, вы можете открыть раздел 1.5.2, где описываются дейтаграммы сетевого уровня, блоки канального уровня и их взаимодействие. В довершение всего, в маркетинговой литературе часто ошибочно используется определение «коммутаторы уровня 3» для маршрутизаторов с интерфейсами Ethernet, но правильнее будет называть их «устройствами уровня 3».

В связи с тем, что темой данной главы является сетевой уровень, в ней будет использоваться термин «маршрутизатор» в значении «пакетный коммутатор» (в т.ч. если речь идет о пакетных коммутаторах в сетях с виртуальными каналами; см. далее).

Установление соединения

Итак, вы знаете, что сетевой уровень выполняет две важных функции — перенаправление и маршрутизация. Но вскоре вы узнаете, что в некоторых компьютерных сетях фактически существует и третья важ-

ная функция сетевого уровня, которая называется **установление соединения** (Connection setup). Вспомните из урока изучения TCP, что квитирование связи (так называемое рукопожатие) требуется выполнить прежде, чем данные начнут передаваться от отправителя к получателю. Это позволяет отправителю и получателю получить необходимую информацию о состоянии (например, порядковый номер и первоначальный размер окна управления потоками). Аналогично и некоторые виды архитектуры сетевого уровня, такие как ATM, FR (ретрансляция кадров) и MPLS (см. раздел 5.8), требуют подтверждения установления связи от каждого маршрутизатора на всем пути от источника до места назначения, чтобы получить данные о состоянии связи, прежде чем начать передавать пакеты в пределах одного соединения. В сетевом уровне такой процесс называется *установлением соединения* (см. раздел 4.2).

4.1.2. Модели служб сетевого уровня

Прежде чем углубиться в изучение сетевого уровня, необходимо получить о нем более широкое представление. Для этого рассмотрим различные типы служб, которые могут использоваться на сетевом уровне. Когда транспортный уровень в системе-отправителе передает пакет в сеть (т. е. передает пакет на сетевой уровень внутри системы-отправителя), можно ли рассчитывать на то, что сетевой уровень самостоятельно обеспечит его доставку в пункт назначения? Когда посылается множество пакетов, будут ли они доставлены на транспортный уровень системы-получателя в том порядке, в котором их отправляли? Будет ли совпадать отрезок времени между последовательной посылкой двух отдельно взятых пакетов с отрезком времени между их приемом? Если информация по каким-то причинам начнет пробуксовывать в сети — получим ли мы уведомление об этом? Как абстрактно можно представить тот канал (то есть, его свойства), который соединяет транспортные уровни отправителя и получателя? Ответы на эти и другие вопросы зависят от **модели служб сетевого уровня**. Выбранная модель определяет особенности сквозной транспортировки пакетов между конечными системами-отправителями и получателями.

Рассмотрим некоторые службы сетевого уровня. В системе-отправителе, когда пакет передается с транспортного уровня на сетевой, тот может предоставить следующие службы (функции):

- *Гарантированная доставка* (Guaranteed delivery). Данная служба гарантирует, что пакет прибудет в пункт назначения.

- *Гарантированная доставка с ограниченной задержкой* (Guaranteed delivery with bounded delay). Такая служба гарантирует доставку пакета с задержкой, не превышающей указанного количества времени (например, не более 100 мс).

Существуют также службы, предназначенные для организации *потока пакетов* по пути от источника до места назначения:

- *Упорядоченная доставка пакетов* (In-order packet delivery). Данная служба обеспечивает получение пакетов в порядке их отправки.
- *Гарантированная минимальная пропускная способность* (Guaranteed minimal bandwidth). Эта служба сетевого уровня эмулирует работу канала связи с определенной скоростью (например, 1 Мбит/с) между отправителем и получателем. Пока система-отправитель передает биты (как элементы пакетов) в пределах указанной пропускной способности, потерь не происходит и каждый пакет прибывает с допустимой задержкой от системы к системе (например, в пределах 40 мс).
- *Гарантированный максимальный джиттер* (Guaranteed maximum jitter). Службой гарантируется, что интервалы времени между последовательной передачей двух пакетов отправителем будут равняться интервалам между их приемом (либо интервалы могут отличаться в пределах указанного значения).
- *Службы безопасности* (Security services). Используя секретный сеансовый ключ, известный только системе-отправителю и системе-получателю, сетевой уровень отправителя может шифровать содержимое всех дейтаграмм, посылаемых в пункт назначения. Сетевой уровень получателя в таком случае будет отвечать за расшифровку содержимого. Данная служба обеспечивает конфиденциальность всех сегментов транспортного уровня (TCP и UDP) между источником и получателем, а также целостность данных и работу служб аутентификации.

Выше перечислена только часть всех тех служб, которые может предоставить сетевой уровень, возможны самые разнообразные сочетания.

Сетевой уровень Интернета представляет единственную службу — **службу негарантированной доставки** (Best-effort service). Из табл. 4.1 может сложиться впечатление, что *данным термином* обозначается *отсутствие какой-либо службы вообще*: не гарантируется ни сохранность пакетов, ни их получение в порядке отправки, ни доставка переданных

пакетов в конечном итоге. Учитывая вышеизложенное, можно сказать, что, даже если сеть не обеспечит доставку *ни одного* пакета в место назначения, такое поведение не будет противоречить определению негарантированной доставки. Тем не менее такая минималистическая модель обслуживания сетевого уровня бывает весьма целесообразной (об этом поговорим чуть позже).

Табл. 4.1. Модели служб архитектур Интернет, ATM CBR и ATM ABR

Архитектура сети	**Модель служб**	**Гарантированная пропускная способность**	**Гарантия отсутствия потерь**	**Порядок получения**	**Временные соотношения**	**Индикация перегрузки**
Интернет	Негарантированная доставка	Нет	Нет	В любом порядке	Не поддерживаются	Не поддерживаются
ATM	Постоянная скорость	Гарантированное постоянное значение	Есть	В порядке отправки	Поддерживаются	Перегрузка не возникает
ATM	Доступная скорость	Гарантированный минимум	Нет	В порядке отправки	Не поддерживаются	Поддерживаются

В других видах сетевых архитектур были определены и реализованы модели служб, которые пошли гораздо дальше негарантированной доставки. Например, архитектура сети ATM[344,56] предусматривает множество моделей служб, поскольку для различных видов связи необходимы разные классы служб внутри одной сети. В этой книге не будут рассматриваться принципы работы сети ATM, необходимо лишь подчеркнуть, что существуют альтернативы службе негарантированной доставки Интернета. Двумя наиболее важными моделями ATM являются служба передачи данных с постоянной скоростью и служба доступной скорости передачи данных:

- ***Служба передачи данных с постоянной скоростью сети ATM*** (Constant bit rate, CBR). Первая модель сети ATM, которая была стандартизирована в связи с заинтересованностью телефонных компаний сетью ATM и возможностью использования служб CBR для передачи аудио- и видеотрафика с постоянной скоростью. Концептуально цель данной модели проста — обеспечить стабильный поток пакетов (ячеек в терминологии ATM) по виртуальному конвейеру таким образом, как если бы между системой-отправителем и получателем существовал специальный канал с постоянной пропускной

способностью. Служба CBR представляет гораздо больше гарантий относительно задержек и потерь ячеек, а также колебаний в длительности задержек (т. е. джиттера). Значения допустимых задержек и потерь согласовываются системой-отправителем при установлении CBR-соединения.

- ***Служба доступной скорости передачи данных сети ATM*** (Available bitrate service, ABR). Данная служба имеет больше гарантий, чем служба негарантированной доставки, действующая в Интернете. И в том и в другом случае возможны потери ячеек (пакетов), однако служба ABR не изменяет порядок ячеек при их получении, а также гарантирует минимальную скорость передачи ячеек (MCR, Minimum cell rate). Если сеть имеет достаточно свободных ресурсов в какой-то момент времени, ячейки могут быть успешно переданы с более высокой скоростью. Кроме того, как уже говорилось в разделе 3.6, служба ATM ABR может обеспечивать обратную связь с системой-отправителем (для уведомлений о перегрузках или заданной скорости отправки данных), которая регулирует уровень скорости от минимальной до пиковой.

4.2. Сети с виртуальными каналами и дейтаграммные сети

Как вы помните из главы 3, транспортный уровень предусматривает режимы работы с установлением соединения и без него. Например, транспортный уровень Интернета предлагает каждому приложению выбор: UDP (без установления соединения) или TCP (с установлением соединения). Аналогичные службы предоставляет и сетевой уровень. Данные режимы на транспортном и сетевом уровнях работают схожим образом. Так, например, режим с установлением соединения сетевого уровня начинает работу с обмена данными о состоянии между хост-системами пунктов отправки и назначения, а режим без установления соединения того же уровня обходится без каких-либо прелюдий.

Несмотря на то, что указанные службы транспортного и сетевого уровня похожи между собой, они также имеют и существенные различия:

- На сетевом уровне эти службы функционируют между несколькими хост-системами для обеспечения работы транспортного уровня. На транспортном же уровне они работают между процессами и передают информацию на прикладной уровень.

- Во всех основных сетевых архитектурах (Интернет, ATM, ретрансляция кадров и др.) в настоящее время на сетевом уровне используются либо режимы с установлением соединения между системами, либо без него, но не оба вида одновременно. Компьютерные сети, использующие только предварительно установленные соединения, называются **сетями с виртуальными каналами** (Virtual Circuit, **VC**), и наоборот — сети, работающие без установления соединения называются **дейтаграммными сетями**.
- Реализации службы с установлением соединения в транспортном и сетевом уровнях существенно отличаются. В предыдущей главе говорилось о том, что данная служба на транспортном уровне выполняется в конечных системах. Далее вы узнаете, что служба с установлением соединения сетевого уровня реализуется как в конечных системах, так и в маршрутизаторах.

Сети с виртуальными каналами и дейтаграммные сети — это два основных класса компьютерных сетей. Принципы обработки информации для передачи данных в них кардинально различаются.

Рассмотрим их реализации более подробно.

4.2.1. Сети с виртуальными каналами

В то время как Интернет — это дейтаграммная сеть, существует множество альтернативных сетевых архитектур, включая сети ATM и ретрансляцию кадров. Эти альтернативные реализации сетей работают с **виртуальными каналами** и потому используют предварительно установленные соединения на сетевом уровне. Изучим, как виртуальные каналы применяются в компьютерной сети.

Виртуальный канал состоит из следующих элементов: (1) маршрут (т. е. серия адресов и маршрутизаторов) между отправителем и адресатом, (2) номера VC, по одному для каждого соединения на всем пути следования и (3) значения в таблицах маршрутизации всех тех маршрутизаторов, через которые проходит пакет.

Пакет, следующий по виртуальному каналу содержит номер VC в заголовке. Поскольку виртуальный канал может иметь различные номера для всех соединений, каждый участвующий в процессе маршрутизатор должен заменить номер в каждом проходящем через него пакете на новый. Новый номер VC берется из таблицы маршрутизации.

Описанный выше принцип проиллюстрирован на рис. 4.3. Номера рядом с соединениями М1 — это номера каналов (интерфейсов) маршрутизатора. Допустим, что система А запрашивает установление виртуального канала с системой Б. Предположим также, что сеть выбирает маршрут А-М1-М2-Б и присваивает соединениям, расположенным на пути следования по виртуальному каналу, номера 12, 22 и 32.

В таком случае, когда пакет покинет пункт А, значение в поле номера VC в заголовке пакета будет равно 12, когда покинет пункт М1, значение будет равно 22 и после выхода из пункта М2 — 32.

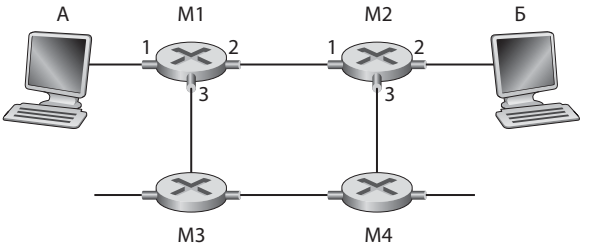


Рис. 4.3. Простая сеть с виртуальным каналом

Как маршрутизатор определяет, каким числом заменить значение номера для проходящего через него пакета? В сетях с виртуальными каналами таблица перенаправления в каждом маршрутизаторе имеет расшифровку номеров VC. Например, таблица перенаправления в маршрутизаторе М1 могла выглядеть примерно так:

Входной интерфейс	Входящий номер VC	Выходной интерфейс	Исходящий номер VC
1	12	2	22
2	63	1	18
3	7	2	17
1	97	3	87
...	...	...	...

Всякий раз, когда устанавливается виртуальное соединение через данный маршрутизатор, в таблице перенаправления добавляются соответствующие записи. И также всякий раз, когда соединение завершается, эти записи удаляются.

Может возникнуть вопрос, почему пакет не может нести один и тот же номер для всех соединений в маршруте. Во-первых, замена номера на новый при смене соединений позволяет сохранять небольшое значение в заголовке пакета. Во-вторых, и что еще более важно, возможность присваивать каждой ссылке отдельный номер значительно упрощает создание виртуальной цепи. Благодаря этому каждому соединению назначается собственный идентификационный номер, отличающий его от других адресов. Если бы для всех соединений в маршруте использовался один и тот же номер, то маршрутизаторам приходилось бы обмениваться данными друг с другом, обрабатывая множество сообщений, чтобы выбрать номер, который будет использоваться в создаваемом виртуальном канале (например, такой номер, который не использовался бы другими виртуальными каналами, проходящими через эти маршрутизаторы).

В сетях с виртуальными каналами маршрутизаторы должны поддерживать **информацию о состоянии соединения** для действующих в данный момент каналов. Каждый раз, когда устанавливается соединение, маршрутизатор должен внести необходимые записи в таблицу перенаправления, и каждый раз, когда соединение прекращено, эти записи должны быть удалены.

Важно отметить, что даже без трансляции номеров VC все равно необходимо отслеживать состояние соединения и связывать номера VC с выходными интерфейсами маршрутизаторов. Вопрос поддержки маршрутизатором информации о состоянии соединения является критически важным и будет еще неоднократно подниматься в данной книге.

Функционирование виртуального канала состоит из трех основных фаз:

- *Установление виртуального соединения*. Во время данной фазы транспортный уровень обращается к сетевому, определяет адрес получателя и ожидает, когда сеть создаст виртуальный канал. Сетевой уровень строит маршрут от отправителя к получателю, т. е. последовательность соединений и маршрутизаторов, через которые будут проходить пакеты. Сетевой уровень также назначает номера для каждого соединения в маршруте. Наконец, сетевой уровень добавляет значения в таблицы перенаправления всех маршрутизаторов в канале. Во время создания канала сетевой уровень может также зарезервировать некоторые ресурсы (например, часть полосы пропускания выбранных соединений).

- *Передача данных.* Как показано на рис. 4.4 после установления соединения, пакеты начинают перемещаться по виртуальному каналу.
- *Освобождение канала.* Система отправителя (или получателя) передает на сетевой уровень сообщение с требованием о прекращении соединения. Сетевой уровень оповещает конечную систему на другом конце цепочки о завершении соединения и обновляет таблицы перенаправления всех маршрутизаторов, удаляя информацию о канале.

Существует тонкое, но важное отличие между установлением соединения на сетевом и транспортном уровнях (вспомните, например, тройное рукопожатие в протоколе TCP, описанное в главе 3). Соединение, установленное на транспортном уровне подразумевает связь лишь между двумя конечными системами. Во время его создания только две конечные системы задают параметры соединения (например, начальный номер и размер окна управления потоками). Маршрутизаторы же не обмениваются информацией с транспортным уровнем. С другой стороны, благодаря сетевому уровню *маршрутизаторы по всей длине цепи участвуют в создании виртуального канала, и каждый маршрутизатор имеет информацию обо всех проходящих через него виртуальных каналах.*

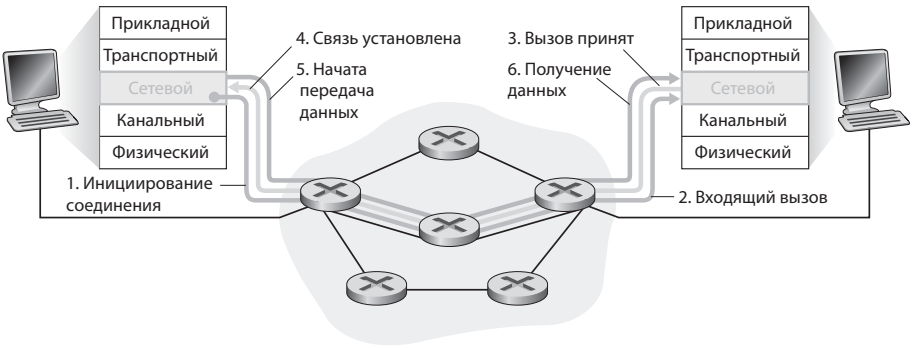


Рис. 4.4. Установление виртуального соединения

Сообщения о том, что конечные системы инициируют или прекращают виртуальное соединение, а также сообщения между маршрутизаторами об установлении связи (необходимые для отслеживания состояния соединения и внесения изменений в таблицы маршрутизации) называются **сигнальными сообщениями**, а протоколы для обмена такими сообщениями часто называют **сигнальными протоколами**. Установление виртуального соединения проиллюстрировано на рис. 4.4.

Сами сигнальные протоколы в данной книге рассматриваться не будут. (См. работу Блэка[57] — обсуждение общих вопросов обмена сигналами в коммутируемых сетях и документ ITU-T Q.2931 1995[257] — спецификация сигнального протокола Q.2931 архитектуры ATM.)

4.2.2. Дейтаграммные сети

В **дейтаграммной сети** каждый раз, когда возникает необходимость отправить пакет, конечная система оставляет на пакете пометку с целевым адресом и просто выпускает его в сеть. Как показано на рис. 4.5 отправке пакетов не предшествует установление соединения, а маршрутизаторы не обмениваются информацией о состоянии виртуального канала (потому что его попросту нет).

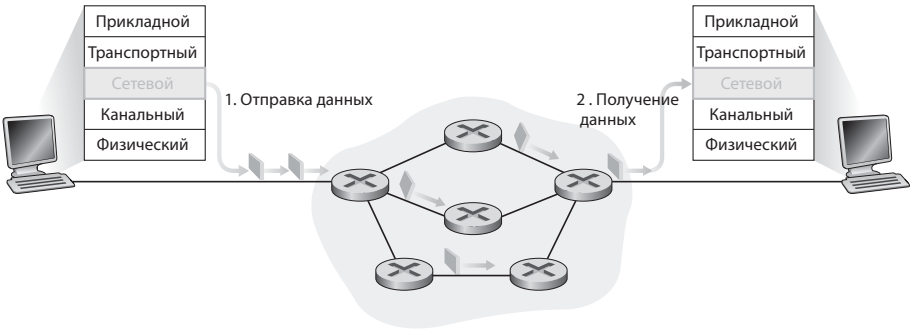


Рис. 4.5. Дейтаграммная сеть

Прежде чем пакет достигнет мес та назначения, он проходит через цепочку маршрутизаторов. Каждый из них использует указанный в пакете целевой адрес для дальнейшей передачи. Каждый маршрутизатор имеет таблицу маршрутизации, в которой хранятся адреса получателей. На эти адреса можно отправлять пакеты; когда пакет достигает маршрутизатора, последний считывает с него адрес получателя и ищет соответствующий выходной интерфейс в таблице маршрутизации и передает пакет в него.

Чтобы понять, как работает операция поиска, рассмотрим небольшой пример. Допустим, что все адреса получателей — 32-разрядные (и в дейтаграммах IP это именно так). При самом примитивном способе обработке в таблице существовало бы по одной единственной записи для каждого возможного адреса получателя. Однако в настоящее время насчитывается более четырех миллиардов возможных адресов.

Теперь предположим, что рассматриваемый маршрутизатор имеет четыре адреса, пронумерованные от 0 до 3, и что пакеты должны быть переданы на их интерфейсы следующим образом:

Диапазон адресов назначения	Интерфейс канала
от 11001000 00010111 00010000 00000000 до 11001000 00010111 00010111 11111111	0
от 11001000 00010111 00011000 00000000 до 11001000 00010111 00011000 11111111	1
от 11001000 00010111 00011001 00000000 до 11001000 00010111 00011111 11111111	2
остальные	3

На данном примере становится понятно, что нет необходимости иметь все четыре миллиарда записей в таблице маршрутизации. Вместо этого таблица может содержать всего четыре записи:

Префикс	Интерфейс канала
11001000 00010111 00010	0
11001000 00010111 00011000	1
11001000 00010111 00011	2
остальные	3

При таком оформлении таблицы маршрутизатор сравнивает **префикс** адреса получателя пакета с записями в таблице, если соответствие найдено, маршрутизатор передает пакет по ссылке, присвоенной заданному значению. Например, пусть адрес получателя пакета выглядит следующим образом: 11001000 00010111 00010110 10100001. Поскольку 21-разрядный префикс соответствует первой записи в таблице, маршрутизатор передает пакет на интерфейс 0. Если бы префикс не соответ-

ствовал первым трем значениям, то маршрутизатор передал бы пакет на интерфейс 3. Несмотря на то что описанный принцип выглядит довольно простым, существует важная деталь. Можно заметить, что адресу получателя может соответствовать более одной записи. Например, первые 24 разряда адреса 11001000 00010111 00011000 10101010 соответствуют второй записи в таблице, а первый 21 разряд того же адреса соответствует третьей записи. При наличии многократных соответствий маршрутизатор использует так называемое **правило наибольшего совпадения**, т. е. находит самую длинную совпадающую с префиксом запись в таблице и передает пакет на соответствующий ей интерфейс. Зачем применяется данное правило, вы узнаете в разделе 4.4, когда будет подробно изучаться Интернет-адресация.

Несмотря на то что маршрутизаторы в дейтаграммных сетях не поддерживают информацию о состоянии соединения, они все же поддерживают обработку этой информации в таблицах маршрутизации. Однако такая информация о перенаправлении изменяется довольно медленно. Дело в том, что в дейтаграммных сетях с таблицами маршрутизации дополнительно работают алгоритмы маршрутизации, обновляющие их примерно через каждые 1–5 минут. В сетях же с виртуальными каналами таблицы перенаправления в маршрутизаторах обновляются при каждом подключении к какому-либо каналу, а также при каждом отсоединении. Если подобные операции осуществляются на магистральном маршрутизаторе первого уровня, то на это затрачиваются доли секунды.

Поскольку таблицы маршрутизации в дейтаграммных сетях могут быть обновлены в любой момент, серия пакетов, отправленных от одной конечной системы к другой, может следовать через сеть разными путями и прибыть к месту назначения в неправильном порядке. В работах Паксона[385] и Джейсвола[265] представлены интересные исследования переупорядочивания пакетов, а также другие явления в общедоступной части Интернета.

4.2.3. Происхождение сетей с виртуальными каналами и дейтаграммных сетей

Эволюция дейтаграммных и VC-сетей отражает их происхождение. Понятие виртуального канала, как центральный принцип организации сети восходит к миру телефонии, где используются физические соединения. Поскольку сеть с виртуальными каналами обладает такими функциями, как установление соединения и поддержка маршрутизаторами

информации о его состоянии, она, пожалуй, превосходит по сложности дейтаграммную сеть (см. интересное сравнение сложности сетей с коммутацией каналов и с коммутацией пакетов[349]). Это также унаследовано из телефонии. В телефонных сетях эти сложности появились задолго до компьютерных, поскольку требовалось соединять весьма неинтеллектуальные конечные устройства вроде дисковых телефонов (для тех, кто не застал их: дисковый телефон — это аналоговое устройство, он не имел кнопок и позволял совершать только голосовые вызовы).

Интернет — это дейтаграммная сеть, однако необходимость соединять компьютеры напрямую давно отпала. Учитывая тот факт, что современные устройства, подключаемые к сети, более совершенны, архитекторы Интернета решили упростить модель обслуживания. Как говорилось в главах 2 и 3, дополнительные функции (такие как доставка в порядке отправки, надежная передача данных, управление перегрузками, получение информации через DNS) выполняются на более высоких уровнях в конечных системах. Из всего вышесказанного о модели телефонной сети можно сделать интересные выводы:

- Поскольку сложившаяся в Интернете модель служб сетевого уровня предоставляет минимальные (практически нулевые!) гарантии, на сетевой уровень накладываются минимальные требования. Данный фактор упрощает соединение между сетями, использующими различные технологии канального уровня (например, Satellite, Ethernet, оптоволокно или радио) и имеющими разные скорости передачи данных и значения потерь. Раздел 4.4 будет посвящен более подробному изучению сопряжения сетей IP с другими сетями.
- Как было сказано в главе 2, такие приложения как электронная почта и Всемирная паутина, а также некоторые виды сетевых служб, например, DNS, выполняются на конечных хост-системах (на серверах). Возможность добавлять новую службу, просто включая сервер в сеть и задавая протокол прикладного уровня (например, HTTP), позволила развернуть новые Интернет-приложения, в частности, Всемирную паутину, за удивительно короткий промежуток времени.

4.3. Маршрутизатор изнутри

Теперь, когда изучены службы и функции сетевого уровня, настало время обратить внимание на такую функцию, как **перенаправление** — фактическая передача пакетов от входящих каналов (интерфейсов)

к соответствующим исходящим внутри конкретного маршрутизатора. В разделе 4.2 уже было рассмотрено несколько аспектов перенаправления, а именно адресация и правило наибольшего совпадения. Также обращаем ваше внимание, что термины «перенаправление» и «коммутация» исследователями компьютерных сетей и практиками зачастую используются как синонимы; в данной книге указанные термины также будут считаться эквивалентными.

Для формирования более точного представления об универсальной архитектуре маршрутизатора см. рис. 4.6. Можно выделить четыре основных компонента маршрутизатора:

- *Входные порты.* Входной порт выполняет несколько ключевых функций. Он функционирует на физическом уровне, завершая входящую физическую связь в маршрутизаторе. Данный процесс изображен в виде крайнего левого блока входного порта и крайнего правого блока входного порта на рис. 4.6. Входной порт также выполняет функции сетевого уровня, требуемые для взаимодействия с сетевым уровнем на другой стороне входящего соединения. Описанный процесс представлен средними блоками входных и выходных портов (рис. 4.6). Функция поиска и выбора — видимо, наиболее важная — также осуществляется во входных портах (крайний правый блок порта на рис. 4.6). Именно в этом компоненте маршрутизатор сверяется с таблицей перенаправления и определяет выходной порт, в который прибывший пакет будет перенаправлен через коммутирующую матрицу. Управляющие пакеты (например, пакеты, содержащие информацию о протоколе маршрутизации) передаются из входного порта в процессор маршрутизации. Обратите внимание на то, что термин «порт» в данном разделе применяется к физическим входным и выходным интерфейсам маршрутизатора и не имеет никакого отношения к «программным» портам, связанным с сетевыми приложениями и сокетами, которые обсуждались в главах 2 и 3.

- *Коммутирующая матрица*. Коммутирующая матрица соединяет входные порты маршрутизатора с выходными портами. Матрица конкретного маршрутизатора работает только в его пределах, создавая условную сеть внутри сетевого маршрутизатора.

- *Выходные порты.* Выходной порт хранит пакеты, полученные от коммутирующей матрицы, и передает их в исходящий канал, выполняя функции сетевого и физического уровней. Когда связь двунаправленная (т. е. имеет потоки данных в обоих направлениях), выходной

порт будет, как правило, спарен с входным портом того же канала на интерфейсной плате (печатная плата, реализующая один или несколько портов и подключенная к коммутирующей матрице).

- *Процессор маршрутизации*. Процессор маршрутизации выполняет протоколы маршрутизации (о которых будет рассказано в разделе 4.6), обрабатывает таблицы маршрутизации и прилагаемую информацию о состоянии соединения, а также составляет таблицу перенаправления для маршрутизатора. Кроме того, он выполняет функции управления сетью (см. главу 9).

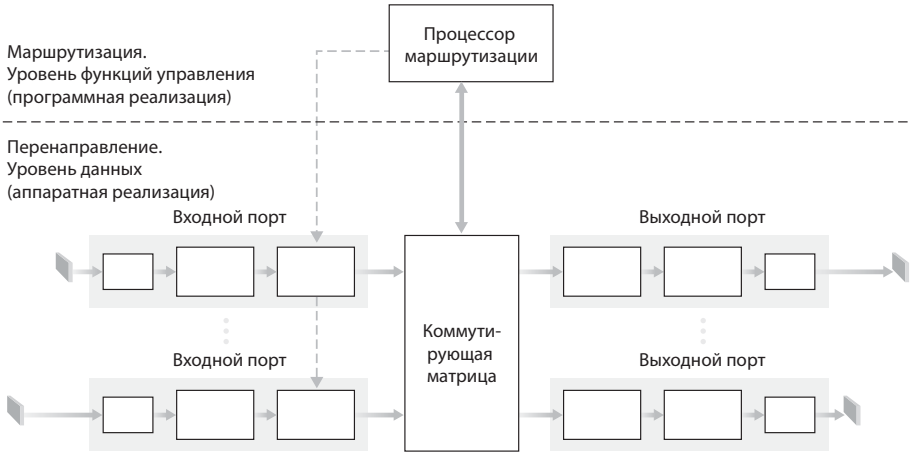


Рис. 4.6. Архитектура маршрутизатора

Как вы помните, в разделе 4.1.1 были описаны различия между функциями перенаправления данных и маршрутизации. Входные и выходные порты маршрутизатора, а также коммутирующая матрица в совокупности осуществляют функцию обработки данных и почти всегда с помощью аппаратных средств, как показано на рис. 4.6. Данные функции обработки в целом иногда называют **уровнем перенаправления данных маршрутизатора**. Чтобы понять, зачем нужна аппаратная реализация, предположим, что входной порт принимает информацию со скоростью 10 Гбит/с, а IP-дейтаграмма имеет размер 64 разряда. Таким образом, у входного порта будет всего 51,2 нс, чтобы обработать дейтаграмму, прежде чем прибудет новая. Если на интерфейсной плате находится (как это часто делается) N портов, то конвейер обработки дейтаграмм должен работать в N раз быстрее, что слишком быстро для программной реализации. Аппаратные средства перенаправления так-

же могут реализовываться с применением как собственной аппаратной архитектуры от производителя маршрутизатора, так и из коммерчески доступных микросхем (например, поставляемых такими компаниями, как Intel и Broadcom).

В то время как уровень перенаправления пакетов оперирует в пределах наносекунд, управляющие функции маршрутизатора — выполнение протоколов маршрутизации, реагирование на подключение и отключение соединений, а также функции управления сетью, которые будут изучены в главе 9 — требуют миллисекунд или даже секунд. Данные функции **уровня управления маршрутизатором** обычно выполняются программным обеспечением на процессоре маршрутизации (как правило, в этих целях используется центральный процессор).

Прежде чем углубиться в изучение уровней управления и обработки данных маршрутизатора, вспомните аналогию из раздела 4.1.1, где перенаправление пакетов сравнивалось с движением автомобиля на перекрестке. Допустим, что перекресток имеет круговое движение и прежде, чем автомобиль въедет на него, водитель должен будет остановиться на въезде и сообщить пункт назначения (не название ближайшей остановки, а конечный пункт маршрута). Оператор у шлагбаума ищет пункт назначения в базе, определяет, на какую дорогу необходимо свернуть с перекрестка, и сообщает ее название водителю. Автомобиль въезжает на перекресток, который может быть заполнен другими транспортными средствами, и в конечном итоге, покинув его, попадает на указанную дорогу.

Воспользовавшись данным примером, можно распознать основные компоненты маршрутизатора на рис. 4.6 — входящей дороге и шлагбауму соответствует входной порт (причем таблица поиска, которая была в будке у оператора, помогает найти локальный исходящий порт); перекрестком является коммутирующая матрица, а выбранной дороге соответствует выходной порт. При помощи данной аналогии можно попытаться найти возможные узкие места. Что произойдет, если транспортный поток значительно увеличится (например, если данный перекресток находится в Германии или Италии), а оператор пункта пропуска будет работать медленно? Как быстро он должен работать, чтобы предотвратить возникновение пробок? Даже если он сам сможет работать исключительно быстро, но машины будут проезжать через перекресток слишком медленно, понадобится ли направлять часть машин в объезд? И если большинство автомобилей въезжает на одну и ту же дорогу, не-

обходима ли дублирующая дорога в том же направлении на перекрестке или в другом месте? Как должен работать пропускной пункт — в первую очередь должен ли он отдавать предпочтение одним транспортным средствам и запрещать въезд на перекресток другим? Все эти вопросы аналогичны критически важным проблемам, с которыми сталкиваются разработчики маршрутизаторов и коммутаторов.

В следующих подразделах более подробно будут изучены функции маршрутизаторов. В работах Айера[259], Чао[86], Чуанга[97], Тернера[640], Маккеона[337] и Партриджа[383] обсуждаются разнообразные варианты архитектуры маршрутизаторов. Для создания более полной картины далее мы поговорим о дейтаграммных сетях, в которых обработка данных базируется на конечных адресах пакетов (в отличие от номеров VC в сетях с виртуальными каналами). Однако соответствующие понятия и методы весьма схожи с понятиями и методами в сетях с виртуальными каналами.

4.3.1. Обработка данных ввода

Подробная схема обработки входящих данных показана на рис. 4.7. Как говорилось выше, реализуемые входным портом функции сопряжения с линией и управления каналом соответствуют физическому и канальному уровням для этого соединения.

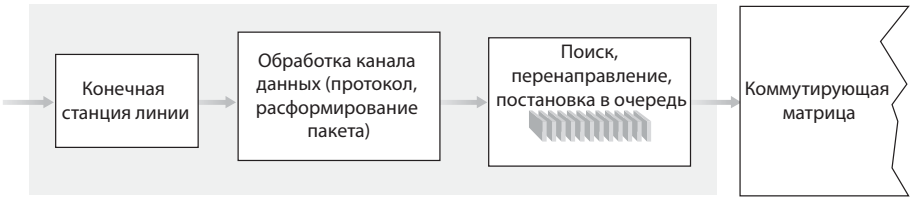


Рис. 4.7. Функционирование входного порта

Поиск, выполняемый входным портом, является важнейшим этапом работы маршрутизатора — именно на этой стадии маршрутизатор использует таблицу перенаправления для поиска выходного порта, в который пакет будет перенаправлен через коммутирующую матрицу. Таблица перенаправления формируется и обновляется процессором маршрутизации с сохранением теневой копии. Обычно такая копия хранится у каждого входного порта. Таблица перенаправления передается от процессора маршрутизации на интерфейсные платы по специальной

шине (например, по шине PCI), обозначенной пунктирной линией на рис. 4.6. Благодаря теневому копированию, обработка пакетов может производиться локально каждым входным портом без привлечения центрального процессора маршрутизации. Поэтому удается избежать узких мест, которые могли бы возникнуть при только централизованной обработке.

ИСТОРИЯ

Cisco Systems: управление сетевым ядром

На момент написания данной книги компания Cisco насчитывала свыше 63 тыс. сотрудников. Как происходило становление этой компании-гиганта? Начало было положено в 1984 г. в самой обычной гостиной в Силиконовой долине.

Лен Босак и его жена Сэнди Лернер работали в Стэндфордском университете, когда у них появилась идея собирать и продавать Интернет-маршрутизаторы исследовательским и академическим учреждениям — ведь именно в таких организациях в те времена наиболее активно развивался Интернет. Сэнди Лернер придумала название Cisco (сокращенное название города Сан-Франциско, San-Fran*CISCO*). Она также придумала логотип компании, на котором изображен мост. Супруги устроили офис прямо у себя в квартире, оплачивали проект собственными кредитками и подрабатывали консультированием. В конце 1986 г. компания Cisco достигла дохода в 250 тыс. долларов в месяц. К концу 1987 г. организации удалось привлечь венчурный капитал размером 2 млн долларов от компании Sequoia Capital в обмен на одну треть компании. В течение следующих нескольких лет Cisco продолжала расти и все больше захватывать рынок. В это время все сильнее углублялись разногласия между супругами-основателями с одной стороны и администрацией Cisco — с другой. В 1990 г. компания Cisco впервые выпустила на биржу собственные акции, и в этот же год Лернер и Босак покинули организацию.

За прошедшие годы компания значительно расширилась, охватив помимо рынка маршрутизаторов такие ниши, как безопасность продаж, платежи в беспроводных сетях, коммутаторы для сетей Ethernet, оборудование дата-центров, организация видеоконференций, а также работа с другими продуктами и услугами. Однако Cisco столкнулась с растущей международной конкуренцией, схлестнувшись, например, с быстро растущей китайской компанией Huawei, работающей в сфере сетевого оборудования. Другими конкурентами Cisco в области продажи маршрутизаторов и коммутаторов Ethernet являются Alcatel-Lucent и Juniper.

В принципе, работа с уже готовой таблицей маршрутизации проста — нужно лишь найти в таблице наиболее длинный префикс, соответствующий искомому адресу, как это описано в разделе 4.2.2. Но при гигабитных скоростях передачи такой поиск должен занимать наносекунды (вспомните предыдущий пример со скоростью канала в 10 Гбит/с и объемом IP-дейтаграммы в 64 байта). Следовательно, кроме аппаратной реализации, придется прибегнуть и к алгоритмам, более совершенным, чем простой линейный поиск по всей таблице. Обзоры «быстрых» алгоритмов поиска сделаны в работах Гапта[196] и Руз-Санчеса[579]. Особое внимание также должно быть уделено количеству обращений к памяти, от которого зависит дизайн схемы со встроенным динамическим ОЗУ (DRAM, Dynamic Random-Access Memory) или более быстрым статическим ОЗУ (SRAM, Static Random-Access Memory). Последний вариант используется в качестве кэш-памяти для DRAM. Троичная ассоциативная память (TCAM, Ternary Content Address Memory) также часто используется для поиска. При таком подходе 32-разрядные IP-адреса размещаются в памяти, которая позволяет выбирать значение из таблицы перенаправления за одинаковое время для любого адреса. Маршрутизаторы Cisco8500 предусматривают режим работы 64 Кбайт ассоциативной памяти у каждого входного порта.

Как только система поиска определит выходной порт, пакет отправляется в коммутирующую матрицу. В некоторых архитектурах он может быть временно задержан перед отправкой, если коммутирующая матрица в данный момент занята пакетами из других входных портов. Заблокированный пакет ставится в очередь внутри входного порта, а затем, когда подходит время, отправляется в коммутирующую матрицу. Более подробно блокирование, формирование очередей, а также механизм их обработки (во входных и выходных портах) будут разобраны в разделе 4.3.4. Помимо поиска выходного порта — важнейшей процедуры в работе входного порта — на этом этапе должно быть выполнено множество других действий: 1) соблюден порядок операций физического и сетевого уровней, о котором говорилось выше; 2) проверены номер версии пакета, поле контрольной суммы и поле с указанием предписанного времени жизни пакета (см. раздел 4.4.1), причем значения в последних двух полях должны быть изменены; 3) обновлены счетчики, используемые для сетевого управления (подсчитывающие, например, количество полученных IP-дейтаграмм) обновлены.

Заканчивая обсуждение обработки данных, рассмотрим входной порт. Необходимо отметить, что процесс поиска («соответствие») IP-адреса входным портом и отправка им пакета в коммутирующую матрицу («действие») являются частным случаем более общей абстракции «соответствие

плюс действие», применяемой многими сетевыми устройствами и маршрутизаторами в том числе. Коммутаторы (описываемые в главе 5) на сетевом уровне ищут адреса назначения, а также могут выполнять некоторые действия в дополнение к отправке кадров в выходной порт через коммутирующую матрицу. Брандмауэры (см. главу 8) фильтруют определенные входящие пакеты: если заголовок пакета соответствует («соответствие») заданным критериям (например, комбинация IP адресов отправителя и получателя, а также номеров портов транспортного уровня), дальнейшая передача такого пакета может быть отменена («действие»). Если номер порта входящего пакета в трансляторе сетевых адресов (NAT, Network address translator; см. раздел 4.4) соответствует («соответствие») заданному значению, транслятор присваивает новое значение номеру перед передачей («действие»). Таким образом, принцип «соответствие плюс действие» весьма эффективен и широко применяется в сетевых устройствах.

4.3.2. Коммутация

Коммутирующая матрица является основой любого маршрутизатора, поскольку благодаря ей пакеты фактически коммутируются (т. е. передаются) от входного порта данных к выходному. Как показано на рис. 4.8, коммутация может быть выполнена несколькими способами:

- *Коммутация через память.* Самые простые, самые ранние маршрутизаторы представляли собой традиционные компьютеры с функцией переключения между входными и выходными портами, осуществляемой под контролем ЦП (процессора маршрутизации). Входные и выходные порты работали, как традиционные устройства ввода-вывода в обычной операционной системе. Входной порт, получивший пакет, с помощью механизма прерывания подавал сигнал процессору маршрутизации. Пакет копировался в память процессора. Затем процессор маршрутизации извлекал адрес получателя из заголовка пакета, искал соответствующий выходной порт в таблице перенаправления и копировал пакет в буфер выходного порта. При таком сценарии, если пропускная способность памяти равна N считанных/переданных пакетов в секунду, то в целом (весь процесс записи пакетов в входной порт и перенаправления их в выходной порт) она составит менее $N/2$ пакетов в секунду. Стоит также отметить, что два пакета не могут быть переданы одновременно, даже если они направляются в различные выходные порты, поскольку на одной разделяемой системной шине за один раз может быть произведена только одна операция чтения/записи.

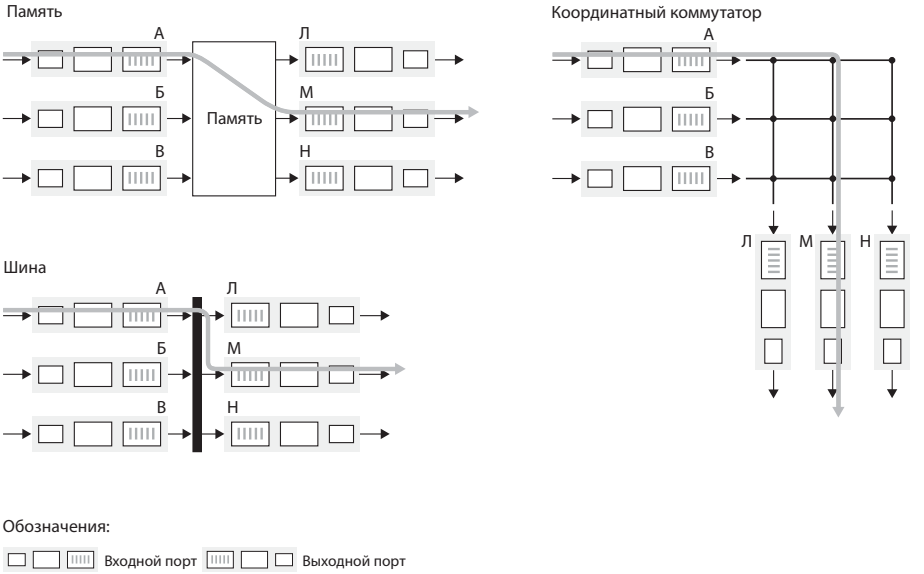


Рис. 4.8. Три способа коммутации данных

Многие современные маршрутизаторы осуществляют коммутацию через память с одним существенным отличием от более ранних моделей. Оно заключается в том, что поиск адреса получателя и сохранение пакета в соответствующую область памяти осуществляются платой входных интерфейсов. В каком-то смысле маршрутизаторы, использующие данный способ коммутации пакетов, похожи на мультипроцессоры с разделяемой общей памятью, коммутируя пакеты интерфейсной платой путем записи их в память соответствующих выходных портов. Маршрутизатор Cisco Catalyst 8500 использует[100] способ коммутации пакетов через разделяемую память.

- *Коммутация через шину*. При данном варианте входной порт передает пакет непосредственно в выходной по разделяемой общей шине без участия процессора маршрутизации. Как правило, это осуществляется путем присоединения входным портом к началу пакета внутренней метки (заголовка), которая указывает локальный выходной порт, куда этот пакет необходимо направить, после чего пакет передается в шину. Пакет получают все выходные порты, но сохранит только порт, указанный в метке. После этого метка удаляется, поскольку она используется только внутри конкретного коммутатора для передачи пакета через шину. Если маршрутизатор одновременно получает несколько пакетов, каждый из которых прибывает в раз-

ные входные порты, в шину направляется только один, т. к. за один раз она может обрабатывать только один пакет, в то время как все остальные пакеты находятся в ожидании. В связи с этим скорость коммутации маршрутизатора ограничена частотой шины. В приведенной ранее аналогии с перекрестком это выглядело бы так: транспортную развязку единовременно может пересекать лишь один автомобиль. Тем не менее, способа коммутации через шину вполне достаточно для маршрутизаторов, используемых в небольших локальных сетях и в сетях предприятий. Коммутатор Cisco 5600[106] использует коммутацию пакетов через шину системной платы со скоростью 32 Гбит/с.

- *Коммутация через соединительную сеть.* Можно преодолеть ограничение пропускной способности единственной разделяемой шины. Он заключается в использовании более сложной схемы соединений, подобной той, что применялась в прошлом, чтобы объединить процессоры в мультипроцессорную компьютерную архитектуру. Координатный коммутатор — это схема соединения, состоящая из $2N$ шин, связывающая N входных портов с N выходных портов, как показано на рис. 4.8. Каждая вертикальная шина пересекает каждую горизонтальную шину. Любая точка пересечения может быть открыта или закрыта в любой момент контроллером коммутирующей матрицы (логика которого задается самой матрицей). Когда пакет прибывает из порта A с назначением в порт Y, контроллер коммутатора закрывает точку пересечения шин A и Y, затем порт A посылает пакет в свою шину, где его получает только шина Y. Обратите внимание: в это же время пакет из порта B может быть передан в порт X, поскольку пакеты AY и BX используют различные шины. Таким образом, в отличие от предыдущих двух вариантов коммутации, коммутация через соединительную сеть допускает параллельную обработку нескольких пакетов. Однако, если два пакета из двух различных входных портов направляются в один и тот же выходной порт, то они будут переданы в него последовательно, т. к. по каждой шине одновременно может быть пропущен только один пакет.

В более сложных соединительных сетях коммутация происходит в несколько этапов, благодаря чему обеспечивается одновременная передача пакетов от различных входных портов к одному и тому же выходному порту через коммутирующую матрицу[637].

Коммутаторы семейства Cisco 12000[99] используют способ коммутации через соединительную сеть.

4.3.3. Обработка исходящих данных

Обработка данных выходным портом схематически изображена на рис. 4.9. Выходной порт передает пакеты, которые ранее были сохранены в его памяти, в исходящее соединение. Этот процесс включает выборку пакета из очереди на передачу и выполнение соответствующих функций передачи канальным и физическим уровнями.

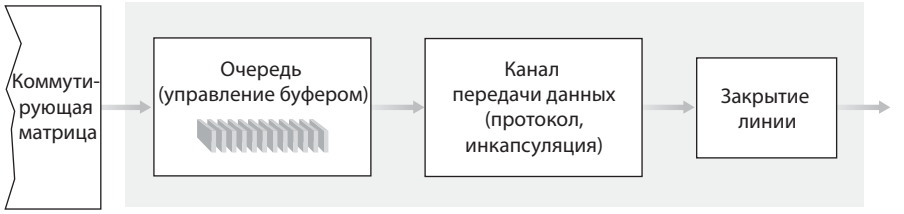


Рис. 4.9. Работа выходного порта данных

4.3.4. Формирование очереди

Если рассмотреть функциональность входных и выходных портов, а также конфигурации, изображенные на рис. 4.8, то станет ясно, что очереди пакетов могут появляться как во входных, так *и* в выходных портах. Аналогично автомобили могут ожидать своей очереди как на въезде на перекресток, так и на выезде. Расположение и длина очереди (и на вводе, и на выводе) зависят от интенсивности трафика, относительной скорости коммутирующей матрицы, а также от скорости линии. Принципы формирования очередей стоит изучить более подробно, т. к. если очередь станет слишком большой, память маршрутизатора исчерпается, новые пакеты негде будет хранить, в связи с чем начнет происходить **потеря пакетов**. Ранее вам могли встречаться фразы «пакеты были потеряны в сети» или «потерян в маршрутизаторе». Причиной тому служили очереди в маршрутизаторах, где пакеты фактически исчезали.

Предположим, что скорость передачи входящих и исходящих соединений одинакова и равна $R_{соедин.}$ пакетов в секунду, а также что существует N входных и N выходных портов. Чтобы упростить дальнейшее повествование, предположим, что все пакеты имеют одинаковую длину и поступают во входные порты синхронно. Таким образом, время отправки пакета на любой канал равно времени получения пакета из любого канала, и в течение данного интервала в один входящий канал может прибыть либо один пакет, либо ни одного. Необходимо определить скорость передачи данных матрицей $R_{матрица}$ (скорость, с которой пакеты

передаются из входного порта в выходной). Если $R_{матрица}$ будет в N раз больше $R_{соедин.}$, то во входных портах станут собираться лишь незначительные очереди. Это связано с тем, что даже в худшем случае, когда все N входящих соединений получают пакеты, которые должны быть переданы в один и тот же выходной порт, каждая группа из N пакетов (по одному пакету от каждого входного порта) может быть обработана коммутирующей матрицей прежде, чем прибудет следующая группа.

Но что тогда будет происходить в выходных портах? Допустим, что значение $R_{матрица}$ все еще в N раз больше $R_{соедин.}$, а пакеты, прибывающие в N входных портов, должны быть переданы в один и тот же выходной порт. Тогда за время, необходимое для отправки одного пакета в исходящий канал, во входной порт прибудет N новых пакетов. Поскольку выходной порт способен отправлять только один пакет за единицу времени (время пакетной передачи), N-е количество пакетов образует очередь ожидания отправки в исходящий канал. К этому времени может прибыть еще N пакетов, а отправлен будет только один пакет из первой группы и т. д. В конечном счете число пакетов в очереди может возрасти настолько, что доступная память выходного порта будет исчерпана, вызвав в дальнейшем потерю пакетов.

Принцип формирования очередей выходного порта проиллюстрирован на рис. 4.10. На момент времени t все входные порты получили по пакету, каждый из которых должен быть передан в правый верхний выходной порт. Учитывая то, что скорость передачи в каналах одинакова, а общая скорость работы коммутатора равна трем скоростям канала, то за одну единицу времени (т. е. за время, затрачиваемое на получение или отправку пакета) все три отдельных пакета были назначены к передаче в выходной порт и поставлены в очередь. За следующую единицу времени один из трех пакетов будет передан в исходящий. В приведенном примере два новых пакета достигли входа коммутатора, и один из них должен быть направлен в верхний правый выходной порт.

Учитывая, что буферы маршрутизатора необходимы, чтобы сглаживать колебания интенсивности трафика, возникает вопрос: *какая* скорость требуется буферному устройству? Достаточно долго приближенное правило расчета для определения скорости буферизации состояло в том, что скорость буферизации (B) должна быть равна среднему времени прохождения сигнала в обоих направлениях (RTT), умноженному на пропускную способность канала (C). Данный способ расчета базируется на анализе динамики формирования очередей с относительно малым количеством TCP-потоков[649]. Таким образом, канал с пропускной

способностью 10 Гбит/с и со средним временем прохождения сигнала в обоих направлениях 250 мск требовал бы скорости буферов, равной $B = RTT \times C = = 2{,}5$ Гбит/с. Однако согласно недавним теоретическим и экспериментальным работам[26] предполагается, что при наличии большого количества TCP-потоков (N), проходящих через канал, необходимая скорость буферизации должна рассчитываться по следующей формуле: $B = RTT \times C/\sqrt{N}$. При большом количестве потоков, как правило, проходящих через маршрутизаторы крупных магистральных каналов[171], значение N может быть велико, и такое сокращение необходимого размера буферов окажется достаточно существенным. В работах[26,46,671] очень доступно описана проблема подбора размеров буфера: теоретическая составляющая, вопросы реализации и эксплуатации буфера.

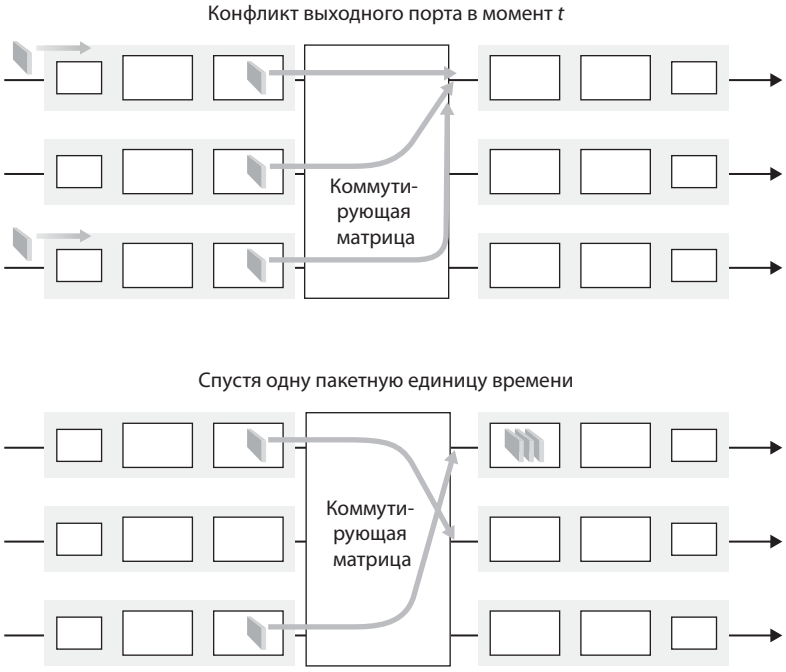


Рис. 4.10. Формирование очередей внутри выходного порта данных

Из образовавшейся очереди внутри выходного порта **планировщик пакетов** должен выбрать один из пакетов. Этот выбор осуществляется по принципу простой очереди, т. е. в порядке поступления (FCFS, First-come-first-served), или по более сложному алгоритму взвешенного обслуживания очередей (WFQ, Weighted fair queuing), который равномерно нагружает исходящий канал пакетами из всех соединений. Пла-

нирование пакетов играет важную роль в обеспечении **гарантий качества обслуживания**. Поэтому в главе 7 эта тема будет разобрана более детально. Планирование пакетов на портах подробно рассматривается в работе Congestion Management Overview[105].

Кроме того, при недостатке памяти буфера для прибывающих пакетов необходимо определить, какие из них должны удаляться — все новые пакеты (так называемое **отбрасывание хвоста**) или некоторые уже находящиеся в очереди, чтобы освободить место для вновь полученных. В некоторых случаях лучше выборочно удалять пакеты или оставлять пометки на их заголовках *до* того, как буфер переполнится, чтобы подготовиться к отправке сигнала о перегрузке отправителю. Политика подсчета удаленных пакетов, а также алгоритм маркировки пакетов (которые вместе называют технологией **активного управления очередью** (Active queue management, **AQM**)) были предложены и проанализированы в публикациях Лабрадора[305] и Холлота[207]. Один из наиболее хорошо изученных и широко применяемых алгоритмов AQM — алгоритм **произвольного раннего обнаружения** (Random Early Detection, **RED**). С данным алгоритмом взвешенное среднее количество пакетов из разных источников сохраняется по всей длине очереди. Если средняя длина очереди меньше минимальной пороговой величины min_{th}, в очередь допускаются все вновь поступившие пакеты. И наоборот, если очередь переполнена или ее средняя длина становится больше максимальной пороговой величины max_{th}, то вновь получаемые пакеты удаляются или маркируются. Наконец, если пакет прибывает в момент, когда длина очереди колеблется в интервале $[min_{th}; max_{th}]$, то пакет либо маркируется, либо удаляется в зависимости от длины очереди и заданных значений минимума и максимума. Разработчики предлагают множество функций выборочной маркировки/удаления, также было смоделировано и/или реализовано большое количество вариантов алгоритма RED. В работах Кристиансена[95] и Флойда[165] приведен обзор этой проблемы и указываются материалы для дальнейшего изучения.

Если скорость работы коммутирующей матрицы недостаточно высока (в сравнении со скоростями входящих соединений), то, чтобы маршрутизатор без задержек обработал *все* поступающие пакеты, очереди могут образовываться также и во входных портах, поскольку каждый прибывший пакет должен дождаться отправки в выходной порт через матрицу. Чтобы понять важность последствий такого способа формирования очередей, необходимо изучить принцип, по которому работает шина передачи данных координатного коммутатора. Предположим, что,

(1) все скорости всех соединений равны, (2) один пакет может быть передан из любого входного порта в заданный выходной порт за то же время, которое необходимо для приема пакета входным интерфейсом, и (3) пакеты перемещаются из заданной входной очереди в целевую выходную очередь в порядке «первым пришел — первым обслужен» (FCFS, First come, first served). Несколько пакетов могут обрабатываться параллельно, если требуемые им выходные порты различны. Однако, если два пакета, стоящие в очереди друг за другом, должны быть направлены в одну и ту же очередь на выводе, то один из пакетов блокируется и ожидает в очереди на вводе, поскольку коммутирующая матрица может передать в одно и то же время только один пакет в один и тот же выходной порт.

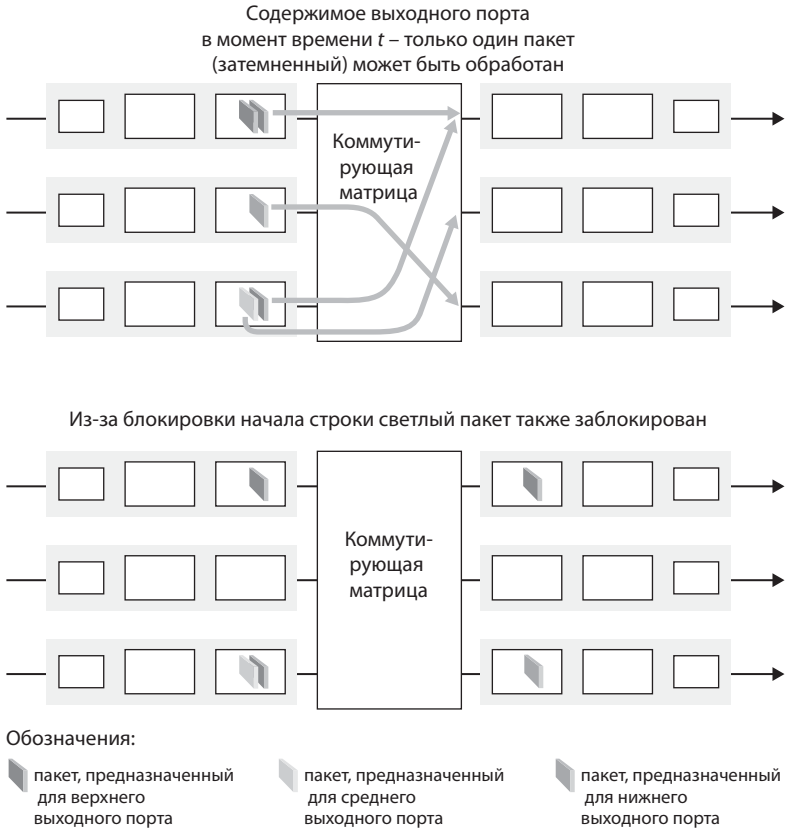


Рис. 4.11. Блокировка головы очереди в коммутаторе с очередями на вводе

На рис. 4.11 показан пример, в котором два пакета, находящиеся в начале своих очередей, должны быть направлены в один и тот же правый верхний выходной порт. Допустим, что шина передачи данных будет пе-

редавать сначала пакеты из верхней левой очереди. В этом случае пакет из нижней левой очереди должен ожидать. Также заблокирован будет и следующий за ним пакет, даже с учетом того, что он предназначен для среднего правого выходного порта и *не* имеет конкурентов. Данное явление называют **блокировкой очереди головным пакетом** (Head of the line **blocking**, **HOL**) в коммутаторе с очередями на входе — пакет, поставленный в очередь на вводе, должен ожидать передачи через матрицу (даже в случае, если его порт назначения свободен), пока будут обработаны заблокированные пакеты в начале очереди. В работе Кэрола[277] читаем, что из-за блокировки головы очереди, очередь на вводе может неограниченно вырастать (что потенциально приводит к потере пакетов) при определенных условиях, например, если частота поступления пакетов из входящих соединений возрастет до 58% их емкости. Маккеон[338] предлагает ряд решений проблем, связанных с блокировкой начала строки.

4.3.5. Уровень управления маршрутизацией

Для предшествующего обсуждения и рис. 4.6 неявно предполагалось, что функции управления маршрутизатором выполняются только процессором маршрутизации, внутри маршрутизатора. В свою очередь, управление маршрутизацией в сети в целом является децентрализованным — отдельные компоненты, то есть алгоритмы маршрутизации, выполняются на разных маршрутизаторах и взаимодействуют друг с другом посредством управляющих сообщений. Действительно, современные Интернет-маршрутизаторы и алгоритмы маршрутизации, которые будут изучены в разделе 4.6, работают именно так. Кроме того, производители маршрутизаторов и коммутаторов объединяют аппаратно реализованное перенаправление данных и программно реализованное управление в рамках закрытых (но взаимодействующих между собой) платформ в вертикально интегрированных продуктах.

Не так давно многие исследователи[70, 72, 339] приступили к разработке новой архитектуры управления маршрутизацией, где часть функций управления реализуется маршрутизатором (например, локальные мониторинг и протоколирование состояния каналов, настройка и обслуживание таблицы маршрутизации), в то время как уровень перенаправления данных и часть управляющих функций могут реализовываться вне маршрутизатора (например, в центральном сервере, который способен выполнять расчет маршрута). Правила взаимодействия этих двух частей задаются строго определенным API. Исследователи утверждают,

что отделение программных функций управления от аппаратного перенаправления данных (с минимальной нагрузкой на маршрутизатор) позволит упростить процесс маршрутизации, заменяя распределенный расчет маршрута централизованным, и преобразит сети, где различные наборы управляющих функций смогут работать с единым высокоскоростным слоем обработки данных.

4.4. Протокол IP: перенаправление и адресация данных в Интернете

До настоящего момента мы обсуждали адресацию и перенаправление данных на сетевом уровне без привязки к типу компьютерной сети. В данном разделе мы уделим внимание тому, как производятся адресация и перенаправление данных в Интернете. Вы увидите, что интернет-адресация и обработка данных — важные компоненты протокола IP (Internet protocol). На сегодняшний день существует два варианта протокола IP. Сначала вы сможете изучить широко распространенную версию этого протокола — IPv4[419], а затем выдвинутую на замену ей версию IPv6[477,532].

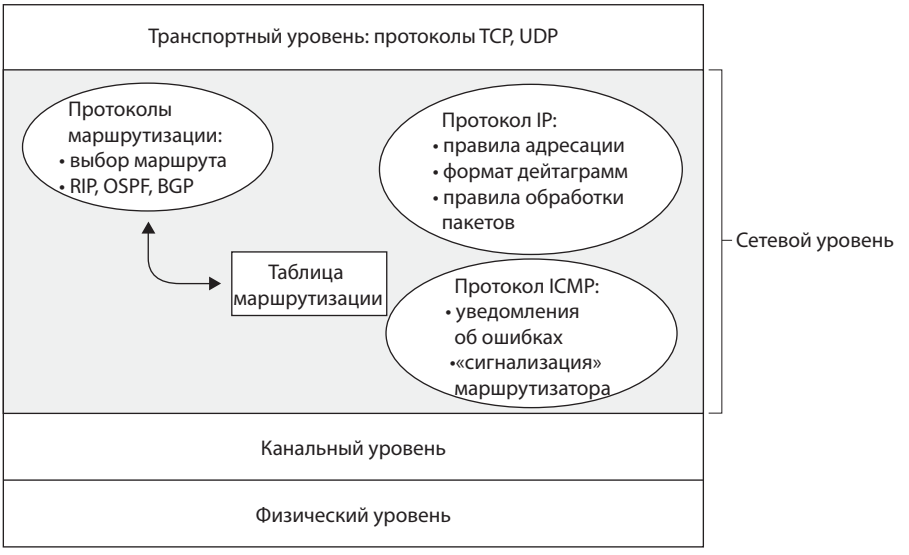


Рис. 4.12. Сетевой уровень Интернета «изнутри»

Прежде чем начать изучение данного протокола, стоит освежить пройденный материал и вспомнить составляющие сетевого уровня Интернета. Как показано на рис. 4.12, сетевой уровень Интернета состоит

из трех компонентов: первый компонент — протокол IP — тема данного раздела, второй и главный компонент — это маршрутизация, определяющая путь, по которому дейтаграммы следуют от источника к месту назначения. Ранее в этой книге уже упоминалось, что протоколы маршрутизации формируют таблицы перенаправления, которые используются для передачи пакетов через сеть. Протоколы маршрутизации Интернета будут изучены в разделе 4.6. Последний, третий компонент сетевого уровня — функции обмена информацией об ошибках в дейтаграммах, а также обмена запрашиваемой информацией на сетевом уровне. Вы сможете узнать больше об ошибках сетевого уровня сети Интернет и о протоколе обмена информацией (Internet Control Message Protocol, ICMP) в разделе 4.4.3.

4.4.1. Формат дейтаграмм

Вы уже знаете, что пакет сетевого уровня называется *дейтаграммой*. Изучение протокола IP начнется с краткого обзора синтаксиса и семантики дейтаграммы IPv4. Возможно, вы считаете, что нет ничего более скучного, чем синтаксис и семантика битов пакета. Однако дейтаграмма играет ключевую роль в изучении Интернета — каждый, от новичка до профессионала, должен понять, принять и усвоить его. Структура дейтаграммы формата IPv4 изображена на рис. 4.13. Ключевыми полями в такой дейтаграмме являются следующие:

Рис. 4.13. Формат дейтаграммы IPv4

- *Номер версии*. Первые 4 бита отводятся номеру версии протокола IP, к которому относится дейтаграмма. Определив версию, маршрутизатор может приступить к дальнейшей интерпретации полей IP-дейтаграммы. Различные версии протокола IP используют различные форматы дейтаграмм. Формат дейтаграммы данной версии — IPv4 — изображен на рис. 4.13. Формат дейтаграммы новой версии (IPv6) будет показан в конце данного раздела.

- *Длина заголовка*. Поскольку дейтаграмма IPv4 может содержать различные опции (которые указываются в заголовке дейтаграмм IPv4), то эти 4 бита необходимы для определения, в какой части строки дейтаграммы начинается собственно информация. Большинство IP-дейтаграмм не содержит опций, поэтому типичная дейтаграмма имеет 20-байтный заголовок.

- *Тип обслуживания* (type of service (TOS)). Биты типа обслуживания включены в заголовок дейтаграмм IPv4, чтобы отличать друг от друга различные типы IP-дейтаграмм (например, дейтаграммы с особыми требованиями к низкой задержке, высокой пропускной способности или надежности). К примеру, удобно отличать дейтаграммы, требующие реального времени (например, те, что используются в IP-телефонии), от так называемого «оффлайнового» трафика (например, FTP). Определение способов обслуживания таких пакетов ложится на плечи администратора маршрутизатора. Тема дифференцированного обслуживания будет изложена в главе 7.

- *Длина дейтаграммы*. Под длиной дейтаграммы подразумевается общая длина IP-дейтаграммы (заголовок плюс данные), измеренная в байтах. Т.к. данное поле имеет длину 16 бит, то теоретический максимальный размер IP-дейтаграммы составляет 65 535 байт. В действительности дейтаграммы редко превышают 1500 байт.

- *Идентификатор, флаги, смещение фрагмента*. Эти три поля имеют отношение к так называемой IP-фрагментации, см. ниже. Новая версия протокола IP, IPv6, не допускает фрагментацию в маршрутизаторах.

- *Время жизни* (time-to-live (TTL)). Наличие поля предписанного времени жизни гарантирует, что дейтаграммы не будут циркулировать по сети вечно (что могло бы случиться, например, из-за зацикливания маршрутизации). С каждым последующим прохождением дейтаграммы через маршрутизатор значение в этом поле постепенно уменьшается. Если значение достигает 0, дейтаграмма должна быть удалена.

- *Протокол*. Это поле используется только, когда дейтаграмма достигает места назначения. Поле определяется протоколом транспортного уровня, на который необходимо передать содержащиеся в дейтаграмме данные. Например, значение 6 указывает, что данные необходимо передать протоколу TCP, в то время как значение 17 определяет протокол UDP. Список всех возможных значений приведен в документе Internet Assigned Numbers Authority, Protocol Numbers[223]. Следует обратить внимание на то, что поле номера протокола в IP-дейтаграмме функционально напоминает поле номера порта в сегменте транспортного уровня. Номер протокола — это звено, связывающее сетевой и транспортный уровни вместе, а номер порта — звено, связывающее транспортный и прикладной уровни. Из главы 5 вы узнаете, что кадр канального уровня также имеет специальное поле, которое связывает канальный и сетевой уровни.

- *Контрольная сумма заголовка*. Контрольная сумма заголовка помогает маршрутизатору обнаружить ошибки в битах полученной IP-дейтаграммы. Контрольная сумма вычисляется путем интерпретации каждых двух байтов заголовка как числа и суммирования получившихся чисел с использованием арифметики дополнения до единицы. Как описано в разделе 3.3, дополненная до единицы сумма — так называемая контрольная сумма Интернета — записывается в поле контрольной суммы. Маршрутизатор вычисляет для каждой полученной IP-дейтаграммы контрольную сумму заголовка и выявляет ошибку, если та, что указана в заголовке дейтаграммы, не соответствует сумме, вычисленной маршрутизатором. Маршрутизаторы, как правило, удаляют дейтаграммы с ошибками. Следует заметить, что контрольная сумма в каждом маршрутизаторе вычисляется повторно и вновь записывается в заголовок дейтаграммы, поскольку поле TTL и, в некоторых случаях, поле опций могут измениться. Интересное обсуждение быстрых алгоритмов для вычисления контрольной суммы Интернета дается в документе RFC 1071[433].Часто задаваемый вопрос по данному полю: почему протоколы TCP/IP выполняют проверку на наличие ошибок и на транспортном, и на сетевом уровнях? Существует несколько причин для такого дублирования. Во-первых, на уровне протокола IP проверяется только заголовок IP, в то время как контрольная сумма TCP/UDP вычисляется по всей длине сегмента TCP/UDP. Во-вторых, протоколы TCP/UDP и IP не всегда имеют одинаковые стеки. Протокол TCP может работать на основе различных протоколов (например, ATM), и протокол IP может передавать данные, которые не попадают в протоколы TCP/UDP.

- *IP-адреса источника и места назначения*. Когда отправитель создает дейтаграмму, он указывает свой IP-адрес в поле источника и IP-адрес получателя в поле места назначения. Зачастую хост-система отправителя определяет адрес получателя через поиск DNS, как говорилось в главе 2. Более подробно IP-адресация будет раскрыта в разделе 4.4.2.
- *Опции*. Поля опций расширяют заголовок. Они предназначены для исключительных случаев, поэтому информация об опциях не включается в заголовок каждой дейтаграммы. Однако само по себе существование опций значительно усложняет ситуацию — в связи с тем, что заголовки дейтаграммы могут иметь переменную длину, невозможно определить априори, в каком месте начинается собственно поле данных. Кроме того, поскольку некоторые дейтаграммы могут потребовать обработки опций, затраты времени на обработку IP-дейтаграмм в маршрутизаторе могут существенно разниться. Вышеизложенные соображения становятся особенно важными для обработки данных в высокопроизводительных маршрутизаторах и хост-системах. По этим и другим причинам IP-опции не указываются в заголовках дейтаграмм IPv6 (см. раздел 4.4.4.).
- *Данные (полезное содержимое)*. Наконец, обсуждение подошло к последнему и самому важному полю — *ради него* и создается дейтаграмма. Чаще всего поле данных IP-дейтаграммы содержит сегмент транспортного уровня (протоколы TCP или UDP), необходимый для ее доставки в место назначения. Однако поле данных может содержать и другие типы данных, такие как сообщения ICMP (см. раздел 4.4.3).

Необходимо помнить, что IP-дейтаграмма имеет 20-байтный заголовок (при условии отсутствия опций). Если дейтаграмма включает сегмент протокола TCP, то каждая (нефрагментированная) дейтаграмма содержит в общей сложности 40 байт заголовка (20 байт заголовка IP плюс 20 байт заголовка TCP) помимо сообщения прикладного уровня.

Фрагментация IP-дейтаграмм

В главе 5 вы узнаете, что все протоколы канального уровня могут одновременно передавать пакеты сетевого уровня. Некоторые протоколы могут обрабатывать большие дейтаграммы, в то время как другие поддерживают обработку лишь малых пакетов. Например, кадры протокола Ethernet способны нести до 1500 байт данных, тогда как кадры

некоторых каналов глобальных сетей могут содержать не более 576 байт в одном пакете. Максимальный объем данных, который может перенести кадр канального уровня, называют *максимальным размером передаваемого блока данных* (Maximum transmission unit, MTU). Поскольку каждая IP-дейтаграмма инкапсулируется в кадр канального уровня для транспортировки от одного маршрутизатора до другого, значение MTU строго ограничивает предел длины IP-дейтаграммы. Наличие жесткого требования к максимальному размеру дейтаграммы не является большой проблемой. Проблема заключается в том, что каждый канал на протяжении всего маршрута от отправителя до получателя может использовать различные протоколы канального уровня, имеющие различные значения MTU.

Чтобы лучше понять, как решить проблему обработки данных, представьте, что *вы* являетесь маршрутизатором и соединяетесь с различными каналами, каждый из которых использует различные протоколы канального уровня с разными значениями MTU. Предположим, что вы получили IP-дейтаграмму с одного из этих каналов. Вы просматриваете свою таблицу обработки данных, чтобы определить исходящий канал, и он имеет значение MTU, меньшее, чем длина дейтаграммы. Кошмар — как сжать эту IP-дейтаграмму до размеров поля данных кадра канального уровня? Это решается разбиением содержимого дейтаграммы на две или более меньших дейтаграмм, вкладыванием каждой из них в отдельный кадр канального уровня и передачей получившихся кадров в исходящий канал. Меньшие дейтаграммы, полученные после фрагментации, называют **фрагментами**.

Фрагменты должны быть собраны воедино прежде, чем попадут на транспортный уровень в системе получателя. Действительно, транспортные уровни протоколов TCP и UDP приспособлены к получению только полных, нефрагментированных сегментов от сетевого уровня. Разработчики IPv4 понимали, что повторная сборка дейтаграмм в маршрутизаторах существенно усложнит работу протокола и снизит производительность маршрутизатора. Если бы вы были маршрутизатором, разве хотелось бы вам заниматься сборкой фрагментов помимо всего прочего? Придерживаясь принципа сохранения простоты сетевого ядра, разработчики IPv4 приняли решение переложить ответственность за сборку фрагментов на конечные системы, а не сетевые маршрутизаторы.

Когда хост-система получателя принимает серию дейтаграмм из одного источника, она должна определить, какие из них являются фраг-

ментами исходной, большей дейтаграммы. Если выяснится, что некоторые дейтаграммы — это фрагменты, далее необходимо будет определить, все ли фрагменты получены и в каком порядке их соединять, чтобы сформировать исходную дейтаграмму. Для выполнения конечной системой указанных задач разработчики IPv4 поместили в строку дейтаграммы поля «Идентификационный номер», «Флаги» и «Смещение фрагмента». При создании новой дейтаграммы система-отправитель вместе с адресами источника и получателя указывает идентификационный номер. Как правило, с новыми дейтаграммами номера постепенно увеличиваются. Когда маршрутизатор приступает к фрагментации дейтаграммы, каждая создаваемая им дейтаграмма (т. е. фрагмент) имеет в заголовке адрес источника, получателя и идентификационный номер исходной дейтаграммы. Когда получатель принимает серию дейтаграмм из одного источника, он проверяет их номера, благодаря чему может определить, какая из них фактически является фрагментом большей дейтаграммы. Поскольку протокол IP ненадежен, один или более фрагментов могут не достигнуть места назначения. Поэтому для того, чтобы конечная система могла определить, что все до одного фрагменты исходной дейтаграммы получены, последний из них помечается значением 0, тогда как все остальные имеют значение 1 в поле флага фрагментации. Кроме того, чтобы конечная хост-система могла определить, не отсутствует ли какой-либо фрагмент, а также смогла собрать все фрагменты в надлежащем порядке, используется поле смещения, в котором указывается, какую часть исходной дейтаграммы представляет данный фрагмент.

На рис. 4.14 продемонстрирован пример. Дейтаграмма размером 4000 байт (20 байт заголовка IP и 3980 байт полезной нагрузки) прибывает в маршрутизатор, она должна быть передана на канал со значением MTU 1500 байт. Это означает, что 3980 байт данных в исходной дейтаграмме должны быть разделены на три фрагмента (каждый из которых станет самостоятельной дейтаграммой IP). Предположим, что исходная дейтаграмма имеет идентификационный номер 777. Характеристики данных трех фрагментов показаны в табл. 4.2. Значения, указанные в данной таблице, отражают требования, согласно которым размер поля данных во всех фрагментах кроме последнего должен быть кратен 8 байтам, а значение смещения определено 8-байтовыми блоками.

У получателя полезное содержимое дейтаграммы передается на транспортный уровень только после того, как уровень IP полностью восстановит исходную дейтаграмму. Если один или более фрагментов не достигают получателя, неполная дейтаграмма не допускается на транс-

портный уровень и удаляется. Но, как вы знаете из предыдущей главы, если на транспортном уровне используется протокол TCP, то он восстановит потерю дейтаграммы путем перезапроса данных из источника.

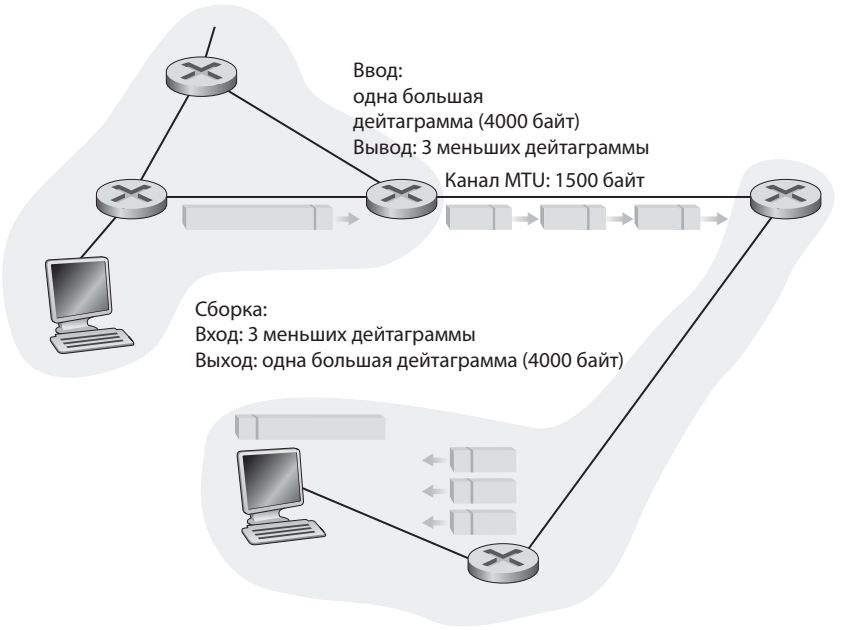


Рис. 4.14. Фрагментация и сборка дейтаграмм IP

Таб. 4.2. IP-фрагменты

Фрагмент	Байты	ID	Смещение	Флаг
№ 1	1480 байт в поле данных дейтаграммы IP	Идентификационный номер = 777	Смещение = 0 (т. е. данные должны быть вставлены в самое начало дейтаграммы)	Флаг = 1 (т. е. есть еще фрагменты)
№ 2	1480 байт данных	Идентификационный номер = 777	Смещение = 185 (т. е. данные должны быть вставлены на 1480-м байте. Обратите внимание, что 185 × 8 = 1480)	Флаг = 1 (т. е. есть еще фрагменты)
№ 3	1020 байт (3980 – 1480 – 1480) данных	Идентификационный номер = 777	Смещение = 370 (т. е. данные должны быть вставлены на 2960-м байте. Обратите внимание, что 370 × 8 = 2960)	Флаг = 0 (т. е. данный фрагмент последний)

Итак, мы узнали, что IP-фрагментация играет важную роль в объединении многочисленных разрозненных технологий канального уровня.

Но фрагментация не обходится без издержек. Во-первых, она усложняет организацию маршрутизаторов и конечных систем, которые должны проектироваться с учетом фрагментации дейтаграмм и их последующей сборки. Во-вторых, фрагментация может использоваться при подготовке катастрофических DoS-атак, в ходе которых злоумышленник забрасывает систему массой странных и неожиданных фрагментов. Классический пример — атака по модели Jolt2, где на целевой хост направляется поток мелких фрагментов, ни один из которых не имеет нулевого смещения. Атакованный хост может просто рухнуть в бесплодных попытках собрать дейтаграммы из поврежденных пакетов. При эксплойтах другого рода хост бомбардируется пересекающимися IP-фрагментами. Значения смещения у подобных фрагментов специально установлены так, что их невозможно правильно выровнять. Уязвимые операционные системы, не умеющие справляться с пересекающимися фрагментами, могут аварийно завершать работу[603]. Как будет показано в конце этого раздела, в IPv6, новой версии IP-протокола удалось полностью избавиться от фрагментации, оптимизировав обработку пакетов по IP-протоколу и снизив уязвимость протокола к атакам.

На сайте **tinyurl.com/2efayv** вы найдете апплет Java, генерирующий фрагменты. При работе с ним указываются размер входящей дейтаграммы, значение MTU и идентификатор входящей дейтаграммы. Апплет автоматически генерирует для вас фрагменты.

4.4.2. Адресация IPv4

Теперь давайте поговорим об адресации IPv4. Возможно, вы еще считаете, что адресация — незамысловатая тема. Надеемся, что к концу этой главы вы убедитесь не только в том, что, Интернет-адресация — тема вкусная, затейливая и интересная, но и в том, что она имеет огромное значение для всей Всемирной Паутины. Тема адресации IPv4 отлично рассмотрена в официальном руководстве компании 3Com Corporation[1] и в первой главе книги Стюарта[619].

Однако прежде, чем переходить к рассмотрению IP-адресации необходимо вкратце разобраться, как хосты и маршрутизаторы объединяются в сеть. Как правило, хост обладает всего одним каналом, который связывает его с сетью. Когда IP-адресу хоста требуется послать дейтаграмму, передача осуществляется именно через этот канал. Точка сопряжения между хостом и физическим каналом называется **интерфейсом**. Теперь давайте рассмотрим маршрутизатор и его интерфейсы.

Поскольку задача маршрутизатора — получить дейтаграмму с одного канала и переправить ее по другому, маршрутизатор обязательно связан с двумя или более каналами. Точка сопряжения маршрутизатора с любым из его каналов также называется «интерфейс». Соответственно, у маршрутизатора несколько интерфейсов, по одному на каждый входящий в него канал. Поскольку все хосты и маршрутизаторы способны посылать и получать IP-дейтаграммы, IP-протокол требует, чтобы у всех интерфейсов каждого хоста и маршрутизатора были собственные IP-адреса. Таким образом, технически IP-адрес ассоциирован с интерфейсом, а не с хостом или маршрутизатором, содержащим этот интерфейс.

Длина каждого IP-адреса составляет 32 бита (то есть 4 байта). Это означает, что всего могут существовать 2^{32} разнообразных IP-адресов. Аппроксимируя величину 2^{10} к 10^3, легко вычислить, что точное количество IP-адресов близко к 4 миллиардам. Как правило, эти адреса записываются в так называемом **точечно-десятичном формате**, где каждый байт адреса представляется в десятичном виде и отделяется точкой от других байтов. Рассмотрим, к примеру, IP-адрес 193.32.216.9. В нем 193 — это десятичный эквивалент первых 8 бит в адресе; 32 — десятичный эквивалент вторых 8 бит в адресе и т. д. Следовательно, адрес 193.32.216.9 будет иметь в двоичной системе следующий вид:

11000001 00100000 11011000 00001001

Каждый интерфейс на каждом хосте и маршрутизаторе во всем глобальном Интернете должен иметь уникальный IP-адрес (за исключением интерфейсов, расположенных за NAT (механизм преобразования сетевых адресов), который мы подробно рассмотрим в конце этого раздела). Адрес не может быть выбран произвольно. Часть IP-адреса будет определяться той подсетью, к которой подключен интерфейс.

На рис. 4.15 предоставлен пример IP-адресации и расположения интерфейсов. Здесь один маршрутизатор с тремя интерфейсами используется для взаимного соединения семи хостов. Обратите внимание на то, какие IP-адреса присвоены интерфейсам хостов и маршрутизатора, а именно на несколько вещей. Во-первых, у трех хостов, расположенных в верхней левой части рис. 4.15, и у интерфейса маршрутизатора, к которому они подключены, IP-адреса имеют вид 223.1.1.xxx. Кроме того, четыре интерфейса соединены друг с другом сетью, *не содержащей маршрутизаторов*. Эта система могла бы быть связана по локальной сети Ethernet, в случае чего взаимодействие между интерфейсами обес-

печивалось через Ethernet-коммутатор (об этом пойдет речь в главе 5), либо через беспроводную точку доступа (см. главу 6).

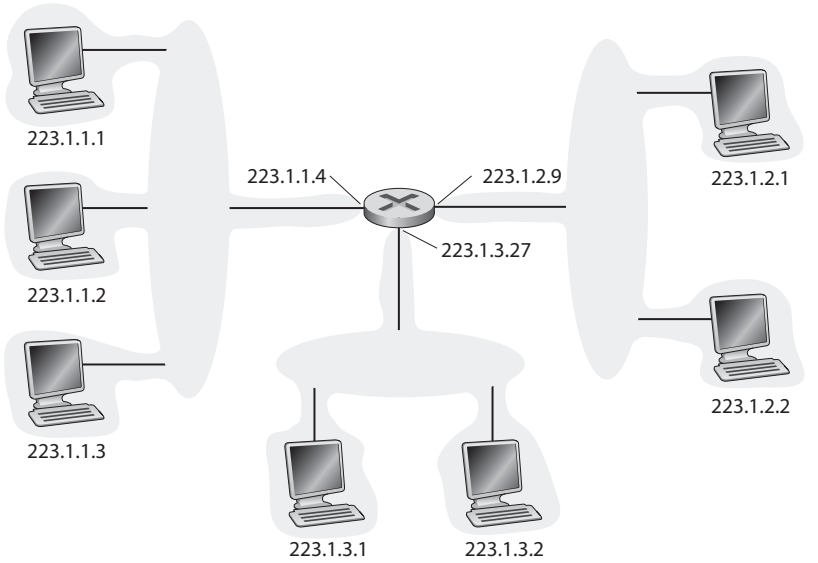


Рис. 4.15. Адреса интерфейсов и подсети

Пока представим сеть без маршрутизаторов, соединяющую указанные хосты, в виде «облачка» и отложим детальное изучение таких сетей до глав 5 и 6.

В контексте IP-протокола принято говорить, что подобная сеть, соединяющая три интерфейса хост-машин и один интерфейс маршрутизатора, представляет собой **подсеть**[426]. В технической литературе по Интернету подсеть также иногда называется *IP-сеть* или просто *сеть*. В ходе IP-адресации данной подсети присваивается адрес 223.1.1.0/24. Часть адреса /24, иногда называемая **маской подсети**, указывает, что 24 крайних левых разряда 32-разрядного значения определяют адрес подсети. Соответственно, подсеть 223.1.1.0/24 состоит из трех интерфейсов хостов (223.1.1.1, 223.1.1.2 и 223.1.1.3) и одного интерфейса маршрутизатора (223.1.1.4). Любые дополнительные хосты, подключаемые к подсети 223.1.1.0/24, *обязательно должны* будут иметь адреса вида 223.1.1.xxx. На рис. 4.15 мы видим еще две подсети: сеть 223.1.2.0/24 и подсеть 223.1.3.0/24. На рис. 4.16 подробнее показаны три IP-подсети, присутствующие на рис. 4.15.

IP-определение подсети не ограничено сегментами Ethernet, соединяющими множество хостов с интерфейсом маршрутизатора. Чтобы

составить впечатление об этой проблеме, рассмотрим рис. 4.17. Здесь показаны три маршрутизатора, все они соединены друг с другом по двухточечному принципу. У каждого маршрутизатора по три интерфейса: по одному для каждого двухточечного соединения и один — для широковещательного канала, который напрямую соединяет маршрутизатор с парой хостов. Какие подсети здесь есть? Их три: 223.1.1.0/24, 223.1.2.0/24 и 223.1.3.0/24, они напоминают подсети, которые мы видели на рис. 4.15. Но обратите внимание, что в данном примере есть и три дополнительные подсети. Одна из них, 223.1.9.0/24, включает интерфейсы, соединяющие маршрутизаторы M1 и M2; другая, 223.1.8.0/24 — интерфейсы, соединяющие маршрутизаторы M2 и M3. Наконец, третья подсеть 223.1.7.0/24 включает в себя интерфейсы, соединяющие маршрутизаторы M3 и M1.

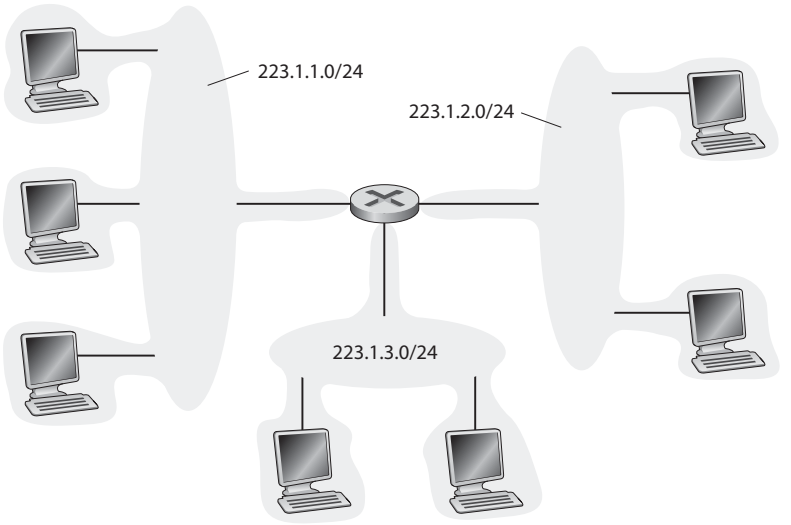


Рис. 4.16. Адреса подсетей

При работе с обычными взаимосвязанными системами из маршрутизаторов и хостов мы можем определять подсети в системе по следующему принципу:

Для определения границ подсетей открепите каждый интерфейс от его хоста или маршрутизатора, создав «островки» — изолированные сети, конечными точками которых являются интерфейсы. Каждая из таких изолированных сетей будет называться ***подсетью****.*

Если применить этот принцип к взаимосвязанной системе, изображенной на рис. 4.17, получим шесть «островков» или подсетей.

Из вышесказанного понятно, что такая организация сетей (например, в большой корпорации или вузе) со множеством Ethernet-сегментов и двухточечных соединений предполагает существование множества подсетей. У всех устройств, относящихся к какой-то из них, будет одинаковый адрес подсети. В принципе, адреса разных подсетей могут существенно отличаться, но на практике они обычно весьма схожи. Чтобы понять, почему так происходит, давайте рассмотрим, как адресация обрабатывается в Интернете на глобальном уровне.

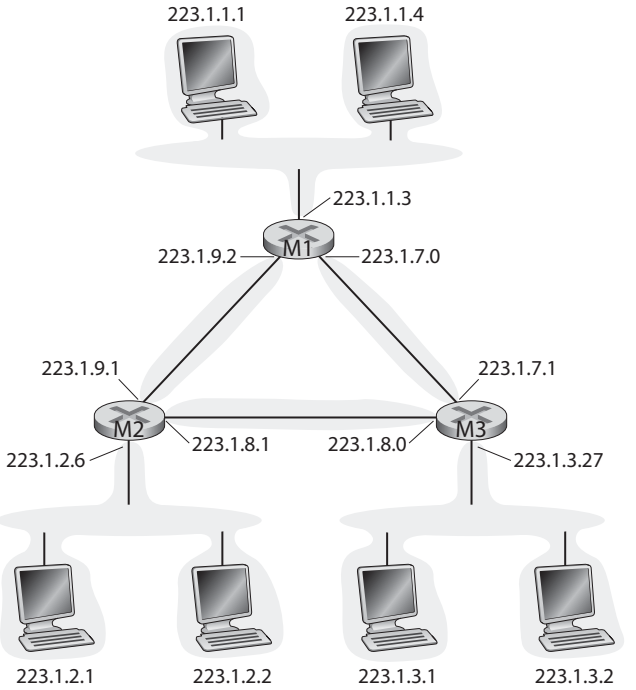


Рис. 4.17. Три маршрутизатора, объединяющие шесть подсетей

Стратегия присваивания адресов в Интернете называется **бесклассовая междоменная маршрутизация** (Classless Interdomain Routing, **CIDR**)[546]. Технология CIDR обобщает представления об адресации в подсетях. Как и при работе с подсетями 32-разрядный IP-адрес делится на две части и имеет точечно-десятичную форму представления *а.б.в.г/х*, где *х* указывает количество разрядов в первой части адреса.

Эти *х* старших разрядов адреса, записанного в форме *а.б.в.г/х*, составляют сетевую часть IP-адреса, которая часто именуется его **префиксом** (*сетевым префиксом*) адреса. Обычно организации присваивается

сплошной непрерывный блок адресов с общим префиксом (см. врезку «Принципы на практике»). В данном случае IP-адреса устройств во всей организации будут совместно использовать общий префикс. В разделе 4.6 мы обсудим действующий в Интернете протокол маршрутизации BGP и убедимся, что лишь эти лидирующие *x* разрядов префикса учитываются маршрутизаторами, работающими вне сети конкретной организации. Таким образом, если действующий вне организации маршрутизатор пересылает дейтаграмму на адрес, расположенный внутри организации, то такой внешний маршрутизатор будет учитывать лишь первые *x* разрядов адреса. Соответственно, размер таблицы маршрутизации в таких устройствах значительно уменьшается; ведь будет достаточно *всего одной* записи в формате *а.б.в.г/x*, чтобы направлять пакеты на *любой* адрес в пределах организации.

ПРИНЦИПЫ В ДЕЙСТВИИ

Ниже рассмотрен гипотетический Интернет-провайдер, подключающий к Интернету восемь организаций. Этот пример наглядно иллюстрирует, как правильно выделенные CIDR-образные адреса помогают оптимизировать маршрутизацию. Предположим (рис. 4.18), что Интернет-провайдер, которого мы условно назовем «Провайдер 1» объявляет всему миру, что нашей конторе следует посылать лишь такие дейтаграммы, первые 20 разрядов в адресе которых соответствуют шаблону 200.23.16.0/20. Таким образом, всему миру становится известно, что адресный блок 200.23.16.0/20 объединяет в себе восемь организаций, каждая из которых обладает собственной подсетью. Такая возможность использовать единственный префикс для объявления множества сетей часто именуется **агрегацией адресов** (или **агрегацией маршрутов**, или **суммированием маршрутов**).

Агрегация адресов работает превосходно, когда IP-адреса целыми блоками выделяются провайдерам Интернет-услуг, а провайдеры, в свою очередь, раздают эти адреса клиентским организациям. Но что делать, если распределение адресов не происходит иерархически? Что случится, если провайдер 1 приобретет компанию 2, сделав ее своим филиалом, а затем Организация 1 подключится к Интернету именно через компанию 2? На рис. 4.18 показано, что филиал компании 2 владеет блоком адресов 199.31.0.0/16, но, к сожалению, IP-адреса Организации 1 не относятся к этому блоку. Как решить подобную проблему? Разумеется, Организация 1 могла бы перенумеровать все свои маршрутизаторы и хосты таким образом, чтобы все новые адреса относились к адресному блоку компании 2.

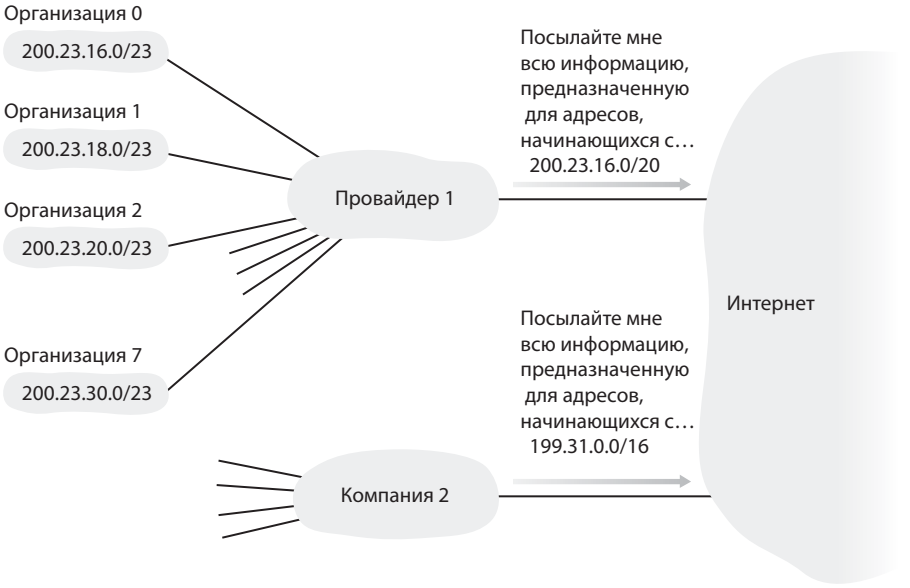


Рис. 4.18. Иерархическая адресация и агрегация маршрутов

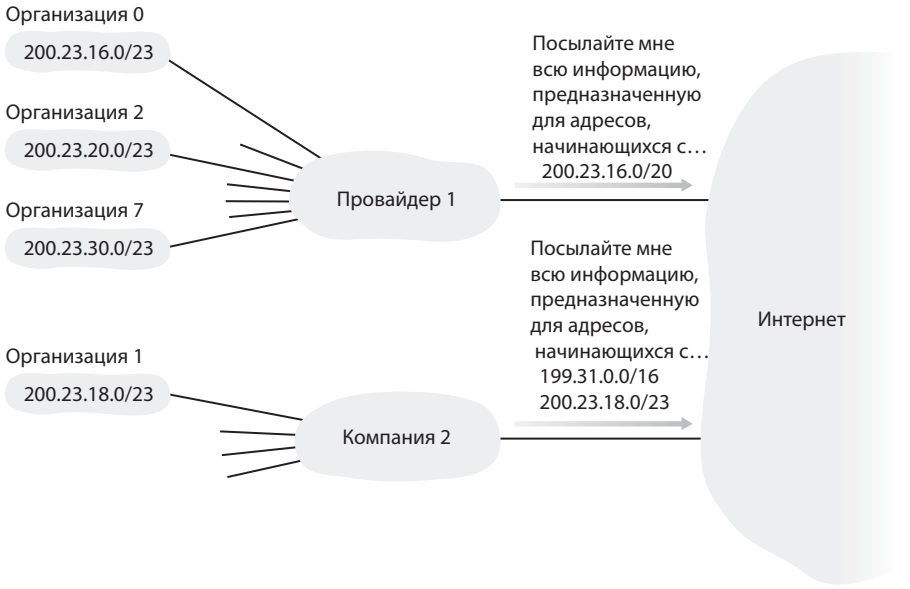


Рис. 4.19. Компания 2 предоставляет более точный маршрут к Организации 1

Однако это дорогостоящее решение, кроме того, не исключено, что в будущем обслуживание Организации 1 будет перепоручено другому филиалу провайдера 1. Как правило, реализуется вариант, при котором IP-адреса Организации 1 остаются в простран-

стве 200.23.18.0/23. В таком случае, как показано на рис. 4.19, провайдер 1 продолжает объявлять адреса из блока 200.23.16.0/20, а компания 2 — из блока 199.31.0.0/16. Однако компания 2 теперь *вдобавок* объявляет и те адреса, которые относятся к блоку Организации 1, то есть, 200.23.18.0/23. Когда другие маршрутизаторы в большом Интернете фиксируют блоки адресов 200.23.16.0/20 (Провайдер 1) и 200.23.18.0/23 (Компания 2), после чего им требуется переслать информацию на адрес из блока 200.23.18.0/23, такие маршрутизаторы будут искать совпадение наиболее длинного префикса (см. раздел 4.2.2) и направлять информацию в Компанию 2, так как эта компания объявляет самый длинный адресный префикс, максимально полно совпадающий с адресом назначения.

Соответственно, если маршрутизатор, расположенный за пределами организации, пересылает дейтаграмму, чей адрес назначения находится внутри организации, то при передаче требуется учитывать лишь *x* ведущих бит в адресе. В результате, размер таблицы перенаправления в таких маршрутизаторах значительно уменьшается, поскольку всего одной записи в формате *а.б.в.г/x* будет достаточно, чтобы направить пакеты по любому адресу в рамках организации.

Остаток 32-разрядного адреса можно использовать для идентификации разных устройств *внутри* организации, все они имеют одинаковый сетевой префикс. Именно эти разряды будут учитываться при пересылке пакетов на маршрутизаторах *внутри* организации. Такие разряды низшего порядка могут иметь (или не иметь) дополнительную структуру подсетей, подобную той, что рассматривалась выше. Предположим, например, что первый 21 разряд CIDR-адреса а.б.в.г/21 указывает сетевой префикс организации и, соответственно, является общим для всех IP-адресов у всех устройств в данной организации. Оставшиеся 11 разрядов далее идентифицируют конкретные хосты в рамках организации. Ее внутренняя структура может позволять использовать 11 крайних правых разрядов для построения подсетей в организации по вышеописанному принципу. Например, адрес а.б.в.г/24 будет относиться к конкретной подсети внутри организации.

До принятия стандарта CIDR длина сетевых фрагментов IP-адресов ограничивалась 8, 16 или 24 битами. Такая схема адресации называется **адресацией на основе классов**, где подсети с адресами в 8, 16 и 24 бит относились, соответственно, к классам 1, 2 и 3. Требование, в соответствии с которым подсетевая часть IP-адреса обязательно должна быть равна 1, 2 или 3 байт, оказалось довольно трудновыполнимым в услови-

ях, когда потребовалось поддерживать стремительно растущее количество организаций с небольшими и средними подсетями. Подсеть класса C (/24) могла вместить в себя лишь $2^8 - 2 = 254$ хоста ($2^8 = 256$, но два адреса резервируются для специального использования) — для многих организаций такое количество оказывается слишком малым. С другой стороны, сеть класса B (/16), включающая в себя до 65 634 адресов, будет слишком велика. При использовании адресации на основе классов организация с 2000 хостов обычно получала адрес подсети класса B (/16). В результате это привело к быстрому израсходованию адресных пространств класса B и нерациональному использованию уже присвоенных адресных пространств. Например, если организация имела всего 2000 хостов, но располагала подсетью класса B, допускающей наличие вплоть до 65 534 интерфейсов, то более 63 000 адресов оставались неиспользуемыми, но при этом не могли быть выделены другим организациям.

Наш рассказ был бы неполон без упоминания еще одного типа IP-адреса. Речь идет о широковещательном адресе 255.255.255.255. Если хост отправляет дейтаграмму на адрес 255.255.255.255, то сообщение приходит на все хосты, расположенные в данной подсети. Предусмотрен вариант, при котором маршрутизаторы пересылают такое сообщение и во все соседние подсети (хотя так обычно не делается).

Итак, мы подробно изучили IP-адресацию. Теперь давайте исследуем, откуда берутся адреса хостов и подсетей. Для начала рассмотрим, как организация получает блок адресов для своих устройств, затем поговорим о том, как конкретному устройству в организации (например, хосту) присваивается адрес из этого блока.

Получение блока адресов

Чтобы получить блок адресов для использования в подсети конкретной организации, администратор вычислительной сети может для начала связаться со своим провайдером Интернета. Этот провайдер выделит администратору набор адресов из более крупного блока, которым располагает сам. Например, конкретному провайдеру Интернета может быть выделен блок адресов 200.23.16.0/20. Провайдер, в свою очередь, может разделить этот большой блок на восемь непрерывных блоков меньшего размера и распределить их между восемью организациями, обслуживанием которых занимается. Для удобства читателя мы подчеркнули в каждом адресе ту часть, которая относится к подсети.

Блок адресов провайдера	200.23.16.0/20	11001000 00010111 00010000 00000000
Организация 0	200.23.16.0/23	11001000 00010111 00010000 00000000
Организация 1	200.23.18.0/23	11001000 00010111 00010010 00000000
Организация 2	200.23.20.0/23	11001000 00010111 00010100 00000000
...	...	...
Организация 7	200.23.30.0/23	11001000 00010111 00011110 00000000

Итак, можно получить адресное пространство в виде цельного блока, выделенного провайдером Интернет-услуг, но это не единственный способ. Разумеется, должна быть предусмотрена возможность получения адресов и для самого провайдера. Существует ли какой-нибудь глобальный орган, который несет полную ответственность за управление IP-адресами и за выдачу блоков адресов провайдерам Интернет-услуг и другим организациям? Конечно есть! Управление IP-адресами находится в компетенции ICANN — Интернет-корпорации по присвоению имен и номеров. Документ ICANN 2012[225] основан на рекомендациях, изложенных в стандарте RFC 2050. Роль некоммерческой организации ICANN, согласно документу NTIA 1998[373] Национального управления информации и телекоммуникаций США, заключается не только в распределении IP-адресов, но и в управлении корневыми серверами системы DNS. Кроме того, организация ICANN занимается очень деликатной работой по присвоению доменных имен и урегулированию споров, которые могут при этом возникать. Служба распределяет адреса между региональным Интернет-регистраторами, среди которых — организации ARIN (Северная Америка), RIPE (Европа, Ближний Восток, Центральная Азия), APNIC (Азия и Тихоокеанский регион), LACNIC (Латинская Америка и Карибский регион). Вместе все эти учреждения образуют Организацию поддержки адресов (ASO) в составе ICANN[28] и отвечают за распределение Интернет-адресов в своих регионах и за управление этими адресами.

Получение адреса хоста: протокол динамической конфигурации хостов

Когда организация получит блок адресов, она может далее присваивать отдельные IP-адреса хостам и интерфейсам маршрутизаторов в своей сети. Как правило, администратор вычислительной сети самостоятельно конфигурирует IP-адреса в маршрутизаторе (зачастую это делается удаленно, при помощи специального сетевого инструмента).

Адреса хостов также можно сконфигурировать вручную, но обычно эта задача решается при помощи **протокола динамического конфигурирования хостов** (Dynamic Host Configuration Protocol, **DHCP**)[466]. Протокол DHCP позволяет хосту автоматически получать IP-адрес. Администратор вычислительной сети может настроить DHCP так, что конкретный хост будет получать один и тот же IP-адрес всякий раз при подключении к сети. Альтернативный вариант предусматривает присваивание хосту **временного IP-адреса**, который будет меняться при каждом подключении хоста к сети. Наряду с присваиванием IP-адреса хосту, протокол DHCP позволяет получать дополнительную информацию, в частности: маску подсети, адрес ее первого маршрутизатора на пути доставки (first-hop router), также нередко называемого «шлюз по умолчанию» (default gateway) и адрес локального DNS-сервера.

Поскольку протокол DHCP позволяет автоматизировать связанные с собственно сетью аспекты включения хоста в сеть, этот протокол зачастую называют **plug-and-play (самонастраивающимся)**. Такая возможность делает его *исключительно* привлекательным для администратора вычислительной сети, которому в противном случае пришлось бы выполнять все задачи такого рода вручную! Кроме того, DHCP очень широко используется как в стационарных сетях доступа к Интернету, так и в беспроводных локальных сетях, где хосты часто подключаются к сети и не менее часто из нее выходят. Предположим, есть студент, который выходит в Интернет с ноутбука. Он постоянно носит его с собой и подключается к сети то из общежития, то из библиотеки, то из аудитории. Вполне вероятно, что в каждом из этих мест студент будет подключаться к новой подсети; соответственно, и в общежитии, и в библиотеке, и в аудитории ему понадобится новый IP-адрес. Протокол DHCP идеально подходит для такой ситуации, поскольку в упомянутых помещениях постоянно сменяются пользователи, а выделяемый IP-адрес нужен каждому из них в течение ограниченного времени. Схожим образом протокол DHCP применяется различными провайдерами Интернета и в сетях стационарного доступа к Интернету. Рассмотрим, например, такого провайдера, который обслуживает 2000 абонентов, но одновременно в сеть выходят не более 400 из них. В таком случае нет необходимости в блоке из 2048 адресов; DHCP-сервер, который динамически присваивает пользователям адреса, может работать с блоком всего из 512 адресов (например, этот блок может иметь вид а.б.в.г/23). По мере подключения хостов к сети и отключения их DHCP-серверу требуется обновлять свой список доступных IP-адресов. Всякий раз, когда в сеть входит новый хост, DHCP-сервер выделяет ему произволь-

ный адрес, взятый из пула тех адресов, которые свободны в настоящий момент. При выходе хоста из сети тот адрес, который ему принадлежал, возвращается в пул.

DHCP — это клиент-серверный протокол. Клиент — это, как правило, новоприбывший хост, которому требуется получить информацию о конфигурации сети, в частности, IP-адрес для этого хоста. В простейшем случае каждая подсеть (в контексте адресации, см. рис. 4.17) будет иметь свой DHCP-сервер. Если в подсети нет отдельного DHCP-сервера, то его заменяет агент-ретранслятор DHCP (как правило, это маршрутизатор), которому известен (на случай необходимости) адрес DHCP-сервера данной сети. На рис. 4.20 показан DHCP-сервер, прикрепленный к подсети 223.1.2/24. Здесь маршрутизатор выступает в качестве агента-ретранслятора для подключающихся клиентов, относящихся к подсетям 223.1.1/24 и 223.1.3/24. В следующих разделах предполагается, что в подсети имеется свой DHCP-сервер.

Для новоприбывшего хоста DHCP-протокол выполняется в четыре этапа, как показано на рис. 4.21. Соответствующая конфигурация сети представлена на рис. 4.20. На данном рисунке значение «Ваш адрес» указывает адрес, выделяемый новоприбывшему клиенту.

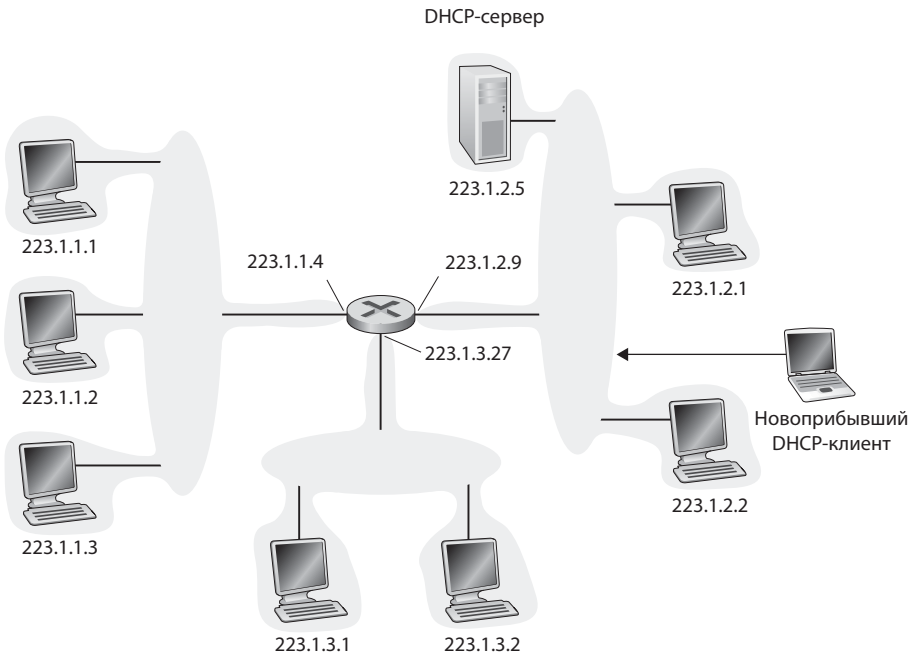


Рис. 4.20. Клиент-серверный сценарий с применением DHCP

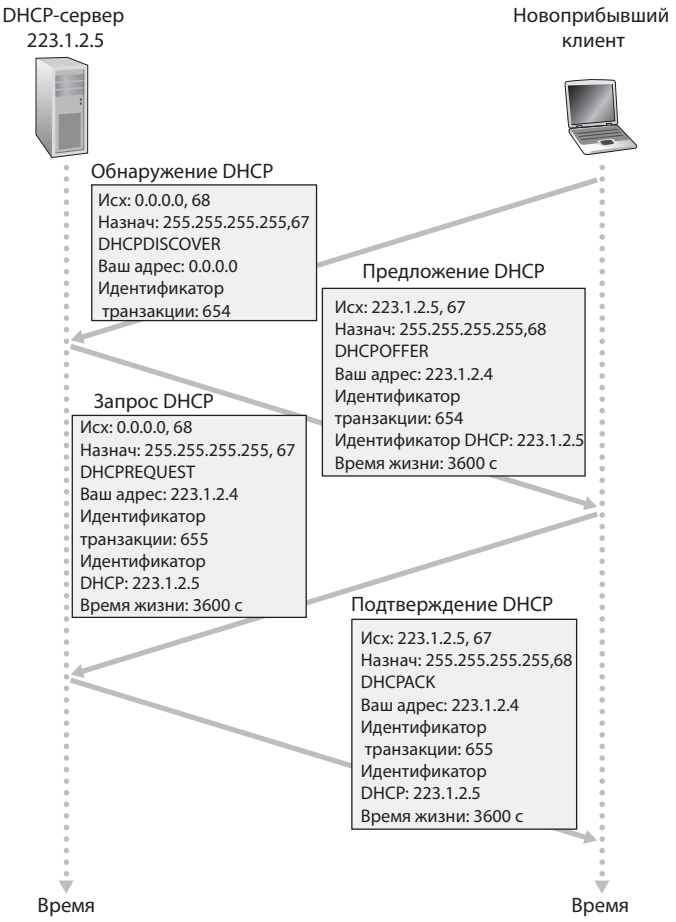


Рис. 4.21. Взаимодействие клиента и сервера по протоколу DHCP

Ниже представлены четыре этапа работы с протоколом DHCP.

- *Обнаружение DHCP-сервера*. Первым делом новоприбывший хост должен найти тот DHCP-сервер, с которым он будет взаимодействовать. Это делается при помощи **сообщения обнаружения DHCP**. Это сообщение клиент отсылает в UDP-пакете на порт 67. UDP-пакет инкапсулируется в IP-дейтаграмму. Но куда должна быть отправлена эта дейтаграмма? Хост не знает даже IP-адреса той сети, к которой он подключен, тем более адреса DHCP-сервера этой сети. В такой ситуации DHCP-клиент создает IP-дейтаграмму с сообщением обнаружения DHCP, широковещательным IP-адресом получателя 255.255.255.255 и IP-адресом отправителя «этот хост» (0.0.0.0). DHCP-клиент передает IP-дейтаграмму на канальный уровень, ко-

торый затем широковещательно рассылает данный кадр на все узлы, входящие в подсеть (мы подробно рассмотрим широковещательную передачу на канальном уровне в разделе 5.4).

- *Предложение(я) DHCP-сервера.* DHCP-сервер, получающий сообщение обнаружения, отсылает клиенту в ответ на него **сообщение с предложением DHCP**, которое широковещательно передается всем узлам подсети. Опять же, для этого используется IP-адрес 255.255.255.255. (предлагаю подумать: а почему этот серверный отклик также нужно передавать широковещательным образом?) Поскольку в подсети могут находиться сразу несколько DHCP-серверов, клиент может оказаться в завидном положении, позволяющем выбрать наиболее подходящий сервер. Каждое сообщение с предложением сервера содержит транзакционный идентификатор полученного сообщения обнаружения, предлагаемый клиенту IP-адрес, маску подсети и **время аренды адреса** — период, в течение которого данный IP-адрес будет валиден. Как правило, сервер предоставляет время аренды от нескольких часов до нескольких дней[146].
- *DHCP-запрос.* Новоприбывший клиент получает серверное предложение или выбирает одно из имеющихся. В качестве отклика на выбранное предложение клиент отсылает **сообщение запроса DHCP**, передавая вместе с ним параметры конфигурации.
- *DHCP-подтверждение (ACK).* Сервер реагирует на сообщение запроса DHCP своим **сообщением подтверждения запроса (ACK)**, подтверждая запрошенные параметры.

После того как клиент получит DHCP ACK, взаимодействие с сервером завершено, и клиент может использовать IP-адрес, выделенный по протоколу DHCP, на протяжении установленного времени аренды. Поскольку пользователю может потребоваться этот адрес и после истечения срока аренды, в протоколе DHCP также предусмотрен механизм, позволяющий обновить аренду.

Ценность подхода «подключи и работай» при работе с DHCP очевидна, учитывая тот факт, что альтернативный вариант требует вручную сконфигурировать IP-адрес хоста. Вернемся к примеру со студентом, который переходит с ноутбуком из общежития в библиотеку, из библиотеки в аудиторию и всякий раз подключается к новой подсети — соответственно, получает новый IP-адрес. Просто невозможно себе представить, как администратор вычислительной сети в принципе мог бы сам переконфигурировать ноутбуки в каждой из таких новых точек. Кроме

того, лишь немногие студенты (кроме тех, кто изучал компьютерные сети) обладают достаточным опытом, чтобы вручную сконфигурировать собственный ноутбук. В мобильных сетях, однако, DHCP имеет ряд недостатков. Поскольку DHCP выдает новый IP-адрес всякий раз, когда узел подключается к новой подсети, мы не можем поддерживать TCP-соединение с удаленным приложением, ведь узел переходит из одной подсети в другую. В главе 6 мы поговорим о мобильных IP — сравнительно новом усовершенствовании IP-инфраструктуры, позволяющем мобильному узлу использовать единый перманентный адрес даже при перемещении между подсетями. Дополнительные подробности о протоколе DHCP изложены в книге Дромса[146] и публикации Dynamic Host Configuration[130]. Свободно распространяемая справочная реализация протокола DHCP предоставляется Консорциумом Интернет-систем[245].

Трансляция сетевых адресов

Итак, мы поговорили об адресах Интернета и о формате дейтаграмм IPv4. Очевидно, что для каждого устройства, поддерживающего связь по IP, требуется свой IP-адрес. При повсеместном распространении подсетей для малых офисов и домашних офисов (SOHO) такая необходимость, казалось бы, подразумевает: всякий раз, когда в офисе формата SOHO нужно настроить локальную сеть для соединения множества устройств, нам не обойтись без диапазона адресов, который мы должны получить от Интернет-провайдера для всех наших машин. Если подсеть начнет расти (например, у наших детей появятся не только собственные ПК, но и смартфоны, и консоли Game Boy с выходом в сеть), то нам придется расширить имеющийся блок адресов. Что же делать, если наш провайдер уже раздал все непрерывные участки того адресного пространства, в котором работает сеть нашего офиса SOHO? Более того, разве средний домовладелец умеет (или готов научиться) управлять IP-адресами своей домашней сети? К счастью, существует более простой способ выделения адресов, который все шире распространяется именно в таких ситуациях, как описанная в этом разделе. Этот подход называется **трансляцией сетевых адресов** (network address translation, **NAT**)[484, 491, 684].

На рис. 4.22 продемонстрирована работа маршрутизатора с поддержкой NAT. Такой маршрутизатор, установленный дома, имеет интерфейс, входящий в состав домашней сети (на рис. 4.22 она изображена справа). Адресация в домашней сети организуется точно так, как и в предыдущих примерах — все четыре интерфейса домашней сети имеют одинаковый

адрес подсети: 10.0.0/24. Адресное пространство 10.0.0.0/8 представляет собой одну из трех частей пространства IP-адресов, зарезервированного для частных сетей согласно стандарту RFC 1918[456]. В данном случае мы имеем **зону** с частными адресами, как в домашней сети на рис. 4.22. *Область частных адресов* — это сеть, чьи адреса являются значимыми лишь для устройств, расположенных в ее пределах. Чтобы понять важность такой организации, обратим внимание на следующий факт: в мире существуют тысячи домашних сетей, причем многие из них используют одно и то же адресное пространство, 10.0.0.0/24.

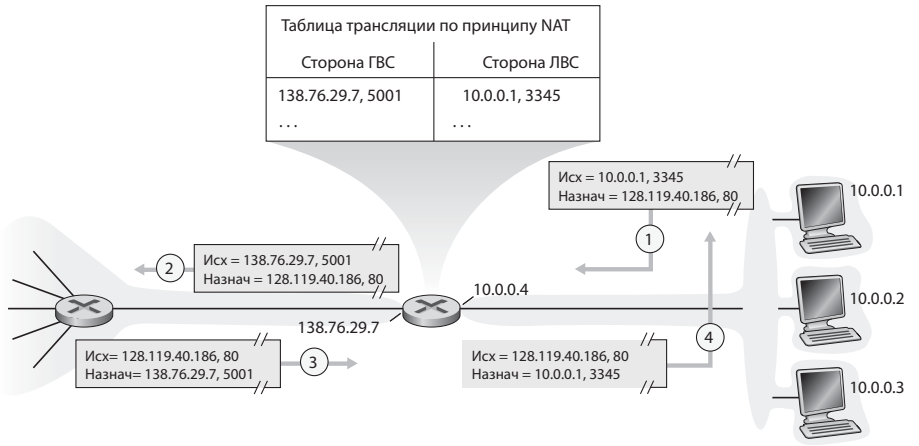


Рис. 4.22. Трансляция сетевых адресов

Устройства в рамках конкретной домашней сети могут обмениваться пакетами при помощи адресации 10.0.0.0/24. Однако те пакеты, которые направляются *за пределы* локальной сети, в глобальный Интернет, разумеется, не могут использовать эти адреса (ни как исходные, ни как целевые), поскольку с ними работают сотни и даже тысячи сетей. Таким образом, адреса из пространства 10.0.0.0/24 могут иметь значение лишь в рамках конкретной домашней сети. Но если частные адреса не имеют значения за пределами домашней сети, как же обрабатывается адресация при получении пакетов из глобального Интернета или при отправке их туда — ведь в Интернете все адреса должны быть уникальны? Чтобы дать ответ на этот вопрос, требуется понимать механизм NAT.

Маршрутизатор с поддержкой NAT не *похож* на маршрутизатор, связывающий сеть с внешним миром. Для внешнего мира маршрутизатор с поддержкой NAT выглядит как *единое* устройство с всего *одним* IP-адресом. На рис. 4.22 весь трафик, исходящий с домашнего маршрути-

затора в глобальный Интернет имеет IP-адрес отправителя 138.76.29.7, а весь трафик, входящий в сеть через этот маршрутизатор, должен иметь адрес получателя 138.76.29.7. В сущности, маршрутизатор с поддержкой NAT скрывает от внешнего мира детали домашней сети. (Кстати, возможно, вы уже заинтересовались, где компьютеры из домашней сети берут себе адреса, и где домашний маршрутизатор берет свой единый IP-адрес. Зачастую ответ одинаков — DHCP! Маршрутизатор получает свой IP-адрес с DHCP-сервера провайдера Интернета, а сам маршрутизатор содержит DHCP-сервер для предоставления адресов компьютерам в адресном пространстве домашней сети, управляемой при помощи NAT-DHCP-маршрутизатора).

Если все дейтаграммы, прибывающие на NAT-маршрутизатор из глобальной сети, имеют один и тот же IP-адрес получателя (а именно — адрес интерфейса, расположенного на стыке глобальной сети и NAT-маршрутизатора), то как же маршрутизатор узнает внутренний хост, на который он должен переслать конкретную дейтаграмму? Для этого на NAT-маршрутизаторе применяется **таблица трансляции сетевых адресов**. В записи этой таблицы включаются IP-адреса и номера портов.

Вернемся к примеру на рис. 4.22. Допустим, пользователь работает в домашней сети на хосте 10.0.0.1 и запрашивает веб-страницу с какого-либо веб-сервера (порт 80) с IP-адресом 128.119.40.186. Хост 10.0.0.1 присваивает (произвольно) номер исходного порта 3345 и посылает дейтаграмму в локальную сеть. NAT-маршрутизатор получает дейтаграмму, генерирует для нее новый номер исходного порта, на этот раз 5001, заменяет исходный IP-адрес соответствующим IP-адресом, расположенным на стороне ГВС (138.76.29.7) и заменяет старый номер исходного порта 3345 новым — 5001. При генерировании нового номера исходного порта NAT-маршрутизатор может выбрать любой, которого пока нет в таблице трансляции сетевых адресов. (Обратите внимание: поскольку поля номеров портов 16-разрядные, протокол NAT способен поддерживать более 60 000 одновременных соединений, обходясь при этом единственным IP-адресом на стыке маршрутизатора и ГВС!) Механизм NAT в маршрутизаторе также добавляет запись в свою таблицу трансляции сетевых адресов. Веб-сервер, совершенно не представляющий, что прибывшая к нему дейтаграмма с HTTP-запросом уже подверглась обработке на NAT-маршрутизаторе, посылает в ответ дейтаграмму, где адрес получателя — это IP-адрес NAT-маршрутизатора, а порт назначения имеет номер 5001. Когда дейтаграмма прибывает на NAT-

маршрутизатор, тот делает выборку из таблицы трансляции сетевых адресов. При этом он использует целевой IP-адрес и номер порта назначения, чтобы получить подходящий IP-адрес (10.0.0.1) и номер порта назначения (3345) для браузера, работающего в домашней сети. Затем маршрутизатор переписывает адрес назначения дейтаграммы и номер порта назначения и пересылает ее в домашнюю сеть.

В последние годы технология NAT получила широкое распространение. Однако необходимо отметить, что в сообществе IETF есть немало консерваторов, решительно критикующих NAT. Во-первых, говорят они, номера портов должны использоваться для идентификации процессов, а не хостов. (Действительно, такое отклонение от правил может вызывать проблемы в работе серверов, установленных в одной и той же домашней сети — ведь, как вы помните из главы 2, серверные процессы ожидают входящие запросы на портах с хорошо известными номерами, которые применяются именно для этой цели.) Во-вторых, указывают критики, маршрутизаторы предназначены для обработки пакетов лишь до уровня 3 включительно. В-третьих, протокол NAT нарушает так называемый «тезис о сквозном взаимодействии», согласно которому хосты должны обмениваться информацией непосредственно друг с другом, без посредников, изменяющих IP-адреса и номера портов. Наконец, по мнению тех же критиков, мы должны работать с системой IPv6 для решения проблемы с дефицитом IP-адресов, а не удовлетворяться временными вариантами-затычками вроде NAT. Но как бы мы к этому ни относились, технология NAT становится важной составляющей Интернета.

Еще одна серьезная проблема, связанная с NAT, заключается в следующем: эта технология мешает работе одноранговых приложений, в частности, приложений для обмена файлами и для голосовой связи по IP. Как вы помните из главы 2, в одноранговом приложении любой участник (пир) А должен иметь возможность инициировать TCP-соединение с любым пиром Б. Суть проблемы заключается в том, что если Б окажется за границей NAT, то он не сможет выступить в роли сервера и принимать TCP-соединения. В одной из задач для домашней работы будет показано, что проблему с NAT можно обойти, если пир А не находится за NAT. В таком случае пир А может сначала связаться с пиром Б через посредника, пира В. Пир В должен работать за пределами NAT, а также иметь установленное TCP-соединение с пиром Б. В таком случае пир А может через пира В запросить пира Б, чтобы тот напрямую установил соединение с пиром А. Как только между пирами А и Б будет установ-

лено прямое одноранговое TCP-соединение, два этих участника смогут обмениваться сообщениями или файлами. Такой прием называется **реверсом соединения** и действительно используется многими одноранговыми приложениями для **обхода NAT**. Если оба пира, А и Б, находятся каждый за своей границей NAT, ситуация несколько осложняется, но все равно решается при помощи ретрансляции приложений, как было показано в главе 2 на примере с ретрансляцией Skype.

Протокол UPnP

Обход NAT все чаще осуществляется при помощи протокола UPnP для универсальной автоматической настройки сетевых устройств. Этот протокол позволяет хосту обнаружить и сконфигурировать близлежащий NAT-маршрутизатор[642]. UPnP требует, чтобы и хост, и NAT были совместимы с протоколом UPnP. При его использовании UPnP приложение, работающее на хосте, может затребовать NAT-отображения следующей информации: (*частный IP-адрес, частный номер порта*) и (*общедоступный IP-адрес, общедоступный номер порта*) для того порта, номер которого мы запросили. Если NAT принимает запрос и создает отображение, то внешние узлы смогут инициировать TCP-соединение с (*общедоступный IP-адрес, общедоступный номер порта*). Более того, протокол UPnP позволяет приложению узнать значение (*общедоступный IP-адрес, общедоступный номер порта*), так, что приложение сможет объявить эту информацию всему окружающему миру.

В качестве примера предположим, что наш хост, расположенный за границей NAT с поддержкой протокола UPnP, имеет частный адрес 10.0.0.1. На порте 3345 этого хоста работает BitTorrent. Кроме того, предположим, что общедоступный адрес этой системы NAT — 138.76.29.7. Естественно, нашему приложению BitTorrent требуется возможность приема соединений с других хостов, чтобы оно могло обмениваться с ними торрент-фрагментами. Для этого приложение BitTorrent у вас на хосте приказывает NAT создать «окно», которое будет отображать (10.0.0.1, 3345) на (138.76.29.7, 5001). Номер общедоступного порта 5001 выбирается приложением. Приложение BitTorrent у вас на хосте также может объявить своему трекеру, что оно доступно по адресу 138.76.29.7, 5001. Внешний хост может послать пакет TCP SYN на адрес 138.76.29.7, 5001. Когда механизм NAT получит пакет SYN, он изменит целевой IP-адрес и номер порта для этого пакета на 10.0.0.1, 3345, после чего будет выполнена переадресация пакета через NAT.

Итак, протокол UPnP позволяет внешним хостам инициировать сеансы связи с хостами, работающими через механизм NAT, при этом используются протоколы TCP или UDP. Границы NAT давно считаются настоящим бичом одноранговых приложений. Протокол UPnP, предоставляющий надежное и эффективное решение для обхода NAT, порой может оказаться спасительным. На этом мы вынуждены закончить наше краткое обсуждение технологий NAT и UPnP. Более подробно механизм NAT рассмотрен в работе Хьюстона[215] и публикации How NAT Works[103].

4.4.3. Протокол управляющих сообщений Интернета

Как вы помните, в состав сетевого уровня Интернета входят три основных компонента: протокол IP, рассмотренный в предыдущем разделе, Интернет-протоколы маршрутизации (в частности, RIP, OSPF и BGP), которым будет посвящен раздел 4.6, и, наконец, протокол ICMP — он нем пойдет речь здесь.

Протокол ICMP, спецификация которого изложена в стандарте RFC 792[420], используется хостами и маршрутизаторами для обмена информацией сетевого уровня друг с другом. Чаще всего протокол ICMP применяется для передачи сообщений об ошибках. Например, если у нас работает сеанс Telnet, FTP или HTTP, то мы можем получить сообщение об ошибке «Destination network unreachable» («Сеть назначения недостижима»). Это сообщение относится именно к протоколу ICMP. В какой-то момент IP-маршрутизатор не смог найти путь к хосту, указанному в вашем приложении Telnet, FTP или HTTP. Тогда маршрутизатор создал и отослал вашему хосту сообщение типа 3 протокола ICMP, указав в сообщении эту ошибку.

ICMP зачастую считается частью протокола IP, однако с архитектурной точки зрения он располагается прямо над IP, поскольку сообщения ICMP передаются внутри IP-дейтаграмм. Таким образом, IP переносит сообщения ICMP в качестве своей полезной нагрузки так же, как он переносит сегменты TCP или UDP. Аналогично, если хост получает IP-дейтаграмму, для которой ICMP указан в качестве протокола верхнего уровня, IP демультиплексирует содержимое дейтаграммы в ICMP, точно так же, как он мог бы демультиплексировать эту информацию в TCP или UDP.

Сообщения ICMP имеют тип и поле с кодом, а также содержат заголовок и первые 8 байт той IP-дейтаграммы, из-за которой и было

сгенерировано ICMP-сообщение (таким образом, отправитель сможет определить, какая дейтаграмма вызвала ошибку). Некоторые типы ICMP-сообщений показаны на рис. 4.23. Обратите внимание: ICMP-сообщения используются не только для сигнализации о возникновении ошибок.

Тип ICMP	Код	Описание
0	0	Эхо-отклик (на команду `ping`)
3	0	Сеть назначения недостижима
3	1	Хост назначения недостижим
3	2	Протокол назначения недостижим
3	3	Порт назначения недостижим
3	6	Сеть назначения неизвестна
3	7	Хост назначения неизвестен
4	0	Подавление источника (борьба с перегрузками)
8	0	Эхо-запрос
9	0	Объявление маршрутизатора
10	0	Обнаружение маршрутизатора
11	0	Предписанное время жизни истекло
12	0	Недопустимый IP-заголовок

Рис. 4.23. Типы сообщений протокола ICMP

Хорошо известная программа ping посылает ICMP-сообщение типа 8 с кодом 0 на указанный хост. Хост назначения, получив этот эхо-запрос, посылает в ответ сообщение ICMP типа 0 с кодом 0. Большинство реализаций протокола TCP/IP поддерживают сервер `ping` непосредственно в операционной системе, то есть сервер не является процессом. В главе 11 книги Стивенса[616] приводится исходный код клиентской программы ping. Обратите внимание: клиентская программа должна иметь возможность проинструктировать операционную систему, что той следует сгенерировать сообщение ICMP типа 8 с кодом 0.

Еще одно интересное сообщение ICMP — это «подавление источника». Оно редко применяется на практике. Изначально это сообщение предназначалось для контроля над перегрузками — если маршрутизатор начинает пробуксовывать, то он может послать на хост — источник трафика — по протоколу ICMP такое сообщение и тем самым принудительно снизить скорость передачи данных с хоста. В главе 3 мы узнали, что

протокол TCP обладает собственным механизмом борьбы с перегрузками, который работает на транспортном уровне. Таким образом, при борьбе с перегрузками обратной связи на сетевом уровне, в частности, отпадает необходимость применять сообщения о подавлении источника по протоколу ICMP.

В главе 1 мы познакомились с программой Traceroute, которая позволяет отслеживать маршрут с хоста к любому другому хосту в мире. Интересно, что программа реализуется с применением сообщений ICMP. Для определения имен и адресов маршрутизаторов между исходным и конечным хостами программа Traceroute на исходном хосте посылает на хост назначения серию обычных IP-дейтаграмм. Каждая из этих дейтаграмм содержит UDP-сегмент с маловероятным номером UDP-порта. Первая имеет предписанное время жизни (TTL) равное 1 «переходу», вторая — 2 «переходам», третья — 3 «переходам» и т. д.

Кроме того, исходный хост запускает таймеры для каждой из дейтаграмм. Когда *n*-ная дейтаграмма прибывает на *n*-ный маршрутизатор, он наблюдает, что предписанное время жизни этой дейтаграммы только что истекло. По правилам протокола IP маршрутизатор избавляется от дейтаграммы и посылает исходному хосту предупредительное сообщение ICMP (тип 11, код 0). В этом сообщении содержится имя маршрутизатора и его адрес. Когда оно прибывает на исходный хост, тот определяет по таймеру время, затраченное на путь в оба конца, а также имя и IP-адрес *n*-ного маршрутизатора (эту информацию он получает из ICMP-сообщения).

Как программа Traceroute с исходного хоста узнает, когда следует прекратить отсылать UDP-сегменты? Вспомните о том, что хост-отправитель постепенно увеличивает значение в поле TTL (предписанное время жизни) для каждой последующей отсылаемой дейтаграммы. Соответственно, рано или поздно одна из дейтаграмм попадет на хост-получатель. Поскольку эта дейтаграмма содержит UDP-сегмент с маловероятным номером порта, хост-получатель посылает в ответ источнику ICMP-сообщение «порт недостижим» (тип 3 код 3). Когда исходный хост получает именно это ICMP-сообщение, он «делает вывод», что больше не требуется отсылать дополнительные зондирующие пакеты. (Кстати, стандартная программа Traceroute отсылает наборы из трех пакетов, имеющих одинаковое значение TTL; соответственно, вывод Traceroute содержит по три результата для каждого из значений TTL.)

О БЕЗОПАСНОСТИ

Инспектирование дейтаграмм: брандмауэры и системы обнаружения вторжений

Допустим, вам поручено администрировать сеть — домашнюю, ведомственную, университетскую или корпоративную. Злоумышленники, зная диапазон IP-адресов вашей сети, вполне могут отсылать на них дейтаграммы. Эти дейтаграммы могут совершать всевозможные вредные операции, в частности выявлять топологию вашей сети при помощи эхо-тестирования адресов и сканирования портов, атаковать уязвимые хосты искаженными пакетами, наводнять серверы потоками ICMP-пакетов, инфицировать хосты, включая в пакеты вредоносное ПО. Что вы, администратор вычислительной сети, можете противопоставить всем этим злоумышленникам, каждый из которых может посылать в вашу сеть вредоносные пакеты? Два популярных механизма защиты, позволяющих предохранить сеть от атак вредоносными пакетами — это брандмауэры и системы обнаружения вторжения (IDS).

Вы как администратор вычислительной сети, вероятно, первым делом попробуете установить брандмауэр (сетевой экран) между вашей сетью и Интернетом. Большинство современных маршрутизаторов доступа оснащены функцией брандмауэра. Он анализирует дейтаграммы и сегментирует поля заголовков, предотвращая проникновение подозрительных дейтаграмм во внутреннюю сеть. Например, он может быть сконфигурирован так, что будет блокировать все ICMP-пакеты, содержащие эхо-запросы. Это лишит злоумышленника возможности выполнить в вашем диапазоне IP традиционное эхо-тестирование адресов. Брандмауэры также могут блокировать пакеты в зависимости от их исходного и конечного IP-адресов и номеров портов. Кроме того, они могут иметь специальную конфигурацию для отслеживания TCP-соединений и допускать в сеть только те дейтаграммы, которые получены через доверенные соединения.

Дополнительная защита обеспечивается при помощи систем обнаружения вторжений (IDS). Система IDS, которая обычно располагается на границе сети, выполняет «глубокую проверку пакетов», при которой учитываются не только поля заголовков, но и поля данных дейтаграмм (включая данные прикладного уровня). В системе IDS есть база данных сигнатур тех пакетов, при помощи которых совершались известные ей атаки. База данных автоматически обновляется по мере обнаружения новых атак. Когда пакеты проходят через систему IDS, система пытается сопоставить их поля заголовков и данных с теми сигнатурами, которые перечислены в ее базе данных. Как только фиксируется такое совпадение, система генерирует

предупредительное сообщение. Существуют также системы предотвращения вторжений (IPS), подобные IDS, с той оговоркой, что система IPS не только генерирует предупреждения, но и блокирует подозрительные пакеты. В главе 8 мы более подробно поговорим о брандмауэрах и системах обнаружения вторжений.

Могут ли брандмауэры и системы обнаружения вторжений полностью защитить вашу систему от всех возможных атак? Разумеется, нет, поскольку злоумышленники постоянно изобретают новые атаки, сигнатуры которых еще не зафиксированы. Но брандмауэры и традиционные системы IDS, работающие на основе сравнения сигнатур, хорошо защищают системы от известных атак.

Таким образом, исходный хост узнает, сколько маршрутизаторов находится между ним и конечным хостом, а также уникальные идентификаторы этих маршрутизаторов и время, затрачиваемое на путь туда и обратно между двумя хостами. Обратите внимание: клиентская программа Traceroute должна иметь возможность потребовать от операционной системы генерировать UDP-дейтаграммы с конкретными значениями TTL (предписанного времени жизни), а также получать от своей операционной системы уведомления о поступлении сообщений протокола ICMP. Теперь, разобравшись в принципах работы программы Traceroute, вам, возможно, захочется вернуться назад и еще немного поэкспериментировать с ней.

4.4.4. IPv6

В начале 1990-х организация IETF (Инженерный совет Интернета) приступила к разработке протокола, призванного заменить IPv4. Основным мотивом для этой работы стало растущее понимание того, что 32-битное пространство IP-адресов достаточно скоро будет израсходовано в силу распространения новых подсетей и IP-узлов, подключаемых к Интернету. Присвоение новых уникальных IP-адресов становилось невероятно стремительным. Чтобы удовлетворить острую потребность в них, был разработан новый протокол — IPv6. Создатели IPv6 также не преминули усовершенствовать и дополнить другие аспекты протокола IPv4, опираясь на богатый накопленный опыт его эксплуатации.

Ведутся активные споры относительно того, когда все адреса IPv4 будут израсходованы (и, соответственно, исчезнет возможность подключения новых сетей к Интернету). Выводы двоих ведущих специали-

стов рабочей группы IETF по оценке времени, оставшегося до окончательного истощения адресного пространства, указывали на 2008 либо на 2018 год соответственно[609]. В феврале 2011 года организация IANA (Администрация адресного пространства Интернета) выделила последний пул не присвоенных адресов IPv4 региональному регистратору. В распоряжении региональных Интернет-регистраторов по-прежнему есть свободные адреса IPv4, но когда все они будут израсходованы, больше не останется ни одного блока адресов, который можно было бы дополнительно выделить из центрального пула[218]. Хотя в середине 1990-х считалось, что полное истощение адресного пространства IPv4 произойдет еще достаточно нескоро, разработчики в то же время, понимали, что на разработку новой технологии такого масштаба потребуется очень значительное время. Поэтому был запущен проект «Новое поколение IP» (IPng)[62; 455]. Результатом этих разработок стало создание спецификации протокола IP версии 6 (IPv6)[477], о которой мы подробно поговорим ниже. Здесь часто задают вопрос: а почему же так и не появился протокол IPv5? Первоначально предполагалось, что роль IPv5 уготована протоколу ST-2, но позже от ST-2 было решено отказаться. Полезные сведения о IPv6 вы подчерпнете из книги Хайтема[213] и сайта IPv6.com[244].

Формат дейтаграмм IPv6

Формат дейтаграмм IPv6 проиллюстрирован на рис. 4.24. Самые важные изменения, отличающие IPv6, очевидны именно в формате дейтаграмм.

- *Расширенные возможности адресации.* В протоколе IPv6 размер IP-адреса увеличивается с 32 до 128 бит. Таким образом гарантируется, что IP-адреса в мире не закончатся никогда. Теперь можно присвоить IP-адрес каждой песчинке на нашей планете. В IPv6 появился новый тип адреса, именуемый **адресом свободной рассылки**, который позволяет доставить дейтаграмму любому хосту из указанной группы. Например, эту возможность удобно применять для отправки HTTP-запроса GET ближайшему сайту-зеркалу из нескольких имеющихся, на котором содержится искомый документ.
- *Оптимизированный 40-байтный заголовок.* Как будет указано ниже, ряд полей IPv4 были упразднены или сделаны необязательными. В результате получились заголовки, имеющие фиксированную длину в 40 байт и обеспечивающие более быструю обработку IP-дейтаграмм. Новая кодировка возможностей обеспечивает их ускоренную обработку.

Рис. 4.24. Формат дейтаграммы IPv6

- *Метка потока и приоритет.* В протоколе IPv6 существует довольно туманное определение **потока**. В стандартах RFC 1752 и RFC 2460 записано, что данная сущность обеспечивает «пометку пакетов, относящихся к конкретным потокам, для которых отправитель затребует специальной обработки — например, обслуживания в локальном времени или качества обслуживания, отличающегося от заданного по умолчанию». Например, передача аудио или видео может интерпретироваться как поток. С другой стороны, при решении более традиционных задач — например, при передаче файлов и электронной почты — сообщаемая информация может и не обрабатываться как потоки. Возможна ситуация, в которой пользовательский трафик с высоким приоритетом (например, если пользователь доплатил за более качественное обслуживание) также будет обрабатываться как поток. Так или иначе, совершенно очевидно, что создатели IPv6 предвидели: рано или поздно потребуется механизм различения потоков, пусть даже точное определение самого термина пока отсутствует. Кроме того, в заголовке IPv6 есть 8-битное поле класса трафика. Это поле, как и поле TOS (тип сервиса обслуживания) в IPv4, может применяться для присвоения приоритета определенным дейтаграммам в потоке либо для предпочтения дейтаграмм, исходящих от одних приложений (допустим, от протокола ICMP) дейтаграммам от других приложений (таких, как сетевые новости).

Как было указано выше, при сравнении рис. 4.24 и рис. 4.13 легко заметить, что дейтаграмма IPv6 обладает более простой и обтекаемой структурой. В дейтаграммах IPv6 определяются следующие поля:

- *Версия*. Это 4-битное поле указывает номер версии протокола IP. Неудивительно, что в протоколе IPv6 здесь записано значение 6. Правда, следует отметить, что, заменив в этом поле 6 на 4, мы не получим валидную дейтаграмму IPv4. (Если бы такой прием действовал, жизнь была бы просто сказкой — см. ниже раздел о переходе с IPv4 на IPv6.)
- *Класс трафика*. Это 8-битное поле функционально напоминает поле TOS (тип обслуживания), знакомое нам по протоколу IPv4.
- *Метка потока*. Как было указано выше, это 20-битное поле применяется для идентификации потока дейтаграмм.
- *Размер полезных данных*. Это 16-разрядное значение трактуется как беззнаковое целое число и указывает количество байт в дейтаграмме IPv6, следующих за заголовком; как вы уже знаете, длина заголовка фиксированная и составляет 40 байт.
- *Следующий заголовок*. В этом поле указывается протокол, на который будет доставлено содержимое (поле с данными) этой дейтаграммы — например, TCP или UDP. В этом поле используются те же значения, что и в поле протокола в заголовке IPv4.
- *Лимит переходов*. Значение, содержащееся в этом поле, уменьшается на единицу каждым из маршрутизаторов, которые пересылают дейтаграмму. Если значение достигает нуля, то дейтаграмма утилизируется.
- *Адреса отправителя и получателя*. Различные форматы 128-разрядного адреса протокола IPv6 описаны в стандарте RFC 4291.
- *Данные*. Это полезное содержимое дейтаграммы IPv6. Когда дейтаграмма достигает места назначения, данные изымаются из нее и передаются протоколу, указанному в поле следующего заголовка.

Выше мы рассмотрели назначение полей, входящих в состав дейтаграммы IPv6. Сравнив формат дейтаграммы IPv6 с рис. 4.24 и формат дейтаграммы IPv4 с рис. 4.13, легко заметить, что некоторые поля дейтаграмм IPv4 в IPv6 отсутствуют:

- *Фрагментация/пересборка*: протокол IPv6 не допускает фрагментацию и пересборку дейтаграмм на промежуточных маршрутизаторах; эти операции могут выполняться лишь на хостах — отправителе и получателе. Если дейтаграмма IPv6, полученная маршрутизатором, слишком велика, и ее не удается передать по исходящему соедине-

нию, то маршрутизатор просто отбрасывает эту дейтаграмму и отсылает отправителю по протоколу ICMP сообщение «Packet Too Big» («Пакет слишком велик») (см. ниже). После этого отправитель может переслать данные, составив более компактную IP-дейтаграмму. На фрагментацию и пересборку дейтаграмм тратится немало времени; при удалении этой функции с промежуточных маршрутизаторов и локализации только в начальной и конечной системе удается значительно ускорить передачу информации в сети по протоколу IP.

- *Контрольная сумма заголовка*. Поскольку протоколы транспортного уровня (TCP и UDP) и канального уровня (например, стандарт Ethernet) в Интернете выполняют проверку контрольных сумм, разработчики IP, вероятно, сочли, что на сетевом уровне эта функциональность является избыточной, и ее можно оттуда удалить. Опять же, основное внимание уделялось ускорению обработки IP-пакетов. Как вы помните из обсуждения протокола IPv4 в разделе 4.4.1, поскольку в заголовке IPv4 есть поле TTL (Предписанное время жизни), подобное полю лимита переходов в IPv6, контрольная сумма заголовка IPv4 должна пересчитываться на каждом маршрутизаторе. Как и фрагментация/пересборка, такой пересчет был признан слишком затратной операцией.
- *Опции*. Поле опций больше не входит в состав стандартного IP-заголовка. Однако оно не исчезло. Теперь поле опций — это один из вариантов поля следующего заголовка, на который содержится указание в IPv6. Таким образом, в качестве следующего заголовка можно указывать как протокол TCP или UDP, так и поле опций. Поскольку поле опций удалено из стандартного заголовка, длина такого IP-заголовка стала фиксированной и составляет 40 байт.

Как вы помните из раздела 4.4.3, протокол ICMP используется IP-узлами, чтобы сообщать о состояниях ошибок и предоставлять краткую информацию (например, эхо-отклика на сообщение ping) конечной системе. Для работы с протоколом IPv6 в стандарте RFC 4443 была определена новая версия ICMP. В ICMPv6 были не только реорганизованы имеющиеся определения типов и кодов, но и добавлены новые типы и коды, необходимые для работы с новым функционалом протокола IPv6. Один из таких типов — «Packet Too Big» (Пакет слишком велик), один из новых кодов ошибки — «unrecognized IPv6 options» («Неопознанные опции IPv6»). Кроме того, протокол ICMPv6 включает в себя функционал протокола IGMP (протокол управления группами Интернета), который мы изучим в разделе 4.7. Протокол IGMP, управляющий

подключением хоста к многоадресным группам и выходом из таких групп, в версии IPv4 был самостоятельным протоколом, не связанным с ICMP.

Переход с IPv4 на IPv6

Итак, рассмотрев технические детали протокола IPv6, давайте обсудим животрепещущий практический вопрос: как осуществить переход всего общедоступного Интернета с IPv4 на IPv6? Проблема заключается в том, что можно обеспечить лишь обратную совместимость систем, работающих по протоколу IPv6, чтобы они могли посылать, переадресовывать и получать дейтаграммы IPv4. Однако уже развернутые системы для работы с IPv4 не смогут обрабатывать дейтаграммы IPv6. Существует несколько вариантов решения этой проблемы[219].

Один из способов — объявить конкретный день (дату и время), в который все машины Интернета будут отключены и модернизированы с IPv4 до IPv6. Последний технологический переход такого масштаба (переход при передаче данных с протокола NCP на TCP) имел место почти 25 лет назад. Даже в те времена[422], когда Интернет еще был миниатюрным и его администрировала горстка «мастеров», уже стало понятно, что назначить подобное «время Ч» невозможно. Если же экстраполировать такой вариант на наши дни, когда сотни миллионов машин обслуживает армия администраторов вычислительной сети и эксплуатируют миллионы пользователей, подобное мероприятие кажется тем более немыслимым. В стандарте RFC 4213 описаны два способа (которые могут использоваться совместно или по отдельности) для постепенной интеграции хостов и маршрутизаторов IPv6 в мир IPv4. Долгосрочная цель такой операции — полный отказ от всех узлов IPv4 и окончательный переход на IPv6.

Вероятно, наиболее прямолинейным способом вести узлы с поддержкой IPv6 является так называемый подход **двойного стека**, где узлы IPv6 также оснащаются полноценной реализацией поддержки IPv4. Такой узел, именуемый IPv6/IPv4 в стандарте RFC 4213, может отсылать и принимать дейтаграммы сразу в двух форматах: IPv6 и IPv4. При взаимодействии с узлом IPv4 узел IPv6/IPv4 может использовать дейтаграммы IPv4, при работе с IPv6 — «говорить на языке» IPv6. Узлы IPv6/IPv4 должны обладать адресами как IPv6, так и IPv4. Кроме того, они должны быть способны определять, для работы с какой версией протокола предназначен узел-собеседник — IPv4 или IPv6. Эта проблема

решаема при помощи системы DNS (см. главу 2), которая сможет возвращать адрес IPv6, если разрешаемое имя узла оказывается пригодным к работе с IPv6, а в противном случае — возвращать именно адрес IPv4.

В случае двойного стека дейтаграммы IPv4 используются если отправитель или получатель пакетов поддерживает только IPv4. В результате может сложиться ситуация, в которой, в сущности, дейтаграммами IPv4 будут обмениваться два узла, поддерживающие IPv6. Такая проблема проиллюстрирована на рис. 4.25. Предположим, узел А поддерживает IPv6, и ему нужно отослать дейтаграмму на узел Е, который также поддерживает IPv6. Узлы А и Б могут обменяться дейтаграммой IPv6. Однако узел Б должен создать дейтаграмму IPv4, чтобы отправить ее на узел В. Конечно же, поле данных дейтаграммы IPv6 можно скопировать в поле данных дейтаграммы IPv4 и выполнить соответствующее совмещение по адресу. Однако при преобразовании дейтаграммы из IPv6 в IPv4 те поля, которые специфичны для IPv6 (например, поле идентификатора потока), не будут иметь аналогов в IPv4. Информация из этих полей окажется потеряна. Соответственно, даже при том, что узлы Д и Е могут обмениваться дейтаграммами IPv6, дейтаграмма IPv4, которая придет на узел Д с узла Г, не будет содержать все те поля, которые были в исходной дейтаграмме IPv6, посланной с узла А.

Альтернатива применению двойного стека, также рассмотренная в RFC 4213 — это так называемое **туннелирование**. Оно позволяет решить вышеописанную проблему, обеспечивая, допустим, получение узлом Д тех дейтаграмм IPv6, которые были отосланы с узла А. Базовая идея, на которой построен механизм туннелирования, такова. Предположим, два узла поддерживают работу по протоколу IPv6 (на рис. 4.25 это, например, узлы Б и Д). Им требуется взаимодействовать, обмениваясь дейтаграммами IPv6, однако между ними находятся маршрутизаторы, поддерживающие только IPv4. Этот ряд промежуточных IPv4-маршрутизаторов между IPv6-хостами называется **туннель**, см. рис. 4.26. При использовании туннелирования узел IPv6, находящийся на стороне отправителя (например, узел Б), принимает *всю* дейтаграмму IPv6 и записывает ее в поле данных (полезной нагрузки) дейтаграммы IPv4.

Затем эта дейтаграмма IPv4 пересылается на узел IPv6 на той стороне туннеля, где находится получатель (например, на узел Д) и отправляется на первый узел в туннеле (например, В). Промежуточные маршрутизаторы IPv4, расположенные в туннеле, пересылают дейтаграмму IPv4 лишь между собой, как и любую другую, совершенно «не подозре-

вая», что полезными данными такой IPv4-дейтаграммы является целая дейтаграмма IPv6. Узел IPv6 на той стороне туннеля, где находится получатель, наконец получает дейтаграмму IPv4 (действительно, ведь это ее место назначения!), определяет, что в качестве данных она содержит дейтаграмму IPv6, извлекает дейтаграмму ее, а затем пересылает точно так же, как если бы получил напрямую с одного из узлов, расположенных на стороне отправителя.

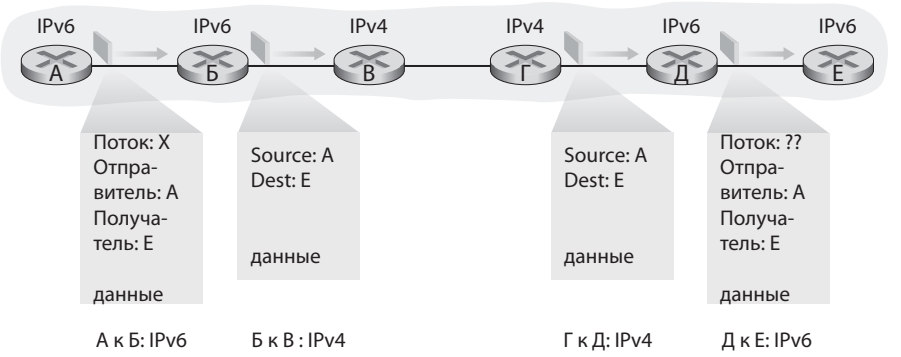


Рис. 4.25. Подход с применением двойного стека

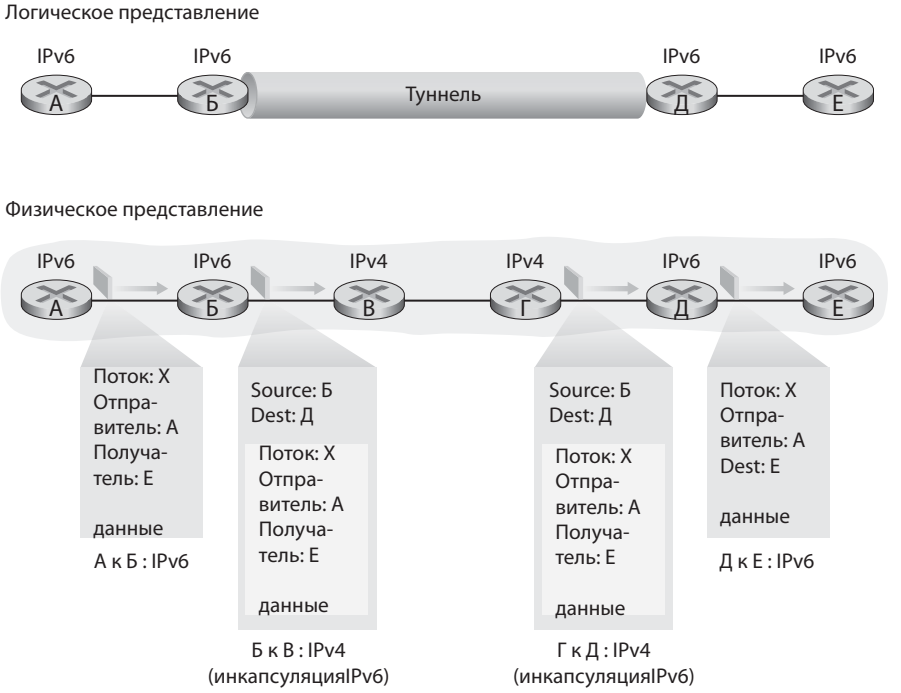


Рис. 4.26. Туннелирование

Завершая этот раздел, отметим, что, хотя переход на IPv6 поначалу протекал медленно[312], недавно этот процесс стал ускоряться. См. работу Хьюстона[217], в которой обсуждается развертывание IPv6 по состоянию на 2008 год. В публикации Estimating IPv6 & DNSSEC Deployment SnapShots[370] сделан «мгновенный снимок» внедрения IPv6 в США. Распространение новых устройств — например, смартфонов с выходом в Интернет и другой мобильной техники — создает дополнительный импульс для все более широкого распространения IPv6. Европейская «Программа партнерства в области технологий третьего поколения»[2] указывает IPv6 как стандартную схему адресации для мобильных мультимедиа.

Важный вывод, который мы можем сделать из опыта работы с IPv6, заключается в том, что на практике исключительно сложно менять протоколы сетевого уровня. С самого начала 1990-х не раз провозглашалось, что очередной новый протокол совершит революцию в Интернете, но до сих пор большинство из них получили весьма ограниченное распространение. В частности, к таким протоколам относятся IPv6, протоколы групповой адресации (раздел 4.7) и протоколы резервирования ресурсов (см. главу 7). Действительно, ввод новых протоколов на сетевом уровне можно сравнить с заменой фундамента в многоквартирном доме. Решить эту задачу очень сложно, если не сносить сам дом и не выселять из него всех жильцов — как минимум, временно. С другой стороны, в истории Интернета известны случаи стремительного развертывания новых протоколов на прикладном уровне. Классические примеры — Всемирная паутина, протокол обмена мгновенными сообщениями, одноранговый обмен файлами. Также можно упомянуть потоковую передачу аудио и видео и распределенные игры. Добавление новых протоколов прикладного уровня можно сравнить с очередной покраской дома. Сделать это относительно просто, а если вы подберете красивый оттенок, то вашему примеру последуют соседи по району. Итак, в будущем вполне могут произойти изменения на сетевом уровне Интернета, но они определенно будут протекать гораздо медленнее, чем сопоставимые изменения на прикладном уровне.

4.4.5. Краткое знакомство с IP-безопасностью

В разделе 4.4.3 мы довольно подробно рассмотрели протокол IPv4 — в частности, обсудили, какие службы он предоставляет, и как реализуются эти службы. Читая тот раздел, вы, вероятно, заметили, что мы

совершенно не затрагивали проблемы безопасности. Действительно, протокол IPv4 разрабатывался в эпоху (1970-е), когда Интернет использовался преимущественно в узком кругу ученых, исследователей сетей, где все друг друга знали и пользовались взаимным доверием. В те годы само создание компьютерной сети, объединяющей в себе множество технологий канального уровня, было задачей не из легких, что уж говорить об обеспечении безопасности.

Но сегодня проблемы безопасности стали исключительно актуальны, и исследователи Интернета приступили к разработке новых протоколов сетевого уровня, предусматривающих целый комплекс служб по обеспечению безопасности. Один из наиболее популярных протоколов из этой категории называется IPsec, и, кстати, очень активно внедряется в виртуальных частных сетях (VPN). Хотя сам IPsec и его криптографические основы довольно подробно рассмотрены в главе 8, в этом разделе мы предлагаем краткое общее введение, в котором расскажем о службах IPsec.

Протокол IPsec проектировался так, чтобы в нем обеспечивалась обратная совместимость с протоколами IPv4 и IPv6. В частности, чтобы полностью пользоваться преимуществами IPsec, нам не требуется заменять стеки протоколов во *всех* маршрутизаторах и хостах в Интернете. Например, при работе в транспортном режиме IPsec (это один из двух его «режимов»), если двум хостам требуется безопасно обмениваться информацией по этому протоколу, то он должен действовать лишь на этих двух хостах. Все остальные маршрутизаторы и хосты, расположенные между ними, могут использовать тривиальный IPv4.

Ради конкретизации изложения мы поговорим здесь только о транспортном режиме IPsec. В таком режиме два хоста сначала устанавливают друг с другом сеанс IPsec. (Таким образом, протокол IPsec ориентирован на соединения!) Когда сеанс установлен, все сегменты TCP и UDP, пересылаемые между этими хостами, в полной мере могут использовать службы IPsec. Со стороны отправителя транспортный уровень передает сегмент по протоколу IPsec. Затем IPsec шифрует сегмент, прикрепляет к нему дополнительные поля, необходимые для обеспечения безопасности, после чего инкапсулирует эти данные в самую обычную IP-дейтаграмму. (На самом деле весь этот процесс немного сложнее, подробнее см. в главе 8.) Затем хост-отправитель отсылает дейтаграмму в Интернет, который уже обеспечивает ее доставку хосту-получателю. Там IPsec расшифровывает сегмент и незашифрованным передает его на транспортный уровень.

В ходе сеанса с применением IPsec предоставляются следующие службы.

- *Криптографическое соглашение.* Это механизм, позволяющий двум хостам согласовать криптографические ключи и алгоритмы, которые будут применяться при обмене информацией.
- *Шифрование* полезного содержимого *IP-дейтаграмм*. Когда хост-отправитель получает сегмент через транспортный уровень, протокол IPsec шифрует данные. Дешифровать их может лишь протокол IPsec на хосте-получателе.
- *Целостность данных.* Протокол IPsec позволяет хосту-получателю проверить поля заголовка дейтаграммы и узнать, не было ли полезное содержимое изменено, пока дейтаграмма находилась в пути.
- *Аутентификация источника.* Когда хост получает дейтаграмму по протоколу IPsec из доверенного источника (с доверенным ключом — см. главу 8), он может быть уверен, что дейтаграмма пришла именно с того IP-адреса, который указан в качестве ее источника.

Когда между двумя хостами установлено соединение по протоколу IPsec, все пересылаемые между ними фрагменты TCP и UDP шифруются и аутентифицируются. Соответственно, протокол IPsec обеспечивает сплошной охват, гарантируя безопасность любого соединения между двумя хостами, какие бы сетевые приложения при этом ни применялись.

Компания может использовать протокол IPsec, чтобы безопасно обмениваться информацией в небезопасном глобальном Интернете. Здесь мы рассмотрим очень простой, чисто иллюстративный пример. Предположим, есть компания, в которой работает большой штат коммивояжеров. У каждого из них есть корпоративный ноутбук. Далее предположим, что эти коммивояжеры должны регулярно обсуждать конфиденциальную корпоративную информацию (например, о ценообразовании и о товарах). Эта информация хранится на сервере в штаб-квартире компании. Кроме того, допустим, что коммивояжеры также должны пересылать друг другу конфиденциальные документы. Как решить все эти задачи при помощи IPsec? Вы уже догадались, что нужно установить IPsec и на сервере компании, и на ноутбуках у всех коммивояжеров. Тогда всякий раз, когда коммивояжеру потребуется связаться с сервером или с другим коллегой, такой сеанс будет безопасным.

4.5. Алгоритмы маршрутизации

Выше в этой главе мы обсуждали в основном только одну функцию сетевого уровня — перенаправление (переадресацию) данных. Мы узнали, что, когда пакет прибывает на маршрутизатор, тот просматривает свою таблицу перенаправления и определяет канальный интерфейс, на который следует послать пакет. Мы также изучили, что алгоритмы маршрутизации, действующие на сетевых маршрутизаторах, обеспечивают обмен информацией, нужной для формирования/заполнения и настройки их таблиц перенаправления. Взаимодействие между алгоритмами маршрутизации и таблицами перенаправления было продемонстрировано на рис. 4.2. Изучив таблицу маршрутизации достаточно подробно, мы перейдем к обсуждению еще одной важной темы этой главы, а именно поговорим о функции маршрутизации сетевого уровня, имеющей огромное значение. Независимо от того, реализует ли сетевой уровень дейтаграммный принцип передачи (в таком случае различные пакеты могут следовать по разным маршрутам между конкретной парой хостов — исходным и конечным), либо работает с виртуальными каналами (тогда все пакеты между исходным хостом и хостом назначения пойдут по одному и тому же пути), именно сетевой уровень должен определять путь каждого пакета от источника к получателю. Как будет показано ниже, задача маршрутизации — определять хорошие пути (то есть, маршруты) от отправителя к получателю, прокладывая их через сеть маршрутизаторов.

Как правило, хост напрямую подключается к одному маршрутизатору, который называется **маршрутизатором по умолчанию** или **первым маршрутизатором на пути доставки**. Маршрутизатор по умолчанию хоста-отправителя мы будем называть **начальным маршрутизатором**, а маршрутизатор по умолчанию хоста-получателя — **конечным маршрутизатором**. Разумеется, задача маршрутизации пакета от хоста-отправителя к хосту-получателю сводится к пересылке пакета от начального маршрутизатора к конечному, о чем и пойдет речь в этом разделе.

Соответственно, цель алгоритма маршрутизации очень проста: имея набор маршрутизаторов, которые соединены каналами, он находит оптимальный путь от начального маршрутизатора к конечному. В простейшем случае оптимальный путь — это наименее затратный. Но на практике приходится учитывать и другие факторы, в частности, детали политики конфиденциальности (например, такое правило: «маршрутизатор *x*, относящийся к организации *Y*, не должен пересылать какие-либо

пакеты, исходящие из сети организации *Z*»). Это значительно усложняет концептуально простые и красивые алгоритмы, теория которых служит основой практики маршрутизации в современных сетях.

Для формулирования проблем маршрутизации используется граф. Как вы помните, **граф** $G = (N,E)$ — это множество из N узлов и набор E ребер, где каждое ребро есть пара узлов из N. В контексте маршрутизации на сетевом уровне узлы графа соответствуют маршрутизаторам — тем точкам, в которых принимаются решения о перенаправлении пакета по тому или иному курсу. Ребра между этими узлами представляют физические каналы между данными маршрутизаторами. Подобное абстрактное представление компьютерной сети в виде графа представлено на рис. 4.27. Примеры графов, отображающих реальные конфигурации сетей, даются в работах Доджа[141] и Чезвика[93]; исследования того, насколько точно различные варианты графов моделируют Интернет, делаются в публикациях Зегуры[682], Фалоутсоса[155] и Ли[318].

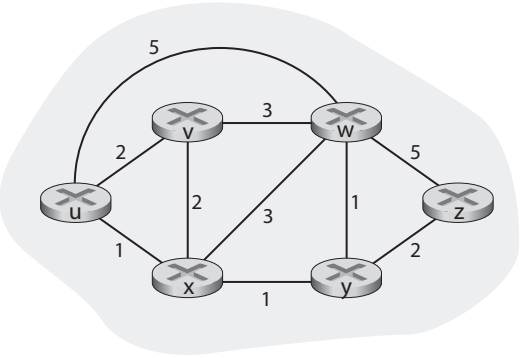


Рис. 4.27. Абстрактная модель компьютерной сети в виде графа

Как показано на рис. 4.27, ребро также имеет значение, характеризующее стоимость этого ребра. Как правило, она отражает физическую длину соответствующего канала в сети (например, трансатлантический канал будет иметь более высокую стоимость, чем короткий наземный канал), скорость передачи данных в этом канале или денежную стоимость передачи информации по нему. В нашем примере мы просто возьмем готовые значения стоимости ребер, не вдаваясь в подробности их расчета. Для каждого ребра (x,y) в E обозначим через $c(x,y)$ стоимость ребра между узлами x и y. Если пара (x,y) не относится к E, то $c(x,y) = \infty$. Кроме того, в этом примере мы будем рассматривать лишь неориентированные графы — то есть такие, ребра которых не имеют направлений.

Таким образом, ребро (x,y) равноценно ребру (y,x), а $c(y,x) = \infty$. Кроме того, узел y называется **соседним** (смежным) узлу x, если (x,y) принадлежит к E.

Учитывая, что при представлении сети в виде графа его ребрам присваиваются различные значения стоимости, естественная цель алгоритма маршрутизации — найти наименее дорогостоящий путь между исходным и конечным хостом. Чтобы точнее сформулировать эту проблему, оговоримся, что **путь** в графе $G = (N,E)$ — это такая последовательность узлов $(x_1, x_2, \ldots x_p)$, где каждая из пар (x_1, x_2), (x_2, x_3), ..., (x_{p-1}, x_p), является ребром в E. Стоимость пути $(x_1, x_2, \ldots x_p)$ — это просто сумма стоимостей всех ребер в его составе, то есть, $c(x_1, x_2) + c(x_2, x_3) + \ldots + c(x_{p-1}, x_p)$. Между любыми двумя узлами x и y как правило пролегает много путей, каждый из которых имеет свою стоимость. Один или несколько из них являются **путями наименьшей стоимости** (в сетевой терминологии пути также именуются **трактами**). Соответственно, проблема наименьшей стоимости формулируется очень просто: найти между начальной и конечной точкой такой путь, который обладает самой низкой стоимостью. Так, на рис. 4.27 путь с наименьшей стоимостью между начальным узлом u и конечным узлом w — это путь (u, x, y, w) со стоимостью 3. Обратите внимание: если все ребра в графе обладают одинаковой стоимостью, то путь с самой низкой в то же время является и **кратчайшим** (то есть он содержит наименьшее количество звеньев между начальной и конечной точками).

В качестве простого упражнения попробуйте найти на рис. 4.27 путь наименьшей стоимости между узлами u и z, а потом задумайтесь, как вы вычислили его. Если вы мыслите как большинство из нас, то поступите следующим образом: рассмотрите рис. 4.27, проведете несколько путей между u и z, а потом логически заключите, что один из них обладает меньшей стоимостью, чем остальные. (Кстати, вы рассмотрели все возможные 17 путей между u и z? Вероятно, нет.) Такое вычисление — пример централизованного алгоритма маршрутизации — такого, который вы целиком прогнали у себя в голове, располагая полной информацией о сети. В широком смысле все алгоритмы маршрутизации можно классифицировать по такому признаку: являются ли они глобальными или децентрализованными.

- **Глобальный алгоритм маршрутизации** вычисляет наименее затратный путь между исходной и конечной точкой, опираясь на исчерпывающие сведения обо всей структуре сети. Соответственно, на вход такого алгоритма поступает информация о соединениях между

всеми узлами и о стоимости всех каналов в сети. В таком случае необходимо, чтобы алгоритм каким-то образом получил всю эту информацию перед тем, как выполнять вычисления. Сами вычисления могут выполняться в одном месте (централизованный глобальный алгоритм маршрутизации), либо воспроизводиться во множестве мест. Основная характерная черта в данном случае такова: глобальный алгоритм располагает полной информацией о соединениях и их стоимости. На практике алгоритмы с такой глобальной информацией зачастую именуются **алгоритмами, учитывающими состояние каналов** (link-state algorithm, **LS**), так как должны обладать информацией о состоянии каждого канала в сети. Мы изучим алгоритмы, учитывающие состояние каналов, в разделе 4.5.1.

- В **децентрализованном алгоритме маршрутизации** расчет пути с наименьшей стоимостью выполняется итерационным распределенным образом. Ни один узел не обладает полной информацией о стоимости всех каналов в сети. Напротив, каждому узлу в начале работы известно о стоимости лишь тех каналов, которые связаны непосредственно с ним. Затем запускается итерационный процесс вычисления пути и обмена информацией с соседними узлами (то есть теми, что расположены на других концах каналов, связанных с конкретным узлом). Узел постепенно высчитывает путь с наименьшей стоимостью к одному или нескольким узлам назначения. Децентрализованный алгоритм маршрутизации мы изучим ниже в разделе 4.5.2, рассмотренный там алгоритм называется **дистанционно-векторным** (distance-vector, DV). Дело в том, что каждый узел поддерживает вектор оценок стоимости (расстояний) до всех остальных узлов в данной сети.

Еще один способ общей классификации алгоритмов маршрутизации — характеристика их по признаку статичности или динамичности. В **статических алгоритмах маршрутизации** маршруты со временем меняются очень медленно, зачастую в результате человеческого вмешательства (например, если администратор вручную отредактирует таблицу перенаправления в маршрутизаторе). **Динамические алгоритмы маршрутизации** изменяют пути в зависимости от активности сетевого трафика или при корректировках топологии сети. Динамический алгоритм может выполняться либо периодически, либо непосредственно в ответ на изменение топологии сети или стоимости каналов. Хотя динамические алгоритмы лучше адаптируются к изменениям в сети, они также более подвержены проблемам, которые могут возникать в маршрутизаторах — например, к появлению петель или осцилляции маршрутов.

Третий подход к классификации алгоритмов маршрутизации — различение их по признаку чувствительности или нечувствительности к нагрузкам. В **алгоритме, чувствительном к нагрузкам**, стоимость каналов динамически варьируется в зависимости от наличия и величины перегрузки в конкретном канале. Если стоимость каналов, которые сейчас перегружены, повышается, то алгоритм маршрутизации будет выбирать пути, позволяющие обойти такой канал. Хотя ранние алгоритмы маршрутизации, действовавшие в сети ARPAnet, были чувствительны к нагрузкам[340], с ними возникал ряд сложностей[213]. Современные алгоритмы маршрутизации, действующие в Интернете (в частности, RIP, OSPF и BGP), **нечувствительны к нагрузкам**. Это означает, что стоимость канала не находится в прямой зависимости от того, насколько он загружен.

4.5.1. Алгоритм маршрутизации, учитывающий состояние каналов

Как вы помните, при применении алгоритма маршрутизации, учитывающего состояние каналов, известны и топология сети, и стоимость каждого отдельного канала. То есть для такого алгоритма все эти сведения служат входными данными. На практике такой алгоритм реализуется на каждом узле путем широковещательной передачи пакетов, учитывающих состояние канала, *всем* другим узлам в сети. В данном случае каждый такой пакет с информацией о состоянии каналов содержит идентификаторы и значения стоимости тех каналов, которые связаны с узлом — отправителем пакета. На практике (например, при работе с Интернет-протоколом маршрутизации OSPF, рассмотренным в разделе 4.6.1) эта задача зачастую решается при помощи **широковещательного** алгоритма **с учетом состояния канала** (link-state broadcast, **LS**)[390]. Широковещательные алгоритмы будут рассмотрены в разделе 4.7. В результате широковещательной работы узлов все они получают идентичное и полное представление о сети. После этого любой узел может выполнить алгоритм с учетом состояния каналов и рассчитать такой же путь с наименьшей стоимостью.

Проиллюстрированный ниже алгоритм маршрутизации с учетом состояния каналов также известен под названием «*алгоритм Дейкстры*». Он очень похож на алгоритм Прима; см. работу Кормена[115], где подробно обсуждаются алгоритмы графов. Алгоритм Дейкстры вычисляет путь с наименьшей стоимостью от одного узла (исходного, обозначим

его u) ко всем другим узлам в сети. Алгоритм является итерационным и обладает следующим свойством: послей k-ной итерации пути с наименьшей стоимостью становятся известны k узлам назначения, а среди всех путей с наименьшей стоимостью, ведущих к узлам назначения, эти k путей будут иметь k наименьших значений стоимости. Давайте определим следующие условные обозначения:

- $D(v)$: цена пути с наименьшей стоимостью от исходного узла к точке назначения v по состоянию на данную итерацию алгоритма.
- $p(v)$: предыдущий узел (соседний с v), расположенный вдоль текущего пути с наименьшей стоимостью от исходного узла к v.
- N': подмножество узлов. Узел v находится на расстоянии N', если точно известен путь с наименьшей стоимостью от исходного узла до v.

Глобальный алгоритм маршрутизации состоит из шага инициализации и следующего за ним цикла. Количество выполнений этого цикла равно количеству узлов в сети. К моменту завершения работы алгоритм рассчитает кратчайшие пути от исходного узла u до любого другого узла в сети.

Алгоритм с учетом состояния каналов для исходного узла u

```
1    Инициализация:
2    N' = {u}
3    для всех узлов v
4       if узел v является соседним с u
5          then D(v) = c(u,v)
6       else D(v) = ∞
7
8    Цикл
9    найти узел w не относящийся к N', чтобы значение D(w) было минимальным
10   добавить w к N'
11   обновить D(v) для каждого v, соседнего для w и не относящегося к N':
12      D(v) = min( D(v), D(w) + c(w,v) )
13   /* новая стоимость для v есть либо старая стоимость v либо известная
14   стоимость кратчайшего пути к w плюс стоимость пути от w до v*/
15   пока N'= N
```

Возьмем в качестве примера сеть с рис. 4.27 и определим пути с наименьшей стоимостью от узла u ко всем возможным узлам назначения. В табл. 4.3 в общем виде представлено вычисление этого алгоритма. В каждой строке таблицы указаны значения переменных алгоритма в конце итерации. Давайте подробно рассмотрим несколько первых этапов.

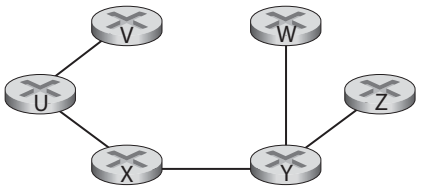
- На этапе инициализации уже известные пути с наименьшей стоимостью от узла u до смежных с ним узлов v, x и w принимают, соответственно, значения 2, 1 и 5. В частности, следует отметить, что стоимость w равна 5 (хотя вскоре мы убедимся, что существует и еще более дешевый путь), поскольку это стоимость пути прямого канала (одного перехода) от u до w. Стоимости переходов к y и z равны бесконечности, поскольку эти узлы не имеют прямой связи с u.
- В первой итерации мы просматриваем узлы, которые еще не относятся к множеству N' и находим узел с наименьшей стоимостью по состоянию на конец предыдущей итерации. Это узел x со стоимостью 1, следовательно, он добавляется ко множеству N'. Затем выполняется строка 12 алгоритма LS, которая обновляет значение $D(v)$ для всех узлов v. Получается результат, показанный во второй строке табл. 4.3 (шаг 1). Стоимость пути к узлу v остается неизменной. Стоимость пути к узлу w (которая на момент завершения инициализации была равна 5) через узел x оказывается равной 4. Соответственно, выбирается путь с меньшей стоимостью, а предшественник w на кратчайшем пути от u становится равен x. Аналогично, пересчитывается стоимость пути до y (через x), она становится равна 2. Таблица соответствующим образом обновляется.
- На следующем этапе итерации выясняется, что узлы v и y обладают путями с наименьшей стоимостью (2). Мы произвольно разрываем связь и добавляем y ко множеству N', так, что множество N' теперь содержит узлы u, x и y. Стоимость пути к оставшимся узлам, еще не принадлежащим ко множеству N' — то есть, к v, w и z, обновляется после выполнения строки 12 алгоритма LS. Получаем результаты, находящиеся в третьей строке табл. 4.3.
- И т. д.

Когда завершается выполнение алгоритма с учетом состояния каналов (LS), мы знаем для каждого узла его узел-предшественник на пути с наименьшей стоимостью от исходного узла. Для каждого предшественника мы также знаем *его* узел-предшественник и т. д.; следовательно, мы можем восстановить весь путь от исходного узла ко всем конечным.

Табл. 4.3. Выполнение алгоритма маршрутизации с учетом состояния каналов для сети, изображенной на рис. 4.27

шаг	*N'*	*D(v),p(v)*	*D(w),p(w)*	*D(x),p(x)*	*D(y),p(y)*	*D(z),p(z)*
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					

Далее, опираясь на эту информацию, можно восстановить таблицу перенаправления на заданном узле — скажем, на узле *u*. Для этого нужно найти путь с наименьшей стоимостью от *u* до узла назначения, сохранив для каждого узла его узел следующего перехода. На рис. 4.28 показаны результирующие пути с наименьшей стоимостью, а на рис. 4.27 — таблица перенаправления из узла *u*.



Назначение	Канал
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

Рис. 4.28. Путь с наименьшей стоимостью и таблица перенаправления для узла *u*

Какова будет трудоемкость этого алгоритма? То есть при наличии n узлов (не считая исходного) какой объем вычислений придется выполнить, чтобы найти пути с наименьшей стоимостью от источника до всех узлов назначения? В первой итерации потребуется просмотреть все n узлов, чтобы определить тот узел w, который не относится к множеству N', но обеспечивает минимальную стоимость. Во второй итерации для определения минимальной стоимости нам придется проверить $n - 1$ узлов, в третьей итерации — $n - 2$ и т. д. Общее количество узлов, которые нам придется перебрать в ходе всех итераций, равно $n(n+1)/2$. Соответственно, для предыдущей реализации алгоритма с учетом состояния каналов наблюдается наивысшая сложность порядка n в квадрате: $O(n^2)$. Более сложная реализация данного алгоритма, в которой используется особая струк-

тура данных под названием «куча», позволяет найти минимум в строке 9 за время, пропорциональное логарифму от числа рассматриваемых узлов. Таким образом, сложность алгоритма значительно снижается.

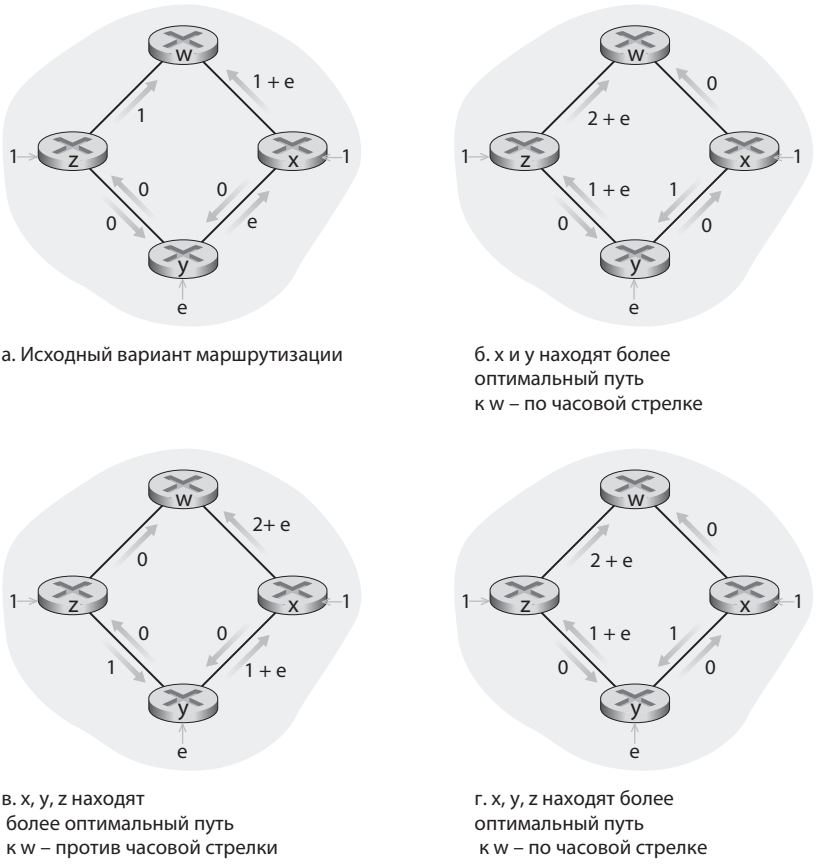
Перед тем как завершить обсуждение алгоритма LS, давайте рассмотрим один потенциально возможный патологический случай. На рис. 4.29 изображена простая топология сети, в которой стоимость канала равна той нагрузке, что по нему передается. Например, на схеме отражена задержка, которая будет возникать на том или ином участке. В данном случае, стоимости каналов несимметричны. Это значит, что $c(u,v)$ равно $c(v, u)$ лишь при условии, что нагрузка, передаваемая в обоих направлениях по каналу между узлами (u, v), будет одинаковой. В данном примере узел z порождает информацию, предназначеную для w, и узел x делает то же самое. Наконец, узел y внедряет сюда объем информации, равный e, также предназначенный для w. Изначальный вариант маршрута показан на рис. 4.29(а), стоимость каналов соответствует объему передаваемого по ним трафика.

При следующем прогоне алгоритма LS узел y определяет (опираясь на стоимости каналов, показанные на рис. 4.29а), что путь до w по часовой стрелке имеет стоимость 1, а путь до w против часовой стрелки (который и был использован в данном случае) — стоимость $1 + e$. Соответственно, теперь путь с наименьшей стоимостью от y до w пролегает по часовой стрелке.

Аналогично x определяет, что его новый путь до w с наименьшей стоимостью также пролегает против часовой стрелки, результирующие значения стоимости даны на рис. 4.29(б). При следующем прогоне алгоритма LS все три узла x, y и z — будут направлять свой трафик против часовой стрелки.

Подобные осцилляции (они могут возникнуть не только в LS, но и в любом другом алгоритме, стоимость каналов в котором зависит от перегрузок и задержек при передаче информации) необходимо предотвращать. Одно из возможных решений — установить, что стоимость каналов не должна зависеть от объема переданного трафика. Правда, такой вариант неприемлем, поскольку сама цель маршрутизации заключается в том, чтобы избегать перегруженных каналов (например, таких, в которых возникают серьезные задержки). Другое решение — обеспечить, чтобы не все маршрутизаторы выполняли алгоритм LS одновременно. Это представляется более разумным, так как мы можем рассчитывать, что даже если все маршрутизаторы будут выполнять алгоритм с оди-

наковой периодичностью, то длительность выполнения на каждом из них не будет одинаковой. Исследователи обнаружили следующую интересную закономерность: оказывается, маршрутизаторы в Интернете могут синхронизироваться друг с другом[166]. Это означает, что даже если изначально маршрутизаторы выполняют алгоритм с одинаковой периодичностью, но в разные моменты времени, то постепенно такие прогоны алгоритма на разных маршрутизаторах могут стать и далее оставаться одновременными. Чтобы избежать такой самопроизвольной синхронизации, можно задать на каждом маршрутизаторе рандомизацию моментов времени, когда он рассылает сведения о каналах.



а. Исходный вариант маршрутизации

б. x и y находят более оптимальный путь к w – по часовой стрелке

в. x, y, z находят более оптимальный путь к w – против часовой стрелки

г. x, y, z находят более оптимальный путь к w – по часовой стрелке

Рис. 4.29. Осцилляции при маршрутизации, чувствительной к нагрузкам

Изучив алгоритм LS, давайте рассмотрим еще один важный алгоритм маршрутизации, который сегодня применяется на практике — дистанционно-векторный.

4.5.2. Дистанционно-векторный алгоритм маршрутизации

В то время как алгоритм с учетом состояния каналов (LS) использует при работе глобальную информацию, **дистанционно-векторный алгоритм** (distance-vector, **DV**) маршрутизации является итерационным, асинхронным и распределенным. Его *распределенность* заключается в том, что, когда узел получает информацию от одного или нескольких *непосредственно связанных* с ним узлов-соседей, этот алгоритм выполняет вычисление, а затем вновь распространяет его результат между узлами-соседями. *Итерационность* алгоритма означает, что процесс продолжается до тех пор, пока узлы-соседи полностью не обменяются той или иной информацией. Интересно отметить, что этот алгоритм также является самозавершающимся — он не требует никакого сигнала о том, что вычисления нужно прекратить, а просто сам завершает работу. *Асинхронность* дистанционно-векторного алгоритма выражается в том, что все участвующие в нем узлы не обязаны «идти в ногу» друг с другом. Очевидно, что асинхронный итерационный самозавершающийся распределенный алгоритм гораздо интереснее и увлекательнее, чем централизованный!

Прежде чем перейти непосредственно к изучению этого алгоритма, не помешает обсудить одну важную взаимосвязь, которая существует между стоимостями наиболее выгодных (кратчайших) путей. Допустим, $d_x(y)$ — это стоимость кратчайшего пути от узла x до узла y. В таком случае, взаимосвязь между наименьшими значениями стоимости будет описываться знаменитым уравнением Беллмана — Форда:

$$d_x(y) = \min_v\{c(x,v)+d_v(y)\} \qquad (4.1)$$

где $\min_v$ — значение, взятое с учетом всех узлов, смежных с x. Уравнение Беллмана — Форда довольно понятное. Действительно, если после перехода от x до v мы изберем кратчайший путь от y до v, то стоимость этого пути будет высчитываться по формуле $c(x,v)+d_v(y)$. Поскольку мы начинаем путь с перехода к какому-либо соседнему узлу v, наименьшая стоимость пути от x до y составит минимальное значение $c(x,v)+d_v(y)$ с учетом всех узлов-соседей v.

Если среди читателей найдутся скептики, сомневающиеся в правильности этого уравнения, предлагаю проверить его для исходного узла u и узла назначения z на рис. 4.27. Исходный узел u имеет трех соседей: это узлы v, x и w. Проследив различные пути в графе, легко

убедиться, что $d_v(z) = 5$, $d_x(z) = 3$ и $d_w(z) = 3$. Подставив эти значения в уравнение 4.1 при стоимостях $c(u,v) = 2$, $c(u,x) = 1$ и $c(u,w) = 5$, имеем $d_u(z) = \min\{2 + 5,5 + 3,1 + 3\} = 4$. Очевидно, это равенство верное, и оно дает ровно такой же результат, как и алгоритм Дейкстры для той же сети. Проверить эти значения можете достаточно быстро, ваш скептицизм тут же исчезнет.

Уравнение Беллмана — Форда — не просто любопытное интеллектуальное достижение. Оно имеет огромное практическое значение. В частности, именно по этому уравнению вычисляются все значения, записанные в таблице маршрутизации узла *x*. Чтобы убедиться, допустим, что *V** — это любой узел-сосед, позволяющий получить минимальный результат в уравнении 4.1. В таком случае, если узлу *x* требуется отослать пакет узлу *y* по пути с наименьшей стоимостью, то сначала этот пакет должен быть переправлен на узел *V**. Соответственно, в таблице маршрутизации узла *x* узел *V** будет указан как цель следующего перехода, с учетом того, что конечной целью маршрутизации является узел *V** Еще один важный практический аспект этого уравнения заключается в том, что в алгоритме DV будет происходить обмен информацией по принципу «от соседа к соседу».

Базовая идея такова. Каждый узел *x* начинает работу, исходя из информации $D_x(y)$ — это оценочная стоимость кратчайшего пути от него до узла *y*, с учетом всех узлов, входящих во множество *N*. Допустим, $\boldsymbol{D}_x = [D_x(y)$:*y* принадлежит к *N*] — это вектор расстояния узла *x*, то есть, вектор, оценивающий расстояние от *x* до всех других узлов *y*, относящихся к *N*. При применении дистанционно-векторного алгоритма каждый узел x будет содержать следующую информацию о маршрутизации.

- Для каждого узла *v* — стоимость $c(x,v)$ от *x* до непосредственной связи с соседом, *v*
- Вектор расстояния узла *x*, то есть, $D_x = [D_x(y)$:*y* во множестве *N*], вектор, оценивающий расстояние от *x* до всех других узлов *y*, относящихся к *N*.
- Вектор расстояния для каждого из узлов-соседей, то есть, $D_v = [D_v(y)$:*y* во множестве *N*] для каждого из соседей *v* узла *x*.

В распределенном асинхронном алгоритме каждый узел время от времени посылает копию своего вектора расстояния каждому из соседей. Когда узел *x* получает новый вектор расстояния от любого из своих

соседей v, он сохраняет вектор расстояния v, а затем использует уравнение Беллмана — Форда, чтобы обновить собственный, вот так:

$$D_x(y) = \min_v\{c(x,v) + D_v(y)\} \qquad \text{Для каждого узла } y \text{ во множестве } N$$

Если в результате такой операции обновления вектор расстояния узла x изменится, то узел x пошлет полученное пересчитанное значение каждому из соседей, которые, в свою очередь, также смогут обновить свои векторы расстояний. И как ни удивительно, в ходе такого асинхронного обмена между узлами, все оценки стоимости $D_x(y)$ постепенно сходятся к $d_x(y)$, фактической стоимости кратчайшего пути от узла x к узлу y[50]!

Дистанционно-векторный алгоритм

Для каждого узла x:

```
1    Инициализация:
2    для всех узлов назначения y во множестве N:
3       Dx(y) = c(x,y) /* если y не является смежным, то c(x,y) = ∞ */
4    для каждого узла-соседа w
5       Dw(y) = ? для всех узлов назначения y во множестве N
6    для каждого узла-соседа w
7       отправить вектор расстояния Dx = [Dx(y): y во множестве N] к w
8
9    цикл
10   ждем (пока не увидим изменения стоимости канала к соседу w или
11      пока не получим дистанционный вектор от того или иного соседа w)
12
13   для каждого узла y во множестве N:
14      Dx(y) = minv{c(x,v) + Dv(y)}
15
16   если Dx(y) изменится для любого узла назначения y
17   посылаем дистанционный вектор Dx = [Dx(y): y во множестве N] всем соседям
18
19   вечно
```

В дистанционно-векторном (DV) алгоритме узел x обновляет свою дистанционно-векторную оценку в двух случаях: либо когда фиксирует изменение стоимости одного из непосредственно связанных с ним каналов, либо когда получает обновленную информацию о дистанционном векторе от одного из своих соседей. Но чтобы обновить собственную таблицу маршрутизации для заданного узла назначения y, узлу x на самом деле требуется знать не кратчайшее расстояние до y, а свой узел-сосед $V^*(y)$, находящийся на расстоянии одного перехода на кратчайшем пути к y. Как вы уже догадались, таким маршрутизатором $V^*(y)$ в нашем случае является сосед v, позволяющий достичь минимального значения в строке 14 алгоритма DV. Если имеется несколько узлов-соседей v, дающих такое минимальное значение, то любой из них можно подставить на место $V^*(y)$. Следовательно, в строках 13–14 для каждого узла назначения y узел x также определяет $v\times(y)$ и обновляет свою таблицу маршрутизации для точки y.

Как вы помните, алгоритм LS является глобальным: это означает, что в нем каждый узел должен сначала получить полную информацию о сети, а уже потом будет выполняться алгоритм Дейкстры. В свою очередь, алгоритм DV является *децентрализованным* и обходится без такой глобальной информации. Действительно, все сведения, которые будут у узла, сводятся к стоимости каналов, связывающих его с непосредственно примыкающими соседями, а также сведениям, полученным от этих соседей. Каждый узел дожидается обновления от любого из своих соседей (строки 10–11), получив обновление, вычисляет новый дистанционный вектор (строка 14), после чего распространяет новый дистанционный вектор среди своих соседей (строки 16–17). DV-подобные алгоритмы применяются во многих протоколах маршрутизации: таковы, например, RIP и BGP, действующие в Интернете, ISO IDRP, Novell IPX и оригинальный алгоритм ARPAnet.

На рис. 4.30 проиллюстрирована работа дистанционно-векторного алгоритма для простой сети, состоящей из трех узлов — она показана в верхней части рисунка. Работа алгоритма показана в синхронном режиме, где все узлы одновременно получают дистанционные векторы от своих соседей, вычисляют новые и информируют своих соседей о том, не изменились ли эти векторы. Изучив этот пример, вы окончательно убедитесь, что алгоритм столь же корректно работает и по асинхронному принципу, когда вычисления на узлах и генерирование/получение обновлений могут происходить в любой момент.

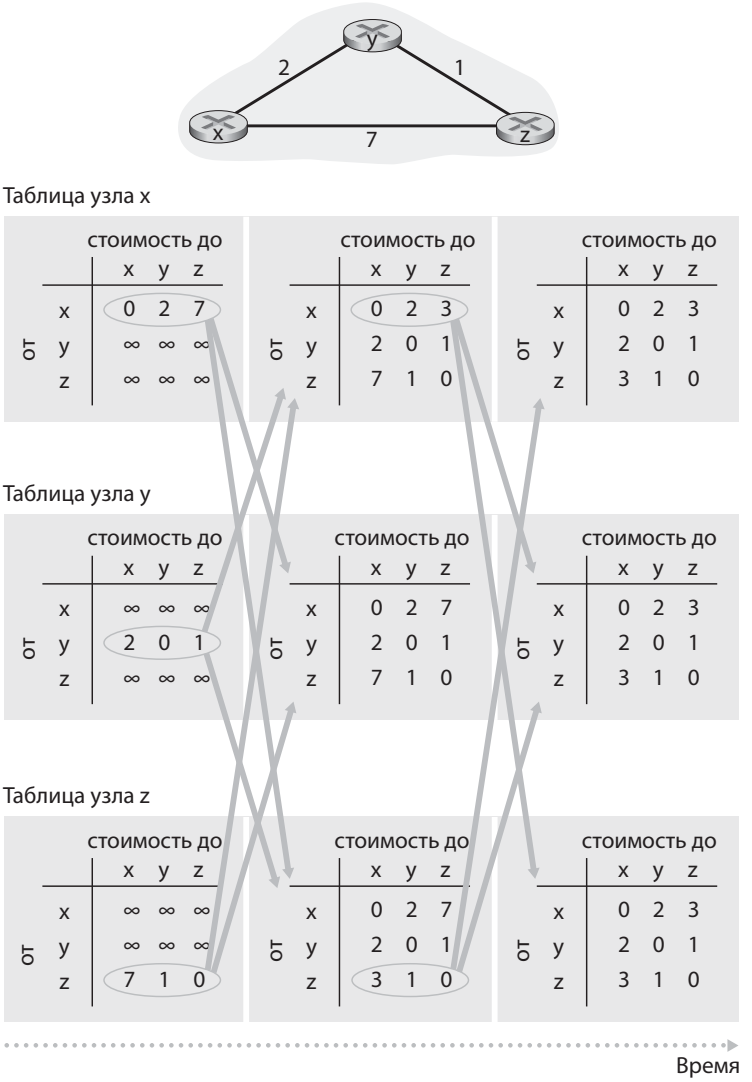


Рис. 4.30. Дистанционно-векторный алгоритм

В крайнем левом столбце на рисунке показаны три исходные **таблицы маршрутизации** для каждого из трех узлов. Например, в верхнем левом углу находится исходная таблица для узла *x*. В рамках каждой отдельно взятой таблицы маршрутизации каждая строка — это дистанционный вектор. Точнее говоря, каждая таблица маршрутизации содержит собственный дистанционный вектор узла и такие дистанционные векторы его узлов-соседей. Итак, в первой строке исходной таблицы маршрутизации узла x имеем $\boldsymbol{D}_x = [D_x(x), D_x(y), D_x(z)] = [0,2,7]$. Во вто-

рой и третьей строках этой таблицы находятся наиболее свежие (недавно полученные) дистанционные векторы от узлов y и z соответственно. Поскольку на момент инициализации узел x еще ничего не получил от узлов y или z, то записи во второй и третьей строках после инициализации результируют в бесконечность.

После инициализации каждый узел посылает свой дистанционный вектор каждому из двух узлов-соседей. На рис. 4.30 это показано при помощи стрелок, направленных из таблиц первого столбца в таблицы второго. Например, узел x отсылает дистанционный вектор $\boldsymbol{D}_x = [0,2,7]$ одновременно узлам y и z. Получив обновления, каждый узел пересчитывает собственный дистанционный вектор. Например, для узла x:

$$D_x(x) = 0$$
$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_y(y)\} = \min\{2+0,7+1\} = 2$$
$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} = \min\{2+1,7+0\} = 3$$

Соответственно, в таблицах второго столбца указаны новые дистанционные векторы для каждого узла, а вместе с ними — дистанционные векторы, только что полученные от узлов-соседей. В частности, обратите внимание на то, что оценка наименьшей стоимости пути от узла x до z, $D_x(z)$, изменилась с 7 на 3. Также отметим, что для узла x его узел-сосед y достигает минимума в строке 14 DV-алгоритма. На узле x имеем $V^*(y) = y$ и $V^*(z) = y$.

После того, как узлы пересчитают свои дистанционные векторы, они опять пошлют обновленные значения своим соседям (если изменения действительно будут). На рис. 4.30 это проиллюстрировано при помощи стрелок, идущих из таблиц второго столбца таблиц в третий. Обратите внимание: лишь узлы x и z присылают обновления. Дистанционный вектор узла y не изменился, поэтому данный узел и не посылает обновлений. После получения обновлений узлы пересчитывают свои дистанционные векторы и обновляют таблицы маршрутизации, что и показано в третьем столбце.

Процесс получения обновленных дистанционных векторов от узлов-соседей, пересчет записей в таблицах маршрутизации и информирование соседей об изменении стоимости кратчайшего пути к узлу назначения продолжаются до тех пор, пока сообщений об обновлении больше не останется. Когда рассылка сообщений об обновлении прекращается, пересчет таблиц маршрутизации прекратится, и алгоритм перейдет

в стационарное состояние. Это значит, что все узлы будут находиться на этапе ожидания, который описан в строках 10–11 алгоритма DV. Алгоритм останется в стационарном состоянии до тех пор, пока вновь не изменится стоимость путей. Об этом пойдет речь ниже.

Дистанционно-векторный алгоритм при изменении стоимостей и неисправностях каналов

Когда узел, на котором работает дистанционно-векторный алгоритм, обнаруживает изменение стоимости канала, направляющегося от него к соседу (строки 10–11), он обновляет свой дистанционный вектор (строки 13–14) и, если наименьшая стоимость пути изменилась, он извещает об этом своих соседей (строки 16–17). Эту ситуацию иллюстрирует рис. 4.31(а). В данном примере стоимость канала от узла x до узла y изменяется с 4 до 1. Здесь мы рассматриваем только записи таблиц расстояний узлов y и z, содержащие стоимости путей до узла x. Дистанционно-векторный алгоритм приводит к следующей последовательности событий.

1. В момент времени t_0 узел y замечает изменение стоимости канала (с 4 до 1) и информирует об этом своих соседей, так как благодаря обновлению стоимости линии меняются минимальные стоимости путей.
2. В момент времени t_1 узел z получает сообщение от узла y и обновляет свою таблицу. Затем он вычисляет новое значение минимальной стоимости маршрута до узла x (она уменьшилась с 5 до 2) и извещает об этом своих соседей.
3. В момент времени t_2 узел y получает сообщение от узла z и обновляет свою таблицу. Его значения минимальных стоимостей остались прежними (хотя стоимость пути до узла x через узел z изменилась), поэтому узел y не посылает сообщений своим соседям. Алгоритм переходит в стационарное состояние.

Соответственно, дистанционно-векторному алгоритму требуется только две итерации, чтобы перейти в стационарное состояние. «Хорошие новости» об уменьшившийся стоимости канала между узлами x и y быстро распространяются по сети.

Рассмотрим теперь, что произойдет в случае *увеличения* стоимости канала. Предположим, что стоимость канала между узлами x и y возросла с 4 до 60, как показано на рис. 4.31(б).

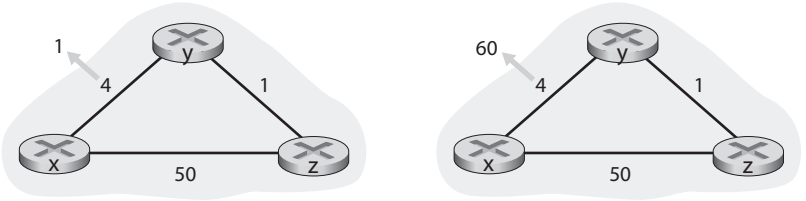


Рис. 4.31. Изменение стоимости канала

1. До изменения стоимости канала $D_y(x) = 4$, $D_y(z) = 1$, $D_z(y) = 1$ и $D_z(x) = 5$. В момент времени t_0 узел y замечает изменение стоимости канала (с 4 до 60). Узел y вычисляет, что новая наименьшая стоимость пути до узла x равна

   $$D_y(x) = \min\{c(x,y) + D_x(x), c(y,z) + D_z(x)\} = \min\{60+0,1+5\} = 6.$$

   Поскольку мы можем видеть всю сеть сразу (глобально), нам понятно, что это новое значение неверно. Но узлу y известно лишь то, что стоимость его прямого соединения с узлом x равна 60 и что узел z в последний раз сообщил узлу y о наличии у узла z пути к узлу x стоимостью 5. Узел y решает, что теперь ему дешевле посылать пакеты узлу x через узел z. Таким образом, в момент времени t_1 у нас образовалась **петля** — узел y направляет пакеты для узла x через узел z, а узел z направляет пакеты для узла x через узел y. Петля подобна черной дыре — пакет, предназначенный узлу x, будет «отфутболиваться» узлами y и z друг другу до тех пор, пока таблицы маршрутизации на этих узлах не изменятся.

2. Так как узел y вычислил новый путь наименьшей стоимости до узла x, он информирует об этом узел z в момент времени t_1.

3. Спустя некоторое время после t_1 узел z получает новый дистанционный вектор с измененной наименьшей стоимостью пути от узла y до узла x (узел y сообщил узлу z, что это значение теперь равно 6). Узел z знает, что он может добраться до узла y по каналу стоимости 1, и вычисляет новое значение стоимости маршрута до узла x (все также через узел y), равное 7. Поскольку наименьшая стоимость пути от узла z до узла x увеличилась, узел z извещает об этом узел y в момент времени t_2.

4. Аналогичным образом узел y обновляет свою таблицу и информирует узел z о том, что значение минимальной стоимости теперь равно 8. Затем узел z обновляет свою таблицу и информирует узел y о том, что значение минимальной стоимости стало равно 9, и т. д.

Как долго может продолжаться этот процесс? Вы можете убедиться в том, что петля будет сохраняться в течение 44 итераций (необходимых узлам y и z для обмена сообщениями) — пока наконец узел z не определит, что стоимость пути до узла x через узел y больше 50. В этот момент узел z (наконец-то!) выяснит, что путь наименьшей стоимости к узлу x пролегает по прямой линии с узлом x. Таким образом, «дурным вестям» об увеличении стоимости канала между узлами x и y для распространения по сети потребовалось очень много времени! Что бы произошло, если бы стоимость канала $c(y,x)$ увеличилась с 4 до 10 000, а стоимость канала $c(z,x)$ была бы равна 9999? Подобная проблема иногда называется проблемой «счета до бесконечности».

Дистанционно-векторный алгоритм и обратная коррекция

Обсуждавшегося выше сценария со счетом до бесконечности можно избежать, если использовать метод, называемый обратной коррекцией, или «*отравлением» обратного пути*. Идея этого метода проста — если узел z направляет пакеты узлу x через узел y, тогда узел z объявит узлу y, что его (узла z) расстояние до узла x равно бесконечности (хотя на самом деле узлу z известно, что $D_z(x) = 5$). Узел z будет продолжать говорить узлу y эту «маленькую ложь» до тех пор, пока узел z направляет пакеты узлу x через узел y. Поскольку узел y полагает, что у узла z нет пути к узлу x, узел y никогда не станет пытаться посылать пакеты узлу x через узел z, пока узел z продолжает посылать пакеты узлу x через узел y (и лгать о том, что у него нет пути к узлу x).

Давайте рассмотрим, как метод обратной коррекции решает проблему петель, с которой мы столкнулись выше на рис. 4.31 (б). В результате обратной коррекции в таблице расстояний узла y в записи стоимости пути до узла x через узел z указана бесконечность (так как узел z сообщил узлу Y, что расстояние от узла z до узла x равно бесконечности). Когда стоимость линии (x,y) увеличивается с 4 до 60, узел y обновляет свою таблицу, но продолжает направлять пакеты узлу x напрямую, несмотря на увеличившуюся стоимость маршрута, а также информирует узел z об изменении стоимости, сообщая, что $D_y(x) = 60$. Получив это извещение в момент времени t_1, узел z немедленно изменяет свой маршрут к узлу x на прямую линию (z,x) стоимости 50. Поскольку маршрут к узлу x изменился и более не проходит через узел y, узел z сообщает узлу y в момент времени t_2, что $D_z(x) = 50$. Получив эту информацию, узел y обновляет свою таблицу расстояний, так что $D_y(x) = 51$. Кроме того, поскольку через узел z теперь проходит путь наименьшей стоимо-

сти от узла *y* к узлу *x*, узел *y* «отравляет» обратный путь от узла *z* до узла *x*, сообщая в момент времени t_3 узлу *z*, что $D_y(x) = \infty$ (хотя на самом деле узлу *y* известно, что $D_y(x) = 51$).

Позволяет ли метод «отравления» обратного пути решить проблему счета до бесконечности в общем случае? Нет. Вы можете убедиться сами, что петли, состоящие из трех и более узлов (а не просто из двух смежных), метод обратной коррекции распознать не сможет.

Сравнение алгоритмов маршрутизации LS и DV

Дистанционно-векторный алгоритм (DV) и алгоритм с учетом состояния каналов (LS) взаимно дополняют друг друга при планировании маршрутизации. В алгоритме DV каждый узел обменивается информацией *только* со своими непосредственными соседями, но предоставляет этим смежным узлам оценку наименьшей стоимости от себя до *всех остальных* узлов в сети (насколько ему это известно). В алгоритме LS каждый узел обменивается информацией со *всеми* остальными узлами в сети (широковещательным образом), но сообщает им стоимости *лишь* тех каналов, которые непосредственно с ним связаны. Давайте завершим наше изучение алгоритмов LS и DV, вкратце сравнив некоторые их свойства. Как вы помните, N — это множество узлов (маршрутизаторов), а E — множество ребер (каналов).

- *Сложность сообщений.* Как мы видели, алгоритм, основанный на состоянии линий, требует от каждого узла знать стоимость каждой линии сети. Для этого необходимо отправить $O(|N|\,|E|)$ сообщений. Кроме того, каждый раз, когда стоимость линии изменяется, об этом следует известить все узлы. Дистанционно-векторный алгоритм требует обмена сообщениями только между напрямую соединенными узлами на каждой итерации. Как было показано, время, необходимое для схождения алгоритма, может зависеть от многих факторов. Когда изменяется стоимость линии, дистанционно-векторный алгоритм распространяет результаты только в том случае, если обновление приводит к изменению пути с наименьшей стоимостью для одного из узлов, присоединенного к этому каналу.
- *Скорость схождения.* Как мы видели, количество вычислений в нашей реализации алгоритма, основанного на состоянии каналов, растет пропорционально квадрату узлов сети, требуя передачи $O(|N|\,|E|)$ сообщений. Дистанционно-векторный алгоритм может сходиться медленно (в зависимости от относительной стоимости путей, как

было показано в примере на рис. 4.10), и во время схождения могут образовываться петли. Кроме того, дистанционно-векторный алгоритм страдает от «приступов» счета до бесконечности.

- *Живучесть*. Что может случиться, если маршрутизатор выйдет из строя, «сойдет с ума» или объявит забастовку? В алгоритме маршрутизации, основанном на состоянии каналов, маршрутизатор может передать всем остальным маршрутизаторам неверные сведения о стоимости одного из присоединенных к нему каналов. Узел может также повредить или потерять один из широковещательных пакетов LS-алгоритма, который он получил. Но узел рассчитывает только собственную таблицу перенаправления. Остальные узлы заполняют свои таблицы сами. Это означает, что в алгоритме, основанном на состоянии каналов, расчеты маршрутов выполняются в значительной степени раздельно, что обеспечивает определенную степень живучести. В случае дистанционно-векторного алгоритма узел может передать другим узлам неверно рассчитанные им значения минимальной стоимости путей. (Такие ситуации встречаются на практике. В 1997 году неисправный маршрутизатор, принадлежащий небольшой компании, занимающейся предоставлением доступа в Интернет, снабжал маршрутизаторы национальной магистрали ошибочной информацией о маршрутах. Это привело к тому, что другие маршрутизаторы завалили трафиком неисправный, в результате большие фрагменты Интернета в течение нескольких часов были отрезаны[365].) Можно сказать, что в дистанционно-векторном алгоритме на каждой итерации результаты вычислений узла непосредственно передаются соседнему узлу, а затем на следующей итерации они попадают к соседу соседа и т. д. Таким образом, в дистанционно-векторном алгоритме некорректно вычисленные данные могут распространиться по всей сети.

В заключение скажем, что ни один алгоритм нельзя считать победителем. Как мы увидим в разделе «Маршрутизация в Интернете», на практике применяются оба эти алгоритма.

Другие алгоритмы маршрутизации

Рассмотренные нами дистанционно-векторный алгоритм и алгоритм, основанный на состоянии каналов, представляют собой не просто популярные алгоритмы маршрутизации. По сути *только* эти два алгоритма и применяются на практике. Тем не менее за последние 30 лет исследователями было предложено множество других алгоритмов маршрутизации, варьирующихся от крайне простых до очень сложных.

В основе широкого класса алгоритмов маршрутизации лежит точка зрения на пакетный трафик как на потоки данных между отправителями и получателями. При таком подходе проблема выбора маршрута может быть сформулирована математически как задача оптимизации при ограничениях, известная как задача сетевых потоков[50]. Еще одно семейство алгоритмов маршрутизации, о которых следует сказать здесь несколько слов, обязаны своим происхождением телефонным сетям. Эти **алгоритмы маршрутизации коммутируемых каналов** представляют интерес для сетей с коммутацией пакетов, когда для каждого соединения резервируются ресурсы линии, такие как пропускная способность части линии связи или объем буферов. И хотя формулировка задачи маршрутизации значительно отличается от метода путей наименьшей стоимости, у подобных алгоритмов довольно много общего, по крайне мере в том, что касается алгоритмов определения маршрутов. Более подробную информацию об исследованиях в области маршрутизации коммутируемых каналов можно найти в публикациях Эша[27], Росса[572] и Жерара[182].

4.5.3. Иерархическая маршрутизация

В предыдущем разделе мы рассматривали сеть просто как множество соединенных друг с другом маршрутизаторов. Один маршрутизатор ничем не отличался от других в том смысле, что на всех работал один и тот же алгоритм для расчета маршрутов через всю сеть. Однако данная модель с однородным набором маршрутизаторов является упрощением и не применяется на практике по двум причинам.

- *Масштабирование*. Когда количество маршрутизаторов оказывается очень большим, накладные расходы на вычисление, хранение данных и обмен данными о маршрутах (такими как обновление состояния каналов или изменения путей наименьшей стоимости) между маршрутизаторами становятся неприемлемыми. Сегодняшний Интернет состоит из сотен миллионов хостов. Хранение информации о маршрутах на каждом из этих хостов потребовало бы памяти огромных размеров. Накладные расходы на широковещательную рассылку пакетов с информацией о состоянии линий между всеми маршрутизаторами Интернета просто не оставили бы пропускной способности для пакетов с данными! А дистанционно-векторный алгоритм при таком огромном количестве узлов сети никогда бы не достиг схождения! Очевидно, необходимо было что-то предпринять, чтобы упростить задачу вычисления маршрутов в такой огромной сети, как Интернет.

- *Административная автономия.* Хотя инженеры часто игнорируют такие вопросы, как пожелания компаний управлять маршрутизаторами, как им хочется (например, использовать алгоритм маршрутизации по своему выбору), или скрыть внутреннюю организацию сети от внешних наблюдателей, важность подобных вопросов может быть высокой. В идеальном случае организация должна иметь возможность управлять своей сетью так, как ей заблагорассудится, не теряя при этом возможности ее соединения с «внешним» миром.

Обе эти задачи могут быть решены путем объединения маршрутизаторов в отдельные области, называемые **автономными системами** (Autonomous System, **АС**). Маршрутизаторы в пределах одной автономной системы используют один и тот же алгоритм маршрутизации (например, дистанционно-векторный алгоритм или алгоритм, основанный на состоянии каналов) и обладают информацией обо всех маршрутизаторах своей автономной системы, как это было в рассмотренном ранее идеальном случае. Алгоритм маршрутизации, используемый внутри автономной системы, называется **протоколом внутренней маршрутизации**. Разумеется, необходимо соединить автономные системы друг с другом, и поэтому один или несколько маршрутизаторов в автономной системе будут отвечать за пересылку пакетов за пределы автономной системы. Эти маршрутизаторы называются **шлюзовыми (граничными) маршрутизаторами**, или просто **шлюзами**.

Данный сценарий проиллюстрирован на рис. 4.32. Здесь изображены три автономные системы, АС1, АС2 и АС3. Жирные линии представляют прямые соединения между парами маршрутизаторов. Тонкие линии, проведенные от маршрутизаторов, представляют подсети, непосредственно подключенные к маршрутизаторам. В автономной системе АС1 имеются четыре маршрутизатора, 1а, 1б, 1в и 1г, на которых работает протокол внутренней маршрутизации, используемый в пределах автономной системы АС1. Соответственно, каждый из этих четырех маршрутизаторов знает, как пересылать пакеты по оптимальным путям на любой целевой узел в системе АС1. Аналогично в автономных системах АС2 и АС3 есть по три маршрутизатора. Обратите внимание, что протоколы внутренней маршрутизации, применяемые в автономных системах АС1, АС2 и АС3, могут быть разными. Шлюзовыми маршрутизаторами являются 1б, 1в, 2а и 3а.

Итак, должно быть ясно, как маршрутизаторы в автономной системе определяют пути доставки для таких пар «исходный узел — конечный узел», которые являются внутренними для автономной системы. Однако

в нашей задаче о маршрутизации из конца в конец по-прежнему нет ответа на один большой вопрос: как маршрутизатор, расположенный внутри автономной системы, сумеет направить пакет на узел, находящийся вне этой автономной системы? Ответить на этот вопрос не составляет труда, если в АС имеется лишь один шлюз, соединяющий ее ровно с одной другой автономной системой. В таком случае, поскольку внутренний алгоритм маршрутизации, действующий в АС, определяет путь с наименьшей стоимостью от любого внутреннего маршрутизатора до шлюза, каждому внутреннему маршрутизатору известно, как пересылать пакеты.

Как только шлюз получает пакет, он пересылает его по единственному каналу, ведущему из шлюза за пределы автономной системы. Вторая автономная система, расположенная на другом конце канала, будет завершать доставку пакета к месту назначения. Рассмотрим пример: маршрутизатор 2б на рис. 4.32 получает пакет, чей узел назначения находится за пределами АС2. В таком случае маршрутизатор 2б перешлет пакет на 2а или 2в — в зависимости от того, что указано в таблице перенаправления 2б, которая была сконфигурирована в соответствии со внутренним протоколом маршрутизации автономной системы АС2. Рано или поздно пакет поступит на шлюз 2а, откуда будет переправлен на 1б. Как только пакет покинет 2а, работа системы АС2 с этим пакетом будет завершена.

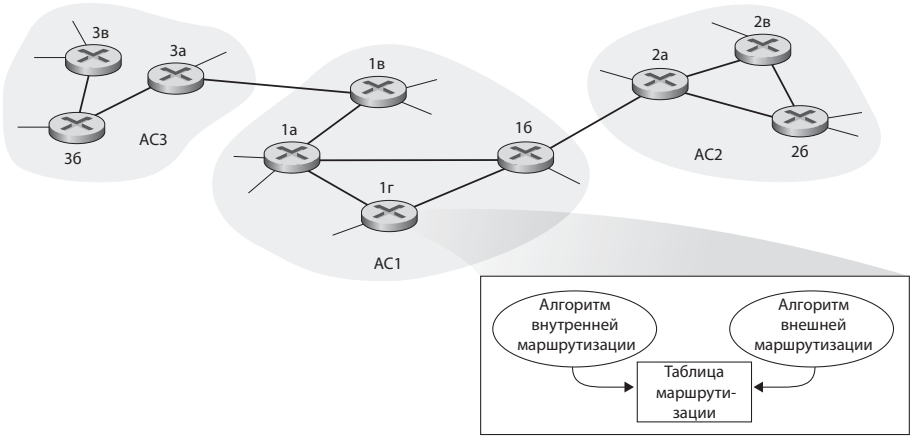


Рис. 4.32. Пример взаимосвязанных автономных систем

Итак, проблема тривиальна, если в автономной системе есть всего один канал, ведущий за ее пределы. Но что если таких каналов будет два и более (соответственно, также будет два и более шлюзов)? В таком случае

определить направление, по которому требуется пустить пакет, становится существенно сложнее. Возьмем для примера маршрутизатор в системе АС1 и предположим, что его нужно отправить на узел, расположенный вне этой АС. Маршрутизатор должен передать пакет на один из шлюзов, 1б или 1в, но на какой именно? Для решения этой проблемы АС1 должна узнать (1) какие места назначения достижимы через АС2, а какие — через АС3 и (2) распространить эту информацию о доступности всем маршрутизаторам в пределах АС1. После этого каждый из маршрутизаторов данной автономной системы сможет соответствующим образом сконфигурировать свою таблицу перенаправления, чтобы иметь возможность работать с такими внешними узлами назначения. Две эти задачи — получение информации о достижимости от соседних АС и распространение этой информации среди всех внутренних маршрутизаторов конкретной автономной системы — решаются при помощи **протокола внешней маршрутизации автономной системы**. Поскольку такой протокол внешней маршрутизации при работе задействует соединения между двумя автономными системами, эти две системы должны использовать один и тот же протокол внешней маршрутизации. Так и есть: все автономные системы, работающие в Интернете, применяют для внешней маршрутизации один и тот же протокол BGP4, о котором мы поговорим в следующем разделе. Как показано на рис. 4.32, каждый маршрутизатор получает информацию от протокола внутренней маршрутизации автономной системы и от протокола внешней маршрутизации, а таблица перенаправления заполняется с учетом информации обоих этих протоколов.

В качестве примера давайте рассмотрим подсеть x (обозначаемую собственным CIDR-адресом) и предположим, что АС1 узнает по внешнему протоколу маршрутизации следующую информацию: подсеть x достижима из системы АС3, *но* недостижима из системы АС2. Система АС1 распространяет эту информацию между всеми своими маршрутизаторами. Когда маршрутизатор 1г узнает, что подсеть x достижима из АС3, то есть в нее можно попасть через шлюз 1в, то уже по информации из внутреннего протокола маршрутизации 1г определяет интерфейс маршрутизации, который расположен на пути с наименьшей стоимостью от 1г к шлюзу 1в. Обозначим этот интерфейс I. В таком случае маршрутизатор 1г может внести в свою таблицу перенаправления запись (x,I). Этот пример, а также другие, приведенные в данной главе, в целом верно иллюстрируют механизмы, действующие в Интернете, но упрощают их. В следующем разделе будет дано более детальное, хотя и более сложное описание этих механизмов — оно понадобится нам при изучении протокола BGP.

Исходя из предыдущего примера, далее предположим, что АС2 и АС3 подключены к другим автономным системам, которые не показаны на схеме. Также допустим, что по внешнему протоколу маршрутизации АС1 узнает, что подсеть *x* достижима как из АС2 через шлюз 1б, так и из АС3 через шлюз 1в. Тогда система АС1 распространит эту информацию между всеми своими маршрутизаторами, включая 1г. Маршрутизатор 1г, чтобы сконфигурировать свою таблицу перенаправления, должен определить, на какой из шлюзов он будет отправлять пакеты, адресованные в подсеть *x* — на 1б или на 1в. В таких случаях на практике часто применяется маршрутизация по принципу **«горячей картофелины»** (hot-potato routing). Другими словами, автономная система должна избавиться от пакета («горячей картофелины») как можно быстрее (точнее — с наименьшими затратами). Для этого система обязывает обычный маршрутизатор послать пакет на тот шлюзовой маршрутизатор, до которого пролегает путь с наименьшей стоимостью, если выбирать из альтернативных шлюзов, расположенных между данным маршрутизатором и хостом назначения. В контексте рассматриваемого нами примера при использовании принципа горячей картофелины на маршрутизаторе 1г мы будем задействовать информацию внутреннего протокола маршрутизации данной автономной системы, чтобы определить стоимости путей до 1в и 1б, а затем выберем путь с наименьшей стоимостью. Когда путь выбран, 1б добавляет в свою таблицу перенаправления запись, соответствующую подсети *x*. На рис. 4.33 обобщенно представлены все действия, которые выполняются на маршрутизаторе 1г для добавления в его таблицу перенаправления новой записи относительно подсети *x*.

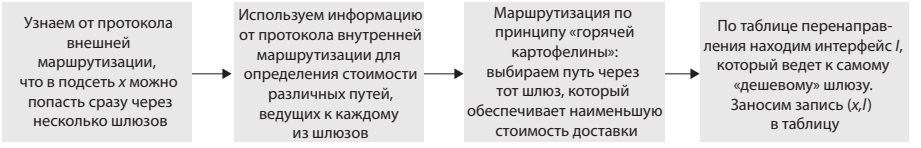


Рис. 4.33. Этапы, в ходе которых хост назначения, расположенный во внешней автономной системе, добавляется в таблицу перенаправления маршрутизатора

Когда АС узнает о месте назначения из соседней АС, первая АС объявляет эту информацию о маршрутизации некоторым из соседствующих с ней АС. Например, АС1 узнает от АС2, что подсеть *x* достижима через АС2. После этого АС1 может сообщить АС3, что подсеть *x* достижима через АС1. Таким образом, если АС3 требуется переслать пакет, предназначенный для доставки в подсеть *x*, то эта автономная система переправит пакет в АС1, а АС1 в свою очередь перешлет этот пакет в АС2. Как будет

показано в разделе о BGP, автономная система может достаточно гибко выбирать те места, о которых она будет объявлять своим соседним автономным системам. Это уже вопрос *политик* администрирования, которые чаще обусловлены экономическими, а не техническими факторами.

Как вы помните из раздела 1.5, Интернет представляет собой иерархическую структуру взаимосвязанных провайдеров. Итак, как же соотносятся Интернет-провайдер и автономная система? Вы могли бы предположить, что все маршрутизаторы одного провайдера образуют одну автономную систему. На практике зачастую это не так, многие провайдеры сегментируют свои сети на множество автономных систем. Например, некоторые провайдеры первого эшелона используют каждый по одной автономной системе для всей своей сети; другие подразделяют свои сети на десятки взаимосвязанных автономных систем.

Итак, проблемы масштабирования и административной автономии решаются путем использования автономных систем. В рамках автономной системы все маршрутизаторы используют для внутренней маршрутизации один и тот же протокол. Автономные системы обмениваются информацией между собой, опять же с применением унифицированного внешнего протокола маршрутизации. Проблема масштабирования решается так: маршрутизатору, расположенному внутри автономной системы, требуется информация лишь о других маршрутизаторах, работающих в этой автономной системе. Проблема административной автономии также легко решается, поскольку организация в своих пределах может использовать для внутренней маршрутизации любой протокол на свой выбор. Однако любые две взаимосвязанные автономные системы должны применять один и тот же протокол внешней маршрутизации, по которому они будут обмениваться информацией о достижимости сетей.

В следующем разделе мы изучим два протокола внутренней маршрутизации в автономных системах (это будут протоколы RIP и OSPF) и протокол внешней маршрутизации BGP, используемые в современном Интернете. Эти практические примеры красиво завершат наш разговор об иерархической маршрутизации.

4.6. Маршрутизация в Интернете

Теперь, когда мы изучили вопросы адресации в Интернете и протокол IP, обсудим протоколы маршрутизации в Интернете, определяющие маршрут следования дейтаграммы от отправителя к получателю.

Мы увидим, что в протоколах маршрутизации в Интернете реализованы многие принципы, с которыми мы познакомились раньше в этой главе. Алгоритм маршрутизации, основанный на состоянии каналов, и дистанционно-векторный алгоритм, рассмотренные в разделах 4.5.1 и 4.5.2, а также автономные системы, упомянутые в разделе 4.5.3, занимают центральное место при обеспечении маршрутизации в сегодняшнем Интернете.

Как рассказывалось в разделе 4.5.3, множество маршрутизаторов, находящихся под общим административным и техническим управлением и использующих один и тот же протокол маршрутизации, называется автономной системой. Каждая автономная система, в свою очередь, как правило, состоит из нескольких подсетей (здесь подсеть — это IP-сеть, описанная в разделе 4.4.2).

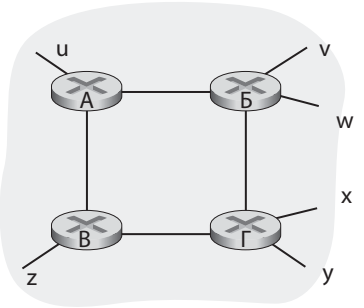
4.6.1. Протоколы внутренней маршрутизации в Интернете: RIP

Протоколы внутренней маршрутизации используются для определения маршрутов внутри автономной системы. Эти протоколы также называют внутренними или **внутришлюзовыми протоколами** (Interior Gateway Protocol, **IGP**). Исторически в качестве протоколов внутренней маршрутизации в Интернете широко применяются два протокола: **протокол маршрутной информации** (Routing Information Protocol, **RIP**) и **открытый протокол выбора кратчайшего маршрута** (Open Shortest Path First, **OSPF**). С OSPF тесно связан еще один протокол маршрутизации — **IS-IS**[438; 390]. Сначала мы обсудим протокол RIP, а затем — OSPF.

Протокол RIP был одним из первых протоколов внутренней маршрутизации, применявшихся в Интернете; он и в наши дни по-прежнему популярен. Своим происхождением и названием он обязан архитектуре XNS (Xerox Network Systems). Широкое распространение протокола RIP объяснялось в первую очередь тем, что он был включен в версию 1982 года операционной системы Berkeley UNIX, поддерживающей стек протоколов TCP/IP. Протокол RIP версии 1 определен в RFC 1058, обратно совместимая версия 2 этого протокола определена в RFC 2453.

Протокол RIP работает по дистанционно-векторному алгоритму и очень напоминает идеализированный протокол, рассматривавшийся нами в подразделе «Алгоритм дистанционно-векторной маршрутизации» раздела 4.5.2. Версия протокола RIP, специфицированная в RFC

1058, в качестве единиц измерения стоимости маршрутов использует количество ретрансляционных участков, то есть стоимость каждого канала считается равной 1. В алгоритме DV, рассмотренном в разделе 4.52, мы ради простоты изложения использовали стоимость канала между парой маршрутизаторов. В протоколе RIP (а также в OSPF) стоимость рассчитывается иначе: от исходного хоста до подсети назначения. При работе с RIP используется понятие «*переход*» (hop), означающий количество подсетей, которые требуется обойти для попадания с исходного маршрутизатора в подсеть назначения (включая подсеть назначения). На рис. 4.34. показана автономная система с шестью ответвляющимися подсетями. Таблица, сопровождающая рисунок, указывает количество переходов от начального хоста А до каждой из ответвляющихся подсетей.



Назначение	Переходы
u	1
v	2
w	2
x	3
y	3
z	2

Рис. 4.34. Количество переходов от исходного маршрутизатора А до различных подсетей

Максимальная стоимость пути ограничена значением 15, таким образом, диаметр автономной системы, поддерживаемой протоколом RIP, не может превышать 15 переходов. Вспомним также, что в дистанционно-векторных протоколах соседние маршрутизаторы обмениваются друг с другом информацией о маршрутах. В протоколе RIP обмен новыми сведениями между соседними маршрутизаторами происходит приблизительно через каждые 30 с, для чего используются так называемые **ответные RIP-сообщения**. Ответное RIP-сообщение, посылаемое маршрутизатором или хостом, содержит список, в котором указаны до 25 сетей-адресатов в пределах автономной системы, а также расстояния до каждой из этих сетей от отправителя. Такие сообщения также иногда называют **RIP-объявлениями**.

Рассмотрим работу RIP-объявлений на простом примере. На рис. 4.35 показан фрагмент автономной системы. Линии, соединяющие маршрутизаторы, обозначают сети. Для удобства мы будем рассматри-

вать только несколько выделенных маршрутизаторов (*А*, *Б*, *В* и *Г*) и сетей (*w*, *x*, *y* и *z*). Пунктирные линии означают, что автономная система не ограничивается помеченными маршрутизаторами и сетями, а распространяется дальше.

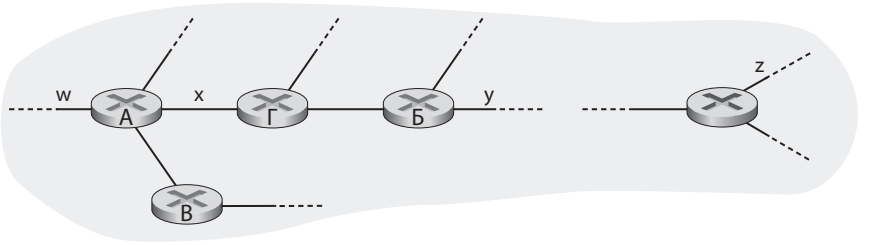


Рис. 4.35. Часть автономной системы

Каждый маршрутизатор ведет RIP-таблицу, которая именуется **таблицей маршрутизации**. Такая таблица одновременно содержит и дистанционный вектор, и таблицу перенаправления. На рис. 4.36 показана таблица маршрутизации для маршрутизатора Г.

Подсеть назначения	Следующий маршрутизатор	Количество переходов до места назначения
w	А	2
y	Б	2
z	Б	7
x	—	1
...	...	...

Рис. 4.36. Таблица маршрутизации в маршрутизаторе *Г* до получения объявления от маршрутизатора *А*

Обратите внимание, что в этой таблице три столбца. В первом столбце указана сеть-адресат, во втором — идентификатор следующего маршрутизатора на кратчайшем пути к сети-адресату, в третьем — количество переходов до сети-адресата на кратчайшем пути, то есть количество сетей, отделяющих отправителя от получателя, включая сеть-адресат. Видно, что для отправки дейтаграммы от маршрутизатора *Г* в сеть *w* дейтаграмму сначала нужно переправить соседнему маршрутизатору *А*. Кроме того, сеть-адресат w находится на расстоянии двух ретрансляционных участков по самому кратчайшему пути. Аналогично, до сети *z* семь переходов, начиная с маршрутизатора *Б*. В принципе, в таблице

перенаправления содержится по одной строке для каждой сети в автономной системе, хотя версия 2 протокола RIP допускает агрегацию маршрутов к сетям при помощи метода, схожего с теми, которые мы рассматривали в разделе 4.4.

Таким образом, табл. на рис. 4.36 и последующие таблицы перенаправления, представленные в этом подразделе, заполнены только частично.

Теперь предположим, что 30 с спустя маршрутизатор *Г* получает от маршрутизатора *А* объявление, показанное на рис. 4.37. Обратите внимание, что это объявление представляет собой не что иное, как информацию из таблицы перенаправления маршрутизатора *А*! Эта информация указывает, в частности, что сеть *z* находится на расстоянии всего четырех переходов от маршрутизатора *А*. Получив это объявление, маршрутизатор *Г* объединяет его (рис. 4.37) со старой таблицей маршрутизации (рис. 4.36) В частности, маршрутизатор *Г* узнает, что теперь появился путь от маршрутизатора *А* до сети *z*, который короче, чем путь через маршрутизатор *Б*. Таким образом, маршрутизатор *Г* обновляет свою таблицу перенаправления, чтобы учесть более короткий путь, как показано на рис. 4.38. Как же так, возможно, спросите вы, кратчайший путь к сети *z* стал еще короче? Возможно, децентрализованный дистанционно-векторный алгоритм все еще находился в процессе схождения (см. раздел 4.5.2) или стали известны новые каналы и/или маршрутизаторы, в результате чего в сети между автономными системами появились новые кратчайшие маршруты.

Рассмотрим теперь некоторые аспекты реализации протокола RIP. Вспомним, что использующие этот протокол маршрутизаторы обмениваются объявлениями приблизительно раз в 30 с. Если маршрутизатор не получает пакетов от своего соседа в течение 180 с, он решает, что данный сосед более недоступен. Это может означать, что сосед вышел из строя или выключен, либо вышла из строя линия связи между ними.

Подсеть назначения	Следующий маршрутизатор	Количество переходов до места назначения
z	B	4
w	—	1
x	—	1
...	...	...

Рис. 4.37. Объявление от маршрутизатора *А*

Подсеть назначения	Следующий маршрутизатор	Количество переходов до места назначения
w	А	2
y	Б	2
z	А	5
...	...	...

Рис. 4.38. Таблица маршрутизации маршрутизатора *Г* после получения объявления от маршрутизатора *А*

Когда такое происходит, протокол RIP изменяет локальную таблицу перенаправления, а затем распространяет эту информацию, рассылая объявления соседним маршрутизаторам (тем, которые все еще доступны). Маршрутизатор может также запросить у соседа информацию о стоимости маршрута от него до заданного адресата при помощи RIP-запроса. Маршрутизаторы пересылают друг другу RIP-запросы и RIP-ответы в UDP-пакетах через порт номер 520. UDP-пакет переносится между маршрутизаторами в стандартном IP-пакете. Тот факт, что протокол RIP пользуется протоколом транспортного уровня (UDP) поверх протокола сетевого уровня (IP), может показаться излишне сложным (так оно и есть!). Чтобы прояснить данный вопрос, необходимо несколько углубиться в реализацию протокола RIP.

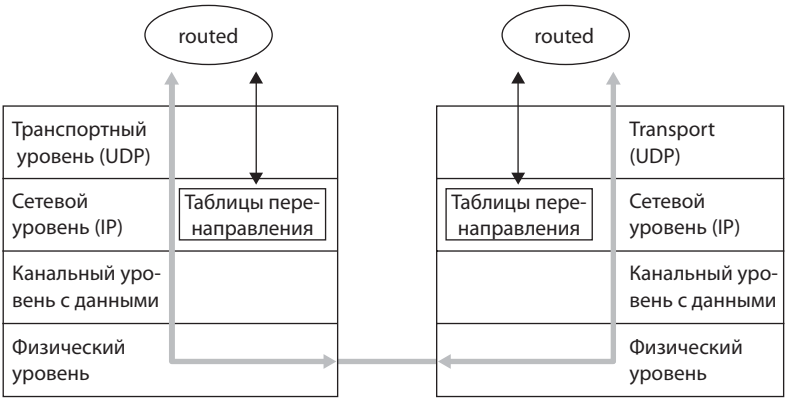


Рис. 4.39. Реализация протокола RIP в виде процесса routed

На рис. 4.39 изображена схема типичной реализации протокола RIP в операционной системе UNIX, например на рабочей станции, играющей роль маршрутизатора. Протокол RIP исполняет процесс, называемый *routed*; он обрабатывает информацию о маршрутах и обменивается со-

общениями с *такими же* процессами, работающими на соседних маршрутизаторах. Поскольку протокол RIP реализован как процесс прикладного уровня (хотя и весьма специфический — например, он способен управлять таблицами перенаправления, находящимися в ядре операционной системы UNIX), он может отправлять и получать сообщения через стандартный сокет и использовать стандартный транспортный протокол. Таким образом, протокол RIP представляет собой протокол прикладного уровня (см. главу 2), работающий поверх протокола UDP. Если вас интересуют детали реализации протокола RIP (либо протоколов OSPF и BGP, которые мы рассмотрим ниже), см. пакет Quagga[401].

4.6.2. Протоколы внутренней маршрутизации в Интернете: OSPF

Как и RIP, открытый протокол выбора кратчайшего маршрута (OSPF) используется для маршрутизации внутри автономной системы. Протокол OSPF и очень похожий на него протокол IS-IS обычно внедряются у интернет-провайдеров верхнего уровня, тогда как RIP характерен для провайдеров более низкого уровня и для корпоративных сетей. Слово «открытый» в названии протокола означает, что спецификация протокола маршрутизации свободно распространяется (в отличие от, к примеру, спецификации протокола EIGRP корпорации Cisco). Последняя (вторая) версия протокола OSPF определена в общедоступном документе RFC 2328[474].

Протокол OSPF считается преемником протокола RIP и обладает рядом дополнительных функций. Однако по своей сути OSPF представляет собой протокол, основанный на учете состоянии каналов и использующий метод лавинной рассылки для распространения информации о состоянии каналов, а также алгоритм определения пути наименьшей стоимости Дейкстры. Маршрутизатор, работающий по протоколу OSPF, формирует полную топологическую карту (направленный граф) всей *автономной системы*. Затем маршрутизатор локально запускает алгоритм определения кратчайшего пути Дейкстры, чтобы найти дерево кратчайших путей ко всем сетям автономной системы. Далее из этого дерева кратчайших путей формируется таблица перенаправления маршрутизатора. Стоимости каналов настраиваются администратором вычислительной сети (см. врезку далее в этой главе). Администратор может установить стоимости всех каналов равными 1, в результате путь наименьшей стоимости совпадет с кратчайшим путем, или установить весовой коэффициент каждой линии обратно пропорциональным про-

пускной способности линии, чтобы маршрутизаторы старались избегать линий с низкой пропускной способностью. Протокол OSPF не занимается определением стоимости каналов (это работа администратора вычислительной сети), а лишь предоставляет механизмы (протокол) определения пути наименьшей стоимости для заданного набора стоимостей каналов.

Маршрутизатор, работающий по протоколу OSPF, путем широковещательной рассылки переправляет информацию о маршрутах *всем* маршрутизаторам автономной системы, а не только соседним. Маршрутизатор рассылает всем информацию о состоянии каналов при каждом изменении состояния какого-либо из них (например, при обновлении стоимости или включении/отключении). Он также рассылает информацию о состоянии линий периодически (по меньшей мере, раз в 30 мин), даже если состояние линии не изменилось. В стандарте RFC 2328[474] отмечается, что «эти периодические объявления о состояниях каналов увеличивают устойчивость алгоритма, основанного на состоянии каналов». Объявления протокола OSPF содержатся в OSPF-сообщениях, напрямую переносимых IP-дейтаграммами, в поле протокола верхнего уровня которых протокол OSPF обозначается кодом 89. Таким образом, протокол OSPF должен сам заниматься такими вопросами, как надежность передачи сообщений и широковещательная рассылка информации о состоянии каналов. Протокол OSPF также проверяет работоспособность каналов (при помощи сообщения HELLO, посылаемого соседу) и позволяет маршрутизатору OSPF получать информацию из базы данных соседнего маршрутизатора о состоянии каналов всей сети.

Ниже перечислены достоинства протокола OSPF.

- *Безопасность*. Весь обмен данными между OSPF-маршрутизаторами (например, новые сведения о состоянии каналов) подвергается процедуре аутентификации. Это означает, что только доверенные маршрутизаторы могут принимать участие в работе протокола OSPF в пределах домена, не позволяя, таким образом, злоумышленникам (или студентам, применяющим полученные знания для развлечений) вставлять в таблицы маршрутизации некорректную информацию. По умолчанию OSPF-пакеты, курсирующие между маршрутизаторами, не аутентифицируются, и их можно подделать. Администратор может сконфигурировать аутентификацию одного из двух типов: простую и с применением MD5 (подробнее о механизме MD5 и аутентификации вообще мы поговорим в главе 8). При простой аутентификации на всех маршрутизаторах устанавливают-

ся одинаковые пароли. Когда маршрутизатор отсылает OSPF-пакет, он включает в этот пакет пароль, записанный обычным текстом. Разумеется, безопасность при такой аутентификации оставляет желать лучшего. Аутентификация MD5 основана на использовании разделяемых секретных ключей, которые конфигурируются на каждом из маршрутизаторов. Для каждого OSPF-пакета, отсылаемого маршрутизатором, маршрутизатор вычисляет MD5-хэш содержимого пакета OSPF, к которому прикрепляется секретный ключ (подробнее о кодах аутентификации сообщений мы поговорим в главе 8). Затем маршрутизатор включает результирующее хеш-значение в пакет OSPF. Маршрутизатор-получатель при помощи заранее сконфигурированного секретного ключа, в свою очередь, вычисляет хэш пакета и сравнивает его с тем хэш-значением, которое уже содержится в пакете. Так проверяется подлинность пакета. Кроме того, при аутентификации по принципу MD5 используются последовательные номера, помогающие защититься от атак с повторным воспроизведением.

- *Несколько путей с одинаковой стоимостью.* Когда у нескольких маршрутов к одному адресату оказывается одинаковая стоимость, протокол OSPF позволяет использовать все (нет проблемы выбора одного из этих маршрутов).
- *Интегрированная поддержка одноадресной и групповой маршрутизации.* Многоадресный вариант протокола OSPF (MOSPF)[449] предоставляет простые расширения для OSPF, обеспечивающие групповую маршрутизацию (этой темы мы подробнее коснемся в разделе 4.7.2). MOSPF использует имеющуюся базу данных по каналам OSPF и добавляет новый тип объявлений о состоянии каналов к существующему в OSPF механизму широковещательного объявления о состоянии каналов.
- *Поддержка иерархичности в отдельно взятом домене маршрутизации.* Вероятно, самое существенное нововведение в OSPF — это возможность иерархически упорядочивать автономные системы. В разделе 4.5.3 уже были рассмотрены некоторые сложные иерархические структуры. В оставшейся части этого раздела рассматривается реализация иерархической маршрутизации в протоколе OSPF.

Автономная система OSPF может быть иерархически структурирована в виде зон. В каждой зоне выполняется собственный экземпляр алгоритма OSPF с учетом состояния каналов, каждый маршрутизатор широковещательно информирует о состоянии собственных каналов другие

маршрутизаторы своей зоны. В каждой зоне один или несколько **граничных маршрутизаторов зоны** занимаются перенаправлением пакетов за ее пределы. Наконец, ровно одна OSPF-зона в автономной системе конфигурируется как **магистральная**. Основная роль магистральной области заключается в том, чтобы направлять трафик между разными зонами автономной системы. Магистральная зона всегда включает в себя все граничные маршрутизаторы других зон автономной системы, а также может включать и негранничные маршрутизаторы. Межзонная маршрутизация в рамках автономной системы требует, чтобы пакет сначала был направлен на граничный маршрутизатор зоны (внутризонная маршрутизация), затем попал в магистральную зону и оттуда был переправлен на другой граничный маршрутизатор, который уже относится к зоне назначения. Этот граничный маршрутизатор уже отправляет пакет куда следует.

ПРИНЦИПЫ В ДЕЙСТВИИ

Настройка весовых коэффициентов в OSPF

Выше при обсуждении состояния каналов предполагалось, что для них уже установлены весовые коэффициенты, алгоритм маршрутизации запущен, потоки трафика идут в соответствии с информацией, заложенной в таблицах перенаправления, которые рассчитаны на основе алгоритма с учетом состояния каналов. Если говорить о причинно-следственных связях, весовые коэффициенты каналов сначала задаются, а потом результируют (по алгоритму Дейкстры) в готовые пути маршрутизации, что позволяет минимизировать общую стоимость. С такой точки зрения можно сказать, что весовые коэффициенты каналов отражают стоимость использования канала (то есть если они обратно пропорциональны пропускной способности, то при использовании каналов с высокой пропускной способностью вес таких каналов окажется меньше; соответственно, они будут более привлекательны с точки зрения маршрутизации). Для снижения общей стоимости применяется алгоритм Дейкстры.

На практике причинно-следственные связи между весом каналов и путями маршрутизации могут быть обратными: администратор вычислительной сети конфигурирует вес каналов, чтобы получить такие пути маршрутизации, которые отвечают определенным инженерным целям[169; 170]. Например, допустим, что у оператора сети есть оценка трафика, поступающего в систему на каждой точке входа и покидающего ее на точке выхода. Затем оператор может задействовать при маршрутизации такие потоки, которые пролегают между определенными точками входа и выхода и сводят к миниму-

му активность использования всех сетевых каналов. Но когда задействуется алгоритм маршрутизации — например, по протоколу OSPF — основными рычагами оператора для настройки потоков маршрутизации в сети становятся именно весовые коэффициенты каналов. Соответственно, чтобы свести к минимуму использование каналов, оператор должен подобрать для них подходящие весовые коэффициенты. Как видим, причинно-следственные связи стали обратными — известны желаемые маршруты потоков, после чего требуется найти в OSPF такие каналы, при выборе которых алгоритм маршрутизации OSPF приведет к формированию таких маршрутов.

OSPF — сравнительно сложный протокол, здесь мы были вынуждены рассмотреть его очень кратко. Более подробно об этом протоколе можно узнать в публикациях Хайтема[213], Мойя[356] и документе RFC 2328[474].

4.6.3. Маршрутизация между автономными системами: протокол BGP

Выше было рассмотрено, как провайдеры Интернета используют протоколы RIP и OSPF для выстраивания оптимальных пар исходных и конечных узлов, расположенных в пределах одной и той же автономной системы. Теперь давайте обсудим, как определяются подобные пары, не ограниченные одной автономной системой. **Протокол граничных маршрутизаторов (BGP)** версии 4, описанный в стандарте RFC 4271 (см. также RFC 4274), *де-факто* является в современном Интернете стандартом маршрутизации между автономными системами. Выступая в такой роли (см. раздел 4.5.3), BGP обеспечивает автономные системы средствами для:

1. Получения информации о достижимости подсетей от смежных автономных систем
2. Распространения информации о достижимости между всеми внутренними маршрутизаторами конкретной автономной системы
3. Определения «хороших» маршрутизаторов для попадания в подсети, на основе информации о достижимости и политики, применяемой в автономной системе.

Наиболее важно, что по протоколу BGP любая подсеть может объявить о своем существовании всему Интернету. Подсеть «кричит»: «Это я, я здесь!», а BGP гарантирует, что все автономные системы в Интер-

нете узнают о существовании этой подсети и о том, как в нее попасть. Если бы протокола BGP не существовало, то каждая подсеть в Интернете оставалась бы изолирована — во всем остальном Интернете о ней ничего не было бы известно.

Основы BGP

Протокол BGP исключительно сложен; теме работы с ним посвящены целые книги, а многие проблемы до сих пор не вполне понятны[678]. Более того, даже прочитав специализированные книги и стандарты RFC, вы, вероятно, не сможете освоить работу с протоколом BGP — чтобы овладеть этим искусством, нужно несколько месяцев (а то и лет) проработать архитектором сетей или администратором вычислительной сети в компании-провайдере Интернета верхнего уровня. Тем не менее, поскольку BGP является критически важным протоколом Интернета — в сущности, именно BGP склеивает воедино всю эту глобальную сеть — мы хотим дать читателю хотя бы приблизительное представление о том, как этот протокол работает. Для начала изучим работу BGP на примере простой иллюстративной сети, которая уже рассматривалась выше, на рис. 4.32. В этом разделе мы будем опираться на описание иерархической маршрутизации, о которой речь шла выше в разделе 4.5.3, рекомендуем перечитать этот материал.

При взаимодействии по протоколу BGP пары маршрутизаторов обмениваются информацией по полупостоянным TCP-соединениям через порт 179. Полупостоянные TCP-соединения для сети с рис. 4.32 изображены на рис. 4.40. Как правило, существует по одному такому TCP-соединению протокола BGP на каждый канал, который непосредственно связывает два маршрутизатора из разных автономных систем. Соответственно, на рис. 4.40 видим два TCP-соединения: между шлюзовыми маршрутизаторами 3а и 1в и между шлюзовыми маршрутизаторами 1б и 2а. Кроме того, полупостоянные TCP-соединения по протоколу BGP имеются и между маршрутизаторами одной автономной системы. В частности, на рис. 4.40 представлена типичная конфигурация TCP-соединений для каждой пары внутренних маршрутизаторов в автономной системе. Таким образом, в автономной системе создается сеть TCP-соединений. Два маршрутизатора, расположенные на концах TCP-соединения, называются **BGP-партнерами** (BGP-пирами) или «равноправными маршрутизаторами», а TCP-соединение вместе со всеми передаваемыми по нему BGP-сообщениями называется **BGP-сеансом**. Далее: BGP-сеанс, охватывающий две автономные системы, называется

внешним BGP-сеансом (eBGP), а сеанс, ограниченный рамками одной автономной системы, называется **внутренним BGP-сеансом (iBGP)**. На рис. 4.40 сеансы eBGP обозначены длинными прерывистыми линиями, а сеансы iBGP — короткими прерывистыми линиями. Обратите внимание: линии BGP-сеансов на рис. 4.40 не всегда соответствуют физическим каналам, обозначенным на рис. 4.32.

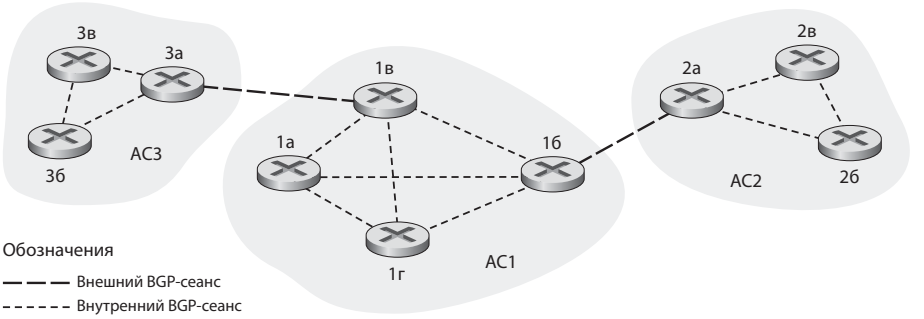


Рис. 4.40. Сеансы eBGP и iBGP

ПРИНЦИПЫ В ДЕЙСТВИИ

Виртуальное представительство в Интернете: решение проблемы

Допустим, вы создали небольшую сеть, в которой есть несколько серверов, в том числе публичный веб-сервер, где любой желающий может получить информацию о товарах и услугах вашей компании, почтовый сервер, где обслуживается электронная переписка ваших сотрудников, а также DNS-сервер. Естественно, вы заинтересованы, чтобы пользователи со всего мира могли заходить на ваш сайт и знакомиться с вашими замечательными предложениями. Более того, требуется, чтобы сотрудники могли вести электронную переписку с потенциальными клиентами со всего мира.

Для достижения этих целей нужно первым делом наладить Интернет-соединение. Для этого заключается договор с Интернет-провайдером, действующим в вашем регионе. Ваша компания получит шлюзовой маршрутизатор, который будет подключен к маршрутизатору этого локального провайдера. Возможны следующие варианты подключения: DSL-соединение поверх имеющейся инфраструктуры телефонной сети, выделенная линия на маршрутизатор вашего Интернет-провайдера, либо одно из многочисленных иных решений, описанных в главе 1. Кроме того, локальный Интернет-провайдер предоставит вам диапазон адресов, например a/24, со-

стоящий из 256 адресов. Таким образом, приобретя физическое соединение и диапазон IP-адресов, вы присваиваете один из них веб-серверу, другой — почтовому серверу, третий — DNS-серверу, четвертый — шлюзу, а все остальные — хостам и сетевым устройствам из вашей корпоративной сети.

Вам потребуется не только заключить договор с Интернет-провайдером, но и связаться с Интернет-регистратором, чтобы получить доменное имя для вашей компании — этот процесс описан в главе 2. Например, если ваша компания называется Xanadu Inc, то вы, естественно, захотите получить доменное имя xanadu.com. Также вашей компании нужно получить представительство в системе DNS. Поскольку посетители будут пытаться установить контакт с вашим DNS-сервером для получения IP-адресов других ваших серверов, вы будете должны, в частности, сообщить Интернет-регистратору IP-адрес вашего DNS-сервера. Регистратор сделает в своей базе данных запись о вашем DNS-сервере (доменное имя и соответствующий ему IP-адрес), эта запись будет относиться к числу доменов верхнего уровня .com (как описано в главе 2). После того как вы справитесь с этим этапом работы, любой пользователь, знающий доменное имя вашего сайта (здесь — xanadu.com) сможет получить IP-адрес вашего DNS-сервера через систему DNS.

Так, чтобы люди могли узнать адреса вашего веб-сервера, вы должны будете разместить на вашем DNS-сервере записи, ассоциирующие хост-имя веб-сервера (здесь — xanadu.com) с его IP-адресом. Понадобится обзавестись подобными записями для всех других общедоступных серверов вашей компании, в том числе, для веб-сервера. Таким образом, если Алиса хочет просмотреть страницы с вашего веб-сервера, система DNS свяжется с вашим DNS-сервером, найдет там адрес веб-сервера и передаст его Алисе. Затем Алиса сможет установить прямое TCP-соединение с вашим веб-сервером.

Остается еще один важный обязательный этап, который необходимо выполнить, чтобы посетители со всего мира получили доступ к вашему веб-серверу. Рассмотрим следующий случай: Алиса, знающая IP-адрес вашего веб-сервера, отсылает IP-дейтаграмму (то есть сегмент TCP SYN) на этот IP-адрес. Данная дейтаграмма будет направлена через Интернет, пройдя через ряд маршрутизаторов, относящихся к различным автономным системам и рано или поздно достигнет вашего веб-сервера. Когда любой маршрутизатор получит ее, он возьмет из своей таблицы перенаправления нужную запись, чтобы определить, на какой исходящий порт он должен передать эту дейтаграмму. Следовательно, все маршрутизаторы должны знать, что диапазон адресов вашей компании имеет префикс /24 (или какую-либо суммарную запись). Как маршрутизатор

может узнать префикс вашей компании? Мы уже убедились, что он получает эту информацию по протоколу BGP! В тот момент, когда ваша компания связывается с локальным Интернет-провайдером и получает префикс (то есть диапазон адресов), этот провайдер использует BGP для объявления данного префикса тем провайдерам, с которыми он связан. Эти провайдеры, в свою очередь, используют протокол BGP для дальнейшего распространения этого объявления. Рано или поздно ваш префикс (или суммарная запись, включающая этот префикс) становится известен маршрутизаторам во всем Интернете. Маршрутизаторы смогут правильно пересылать дейтаграммы, адресованные на ваш веб-сервер и почтовый сервер.

Протокол BGP позволяет каждой автономной системе узнать, какие места достижимы через ее соседние автономные системы. В протоколе узлы назначения обозначаются не в виде хостов, а в виде CIDR-**префиксов**. Каждый префикс представляет собой подсеть или коллекцию подсетей. Итак, давайте предположим, что к автономной системе AC2 подключены следующие подсети: 138.16.64/24, 138.16.65/24, 138.16.66/24 и 138.16.67/24. Система AC2 сможет суммировать префиксы всех четырех этих подсетей и объявить автономной системе AC1 единый префикс для доступа к ним: 138.16.64/22. Другой пример: допустим, что только три первые из вышеупомянутых подсетей относятся к автономной системе AC2, а четвертая подсеть, 138.16.67/24, находится в AC3. Затем, как было описано во врезке «Принципы и практика» к разделу 4.4.2, поскольку маршрутизаторы используют при перенаправлении дейтаграмм совпадение наиболее длинного префикса, AC3 может объявить AC1 более конкретизированный префикс 138.16.67/24, а AC2 *по-прежнему* может объявить AC1 суммарный префикс 138.16.64/22.

Теперь давайте обсудим, как протокол BGP распространял бы информацию о достижимости подсетей с теми или иными префиксами в рамках сеансов, показанных на рис. 4.40. Как вы уже догадываетесь, установив сеанс eBGP между шлюзовыми маршрутизаторами 3а и 1в, AC3 пошлет AC1 список тех префиксов, которые достижимы из автономной системы AC1. Аналогично, автономные системы AC1 и AC2 обмениваются информацией о достижимости префиксов через свои шлюзовые маршрутизаторы 1б и 2а. Также, как вы понимаете, когда шлюзовой маршрутизатор (в любой автономной системе) получает префиксы, выясненные в ходе eBGP-сеанса, он использует свой iBGP-сеанс для распространения информации о префиксах между другими

маршрутизаторами своей автономной системы. Итак, все маршрутизаторы AC1, включая шлюзовой 1б, узнают префиксы подсетей из AC3. Шлюзовой маршрутизатор 1б (в AC1) может повторно объявить префиксы AC3 в автономной системе AC2. Когда маршрутизатор (шлюзовой или обычный) узнает о новом префиксе, он создает запись для этого префикса в своей таблице перенаправления, как это описано в разделе 4.5.3.

Атрибуты пути и маршруты BGP

Итак, мы получили предварительное представление о BGP, теперь давайте исследуем его чуть подробнее (правда, все равно придется опускать некоторые второстепенные детали). Протокол BGP распознает автономные системы (RFC 1930) по глобально уникальным **номерам автономных систем** (Autonomous System Number, **ASN**). На практике не у каждой автономной системы есть номер. В частности, так называемая тупиковая автономная система (автономная система-заглушка), переносящая только трафик, для которого она же является отправителем или получателем, как правило, не имеет номера. В нашем обсуждении мы опустим эти детали. Номера автономных систем, как и IP-адреса, назначаются региональными организациями ICANN[225].

Когда маршрутизатор объявляет префикс в рамках BGP-сеанса, вместе с префиксом в это объявление включается ряд **BGP-атрибутов**. В терминологии BGP совокупность префикса и всех его атрибутов называется **маршрутом**. Соответственно, BGP-партнеры объявляют между собой маршруты друг другу. Остановимся на двух наиболее важных атрибутах: AS-PATH и NEXT-HOP.

- *AS-PATH*: в этом атрибуте указываются автономные системы, через которые прошло объявление префикса. Когда префикс передается через автономную систему, она добавляет свой номер ASN к атрибуту AS-PATH. Например, вернемся к рис. 4.40 и предположим, что префикс 138.16.64/24 сначала объявляется системой AC2 системе AC1. Если после этого AC1 объявит префикс системе AC3, то при этом будет использоваться атрибут AS-PATH со значением AC2 AC1. Маршрутизаторы применяют атрибут AS-PATH для обнаружения и предотвращения петлевых объявлений. Так, если маршрутизатор обнаружит, что его автономная система уже упомянута в атрибуте AS-PATH, он отклонит такое объявление. Вскоре мы поговорим о том, как маршрутизаторы применяют атрибут AS-PATH при выборе одного из многих возможных путей к искомому префиксу.

Атрибут NEXT-HOP обеспечивает критически важную связь между внутренними и внешними протоколами маршрутизации, обслуживающими автономные системы. При всей малозаметности этот атрибут просто незаменим.

- *NEXT-HOP* соответствует маршрутизатору-интерфейсу, с которого начинается путь *AS-PATH*. Чтобы лучше познакомиться с этим атрибутом, вновь обратимся к рис. 4.40. Рассмотрим, что происходит, когда шлюзовой маршрутизатор 3а в автономной системе АС3 объявляет путь к шлюзу 1 в автономной системе АС1 в рамках сеанса eBGP. В маршруте содержится объявляемый префикс, который мы обозначим через *x*, а также атрибут AS-PATH, описывающий путь к префиксу. В этом объявлении есть и атрибут NEXT-HOP, представляющий собой IP-адрес интерфейса маршрутизатора 3а, ведущего к 1в. Как вы помните, у маршрутизатора несколько IP-адресов, по одному для каждого из его интерфейсов. Теперь давайте рассмотрим, что произойдет, когда маршрутизатор 1г узнает о данном маршруте в рамках сеанса iBGP. Узнав об этом маршруте к *x*, маршрутизатор 1г может выбрать именно его для пересылки пакетов к x — то есть включить в свою таблицу перенаправления запись (*x*, *l*). Здесь *l* — это интерфейс данного маршрутизатора, с которого начинается путь наименьшей стоимости от 1г к шлюзовому маршрутизатору 1в. Для определения *l* маршрутизатор 1г сообщает внутреннему модулю своей автономной системы нужный IP-адрес, это делается в атрибуте NEXT-HOP. Обратите внимание: внутренний алгоритм маршрутизации данной автономной системы определяет путь наименьшей стоимости ко всем подсетям, подключенным к маршрутизаторам АС1. В их числе оказывается и та подсеть, к которой относится канал связи между 1в и 3а. На основании этого пути с наименьшей стоимостью, ведущего от 1г к подсети с каналом 1в-3а, маршрутизатор 1г и определяет интерфейс *l*, с которого начинается данный путь. Затем запись (*x*, *l*) заносится в таблицу перенаправления. Итак, атрибут NEXT-HOP нужен маршрутизаторам, чтобы правильно заполнять их таблицы перенаправления.

На рис. 4.41 проиллюстрирована иная ситуация, в которой требуется использовать атрибут NEXT-HOP. На этом рисунке автономные системы АС1 и АС2 соединены двумя равноправными каналами. Маршрутизатор в системе АС1 может узнать о двух разных маршрутах, ведущих к одному и тому же префиксу *x*. У двух этих маршрутов могут быть одинаковые атрибуты AS-PATH, но разные значения NEXT-HOP, соответствующие разным равноправным каналам. При помощи значений NEXT-HOP

и внутреннего алгоритма маршрутизации автономной системы маршрутизатор может определить стоимость пути по каждому из равноправных каналов, а затем применить маршрутизацию по методу «горячей картофелины» (см. раздел 4.5.3) для подбора подходящего интерфейса.

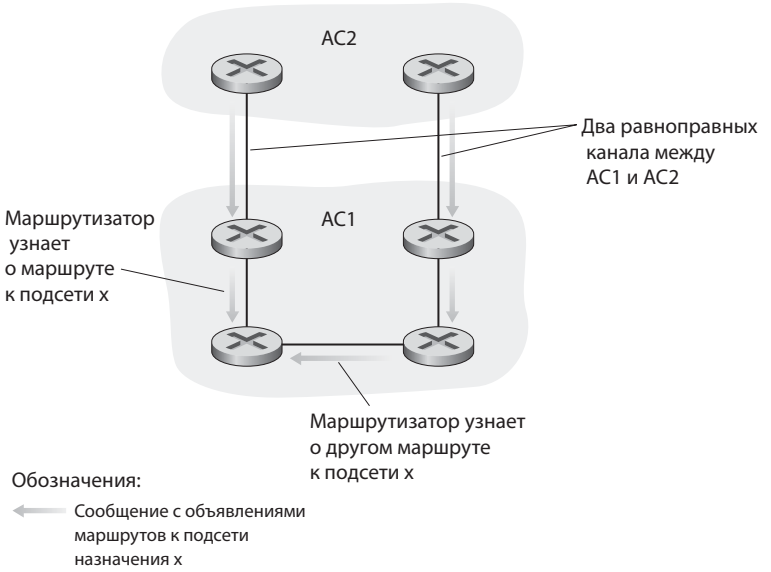


Рис. 4.41. Атрибуты NEXT-HOP при объявлениях используются для определения того, какой из равноправных каналов использовать

Кроме того, в состав протокола BGP входят атрибуты, позволяющие маршрутизаторам присваивать определенные приоритеты маршрутам, и атрибут, указывающий, каким образом префикс был внедрен в протокол BGP в исходной автономной системе. Подробное обсуждение атрибутов маршрутов дается в работах Стюарта[619], Гриффина[192], Халаби[198], Фемстера[157], и документе RFC 4271[529].

Когда шлюзовой маршрутизатор получает объявление о маршруте, он задействует свою **политику импорта** (import policy), чтобы решить, принять или пропустить этот маршрут, а также следует ли снабдить его теми или иными атрибутами, например, параметрами приоритета. Политика импорта может привести к отбрасыванию маршрута, поскольку для данной АС может быть нежелательно пересылать трафик через ту или иную АС, упомянутую в атрибуте AS-PATH этого маршрута. Шлюз также может отклонить маршрут, поскольку ему уже известен более предпочтительный путь к тому же префиксу.

Выбор маршрута при работе с протоколом BGP

Как было описано выше в этом разделе, протокол BGP использует сеансы eBGP и iBGP для распределения маршрутов между всеми маршрутизаторами конкретной автономной системы. При таком распределении маршрутизатор может узнать более чем об одном пути к любому префиксу. В таком случае маршрутизатору придется выбрать один из возможных маршрутов. Входными данными для такого процесса выбора служит информация обо всех маршрутах, которые были учтены и приняты данным маршрутизатором. Если имеется два и более маршрутов к одному и тому же префиксу, протокол BGP последовательно применяет следующие правила исключения, пока не останется всего один.

- Каждому маршруту в качестве одного из атрибутов присваивается значение локального предпочтения. Локальное предпочтение маршрута может быть задано самим маршрутизатором или получено от другого маршрутизатора из данной автономной системы. Такое решение из области политики принимается администратором вычислительной сети. Чуть ниже мы подробнее обсудим вопросы политики, действующей в BGP. Выбираются маршруты с наивысшими значениями локальных предпочтений.
- Из оставшихся маршрутов (имеющих наивысшее значение локального предпочтения) выбирается вариант с кратчайшим AS-PATH. Если бы это было единственное правило то BGP выбирал бы путь по дистанционно-векторному алгоритму, где в качестве величины расстояния учитывается количество переходов между автономными системами, а не между маршрутизаторами.
- Из оставшихся маршрутов (обладающих одинаковым значением локального предпочтения и одинаковой длиной пути согласно атрибуту AS-PATH) выбирается тот маршрут, в атрибуте NEXT-HOP которого указан ближайший маршрутизатор. В данном случае им считается тот, путь к которому самый недорогой (это определяется по внутреннему алгоритму автономной системы). Как говорилось выше в разделе 4.5.3, такой механизм называется «маршрутизацией по принципу горячей картофелины».
- Если после всех предыдущих шагов остается более одного маршрута, то для окончательного выбора маршрутизатор использует BGP-идентификаторы[619].

На практике правила исключения еще сложнее, чем те, что описаны выше. Но лучше изучать их постепенно, чтобы протокол BGP не снился вам в ночных кошмарах.

ПРИНЦИПЫ В ДЕЙСТВИИ

Все вместе: как новая запись попадает в маршрутизаторе в таблицу перенаправления?

Как вы помните, каждая запись в таблице перенаправления состоит из префикса (например, 138.16.64/22) и соответствующего выходного порта маршрутизатора (допустим, порт 7). Когда пакет прибывает в маршрутизатор, IP-адрес назначения этого пакета сравнивается с префиксами из таблицы перенаправления, чтобы найти префикс с наиболее длинным совпадением. После этого пакет пересылается (маршрутизатором) на порт маршрутизатора, ассоциированный с этим префиксом. Теперь давайте резюмируем, как запись о маршрутизации (префикс и ассоциированный с ним номер порта) попадает в таблицу перенаправления. Это простое упражнение поможет нам объединить многие знания о маршрутизации и перенаправлении, приобретенные нами на настоящий момент. Чтобы получилось интересней, предположим, что мы имеем дело с «чужим префиксом», который относится не к автономной системе маршрутизатора, а к другой автономной системе.

Чтобы префикс попал в таблицу перенаправления маршрутизатора, тот сначала должен узнать об этом префиксе (соответствующем подсети или группе подсетей). Как мы изучили выше, маршрутизатор узнает о префиксе из объявления BGP о маршруте. Такое объявление может быть передано маршрутизатору в рамках сеанса eBGP (от маршрутизатора из другой автономной системы) либо iBGP (от маршрутизатора, расположенного в той же автономной системе).

После того как маршрутизатор узнает о префиксе, он должен определить подходящий выходной порт, через который будут передаваться дейтаграммы, предназначенные для отправки в подсеть с данным префиксом. Лишь после этого маршрутизатор сможет внести такой префикс в свою таблицу перенаправления. Если маршрутизатор получит более одного объявления, связанного с данным префиксом, то он задействует применяемый в BGP процесс выбора путт, описанный выше в данном подразделе, чтобы найти наилучший маршрут к префиксу. Допустим, такой оптимальный маршрут уже выбран. Как вы уже знаете, выбранный маршрут имеет атрибут NEXT-HOP, представляющий собой IP-адрес ближайшего маршрутизатора из соседней автономной системы, связанного с исходным маршрутизатором и расположенного по оптимальному маршруту. Как было описано выше, далее маршрутизатор задействует протокол маршрутизации, применяемый внутри его автономной системы (как правило, это OSPF) для определения кратчайшего пути к маршрутизатору NEXT-HOP. Затем маршрутизатор определяет

ассоциированный с префиксом номер порта, выявив первый канал, идущий вдоль кратчайшего пути. После этого маршрутизатор может (наконец-то!) ввести в свою таблицу перенаправления пару «порт-префикс». Таблица перенаправления, рассчитываемая процессором маршрутизации (см. рис. 4.6) отправляется на входные интерфейсные карты маршрутизатора.

Политика маршрутизации

Давайте изучим основы политики BGP, применяемой при маршрутизации, на простом примере. На рис. 4.42 показаны шесть соединенных друг с другом автономных систем: *А*, *Б*, *В*, *К*, *Л* и *М*. Необходимо отметить, что *А*, *Б*, *В*, *К*, *Л* и *М* — это сети, а не маршрутизаторы.

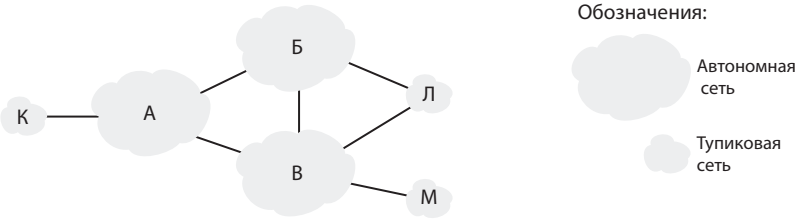


Рис. 4.42. Простой сценарий BGP

Пусть автономные системы *К*, *Л* и *М* представляют собой тупиковые сети, а автономные системы *А*, *Б* и *В* являются сетями магистральных Интернет-провайдеров. Весь попадающий в **тупиковую сеть** трафик должен предназначаться этой сети, а весь покидающий — генерироваться ее узлами. Если взглянуть на рисунок, должно быть очевидно, что сети *К* и *М* являются тупиковыми, а сеть *Л* называют **многоинтерфейсной тупиковой сетью**, так как она соединяется с остальными сетями через двух разных Интернет-провайдеров (ситуация, становящаяся все более популярной в последнее время). Однако, как и сети *К* и *М*, сеть *Л* должна быть источником/адресатом всего исходящего/входящего трафика сети *Л*. Но как реализовать и гарантировать такое поведение? Как сеть *Л* может не пропускать трафик между сетями *Б* и *В*? Этого легко добиться, контролируя способ, которым объявляются BGP-маршруты. В частности, сеть *Л* будет функционировать как тупиковая, если она объявит (своим соседям, сетям *Б* и *В*), что у нее нет путей к другим адресатам, кроме самой сети *Л*. То есть даже если сети *Л* известен маршрут до сети *М*, например *ЛВМ*, она не будет сообщать этот маршрут сети *Б*. Поскольку сеть *Б* не знает, что у сети *Л* есть маршрут до сети *М*, сеть *Б*

никогда не станет направлять трафик, предназначенный сети *M* (или *B*) через сеть *Л*. Этот простой пример иллюстрирует механизм выборочного объявления о маршрутах для реализации связанных с маршрутизацией отношений между клиентом и поставщиком услуг.

ПРИНЦИПЫ В ДЕЙСТВИИ

Почему внутри автономных систем и между ними применяются разные протоколы маршрутизации?

Изучив детали конкретных протоколов для внутренней и внешней маршрутизации, реализованных в сегодняшнем Интернете, закончим наше изучение, возможно, одним из наиболее фундаментальных вопросов: зачем нужны разные протоколы для внутренней и внешней маршрутизации? Ответ на этот вопрос кроется в различном назначении маршрутизации внутри автономных систем и между автономными системами.

- *Политика*. Среди автономных систем вопросы политики доминируют. Бывает важно, чтобы трафик, сгенерированный в некоторой автономной системе, не мог проходить через другую конкретную автономную систему. Аналогично, конкретная автономная система может пожелать контролировать транзитный трафик, переносимый между другими автономными системами. Мы видели, что протокол BGP передает атрибуты маршрутов и предоставляет возможность контролируемого распределения информации о маршрутах, что позволяет принимать политические решения о выборе маршрутов в автономной системе.
- *Масштабирование*. Способность алгоритма маршрутизации и его структур данных к масштабированию в целях поддержания большого количества сетей является ключевой для внешней маршрутизации. В пределах одной автономной системы значимость масштабирования не так велика, поскольку, если какой-либо административный домен становится слишком большим, его всегда можно разбить на две автономные системы поменьше и соединить их с помощью механизмов внешней маршрутизации. (Вспомним также, что протокол OSPF позволяет разделять автономную систему на отдельные зоны, создавая таким образом иерархическую структуру)
- *Производительность*. Поскольку вопросы политики играют во внешней маршрутизации столь важную роль, качество обслуживания в используемых маршрутах (например, производительность) часто оказывается на втором плане (то есть более длинному или дорогому маршруту, удовлетворяющему определенным полити-

ческим критериям, может быть оказано предпочтение по сравнению с более коротким маршрутом, который этим критериям не удовлетворяет). Действительно, мы видели, что при маршрутизации между автономными системами стоимость даже не упоминается (не считая количества переходов). Однако в пределах одной автономной системы подобные политические соображения не столь важны, что позволяет при выборе маршрута больше внимания уделить производительности.

Рассмотрим теперь сеть поставщика услуг, например автономную систему *Б*. Предположим, сеть *Б* узнала (от сети *А*), что у сети *А* есть путь *АК* к сети *К*. В результате сеть *Б* может добавить в свою маршрутную базу данных маршрут *БАК*. Естественно, сеть *Б* захочет сообщить о маршруте *БАК* своему клиенту *Л*, чтобы клиент *Л* знал, что он может направлять данные сети *К* через сеть *Б*. Но должна ли сеть *Б* рекламировать маршрут *БАК* сети *В*? Если она это сделает, тогда сеть *В* сможет направлять трафик в сеть *К* по маршруту *ВБАК*. Если все сети *А*, *Б* и *В* представляют собой магистральные сети Интернет-провайдеров, тогда сеть *Б* может решить, что ей нет резона взваливать на себя бремя переноса трафика между сетями *А* и *В*. Поскольку владельцы сетей *А* и *В* взимают со своих клиентов плату за пересылку данных, будет только справедливо, если они воспользуются прямым соединением между ними. Сегодня нет официальных «стандартов», определяющих, как Интернет-провайдеры должны поступать с трафиком друг друга. Однако на практике Интернет-провайдеры придерживаются простого правила, состоящего в том, что любой трафик, пересекающий магистральную сеть Интернет-провайдера, должен либо генерироваться, либо приниматься в сети, являющейся клиентом этого Интернет-провайдера. В противном случае он будет воспринимать подобный трафик как «безбилетного пассажира». Индивидуальные соглашения, регулирующие подобные спорные вопросы, как правило, заключаются парами Интернет-провайдеров и зачастую являются конфиденциальными. Интересное обсуждение подобных соглашений содержится в работе Хастона[214]. Подробное описание взаимосвязей между политикой маршрутизации и коммерческими отношениями различных Интернет-провайдеров изложено в работах Гао[178] и Дмитиропоулоса[138]. Обсуждение политик маршрутизации BGP с точки зрения Интернет-провайдера изложено в работе Цезаря[71].

Как было указано выше, протокол BGP *де-факто* является стандартом выполнения маршрутизации между автономными системами в общедоступном Интернете. Если вы хотите посмотреть (большую!)

подборку содержимого различных BGP с маршрутизаторов Интернет-провайдеров верхнего уровня, посетите сайт **www.routeviews.org**. Зачастую BGP-таблицы маршрутизации содержат десятки тысяч префиксов и соответствующих им атрибутов. Статистические данные о размерах и характеристиках BGP-таблиц маршрутизации представлены на графике на сайте **bgp.potaroo.net**[399].

На этом мы завершаем наше краткое вводное исследование протокола BGP. Понимать его важно, поскольку он играет центральную роль в Интернете. Рекомендуем вам подробнее познакомиться с этим протоколом в работах Стюарта[619], Хайтема[213], Гриффина[192], Халаби[198], Фемстера[157], Гао[178], Цезаря[71], Лабовица[302] и Ли[319].

4.7. Широковещательная и групповая маршрутизация

Выше в этой главе мы говорили в основном о таких протоколах маршрутизации, которые поддерживают одноадресное взаимодействие по двухточечному принципу («точка-точка»). В таких случаях имеется всего один хост-отправитель, отсылающий пакет только одному хосту-получателю. В этом разделе мы изучим протоколы для широковещательной и групповой маршрутизации. При **широковещательной маршрутизации** (broadcast routing) сетевой уровень обеспечивает доставку пакета от отправителя всем остальным хостам в сети. При **групповой маршрутизации** (multicast routing) один отправитель отсылает копии пакета подмножеству других хостов сети. В разделе 4.7.1 рассматриваются алгоритмы широковещательной маршрутизации и их реализация в соответствующих протоколах. Групповая маршрутизация изучается в разделе 4.7.2.

4.7.1. Алгоритмы широковещательной маршрутизации

Вероятно, простейший способ обеспечения широковещательной коммуникации таков: хост-отправитель отправляет по одной копии пакета каждому из хостов-получателей, как показано на рис. 4.43(а). Имея N получателей, исходный узел просто создает N копий пакета, посылает каждую копию на конкретный адрес при помощи одноадресной маршрутизации. Такой метод **N-адресной** широковещательной маршрутизации прост и не требует никаких новых протоколов маршрутизации сетевого уровня, дублирования пакетов или специального функционала

пересылки. Однако у этого подхода есть ряд недостатков. Первый из них — неэффективность. Если исходный узел соединен с остальной сетью единственным каналом, то *N* отдельных копий (одного и того же) пакета будут проходить по этому каналу. Разумеется, было бы более целесообразно послать одну копию пакета на расстояние одного перехода (до первого транзитного узла), а затем приказать этому транзитному узлу создать нужное количество копий пакета на основе первой и уже оттуда переслать их все куда следует. Соответственно, работа строилась бы более эффективно, если бы разные узлы сети (а не только исходный) умели создавать дополнительные копии пакета. Например, на рис. 4.43(б) по каналу M1-M2 проходит всего одна копия пакета. Затем на узле M2 этот пакет копируется, и по каналам M2-M3 и M2-M4 отсылаются полученные таким образом два экземпляра пакета.

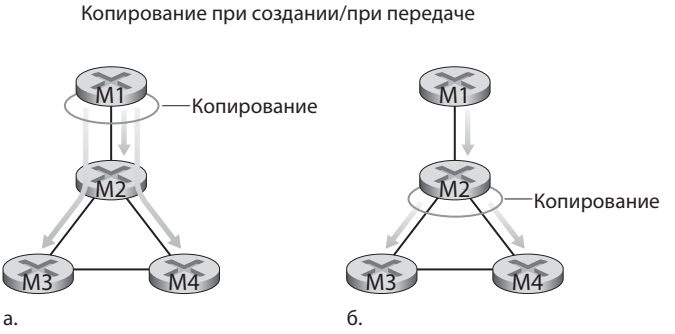


Рис. 4.43. Сравнение копирования на исходном узле и копирования на транзитных узлах

Другие недостатки N-адресной маршрутизации не столь очевидны, но не менее существенны. При N-адресной маршрутизации неявно предполагается, что отправителю известны адреса всех хостов-получателей широковещательной рассылки. Но как он получает эту информацию? Скорее всего, при этом потребуется задействовать дополнительные механизмы работы с протоколами (например, широковещательный запрос членства или протокол регистрации узлов назначения). В результате возникнет еще больше издержек и, что гораздо важнее, сам протокол значительно усложнится, тогда как в исходном варианте он казался довольно простым. Наконец, последний недостаток N-адресной маршрутизации заключается в том, для каких целей обычно применяется широковещание. В разделе 4.5 мы узнали, что протоколы маршрутизации с учетом состояния каналов применяются для широковещательного распространения информации о состоянии каналов, которая, в свою

очередь, используется для расчета одноадресных маршрутов. Разумеется, в тех случаях, когда широковещание применяется для создания и обновления одноадресных маршрутов, будет (мягко говоря!) неразумно выстраивать широковещательную передачу информации на одноадресной архитектуре.

С учетом недостатков N-адресной широковещательной маршрутизации больший интерес вызывают такие подходы, при которых сами сетевые узлы активно участвуют в размножении и перенаправлении пакетов, а также в расчете широковещательных маршрутов. Ниже мы исследуем ряд таких подходов в виде графа, который мы уже рассматривали в разделе 4.5. Итак, вновь представим сеть как граф $G = (N,E)$, где N — это множество узлов и набор ребер E, причем каждое ребро объединяет пару узлов из множества N. В нашей нотации мы допустим небольшую неаккуратность и будем обозначать через N и множество узлов, и мощность (размер) этого множества ($|N|$) в случаях, когда значение N является недвусмысленным.

Неуправляемая лавинообразная маршрутизация

Самый простой подход к широковещательной передаче данных называется **лавинообразной маршрутизацией** (флудингом). В таком случае исходный узел посылает копию пакета всем своим соседним узлам. Как только узел получает пакет, он копирует его и пересылает всем своим соседям (кроме того, от которого он получил пакет). Разумеется, если граф является связанным, то такая схема рано или поздно приведет к доставке широковещательно распространяемого пакета ко всем его узлам. Хотя эта схема проста и красива, у нее есть катастрофический недостаток (прежде чем читать далее, попробуйте описать его сами). Если в графе есть циклы, то одна или несколько копий широковещательного пакета попадут в бесконечный цикл. Например, на рис. 4.43 лавинообразная маршрутизация пойдет следующим образом: от M2 к M3, от M3 к M4, от M4 к M2, а от M2 (опять!) к M3 и т. д. В таком простом сценарии в бесконечный цикл попадают два пакета, один движется по часовой стрелке, другой — против часовой. Но возможен еще более катастрофический исход: если узел соединен более чем с двумя другими узлами, он начнет создавать и передавать множество копий пакета (которые будут только множиться на других узлах, соединенных более чем с двумя соседями) и т. д. Такое состояние называется **широковещательным штормом** и возникает по причине бесконечного размножения передаваемых пакетов. Вскоре пакетов станет настолько много, что сеть

будет буквально забита ими. В домашнем задании к этой главе есть вопросы, посвященные анализу этой проблемы и расчету скорости нарастания широковещательного шторма.

Управляемая лавинообразная маршрутизация

Чтобы не допустить возникновения широковещательного шторма, узел должен осмысленно выбирать, когда следует и когда не следует добавлять пакет в лавину (например, пакет не должен попадать в лавину, если этот узел уже получал и отправлял в лавину его копию). На практике такой механизм реализуется одним из нескольких способов.

При **контролируемой лавинообразной маршрутизации с использованием порядковых номеров** исходный узел записывает в широковещательный пакет свой адрес (или другой уникальный идентификатор), а также **порядковый широковещательный номер**. Затем узел отправляет этот пакет всем своим соседям. Каждый узел ведет список исходных адресов и порядковых номеров всех пакетов, которые он уже получил, скопировал и переслал широковещательным образом. Когда узел получает широковещательный пакет, он сначала проверяет, указан ли этот пакет в его списке. Если это так, то пакет отбрасывается; в противном случае пакет копируется и по одному его экземпляру передается всем узлам, смежным с данным (кроме того узла, от которого был получен пакет). В протоколе Gnutella, рассмотренном в главе 2, управляемая лавинообразная маршрутизация с применением порядковых номеров используется для широковещания запросов по наложенной (оверлейной) сети. В протоколе Gnutella копирование и передача сообщений выполняется на прикладном, а не на сетевом уровне.

Другой способ управляемой лавинообразной маршрутизации именуется **переадресацией в обратном направлении** (reverse path forwarding, **RPF**)[123], иногда также называется «широковещание в обратном направлении» (reverse path broadcast, RPB). В основе этого механизма лежит простая, но элегантная идея. Когда при широковещательной передаче маршрутизатор получает пакет с заданным адресом, он пересылает этот пакет по всем своим исходящим каналам (кроме того, по которому он поступил), но лишь при условии, что пакет прибыл по тому каналу, который находится на кратчайшем одноадресном пути к источнику пакета. В противном случае узел просто отбрасывает такой входящий пакет, не пересылая его ни по одному из своих исходящих каналов. Такой пакет действительно можно смело отбросить, поскольку маршрутизатору из-

вестно, что он либо уже получал такой пакет, либо получит его именно по тому каналу, который расположен на кратчайшем пути от данного узла к узлу-отправителю. Можете сами подробнее исследовать этот случай и убедиться, что он действительно не допускает возникновения ни петель, ни широковещательного шторма. Обратите внимание: в механизме RPF не применяется одноадресная маршрутизация для доставки пакета к месту назначения; кроме того, от маршрутизатора не требуется знать полный кратчайший путь от себя до источника пакета. RPF требует, чтобы маршрутизатор знал лишь ближайший узел, смежный с этим маршрутизатором и расположенный на кратчайшем пути к отправителю. Он использует идентификатор смежного узла лишь для того, чтобы определить, отправлять или нет полученный пакет в лавинообразную рассылку.

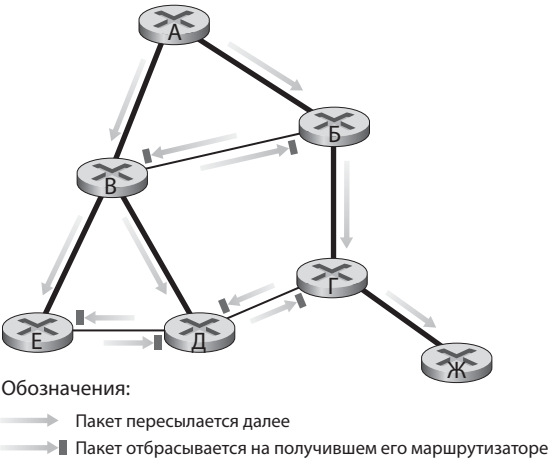
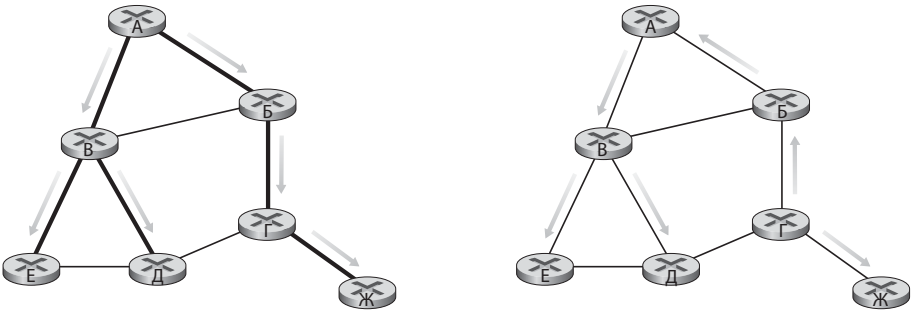


Рис. 4.44. Переадресация в обратном направлении

Механизм RPF проиллюстрирован на рис. 4.44. Предположим, что каналы, обозначенные жирными линиями, соответствуют путям с наименьшей стоимостью от адресатов к исходному узлу (*А*). Узел *А* сначала широковещательно передает пакет из источника *А* узлам *Б* и *В*. Узел *Б* перешлет пакет, полученный из источника *А* (поскольку *А* находится на кратчайшем пути от *Б* к *А*) узлам *В* и *Г*. Узел *Б* проигнорирует (то есть, отбросит, не пересылая) любые пакеты из источника *А*, которые он получит от других узлов (например, от маршрутизаторов *В* или *Г*). Теперь давайте обратим внимание на узел *В*, который получит пакет *А* и непосредственно от *А*, и от *Б*. Поскольку *Б* не находится на кратчайшем пути от *В* до *А*, *В* проигнорирует все пакеты с источника *А*, полученные от *Б*. Но если *В* получит пакет *А* непосредственно от *А*, то он перешлет этот пакет на узлы *Б*, *Д* и *Е*.

Широковещательная маршрутизация по методу связующего дерева

Итак, при управляемой лавинообразной маршрутизации с применением порядковых номеров и при RPF удается не допускать широковещательных штормов, но эти алгоритмы не позволяют полностью исключить широковещательную передачу избыточных пакетов. Так, на рис. 4.44 узлы *Б, В, Г, Д* и *Е* получат по одному или по два лишних пакета каждый. В идеале при широковещательной передаче каждый узел должен принимать всего по одному экземпляру пакета. Изучив изображенное на рис. 4.45(а) дерево узлов, соединенных жирными линиями, мы видим, что если бы при широковещательной передаче пакеты шли только по каналам из этого дерева, то каждый узел в сети получил бы ровно по одному экземпляру широковещательно передаваемого пакета. Именно этого мы и добиваемся! Здесь мы видим пример **связующего дерева** — такое дерево включает в себя абсолютно все узлы графа. Более строгая формулировка будет звучать так: связующее дерево графа $G = (N,E)$ — это такой граф $G' = (N,E')$, в котором E' является подмножеством E, граф G является связным, граф G' не содержит циклов и включает в себя все исходные узлы G. Если каждый канал имеет определенную стоимость, а стоимость дерева равна сумме стоимостей всех его каналов, то связующее дерево графа, обладающее минимальной стоимостью, выделяется из всех связующих деревьев графа и (что неудивительно) называется **минимальным связующим деревом** этого графа.



а. Широковещание инициировано на узле А

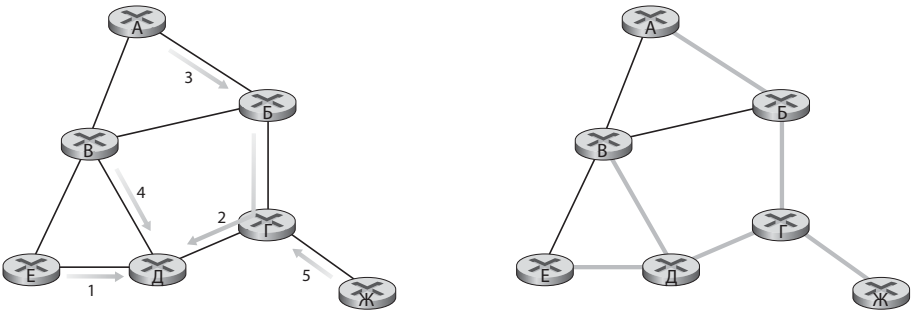
б. Широковещание инициировано на узле Г

Рис. 4.45. Широковещание по связующему дереву

Итак, альтернативный подход к обеспечению широковещательной передачи данных в сети — создание связующего дерева. Когда исходному узлу требуется послать широковещательный пакет, он рассылает его по всем инцидентным каналам, относящимся к связующему дереву. За-

тем узел, получивший широковещательный пакет, рассылает его всем своим соседним узлам в связующем дереве (кроме того, от которого он получил этот пакет). Связующее дерево позволяет не только избавиться от лишних широковещательных пакетов, но и начинать широковещательную передачу с любого узла, как показано на рис. 4.45(а) и 4.45(б). Обратите внимание: узлу не требуется информация обо всем дереве; он должен лишь знать, какие из его узлов-соседей по графу *G* относятся к связующему дереву.

Основная сложность, возникающая при использовании связующих деревьев, заключается в необходимости их создания и поддержки. Разработаны многочисленные распределенные алгоритмы для создания связующих деревьев[177, 180]. Здесь мы рассмотрим только один такой алгоритм. Это **центрированный** подход, при котором в дереве определяется центральный узел (также называемый **точкой рандеву** или **ядром**). Далее все узлы одноадресно передают центральному узлу сообщения для объединения дерева. Такое сообщение пересылается методом одноадресной маршрутизации к центру, пока не достигнет либо одного из узлов, уже относящихся к связующему дереву, либо, собственно, центра. Можно сказать, что такой путь «пересаживается» или «прививается» к имеющемуся связующему дереву.



а. Пошаговое создание связующего дерева

б. Готовое связующее дерево

Рис. 4.46. Создание связующего дерева, начиная с центрального узла

На рис. 4.46 продемонстрировано построение связующего дерева с центральной точкой. Предположим, узел *Д* выбран в качестве центра дерева. Далее допустим, что узел *Е* первым присоединяется к дереву и отправляет узлу *Д* объединяющее сообщение. Один канал *ДЕ* становится «ростком» связующего дерева. Затем к этому дереву подключается узел *Б*, отправляющий узлу *Д* объединяющее сообщение. Предположим, что одноадресный маршрут от *Д* до *Б* пролегает через *Г*. В результате имеем

«прививку» пути *БГД* к связующему дереву. Следующим к группе подключается узел *А*, направляя объединяющее сообщение узлу *Д*. Если одноадресный путь от *А* до *Д* пролегает через *Б*, то, поскольку узел *Б* уже относится к связующему дереву, прибытие объединяющего сообщения от узла *А* к узлу *Б* приведет к образованию канала *АБ*, который будет сразу же «привит» к связующему дереву. Следующим к дереву присоединится узел *В*, который направит объединяющее сообщение непосредственно к *Д*. Наконец, поскольку одноадресная маршрутизация от узла *Ж* к *Д* должна идти через узел *Г*, канал *ЖГ* «прививается» к связующему дереву на узле *Г*.

Широковещательные алгоритмы на практике

На практике широковещательные протоколы применяются как на прикладном, так и на сетевом уровне. Протокол Gnutella[184] использует широковещательную передачу данных на прикладном уровне для рассылки запросов содержимого среди равноправных хостов Gnutella. В данном случае связь между двумя распределенными равноправными процессами прикладного уровня по протоколу Gnutella представляет собой обычное TCP-соединение. Gnutella задействует своеобразный вариант управляемой лавинообразной маршрутизации с применением порядковых номеров, где 16-битный идентификатор и 16-битный описатель полезного содержимого (указывающий тип сообщения Gnutella) используются для определения того, что ранее происходило с полученным широковещательным запросом: был ли он получен, копирован и переадресован. Протокол Gnutella также применяет поле TTL (предписанное время жизни), позволяющее ограничить количество транзитных узлов (переходов), которые может преодолеть лавинообразно направленный запрос. Когда процесс Gnutella получает и копирует запрос, этот процесс сначала понижает значение в поле TTL на единицу, а только потом пересылает запрос. Соответственно, лавинообразный запрос в Gnutella сможет достичь лишь тех узлов, количество переходов до которых на прикладном уровне не превышает исходного значения в поле TTL этого процесса. Поэтому применяемый в протоколе Gnutella механизм лавинообразной маршрутизации иногда именуется «*лавинообразная маршрутизация с ограниченным распространением*».

Кроме того, особая разновидность управляемой лавинообразной маршрутизации с применением порядковых номеров используется для широковещательной рассылки объявлений о состоянии каналов (LSA) по протоколу OSPF[390, 474], а также по протоколу IS-IS (протокол маршру-

тизации промежуточных систем)[438; 390]. Для идентификации таких объявлений протокол OSPF использует 32-битные порядковые номера, а также 16-битное поле устаревания. Как вы помните, узел протокола OSPF широковещательно распространяет объявления LSA по непосредственно связанным с этим узлом каналам, в случае, когда стоимость пути до узла-соседа изменяется, либо когда канал включается/отключается. Порядковые номера объявлений о состоянии канала используются для выявления дублирующихся объявлений такого рода, но также выполняют в протоколе OSPF и другую важную функцию. При лавинообразной маршрутизации не исключено, что объявление LSA, сгенерированное исходным хостом в момент времени t, прибудет на узел назначения уже *после* наступления момента $t+\delta$, в который исходный узел успеет сгенерировать новое объявление об изменении состояния канала. Поскольку исходный узел использует порядковую нумерацию сообщений, более старое сообщение можно отличить от более нового. Поле устаревания функционально похоже на поле предписанного времени жизни (TTL). Сначала в этом поле записано значение 0, а после каждого перехода в процессе лавинообразной рассылки оно увеличивается. Это значение увеличивается даже в то время, пока пакет находится в памяти и ожидает, пока его направят в лавинообразную рассылку. Здесь мы не можем более подробно обсудить алгоритм лавинообразной рассылки LSA, но отметим, что разработка протоколов для такой маршрутизации является очень сложным делом. В работе Перлмана[390] и документе RFC 789[418] описан случай, в котором неправильно переданные двумя неисправными маршрутизаторами объявления о состоянии каналов вызвали крах всей сети ARPAnet, где применялась одна из ранних версий алгоритма лавинообразной маршрутизации объявлений LSA.

4.7.2. Групповая маршрутизация

В предыдущем разделе мы убедились, что при широковещательном распространении данных пакет доставляется всем без исключения узлам в сети. В этом разделе мы поговорим о **групповой** (многоадресной) маршрутизации, при которой пакеты доставляются лишь *подмножеству* узлов в сети. Ряд недавно появившихся сетевых приложений требуют доставки пакетов от одного или более отправителей к группе получателей. Среди таких задач требуется отметить передачу массива данных — например, передачу пакета обновлений приложения пользователям, которые в них нуждаются; потокову передачу мультимедийной информации (например, аудио, видео и текста распределенной ау-

дитории во время виртуальной лекции); приложения с разделяемыми данными (например, приложения для телеконференций и виртуальных лекционных досок, которые совместно используются группой распределенных клиентов), потоки данных (например, для представления биржевых котировок), обновление веб-кэша, интерактивные игры в Интернете (например, распределенные интерактивные виртуальные среды или многопользовательские игры).

При групповой рассылке мы сразу же сталкиваемся с двумя проблемами: как идентифицировать получателей многоадресной дейтаграммы и как адресовать эту дейтаграмму.

В случае одноадресной рассылки IP-адрес получателя содержится в каждой одноадресной IP-дейтаграмме и означает единственного получателя. Но в случае групповой рассылки имеется несколько получателей. Есть ли смысл помещать в каждую групповую дейтаграмму IP-адреса всех получателей, входящих в группу? Хотя такой подход может работать для небольшого количества получателей, он плохо подходит для случая, когда их сотни и тысячи. Адресные данные просто займут весь объем дейтаграммы, вытеснив из нее полезную информацию. Кроме того, чтобы явно указать *всех* получателей, отправитель должен знать все их идентификаторы и адреса. Далее будет показано, что в некоторых ситуациях такое требование может быть нежелательным.

По вышеуказанным причинам в архитектуре Интернета (а также в других архитектурах, таких, как ATM-сети[56]) групповая дейтаграмма **адресуется косвенно**. То есть для группы получателей используется один идентификатор, а копия дейтаграммы, адресованной группе получателей при помощи этого единственного идентификатора, доставляется всем членам этой группы. В Интернете единый идентификатор, соответствующий группе получателей, представляет собой групповой адрес класса D (см. раздел «Интернет-протокол»). Группа получателей, ассоциированная с адресом класса D, называется **группой рассылки**. На рис. 4.47 четыре хоста (с указанными IP-адресами) ассоциированы с группой рассылки с IP-адресом 226.17.30.197. Эти хосты будут получать все дейтаграммы, направляемые по данному адресу. Сложность заключается в том, что у каждого хоста имеется уникальный IP-адрес, полностью независимый от адреса группы рассылки.

Хотя идея группы рассылки проста, она поднимает ряд вопросов. Как создается группа и как она прекращает свое существование? Каким образом выбирается групповой адрес? Как к группе добавляются новые

хосты (в качестве отправителей или получателей)? Может ли кто угодно присоединиться к группе (и передавать или принимать данные, посылаемые этой группе) или членство в группе ограничивается, и если да, то кем? Известны ли членам группы идентификаторы других членов группы? Как сетевые маршрутизаторы взаимодействуют друг с другом, чтобы доставить групповую дейтаграмму всем членам группы? В Интернете на все эти вопросы отвечает протокол IGMP[506]. Мы рассмотрим протокол IGMP, а затем вернемся к более общим вопросам.

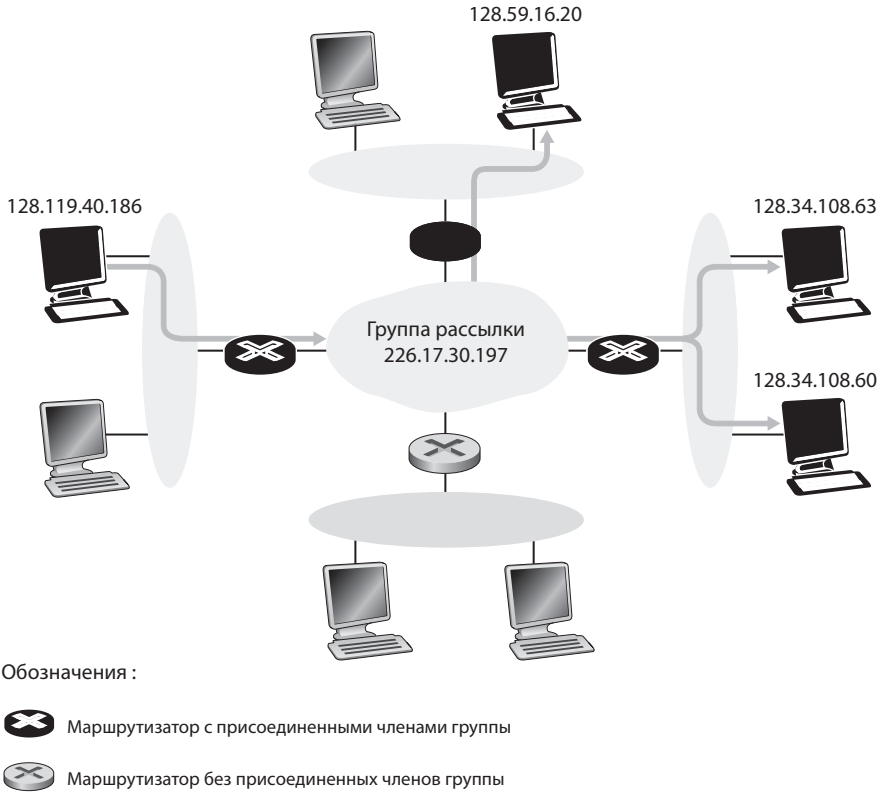


Рис. 4.47. Адресованная в группу рассылки дейтаграмма доставляется всем членам группы

Протокол управления группами Интернета

Протокол IGMP (Internet Group Management Protocol — протокол управления группами Интернета) версии 3[506] работает между хостом и соединенным с ним напрямую маршрутизатором (который можно рассматривать как первый маршрутизатор на пути следования входящих

дейтаграмм или последний маршрутизатор на пути следования исходящих дейтаграмм), как показано на рис. 4.48. На рис. 4.48 изображены три групповых маршрутизатора (расстояние между любыми двумя из них составляет один переход), каждый из которых соединен с парой хостов через локальный интерфейс. В данном примере локальный интерфейс связан с локальной сетью, и, как правило, несколько хостов локальной сети одновременно являются членами той или иной группы рассылки.

Протокол IGMP предоставляет хосту средство информировать соединенный с ним маршрутизатор о том, что работающее на хосте приложение желает присоединиться к определенной группе рассылки. Учитывая ограниченность сферы действия протокола IGMP хостом и соединенным с ним маршрутизатором, для координирования групповых маршрутизаторов (включая присоединенные) в Интернете, очевидно, необходимы другие протоколы, позволяющие доставлять групповые дейтаграммы к местам назначения. Это обеспечивается при помощи алгоритмов групповой маршрутизации сетевого уровня, примеры которых мы вскоре рассмотрим. Соответственно, групповая передача данных на сетевом уровне в Интернете состоит из двух взаимодополняющих компонентов: протокола IGMP и протоколов групповой маршрутизации.

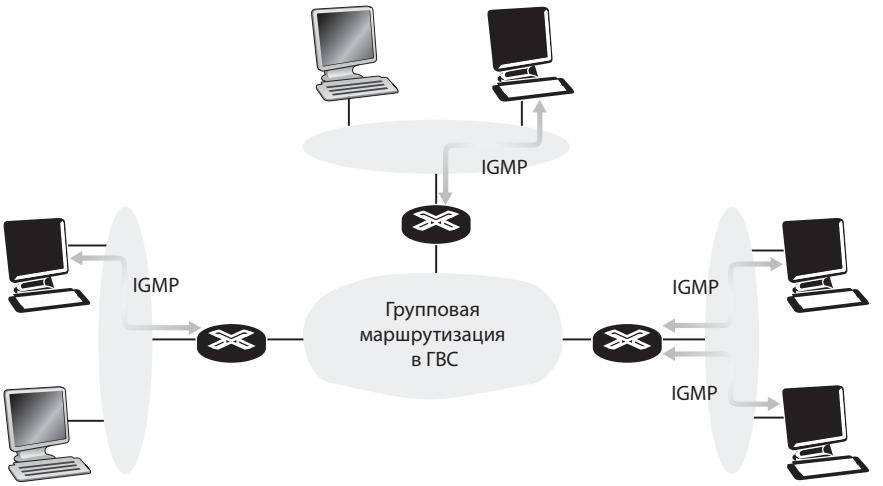


Рис. 4.48. Два компонента групповой маршрутизации на сетевом уровне в Интернете: протокол IGMP и протоколы групповой маршрутизации

В протоколе IGMP используются только три типа сообщений. Подобно ICMP, сообщения IGMP, вкладываются (инкапсулируются) в IP-дейтаграммы, с номером протокола 2. Общее сообщение `membership_`

query посылается маршрутизатором всем хостам, присоединенным к его интерфейсу (например, всем хостам локальной сети), чтобы узнать обо всех группах рассылки, членами которых стали хосты данного интерфейса.

Хосты отвечают на сообщение membership_query IGMP-сообщением membership_report. Хост может также генерировать сообщения membership_report, не ожидая сообщения membership_query от маршрутизатора, когда приложение впервые присоединяется к группе рассылки. Последний тип IGMP-сообщений — это leave_group. Интересно отметить, что это сообщение не является обязательным! Но как тогда маршрутизатор определяет, что в данной локальной сети не осталось хостов, входящих в определенную группу? Оказывается, маршрутизатор *логически заключает*, что в данную группу рассылки более не входят присоединенные к нему хосты, если ни один хост не отвечает на его сообщение membership_query с конкретным групповым адресом. Интернет-протоколы, в которых по истечении некоторого интервала времени информация об адресах удаляется, иногда называют протоколами **с неустойчивым состоянием**. Таким протоколом является протокол IGMP, в котором информация о наличии членов определенной группы рассылки среди хостов локальной сети удаляется по истечении заданного интервала времени (в этом случае интервал задает периодически посылаемое маршрутизатором сообщение membership_query), если оно не обновляется явно (при помощи посылаемого хостом сообщения membership_report).

Термин «неустойчивое состояние» был предложен Кларком[109], описавшим идею периодических сообщений об обновлении состояния, отсылаемых конечной системой, и предположил, что при их применении состояние можно восстанавливать на конечной системе даже после аварийного выключения сети — на основании информации, содержащейся в последующих сообщениях об обновлении. Такой механизм представляется совершенно прозрачным для конечной системы и не требует вызывать в ней какие-либо явные восстановительные процедуры:

> *«...информация о состоянии представляется некритичной при поддержании нужного типа обслуживания в ходе работы с потоком. Действительно, такой способ обслуживания будет жестко регламентироваться конечными системами, которые периодически обмениваются сообщениями, чтобы гарантировать, что с потоком информации ассоциировано нужное обслуживание. Таким образом, информация о состоянии, ассоциированная с потоком, может быть потеряна при аварийном отключении, но без необратимого уничто-*

жения тех функциональных возможностей, которые при этом используются. Такую концепцию я называю «неустойчивым состоянием», и она вполне позволяет нам достичь наших приоритетных целей, связанных с жизнеспособностью и гибкостью...»

Утверждается, что протоколами с неустойчивым состоянием проще управлять, нежели протоколами с устойчивым состоянием. Для последних нужны не только механизмы явного добавления или удаления состояний (то есть информации о членстве в группах), но также какие-то средства восстановления в ситуации, когда один из этих механизмов преждевременно завершит работу или вообще выйдет из строя. Интересные соображения о неустойчивом состоянии содержатся в работах Рамана[406], Джи[266] и Луи[327].

Алгоритмы групповой маршрутизации

Рисунок 4.49 призван проиллюстрировать **задачу групповой маршрутизации**. Хосты, являющиеся членами группы рассылки, показаны на рисунке темными, как и непосредственно соединенные с ними маршрутизаторы. Как видно из рисунка, принимать групповой трафик нужно не всем маршрутизаторам, а только тем, чьи хосты являются членами группы рассылки, то есть маршрутизаторам А, Б, Д и Е. Маршрутизаторам В и Г не нужен групповой трафик, так как присоединенные к маршрутизатору Г хосты не являются членами группы рассылки, а у маршрутизатора В вообще нет непосредственно присоединенных хостов. Цель групповой маршрутизации заключается в том, чтобы построить дерево, связывающее все маршрутизаторы, присоединенные хосты которых относятся к данной группе рассылки. При этом групповые пакеты будут направляться по этому дереву от отправителя ко всем хостам, входящим в дерево группы рассылки. Разумеется, дерево может содержать маршрутизаторы, не имеющие хостов, относящихся к данной группе рассылки (например, как видно из рисунка, невозможно связать маршрутизаторы А, Б, Д и Е в дерево, не включив в него маршрутизатор В или Г).

На практике для построения дерева групповой маршрутизации применяются два подхода, отличающиеся тем, используется ли общее дерево для *нескольких* отправителей или для каждого отправителя создается специальное.

- *Групповая маршрутизация с общим деревом*. Как и в случае широковещательной передачи данных с применением связующего дерева, групповая маршрутизация с общим деревом осуществляется путем

построения дерева, включающего в себя все граничные маршрутизаторы и связанные с ними хосты, относящиеся к групповой рассылке. На практике дерево для групповой маршрутизации выстраивается начиная от центра, включает в себя все граничные маршрутизаторы и связанные с ними хосты, которые подключаются к этой группе, отсылая центральному узлу (одно)адресные сообщения о присоединении. Как и в широковещательном случае, сообщение о присоединении отправляется по направлению к центральному узлу одноадресным методом до тех пор, пока либо не попадет на маршрутизатор, относящийся к группе, либо не достигнет центра. Все маршрутизаторы, расположенные по пути, преодолеваемому сообщением по направлению к центру, затем переадресуют полученные групповые пакеты тому маршрутизатору, который инициировал подключение к групповой передаче. При организации такой маршрутизации критически важен вопрос выбора центрального узла. Соответствующие алгоритмы рассматриваются в работах Уолла[656], Талера[634] и Эстрина[153].

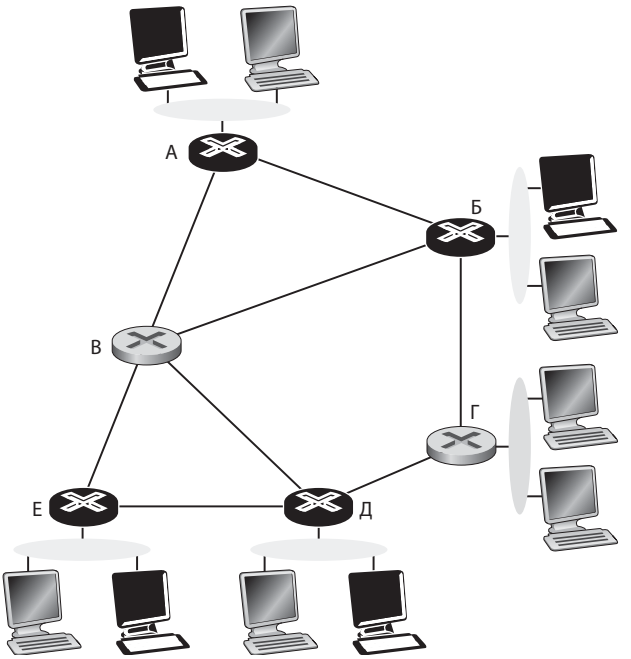


Рис. 4.49. Хосты при групповой передаче данных, примыкающие к ним маршрутизаторы и другие маршрутизаторы

- *Групповая маршрутизация с использованием дерева для каждого источника.* В то время как при групповой передаче данных с использо-

ванием общего дерева создается единое разделяемое дерево, по которому направляются пакеты от *всех* отправителей, рассматриваемый подход предполагает построение деревьев групповой маршрутизации для *каждого* из источников, находящихся в группе. На практике применяется алгоритм *переадресации в обратном направлении* (RPF) с исходным узлом *x*, при помощи которого создается дерево групповой передачи дейтаграмм, исходящих с источника *x*. Широковещательный вариант алгоритма RPF, изученный нами ранее, нужно немного усовершенствовать для применения в групповой передаче данных. Чтобы уточнить этот момент, рассмотрим маршрутизатор Г на рис. 4.50.

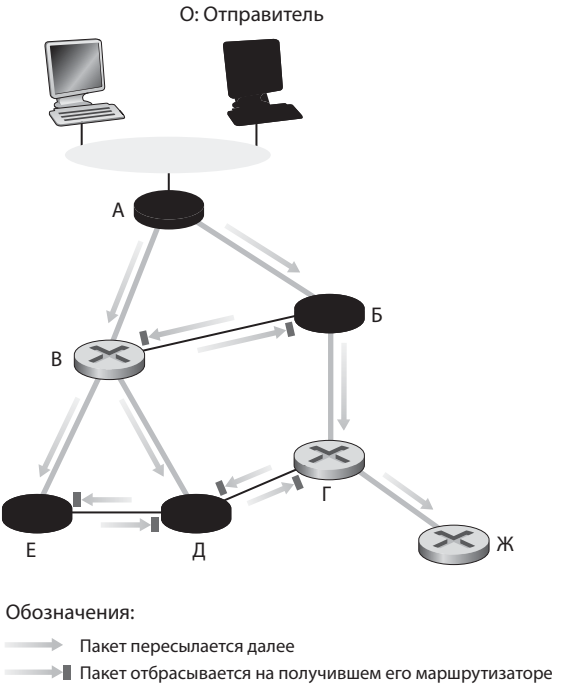


Рис. 4.50. Переадресация в обратном направлении при групповой передаче данных

При широковещательной передаче в обратном направлении он пересылал бы пакеты маршрутизатору Ж, несмотря на то, что у Ж нет примыкающих хостов, которые относились бы к подмножеству групповой рассылки. В данном сценарии это обстоятельство не доставляет неудобств, так как «ниже по течению» от Г располагается всего один маршрутизатор — Ж, но можете себе представить, что

случилось бы, если бы насчитывались тысячи таких маршрутизаторов! Каждый из них получал бы ненужные пакеты, относящиеся к групповой рассылке.

Этот сценарий вовсе не так фантастичен, как это может показаться. Первая глобальная сеть групповой рассылки Mbone[77, 330] вначале страдала именно таким недостатком!)

Решение проблемы с доставкой ненужных групповых пакетов в алгоритме RPF называют **отсечением**. Маршрутизатор, получающий групповые пакеты, у которого нет соединенных с ним напрямую хостов, принадлежащих этой группе, посылает отсекающее сообщение маршрутизатору «выше по течению». Если маршрутизатор получает отсекающие сообщения от всех своих маршрутизаторов «ниже по течению», тогда он также может послать отсекающее сообщение маршрутизатору «выше по течению».

Групповая маршрутизация в Интернете

Первым протоколом групповой маршрутизации, использовавшимся в Интернете, был **DVMRP** (Distance-Vector Multicast Routing Protocol) — **протокол дистанционно-векторной групповой маршрутизации**[434]. В протоколе DVMRP применяются деревья для каждого источника с переадресацией в обратном направлении и отсечением. В DVMRP используется алгоритм RPF с отсечением, рассмотренный выше. Вероятно, самым широко используемым **протоколом групповой маршрутизации** в Интернете является **PIM** (Protocol-Independent Multicast, **протоколо-независимая групповая маршрутизация**), в котором явно различается два сценария распространения многоадресной информации. В уплотненном режиме[525] члены группы многоадресной передачи данных расположены поблизости друг от друга; таким образом, при передаче групповых дейтаграмм задействуются многие маршрутизаторы в этой зоне или большинство из них. В уплотненном режиме механизм работы PIM напоминает лавинную рассылку и обратную передачу данных, то есть, принципиально похож на функционал DVMRP.

В разреженном режиме[543] количество маршрутизаторов, к которым подключены члены группы, сравнительно невелико по сравнению с общим количеством маршрутизаторов. Члены группы достаточно широко разбросаны. В разреженном режиме протокол PIM использует точки встречи для построения дерева, по которому осуществляется распределенная групповая маршрутизация. При **групповой маршрутизации, за-**

висящей от отправителя (source-specific multicast, **SSM**)[519, 544] лишь один отправитель может отсылать трафик в дерево групповой маршрутизации, что существенно упрощает построение и техническую поддержку дерева.

Когда в некоторой зоне одновременно используются протоколы PIM и DVMRP, администратор вычислительной сети может сконфигурировать IP-маршрутизаторы групповой рассылки внутри этой зоны во многом так же, как обычно конфигурируются протоколы одноадресной рассылки — RIP, IS-IS и OSPF. Но что делать, если групповая маршрутизация должна осуществляться между разными зонами? Существует ли групповой эквивалент внутреннего протокола BGP? Ответ — да. Стандарт RFC 4271[529] описывает многопротокольные расширения BGP, позволяющие передавать по этому нему маршрутную информацию от других протоколов, в том числе — связанную с групповой маршрутизацией. MSDP (протокол обнаружения источников группового вещания)[521, 545] может использоваться для соединения точек встречи в различных зонах, использующих разреженный режим работы с протоколом PIM. Отличный обзор актуального состояния групповой маршрутизации в Интернете описан в стандарте RFC 5110[551].

Завершая обсуждение групповой маршрутизации в IP-сетях, следует отметить, что она пока не получила широкого распространения. Интересные соображения о модели групповой маршрутизации в Интернете и о проблемах, связанных с ее развертыванием, высказываются в работах Диота[136] и Шарма[599]. Тем не менее, несмотря на отсутствие широкого распространения, слухи о смерти групповой маршрутизации на сетевом уровне сильно преувеличены. Уже много лет групповой трафик передается в Интернете-2 и подобных сетях[243]. Так, в Великобритании компания BBC в экспериментальном режиме распространяет контент при помощи группового вещания по IP[45]. В то же время, групповая маршрутизация на прикладном уровне, которую мы рассматривали в главе 2, говоря о технологии PPLive и других одноранговых системах, например, End System Multicast[96] обеспечивает групповое распространение содержимого среди равноправных узлов по протоколам прикладного, а не сетевого уровня. Сегодня распространение контента в одноранговых сетях пользуется огромной популярностью, благодаря чему можно сделать вывод, что, по крайней мере, в ближайшем будущем групповая маршрутизация будет тяготеть к прикладному уровню. Тем не менее, групповая IP-маршрутизация продолжает развиваться. Рано или поздно эти состязания должны привести к стабильному решению.

4.8. Заключение

В этой главе мы начали знакомство с ядром сети. Мы узнали, что на сетевом уровне работают абсолютно все хосты и маршрутизаторы сети. Поэтому сетевые протоколы являются одними из самых сложных в стеке протоколов.

Мы узнали, что больших компьютерных сетях маршрутизатору может потребоваться одновременно обрабатывать миллионы потоков пакетов между различными парами отправитель — получатель. За годы работы разработчики сетей пришли к важному выводу: чтобы маршрутизатор мог обрабатывать такое большое количество потоков, функции маршрутизатора должны быть максимально простыми. Для упрощения работы маршрутизатора доступны разные методы, включая использование вместо виртуальных каналов дейтаграмм на сетевом уровне, применение упрощенного заголовка фиксированной длины (как в протоколе IPv6), устранения фрагментации (что также сделано в IPv6), предоставление минимального обслуживания (обслуживание по остаточному принципу). Возможно, наиболее важный принцип здесь заключается в том, чтобы *не* отслеживать отдельные потоки, а принимать решения о выборе маршрутов исключительно на базе иерархически структурированных адресов получателей, указанных в пакетах. Интересно отметить, что почтовая служба применяла подобный принцип на протяжении многих лет.

В этой главе мы также рассмотрели основополагающие принципы алгоритмов маршрутизации. Мы узнали, что разработчики алгоритмов маршрутизации используют абстракцию компьютерной сети в виде графа с узлами и ребрами (соединениями между узлами). Это позволяет задействовать хорошо развитую теорию графов и алгоритмы нахождения кратчайших маршрутов, разработанные за последние 40 лет. Мы увидели, что существует два основных подхода — централизованный, при котором каждый узел получает полную карту сети и независимо от других узлов применяет алгоритм определения кратчайшего маршрута, и децентрализованный, при котором отдельные узлы обладают только частичным представлением обо всей сети, однако, работая вместе, доставляют пакеты по кратчайшим маршрутам. Мы также рассмотрели, как при помощи иерархических структур решается проблема масштабирования, позволяющая сегментировать большие сети в независимо управляемые подсети (домены), называемые «административными системами» (АС). Каждая АС самостоятельно организует потоки дей-

таграмм внутри себя, точно как каждая страна независимо от других организует обслуживание своих почтовых отправлений. Мы изучили, как централизованные, децентрализованные и иерархические методы реализуются в основных протоколах маршрутизации в Интернете: RIP, OSPF и BGP. В завершение нашего исследования алгоритмов маршрутизации мы рассмотрели широковещательную и групповую передачу данных.

Заканчивая обсуждение сетевого уровня, мы переходим на канальный уровень, расположенный в стеке протоколов еще глубже. Канальный уровень, как и сетевой, входит в состав ядра. Но в следующей главе мы убедимся, что он решает гораздо более узкую задачу и занят перемещением пакетов между узлами в рамках одного и того же канала или ЛВС. Хотя на первый взгляд эта задача может показаться тривиальной по сравнению с сетевыми, вы вскоре убедитесь, что на канальном уровне решается ряд важных и интересных проблем, на изучение которых у нас уйдет немало времени.

Глава 5
КАНАЛЬНЫЙ УРОВЕНЬ: КАНАЛЫ, СЕТИ ДОСТУПА И ЛВС

Из предыдущей главы мы узнали, что сетевой уровень обеспечивает взаимодействие между *любыми* двумя хостами в сети. Дейтаграммы движутся между двумя хостами по цепочке коммуникационных каналов, среди которых есть как проводные, так и беспроводные. Передача начинается с исходного хоста, затем информация проходит через ряд коммутаторов пакетов (коммутаторов и маршрутизаторов), пока наконец, не оказывается на хосте назначения. В этой главе мы переходим еще на шаг ниже в стеке протоколов, с сетевого уровня на канальный. Здесь мы поговорим о том, как пакеты переходят по *отдельным каналам*, из которых состоит весь путь передачи данных. Как дейтаграммы сетевого уровня инкапсулируются в кадры канального уровня для передачи по каналу? Могут ли в разных каналах на пути передачи данных применяться различные протоколы канального уровня? Как разрешаются конфликты, которые могут возникать в каналах при широковещательной передаче данных? Существует ли на канальном уровне адресация, и если да — как она сочетается с адресацией сетевого уровня, изученной нами в главе 4? В чем именно заключается разница между маршрутизатором и коммутатором? В данной главе мы ответим на этот и на другие важные вопросы.

Во время обсуждения канального уровня мы обнаружим, что на нем существуют два принципиально разных типа каналов. Первый тип — это широковещательные каналы, распространенные в локальных, беспроводных локальных и спутниковых сетях, а также в гибридных оптоволоконных сетях. В широковещательном канале несколько хостов присоединены к одному и тому же каналу связи, а для того чтобы координировать передачу требуется протокол доступа к несущей. Второй тип канала — это обычная двухточечная линия связи, соединяющая, например, два маршрутизатора или офисный компьютер пользователя с Ethernet-коммутатором. Управление доступом к каналу в подобном двухточечном соединении является тривиальным. При помощи протокола двухточечной передачи (PPP) задаются разнообразные настройки:

от коммутируемого доступа по телефонной линии до высокоскоростной двухточечной передачи кадров по оптоволоконным сетям.

В этой главе мы также ознакомимся с несколькими важными технологиями канального уровня. Мы подробно поговорим об обнаружении и исправлении ошибок — эту тему мы кратко затрагивали в главе 3. Мы рассмотрим разнообразные сети доступа и коммутируемые локальные сети, в частности, Ethernet — в настоящее время эта технология является доминирующей в сфере локальных сетей. Кроме того, мы рассмотрим виртуальные локальные сети и сети дата-центров. Хотя Wi-Fi и, в более общем смысле, все беспроводные локальные сети тематически относятся к канальному уровню, мы пока отложим изучение этих важных тем и подробно обсудим их в главе 6.

5.1. Обзор канального уровня

Начнем изучение канального уровня с терминологии. В этой главе мы будем именовать узлом любое устройство, которое использует при работе протокол канального уровня (уровня 2). Различные виды узлов — это хосты, маршрутизаторы, коммутаторы и точки доступа Wi-Fi. О точках доступа мы поговорим в главе 6. Кроме того, мы будем называть каналами линии связи, которые связывают смежные узлы вдоль линии передачи данных. Чтобы дейтаграмма была передана с исходного на конечный хост, ее нужно переслать по всем *отдельным каналам* полного пути от хоста к хосту. Допустим, в сети предприятия, изображенной на рис. 5.1, нужно передать дейтаграмму с одного из беспроводных хостов на один из серверов. В таком случае, данная дейтаграмма пройдет по шести каналам: канал Wi-Fi между исходным хостом и точкой доступа; канал Ethernet между точкой доступа и коммутатором канального уровня; канал между коммутатором канального уровня и маршрутизатором; канал между двумя маршрутизаторами; канал Ethernet между маршрутизатором и коммутатором канального уровня; наконец, канал Ethernet между коммутатором и сервером. В каждом из этих каналов дейтаграмма инкапсулируется в кадр канального уровня, который и переносит кадр по каналу.

Чтобы лучше понимать, как работает канальный уровень и как он взаимодействует с сетевым, приведем аналогию с транспортом. Представьте себе агента бюро путешествий, планирующего тур из Принстона, штат Нью-Джерси, в Лозанну, Швейцария.

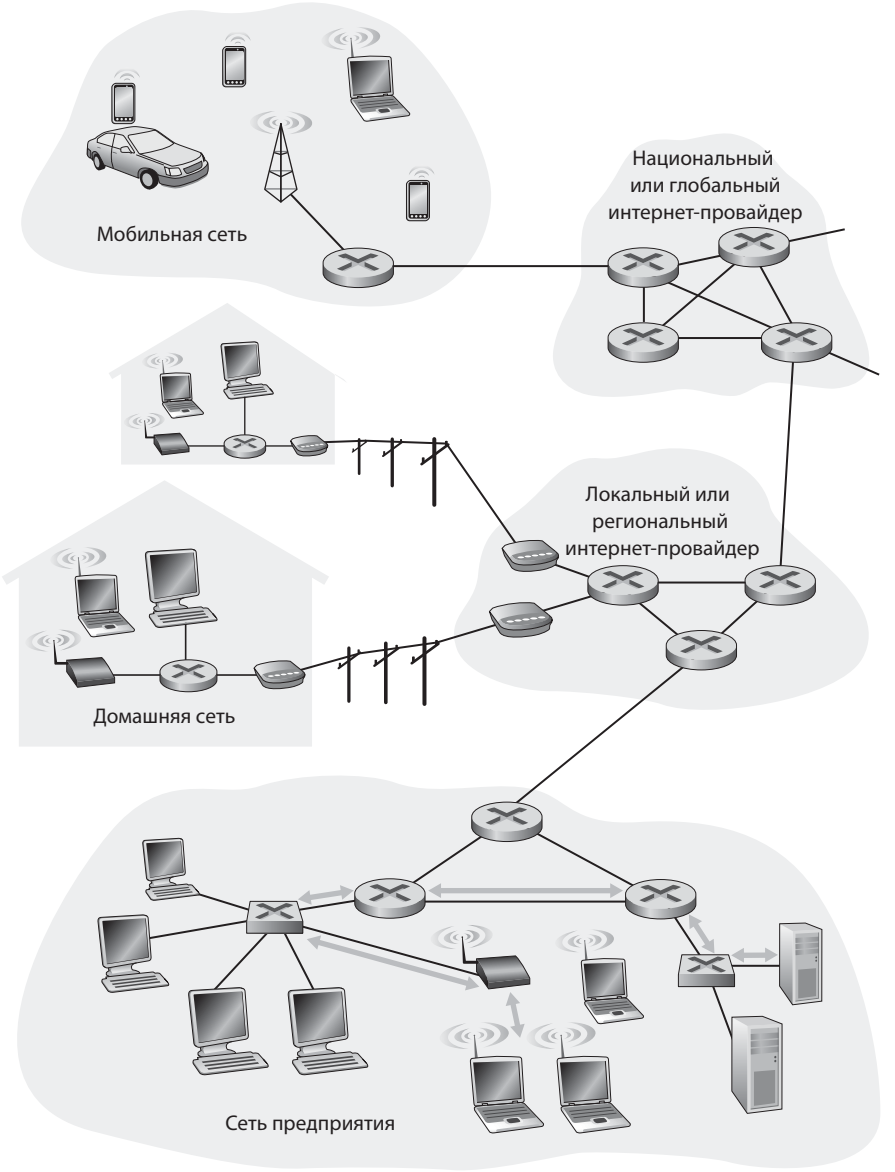


Рис. 5.1. Шесть переходов на канальном уровне между беспроводным хостом и сервером

Предположим, турагент решает, что туристам удобнее всего отправиться из Принстона в аэропорт Кеннеди на такси, откуда на самолете перелететь в Женеву и, наконец, добраться до Лозанны на поезде. Туроператор делает три заказа, после чего за доставку туриста из Принсто-

на в аэропорт отвечает Принстонский таксопарк, за доставку туриста из аэропорта JFK в Женеву ответственность несет авиакомпания, а за доставку туриста из Женевы в Лозанну — Швейцарская железнодорожная служба. Каждый из трех сегментов маршрута «напрямую» связывает два «соседних» узла. Обратите внимание, что тремя сегментами маршрута управляют три разные компании и на этих сегментах используются совершенно разные виды транспорта (такси, самолет и поезд). Несмотря на это каждый сегмент предоставляет основную службу, обеспечивающую перемещение пассажиров в соседний узел. В данной аналогии турист соответствует дейтаграмме, каждый сегмент — линии связи, вид транспорта — протоколу канального уровня, а турагент, планирующий весь маршрут, — протоколу маршрутизации.

5.1.1. Службы канального уровня

Хотя основная услуга любого канального уровня заключается в «перемещении» дейтаграммы между двумя узлами по одной линии связи, полный перечень услуг зависит от конкретного протокола канального уровня, используемого на данной линии связи. К числу таких служб относятся следующие:

- *Формирование кадра*. Почти все протоколы канального уровня помещают каждую дейтаграмму сетевого уровня в кадр канального уровня, перед тем как отправить ее по каналу. Кадр состоит из поля данных, в которое помещается дейтаграмма сетевого уровня, и нескольких полей заголовка. Структура кадра специфицируется канальным протоколом передачи данных. Во время обсуждения конкретных протоколов канального уровня во второй половине этой главы мы рассмотрим несколько различных форматов кадров.
- *Доступ к каналу связи*. Протокол управления доступом к среде передачи (MAC, Media Access Control) определяет правила передачи кадра в канал. Для двухточечных каналов с единственным отправителем на одном конце и единственным получателем на другом протокол MAC очень прост (или вообще отсутствует) — отправитель может передать кадр в любой момент, когда линия свободна. Более интересный случай представляет конфигурация, в которой несколько узлов совместно используют один широковещательный канал. В этом случае возникает так называемая проблема множественного доступа, и протокол MAC призван координировать передачу кадров многих узлов.

- *Надежная доставка*. Когда протокол канального уровня предоставляет услугу по надежной доставке, он гарантирует перемещение каждой дейтаграммы сетевого уровня по линии связи без ошибок. Вспомним, что некоторые протоколы транспортного уровня (как, например, TCP) также обеспечивают надежную доставку. Как и на транспортном уровне, служба надежной доставки канального уровня поддерживается с помощью механизмов подтверждений и повторных передач (см. раздел 3.4). Служба надежной доставки транспортного уровня часто обслуживает линии связи с высокой вероятностью ошибок, например беспроводные. Таким образом, на канальном уровне ошибки исправляются локально — на том канале, где они возникают, что позволяет отказаться от повторной передачи данных протоколом транспортного или прикладного уровня. Однако в линиях с низкой вероятностью ошибок надежная доставка на канальном уровне может оказаться излишней. К таким линиям относятся волоконно-оптические и экранированные кабели, а также различные категории линий типа «витая пара». Поэтому многие протоколы для кабельных линий не предоставляют отдельной услуги по надежной доставке.
- *Обнаружение и исправление ошибок*. Принимающий узел может неверно посчитать, что значение бита в кадре равно нулю, в то время как передавалась единица, и наоборот. Подобные битовые ошибки вызываются ослаблением сигнала и электромагнитными помехами. Поскольку нет смысла передавать дальше дейтаграмму, содержащую ошибки, многие протоколы канального уровня предоставляют услугу по обнаружению ошибок в кадре. Для этого передающий узел добавляет к кадру биты обнаружения ошибок, а получающий узел выполняет проверку контрольной суммы. Как было показано в главах 3 и 4, транспортный и сетевой уровни в Интернете также предоставляют ограниченную услугу по обнаружению ошибок. На канальном уровне обнаружение ошибок сложнее и, как правило, реализуется аппаратно. Исправление ошибок подобно их обнаружению, за тем исключением, что получатель не только находит ошибки в кадре, но и определяет, где именно они произошли, а затем исправляет их.

5.1.2. Протоколы, реализующие канальный уровень

Прежде чем приступить к подробному изучению канального уровня, давайте завершим вводную часть этой главы и рассмотрим, где реализу-

ется канальный уровень. Здесь мы сосредоточимся на конечной системе, поскольку в главе 4 мы уже выяснили, что в маршрутизаторе канальный уровень реализуется интерфейсной платой. Каким же образом реализован канальный уровень хоста — аппаратно или программно? Отвечает ли за него отдельная плата, как он состыкуется с остальным аппаратным обеспечением хоста и с компонентами операционной системы?

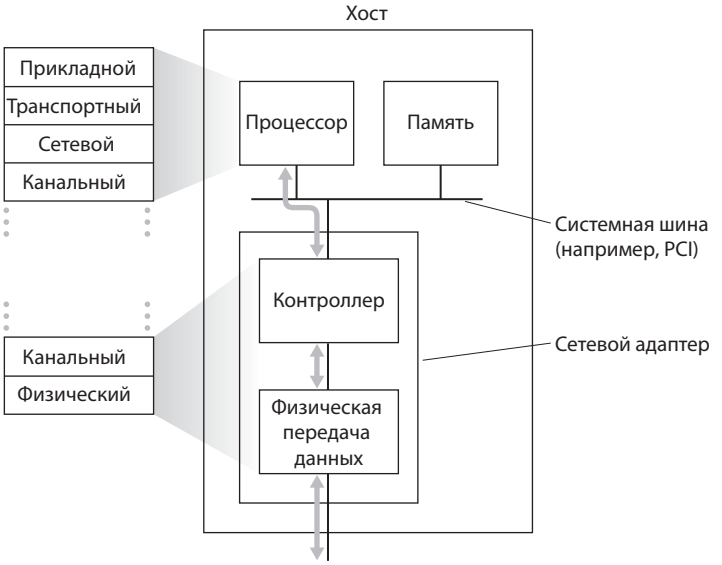


Рис. 5.2. Сетевой адаптер и его отношение к другим компонентам хоста и к функционалу стека протоколов

На рис. 5.2 представлена типичная архитектура хоста. Канальный уровень в основном реализован в **сетевом адаптере**, также называемом **сетевой интерфейсной платой** (network interface card, NIC). Центральным элементом сетевого адаптера является контроллер канального уровня, реализующий многие канальные службы (формирование кадров, доступ к каналу, обнаружение ошибок и т. д.). Соответственно, большая часть функционала канального контроллера реализуется аппаратно. Например, контроллер Intel 8254x[241] реализует протоколы Ethernet, о которых мы поговорим в разделе 5.5; контроллер Atheros AR5006[30] реализует протоколы Wi-Fi 802.11, которые мы изучим в главе 6. До конца 1990-х большинство сетевых адаптеров представляли собой физически отдельные карты (например, карта PCMCIA или карта, подключаемая в PCI-разъем ПК). Но со временем сетевые адаптеры стали все чаще встраивать в материнскую плату хоста — такая конфигурация получила

название LOM (LAN-on-motherboard — сетевые решения на материнской плате).

Со стороны отправителя контроллер берет дейтаграмму, которая была создана и сохранена в памяти вышестоящими уровнями стека протоколов, инкапсулирует ее в кадр канального уровня (заполняя различные поля кадра), а затем передает кадр по линии связи согласно протоколу доступа к каналу. На стороне получателя контроллер принимает целый кадр и извлекает из него дейтаграмму сетевого уровня. Если канальный уровень выполняет обнаружение ошибок, то именно контроллер на стороне отправителя должен установить биты обнаружения ошибок в заголовке кадра, а обнаружение ошибок выполнит уже контроллер на стороне получателя.

На рис. 5.2 показан сетевой адаптер, подключенный к системной шине хоста (PCI или PCI-X). Он весьма похож на другие устройства ввода-вывода, связанные с другими компонентами хоста. На рис. 5.2 также видим, что, хотя большая часть канального уровня реализована аппаратно, в нем есть и программная часть, работающая на процессоре хоста. Программные компоненты канального уровня реализуют высокоуровневый функционал — в частности, сборку адресной информации канального уровня и управление аппаратурой контроллера. На стороне получателя программная часть канального уровня реагирует на прерывания контроллера (например, при получении одного или нескольких кадров), обработку ошибок или передачу дейтаграммы на сетевом уровне. Итак, на канальном уровне сочетаются программная и аппаратная часть, именно в этой части стека протоколов «железо» и код встречаются друг с другом. На сайте **www.intel.ru**[241] приведен обзор (а также дается подробное описание) контроллера 8254x с программной точки зрения.

5.2. Приемы обнаружения и исправления ошибок

В предыдущем разделе мы отмечали, что на канальном уровне часто предоставляются услуги по **обнаружению и исправлению ошибок в отдельных разрядах кадра**, передаваемого между двумя физически соединенными узлами. Как было показано в главе 3, аналогичные услуги часто предоставляются также на транспортном уровне. В данном разделе мы рассмотрим некоторые простейшие методы обнаружения и исправления однобитовых ошибок. Данному вопросу посвящены целые книги (см., например, публикации Швартца[593] или Бертсекаса[50]), а здесь мы дадим лишь краткое введение в тему. Цель нашего обсуждения —

сформировать представление о возможностях этих методов, а также показать, как работают наиболее простые из них и как они практически используются для обнаружения и исправления ошибок на канальном уровне.

На рис. 5.3 представлена схема передачи данных, иллюстрирующая нашу тему. На передающем узле к данным *D* добавляются разряды *EDC* (Error Detection and Correction — обнаружение и исправление ошибок), позволяющие обнаруживать и исправлять ошибки. Как правило, таким образом защищается не только дейтаграмма, передаваемая сетевым уровнем для транспортировки по линии связи, но также и адресная информация канального уровня, включая порядковые номера и другие поля заголовка кадра канального уровня. В кадре канального уровня принимающему узлу передаются как данные *D*, так и биты обнаружения и исправления ошибок *EDC*. Принимающий узел получает разряды *D'* и *EDC'* соответственно. Обратите внимание, что значения *D'* и *EDC'* могут отличаться от исходных значений *D* и *EDC* вследствие ошибок при передаче.

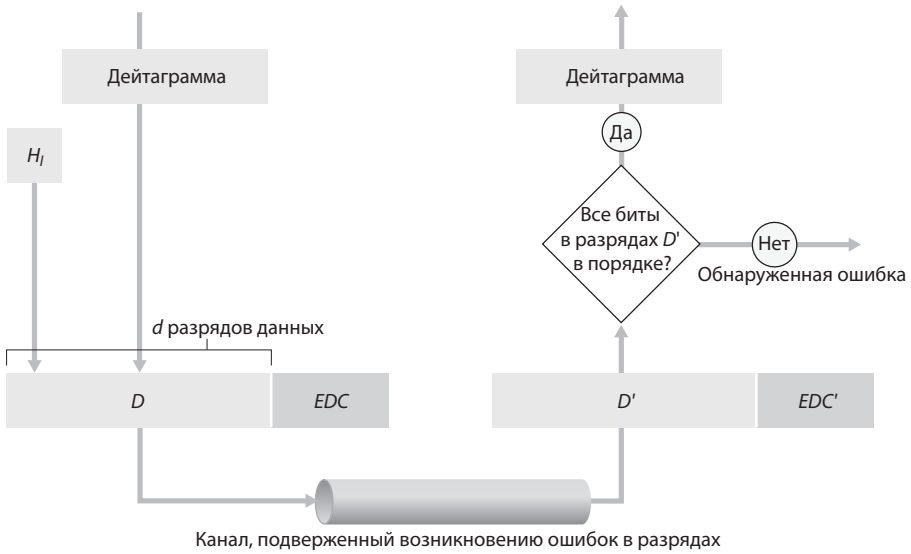


Рис. 5.3. Сценарий обнаружения и исправления ошибок

На основании принятых полей *D'* и *EDC'* получатель должен определить, совпадают ли разряды *D'* с разрядами *D*. Обратите внимание, что вопрос заключается в том, обнаружена ли ошибка, а не в том, прои-

зошла ли она! Методы обнаружения и исправления ошибок иногда, *но не всегда*, позволяют получателю обнаруживать, что произошла ошибка в одном или нескольких разрядах кадра. Другими словами, даже при добавлении к данным дополнительных бит с этой целью остается вероятность, что **ошибка не будет обнаружена**, а получатель не получит информации об искажении полученных данных. В результате канальный уровень получателя может передать сетевому уровню поврежденную дейтаграмму или не заметить повреждения какого-либо другого поля в заголовке кадра. Поэтому следует выбирать такую схему определения ошибок, при которой вероятность подобных событий мала. Как правило, чем изощреннее методы обнаружения и исправления ошибок (снижающие вероятность появления необнаруженных ошибок), тем больше издержки — требуется больше операций для вычисления контрольной суммы и больше времени для передачи дополнительной информации.

Мы рассмотрим три метода обнаружения ошибок в переданных данных: контроль четности (чтобы проиллюстрировать основные идеи, лежащие в основе методов обнаружения и исправления ошибок), метод контрольных сумм (больше характерный для транспортного уровня) и коды циклического контроля (как правило, используемые в адаптерах канального уровня).

5.2.1. Контроль четности

Возможно, простейшая форма обнаружения ошибок заключается в использовании одного **бита четности**. Предположим, что на рис. 5.3 передаваемые данные D имеют длину d разрядов. При проверке на четность отправитель просто добавляет к данным один бит, таким образом, чтобы сумма всех единиц в d+1 бит (исходная информация плюс бит четности) оказалась четной. В схемах с проверкой нечетности значение бита выбирается таким образом, чтобы получилось нечетное количество единиц. На рис. 5.4 проиллюстрирован контроль четности, единственный бит четности сохраняется в отдельном поле.

Действия, выполняемые получателем при использовании такой схемы, также очень просты. Он должен всего лишь сосчитать количество единиц в полученных d+1 разрядах. Если при проверке на четность получатель обнаруживает, что в принятых им данных нечетное количество единичных разрядов, он понимает, что произошла ошибка по меньшей мере в одном разряде. В общем случае это означает, что в полученных

данных инвертировано *нечетное* количество разрядов (произошла ошибка нечетной кратности).

Рис. 5.4. Контроль четности с добавлением 1 бита

Что произойдет, если в прибывшем пакете данных произойдет четное количество однобитовых ошибок? В этом случае получатель не сможет их обнаружить. Если вероятность ошибки в одном разряде мала и можно предположить, что ошибки в отдельных разрядах возникают независимо друг от друга, тогда вероятность нескольких ошибок в одном пакете крайне низкая. В таком случае единственного бита четности может быть достаточно. Однако практические наблюдения показали, что в действительности ошибки не являются независимыми, а часто группируются в пакеты ошибок. В случае пакетных ошибок вероятность того, что получатель не обнаружит ошибку в пакете, может приблизиться к величине 50%[610]. Очевидно, в такой ситуации требуется более надежная схема обнаружения ошибок (и она действительно применяется на практике). Но прежде чем перейти к изучению схем, применяемых на практике, рассмотрим простой пример, который обобщает предыдущую схему одноразрядного контроля четности и помогает понять принцип работы методов исправления ошибок.

На рис. 5.5 показано двумерное обобщение одноразрядной схемы проверки на четность. В данной схеме d разрядов пакета данных D разделяются на i строк и j столбцов, образуя прямоугольную матрицу. Значение четности вычисляется для каждой строки и каждого столбца. Получающиеся в результате $i+j+1$ битов четности образуют разряды обнаружения ошибок кадра канального уровня.

Предположим теперь, что в исходном блоке данных из d разрядов происходит однократная ошибка. В такой **двумерной схеме контроля четности** об ошибке будут одновременно сигнализировать контрольные разряды строки и столбца. Таким образом, получатель сможет не только *обнаружить* сам факт ошибки, но и по номерам строки и столбца найти поврежденный бит данных и *исправить* его. На рис. 5.5 показан пример, в котором поврежден бит в позиции (2, 2) — он изменил свое значение

с 1 на 0. Такую одиночную ошибку получатель может не только обнаружить, но и исправить. Хотя нас, в первую очередь, интересует обнаружение и исправление ошибок в исходных d разрядах, данная схема позволяет также обнаруживать и исправлять одиночные ошибки в самих битах четности. Кроме того, данная двумерная схема контроля четности позволяет обнаруживать (но не исправлять!) любые комбинации из двух одиночных ошибок (то есть двойные ошибки) в пакете. Другие свойства двумерной схемы рассматриваются в упражнениях в конце этой главы.

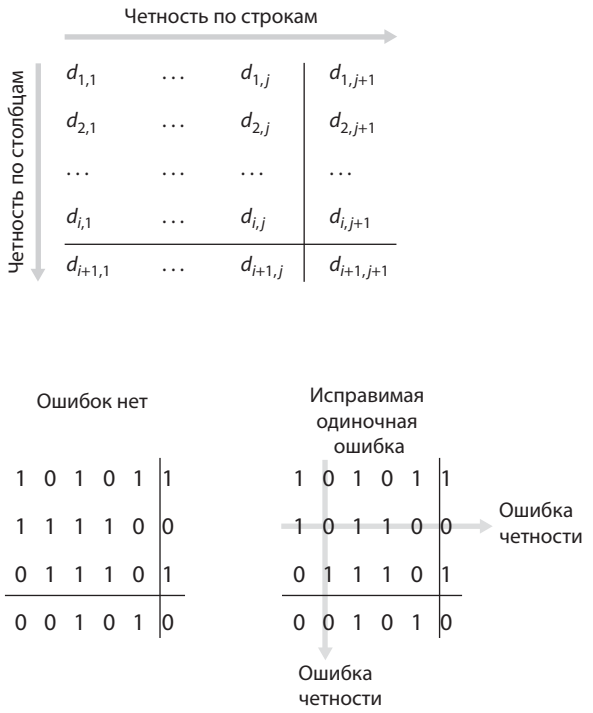


Рис. 5.5. Двумерный контроль четности

Способность получателя обнаруживать и исправлять ошибки иногда называют **прямым исправлением ошибок** (Forward Error Correction, **FEC**). Подобные приемы широко применяются в устройствах хранения и воспроизведения звука, например на лазерных компакт-дисках. В сетях методы обнаружения и исправления ошибок могут использоваться сами по себе, а также в сочетании с автоматическими запросами на повторную передачу, которые мы рассматривали в главе 3. Методы прямого обнаружения и исправления ошибок очень полезны, так как по-

зволяют снизить необходимое количество повторных передач. Кроме того (что, возможно, даже важнее), эти методы позволяют получателю немедленно исправлять ошибки. Таким образом, получателю данных не нужно ждать, пока отправитель примет его сообщение об ошибке и вышлет пакет еще раз, что может быть существенным преимуществом в сетевых приложениях реального времени[577] или при использовании каналов, для которых характерны длительные задержки распространения (например, таковы каналы для передачи информации в космосе). Исследования методов прямого обнаружения и исправления ошибок, представлены в работах Бирсека[52], Нонненмахера[372], Байерса[67] и Шачема[597].

5.2.2. Методы контрольных сумм

Методы, основанные на использовании контрольных сумм, обрабатывают d разрядов данных (см. рис. 5.4) как последовательность k-разрядных целых чисел. Наиболее простой метод заключается в простом суммировании этих k-разрядных целых чисел и использовании полученной суммы в качестве битов определения ошибок. Так работает *алгоритм вычисления контрольной суммы, принятый в Интернете*, — байты данных группируются в 16-разрядные целые числа и суммируются. Затем от суммы берется обратное значение (дополнение до 1), которое и помещается в заголовок сегмента. Как уже отмечалось в разделе 3.3, получатель проверяет контрольную сумму, вычисляя дополнение до 1 от суммы полученных данных (включая контрольную сумму), и сравнивает результат с числом, все разряды которого равны 1. Если хотя бы один из разрядов результата равен 0, это означает, что произошла ошибка. Принятый в Интернете алгоритм вычисления контрольной суммы и его реализация подробно описываются в RFC 1071. В протоколах TCP и UDP контрольная сумма вычисляется по всем полям (включая поля заголовка и данных). В других протоколах, например XTP[623], вычисляются две контрольные суммы, одна для заголовка, а другая для всего пакета.

Метод вычисления контрольной суммы для пакетов требует относительно небольших накладных расходов. Например, в протоколах TCP и UDP для контрольной суммы используются всего 16 бит. Однако подобные методы предоставляют относительно слабую защиту от ошибок по сравнению с обсуждаемым далее методом контроля с помощью кода циклического контроля, который часто использу-

ется на канальном уровне. Разумеется, возникает вопрос: почему на транспортном уровне применяют контрольные суммы, а на канальном уровне — код циклического контроля? Вспомним, что транспортный уровень, как правило, реализуется на хосте программно как часть операционной системы хоста. Поскольку обнаружение ошибок на транспортном уровне реализовано программно, важно, чтобы схема обнаружения ошибок была простой. В то же время обнаружение ошибок на канальном уровне реализуется аппаратно в адаптерах, способных быстро выполнять более сложные операции по вычислению кода циклического контроля.

Филдмайер[160] анализирует быстрые методы программной реализации для вычисления не только взвешенных кодов контрольных сумм, но и циклических (см. ниже), а также других кодов.

5.2.3. Код циклического контроля

Широко применяемый в современных компьютерных сетях метод обнаружения ошибок основан на использовании кода. **Коды циклического контроля** (Cyclic Redundancy Check, **CRC**) также называют **полиномиальными кодами**, так как при их вычислении битовая строка рассматривается как многочлен (полином), коэффициенты которого равны 0 или 1, и операции с этой битовой строкой можно интерпретировать как операции деления и умножения многочленов.

Циклические коды работают следующим образом. Рассмотрим фрагмент данных *D,* которые передающий узел хочет отправить принимающему узлу. Отправитель и получатель должны договориться о последовательности из $r+1$ бит, называемой **образующим многочленом** (или **генератором**), который мы будем обозначать *G*. Старший (самый левый) бит генератора *G* должен быть равен 1. Ключевую идею кода циклического контроля иллюстрирует рис. 5.6. Для заданного фрагмента данных *D* отправитель формирует r дополнительных разрядов *R*, которые он добавляет к данным *D* так, что получающееся в результате число, состоящее из $d+r$ бит, делится по модулю 2 на генератор (число) *G* без остатка. Таким образом, процесс проверки данных на наличие ошибки относительно прост. Получатель делит полученные $d+r$ бит на генератор *G*. Если остаток от деления не равен нулю, это означает, что произошла ошибка. В противном случае данные считаются верными и принимаются.

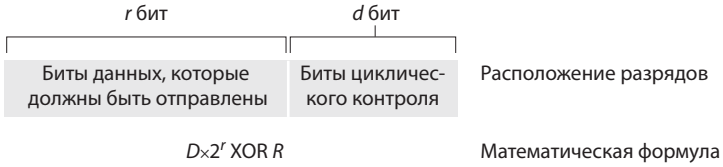


Рис. 5.6. Код циклического контроля

Все операции по вычислению CRC-кода производятся в арифметике по модулю 2 без переносов в соседние разряды. Это означает, что операции сложения и вычитания идентичны друг другу и эквивалентны поразрядной операции исключающего ИЛИ (exclusive OR, XOR). Например:

```
1011 XOR 0101 = 1110
1001 XOR 1101 = 0100
```

Кроме того, мы получим:

```
1011 – 0101 = 1110
1001 – 1101 = 0100
```

Операции умножения и деления аналогичны соответствующим операциям в обычной двоичной арифметике с той разницей, что любая требующаяся при этом операция сложения или вычитания выполняется без переносов в соседний разряд. Как и в обычной арифметике, умножение на 2^k означает сдвиг числа на k разрядов влево. Таким образом, при заданных значениях D и R величина $D{\times}2^r$ XOR R соответствует последовательности из $d{+}r$ бит, показанной на рис. 5.6. В нашем дальнейшем обсуждении мы будем использовать именно эту алгебраическую форму для обозначения последовательности из $d{+}r$ бит.

Вернемся теперь к главному вопросу: как отправитель вычисляет R? Вспомним, что нам требуется найти такое значение R, чтобы для некоторого n выполнялось следующее равенство:

$$D{\times}2^r \text{ XOR } R = nG$$

То есть требуется выбрать такое значение R, чтобы $D{\times}2^r$ XOR R делилось на G без остатка. Если прибавить к каждой части уравнения значение R по модулю 2, то мы получим

$$D{\times}2^r = nG \text{ XOR } R$$

Отсюда следует, что если мы разделим $D{\times}2^r$ на G, то значение остатка будет равно R. Таким образом, мы можем вычислить R как

$$R = \text{остаток } \frac{D{\times}2^r}{G}$$

На рис. 5.7 показан пример вычисления R в случае $D = 101110$, $d = 6$, $G = 1001$ и $r = 3$. В этом случае отправитель передает следующие 9 бит: 101110011. Вы можете сами проверить правильность расчетов, а также убедиться, что $D{\times}2^r = 101011 \times G$ XOR R.

Для 8-, 12-, 16- и 32-разрядных генераторов G определены международные стандарты. В стандарте CRC-32, принятом в ряде IEEE-протоколов канального уровня, используется генератор вида

$$G_{CRC-32} = 10000010011000001000111011011 0111$$

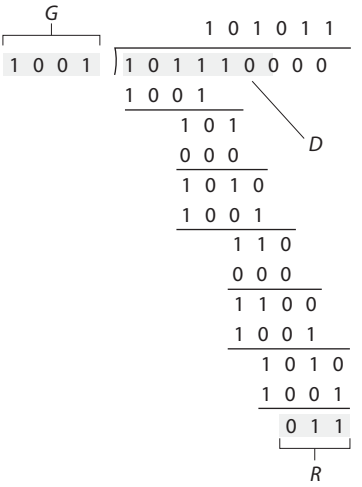


Рис. 5.7. Пример вычисления CRC

Каждый стандарт CRC-кода способен обнаруживать ошибочные пакеты длиной не более r разрядов. Кроме того, при соответствующих допущениях ошибочный пакет длиной более чем r разрядов будет обнаружен с вероятностью $1 - 0{,}5^r$. Помимо этого каждый стандарт CRC-кода может обнаруживать ошибки нечетной кратности. В работе Вильямса[667] рассматриваются вопросы осуществления проверок при помощи кода циклического контроля. Теория CRC-кодов и еще более мощных кодов выходит за рамки данной книги. Прекрасное введение в эту тему можно найти в работе Швартца[593].

5.3. Протоколы и каналы множественного доступа

В начале главы мы отметили, что существуют два типа сетевых каналов: двухточечные и широковещательные. **Двухточечная линия связи** соединяет отправителя на одном конце линии и получателя на другом. Для двухточечных линий связи разработано множество протоколов канального уровня. Ниже в этой главе будет рассказано о двух таких протоколах: PPP (Point-to-Point Protocol — протокол передачи от точки к точке) и HDLC (High-level Data Link Control — высокоуровневый протокол управления каналом). Второй тип канала, **широковещательный канал**, может иметь несколько передающих и принимающих узлов, присоединенных к одному и тому же совместно используемому широковещательному каналу. Термин «широковещание» используется здесь потому, что, когда один из узлов передает кадр, этот кадр принимается всеми остальными узлами, присоединенными к каналу. Примерами использования широковещательной технологии канального уровня являются Ethernet-сети и беспроводные локальные сети. В данном разделе мы сделаем небольшое отступление от темы протоколов канального уровня и сначала рассмотрим крайне важную проблему, заключающуюся в координации доступа множества передающих и принимающих узлов к общему широковещательному каналу, — так называемую **проблему множественного доступа**. Широковещательные каналы часто применяют в локальных сетях, то есть в сетях, географически сконцентрированных в одном здании (или комплексе зданий, принадлежащих одной организации, например университету или компании). В конце этого раздела мы познакомимся с тем, как используются каналы коллективного доступа в локальных сетях.

Нам всем известно понятие широковещательной рассылки, так как эта технология передачи данных используется в телевидении с момента его изобретения. Но в традиционном телевидении широковещательная рассылка является односторонней, так как там один фиксированный узел передает информацию множеству получающих узлов. Между тем узлы компьютерной широковещательной сети могут как принимать данные, так и передавать их. Возможно, более близкой к компьютерной широковещательной сети аналогией является вечеринка с коктейлями, где множество людей собираются в большой комнате (средой широковещательного канала при этом является воздух), чтобы поговорить и послушать. Вторая аналогия, хорошо знакомая многим читателям, — аудитория, в которой преподаватель и студенты совместно используют одну общую широковещательную среду передачи. Центральная пробле-

ма обоих сценариев заключается в том, чтобы решить, кто и когда получает право говорить (передавать по каналу). Для себя люди разработали сложный набор правил коллективного использования канала:

«дайте возможность поговорить каждому»;

«не говорите, пока с вами не заговорят»;

«не монополизируйте беседу»;

«если у вас есть вопрос, поднимите руку»;

«не прерывайте говорящего громким храпом».

Обмен данными в компьютерных сетях также управляется наборами правил, составляющих так называемые **протоколы множественного доступа**. Как показано на рис. 5.8, протоколы множественного доступа применяются в сетях самых разных конфигураций, включая кабельные и беспроводные локальные сети, а также спутниковые сети. Хотя технически каждый узел получает доступ к широковещательному каналу через адаптер, в данном разделе и передающее, и принимающее устройства мы будем называть *узлом*. На практике по одному широковещательному каналу могут обмениваться данными сотни или даже тысячи узлов.

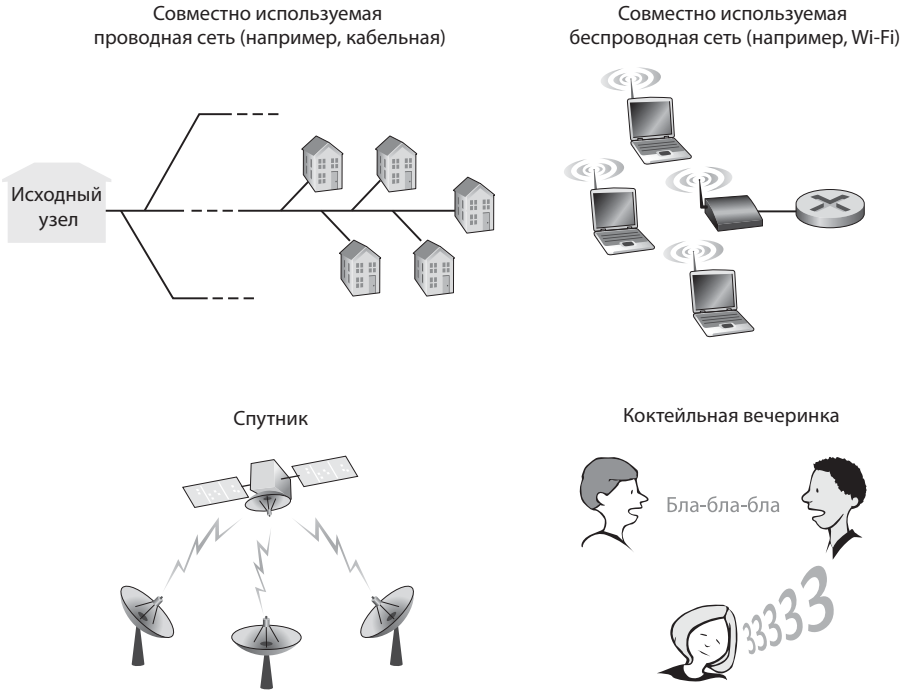


Рис. 5.8. Различные каналы множественного доступа

Поскольку передавать кадры могут все узлы, возможна ситуация, когда одновременно начнут передачу несколько узлов. Когда такое происходит, каждый из узлов одновременно получает несколько кадров, то есть на принимающих узлах имеет место **коллизия** переданных кадров. Как правило, в случае коллизии принимающие узлы не могут корректно обрабатывать принятые кадры, так как сигналы таких кадров накладываются друг на друга. Таким образом, все вовлеченные в коллизию кадры теряются, а все время, пока они передавались, оказывается потраченным впустую. Очевидно, что наличие множества узлов, требующих частой передачи данных, означает высокую вероятность коллизий и низкий коэффициент использования канала.

Чтобы гарантировать высокую производительность канала при большом количестве активных узлов, необходимо каким-то образом координировать передачу ими кадров. За эту координацию отвечает протокол коллективного доступа. За последние 40 лет по теме протоколов коллективного доступа были написаны тысячи статей и защищены сотни докторских диссертаций. Всесторонний обзор этой работы можно найти в книге Рома[569]. Более того, активные исследования протоколов коллективного доступа продолжаются до сих пор, что вызвано появлением новых типов линий связи, например новых беспроводных каналов.

За годы работы с помощью широкого спектра технологий канального уровня были реализованы десятки протоколов множественного доступа. Тем не менее практически любой из этих протоколов мы можем отнести к одной из трех категорий: **протоколы разделения канала, протоколы произвольного доступа** и **протоколы поочередного доступа**. Мы обсудим эти категории протоколов коллективного доступа в следующих трех подразделах.

В заключение этого обзора отметим, что в идеальном случае протокол множественного доступа для широковещательного канала со скоростью передачи данных R бит/с должен обладать следующими характеристиками.

1. Когда данные для передачи есть только у одного узла, этот узел обладает пропускной способностью в R бит/с.
2. Когда данные для передачи есть у M узлов, каждый из этих узлов обладает пропускной способностью в R/M бит/с. Это не означает, что каждый из M узлов в каждый момент времени может передавать данные со скоростью R/M бит/с, — это средняя скорость передачи данных каждого из узлов.

3. Протокол является децентрализованным, то есть не существует управляющих узлов, выход из строя которых может остановить работу всей сети.
4. Протокол прост и дешев в реализации.

5.3.1. Протоколы разделения канала

Вспомним раздел 1.3, где шла речь о двух методах разделения пропускной способности широковещательного канала между всеми узлами, использующими этот канал. Мы говорили о мультиплексировании с временным разделением (Time-Division Multiplexing, TDM) и мультиплексировании с частотным разделением (Frequent-Division Multiplexing, FDM). Предположим для примера, что канал поддерживает N узлов и скорость передачи данных в канале равна R бит/с. При временном разделении канала время делится на интервалы, называемые **кадрами**, каждый из которых делится на N элементарных интервалов времени, называемых **квантами**. (Не путайте кадр времени в контексте TDM с *кадром* как минимальной единицей передачи данных на канальном уровне между исходным и конечным адаптерами. Чтобы не допускать излишней путаницы, в этом разделе мы будем называть кадр обмена данными на канальном уровне словом «пакет».) Затем каждому из N узлов назначается один временной квант. Когда у узла есть кадр для отправки, он передает биты этого кадра в течение назначенного ему элементарного интервала времени. Как правило, длина кванта выбирается таким образом, чтобы за этот интервал можно было передать один кадр. На рис. 5.9 показан простой пример мультиплексирования с временным разделением для канала с четырьмя узлами. Если вернуться к нашей аналогии с вечеринкой, то при такой схеме каждый участник вечеринки сможет говорить в течение некоторого фиксированного интервала времени, после чего такое же право получает другой участник, и т. д. После того как возможность поговорить получат все участники вечеринки, она снова переходит к первому участнику, и все повторяется.

Привлекательность временного разделения канала заключается в том, что такая схема полностью устраняет конфликты (коллизии) и обладает идеальной справедливостью: каждый узел получает выделенную скорость передачи данных, равную R/N бит/с в течение каждого временного кадра. Однако у такой схемы есть два существенных недостатка. Во-первых, каждый узел ограничен средней скоростью передачи данных в R/N бит/с даже в том случае, когда этот узел единственный,

кому нужно отсылать данные в этот момент. Во-вторых, при передаче узел всегда должен ждать своей очереди, даже когда кроме него никто не отправляет данные. Сложно представить себе вечеринку, где лишь одному гостю есть что сказать (и тем более маловероятно, что все остальные гости готовы его слушать). Таким образом, очевидно, что временное разделение канала плохо подходит для протокола коллективного доступа.

Метод мультиплексирования с частотным разделением делит канал с пропускной способностью R бит/с на частотные диапазоны с полосой пропускания R/N бит/с, при этом каждому узлу выделяется собственный частотный диапазон. Таким образом, при методе частотного разделения из одного канала с пропускной способностью R бит/с создается N каналов с пропускной способностью R/N бит/с. Мультиплексирование с частотным разделением канала обладает теми же преимуществами и недостатками, что и с временным. Устраняются коллизии и обеспечивается справедливое распределение пропускной способности между узлами, но пропускная способность каждого узла ограничена значением R/N бит/с, независимо от текущей загруженности канала.

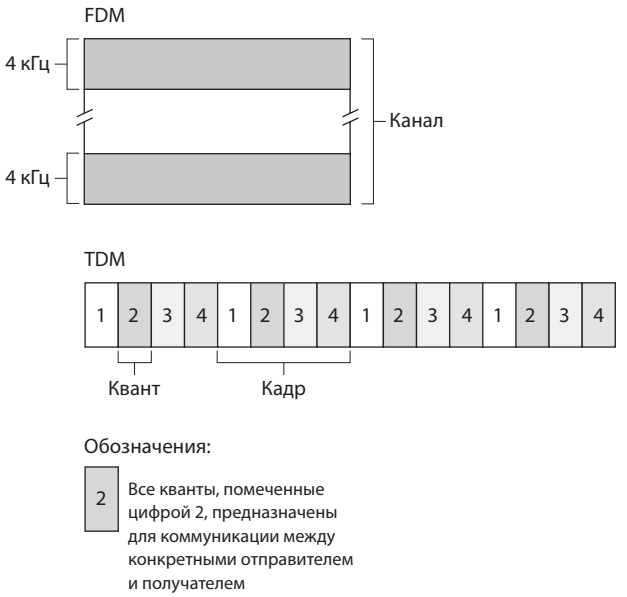


Рис. 5.9. Примеры мультиплексирования с временным и частотным разделением для системы с четырьмя узлами

Еще один метод совместного использования общего канала, который мы рассмотрим, предлагает протокол **множественного доступа с кодо-**

вым разделением (Code Division Multiple Access, **CDMA**). В отличие от схем мультиплексирования с частотным и временным разделением канала, предоставляющих узлам частотные диапазоны или временные интервалы, протокол CDMA назначает каждому узлу собственный *код*. Затем каждый узел использует этот уникальный код для кодирования передаваемых данных. Как мы увидим, протокол CDMA позволяет нескольким узлам передавать данные *одновременно*, при этом получатели могут корректно их принимать (при условии, что получателю известен код передатчика), даже при воздействии помех со стороны других узлов. Протокол CDMA в течение некоторого времени использовался в военных системах связи (благодаря своей устойчивости к попыткам подавления сигнала), а в настоящее время получает все более широкое распространение в гражданских беспроводных средствах связи коллективного доступа. Поскольку использование CDMA так тесно связано с темой беспроводных каналов, мы отложим его подробное рассмотрение до главы 6. Пока будет достаточно сказать, что коды CDMA, подобно квантам времени в TDM и частотам в FDM, могут выделяться сразу многим пользователям канала коллективного доступа.

5.3.2. Протоколы произвольного доступа

Второй широкий класс протоколов коллективного доступа составляют так называемые протоколы произвольного доступа. В протоколе произвольного доступа передающий узел всегда передает данные в канал с максимальной скоростью, то есть R бит/с. Когда возникает коллизия, каждый вовлеченный в нее узел отправляет свой кадр (то есть пакет) повторно до тех пор, пока ему не удастся осуществить передачу без коллизий. Однако, претерпев коллизию, узел, как правило, не повторяет передачу сразу же, а *выжидает в течение случайного интервала времени*. Благодаря разной длительности случайных интервалов времени существует ненулевая вероятность того, что интервал, выбранный одним из узлов, окажется меньше, чем у других вовлеченных в коллизию узлов, и он успеет вытолкнуть свой кадр в канал беспрепятственно.

В литературе описаны десятки, если не сотни, протоколов произвольного доступа[569, 50]. В данном разделе мы опишем некоторые из наиболее популярных, включая протоколы ALOHA[5, 6, 7] и CSMA[289]. Кстати, Ethernet[342] — это популярный и широко используемый протокол CSMA.

Дискретный протокол ALOHA

Начнем наше изучение протоколов произвольного доступа с одного из наиболее простых, так называемого дискретного протокола ALOHA. В нашем описании мы будем предполагать следующее:

- все кадры состоят ровно из L бит;
- время разделено на интервалы времени (кванты) длительностью L/R с (это время, за которое передается один кадр);
- узлы начинают передачу кадров только в момент начала очередного кванта;
- узлы синхронизируются так, что каждый из них знает, когда начинается квант;
- если в течение данного временного кванта сталкиваются несколько кадров, тогда все узлы обнаруживают факт коллизии, прежде чем закончится данный квант.

Обозначим через p вероятность — то есть p будет иметь значение 0 или 1. Работа дискретного протокола ALOHA на каждом узле проста:

- Когда у узла появляется новый кадр для отправки, он дожидается начала следующего кванта времени и за этот квант передает весь кадр.
- Если коллизия не возникает, это значит, что узел успешно передал свой кадр и повторная передача не требуется. (Если у узла есть новый кадр, то он может подготовить его к отправке.)
- Если коллизия возникает, то узел замечает ее до окончания кванта времени. Узел будет пытаться передать свой кадр в течение последующих квантов времени, что рано или поздно удастся ему с вероятностью p.

Под повторной передачей кадра с вероятностью p мы подразумеваем, что узел как бы подбрасывает монету. При этом кадр передается повторно, только если выпадает решка, что происходит с вероятностью p. При выпадении орла, что происходит с вероятностью $(1 - p)$, узел пропускает данный временной интервал и подбрасывает монетку еще раз. Все узлы, вовлеченные в коллизию, подбрасывают свои монетки независимо друг от друга. Казалось бы, дискретный протокол ALOHA обладает рядом достоинств. В отличие от протоколов мультиплексирования с разделением канала, он позволяет единственному активному в сети узлу передавать

кадры без перерыва с максимальной скоростью R (узел называется активным, если у него есть кадр для передачи). Дискретный протокол ALOHA является в большой степени децентрализованным, так как каждый узел сам обнаруживает факт коллизии и независимо от других узлов принимает решение о времени повторной передачи. (Однако для использования дискретного протокола ALOHA требуется синхронизация узлов. Далее мы кратко обсудим непрерывную версию протокола ALOHA, а также протоколы CSMA, не требующие подобной синхронизации.) Кроме того, ALOHA — чрезвычайно простой протокол.

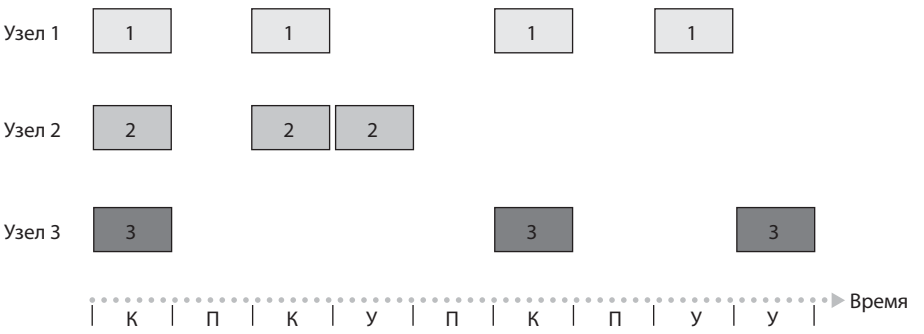


Рис. 5.10. Между узлами 1, 2 и 3 возникает коллизия в первый квант времени. Узел 2 наконец срабатывает в четвертом кванте времени, узел 1 — в восьмом, а узел 3 — в девятом

Дискретный протокол ALOHA хорошо работает в тех ситуациях, когда имеется только один активный узел, но какова его эффективность при наличии нескольких активных узлов? Ее снижают два фактора. Во-первых, как показано на рис. 5.10, когда в сети несколько активных узлов, определенная доля квантов тратится впустую из-за коллизий. Во-вторых, другая доля квантов тратится напрасно, когда все активные узлы одновременно отказываются от передачи в силу ее вероятностного механизма. Дискретный протокол ALOHA работает продуктивно только в те кванты, когда передавать требуется ровно одному узлу. Квант, на протяжении которого передает только один узел, называется **успешным квантом**. **Эффективность** дискретного протокола коллективного доступа определяется долей успешных квантов при наличии большого количества активных узлов, у каждого из которых всегда есть большое

количество кадров для передачи. Обратите внимание, что, если вообще не использовать протоколы коллективного доступа и после коллизии немедленно повторять передачу каждым из узлов, эффективность сети была бы равна нулю. Дискретный протокол ALOHA очевидно увеличивает эффективность сети, но насколько?

Попытаемся определить максимальную эффективность дискретного протокола ALOHA. Чтобы упростить наши вычисления, немного изменим протокол, предположив, что каждый узел с вероятностью p пытается передать кадр с наступлением каждого нового кванта. То есть мы предполагаем, что у каждого узла всегда есть кадр для передачи и узел всегда с вероятностью p пытается передать его независимо от того, новый это кадр или передаваемый повторно. Пусть в сети будет N узлов. В этом случае квант оказывается успешным, если один из узлов передает, а $N-1$ узлов воздерживаются от передачи. Вероятность того, что некий конкретный узел будет передавать, равна p. Вероятность того, что остальные $N-1$ узлов не будут передавать, равна $(1-p)^{N-1}$. Таким образом, вероятность того, что данному узлу удастся успешно передать кадр, равна $p(1-p)^{N-1}$. Поскольку существует N узлов, вероятность того, что повезет одному (любому) из них, равна $Np(1-p)^{N-1}$.

Таким образом, при наличии N активных узлов эффективность дискретного протокола ALOHA равна $Np(1-p)^{N-1}$. Чтобы определить *максимальную* эффективность протокола при N активных узлах, нам нужно найти такое значение вероятности p^*, при котором данное выражение достигает максимума (см. упражнения в конце этой главы, относящиеся к данной теме). А чтобы получить максимальную эффективность протокола для большого количества активных узлов, мы можем найти предел значения $Np^*(1-p^*)^{N-1}$ для значения N, стремящегося к бесконечности (опять же, см. упражнения в конце главы). Выполнив все эти вычисления, мы обнаружим, что максимальная эффективность протокола составляет $1/e = 0{,}37$. Таким образом, когда у большого количества узлов имеется много кадров для передачи, то (в лучшем случае) только около 37% квантов канал будет тратить на полезную работу. То есть эффективная пропускная способность канала равна не R бит/с, а всего лишь 0,37 R бит/с! Оказывается, что около 37% времени канал будет простаивать и еще около 26% времени тратить на обработку коллизий. Представьте себе несчастного администратора вычислительной сети, приобретшего систему с дискретным ALOHA с пропускной способностью 100 Мбит/с и собирающегося использовать ее для обслуживания трафика между большим количеством пользователей с суммарной про-

пускной способностью около 80 Мбит/с! Несмотря на то, что мгновенная пропускная способность канала равна 100 Мбит/с, его эффективная пропускная способность составит менее 37 Мбит/с.

«Чистый» протокол ALOHA

Дискретный протокол ALOHA требует, чтобы все узлы синхронизировали время начала передачи кадров. Собственно, первый протокол ALOHA[5] не был дискретным и являлся полностью децентрализованным. В так называемом чистом протоколе ALOHA, когда прибывает первый кадр (то есть дейтаграмма сетевого уровня передается на более низкий уровень передающего узла), узел немедленно передает весь кадр целиком в широковещательный канал. Если переданный кадр сталкивается с одним или несколькими другими кадрами (происходит коллизия), с вероятностью p узел немедленно передает кадр повторно. В противном случае узел выжидает в течение времени, необходимого для передачи одного кадра, после чего опять с вероятностью p передает кадр либо пережидает еще один интервал времени с вероятностью $1-p$.

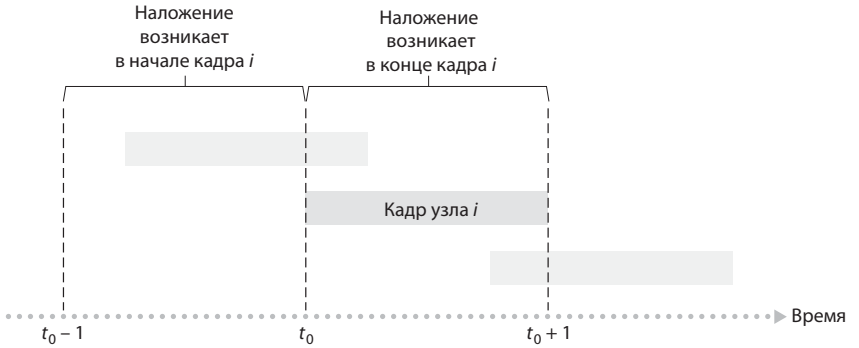


Рис. 5.11. Наложение передачи данных в чистом протоколе ALOHA

Чтобы определить максимальную эффективность чистого протокола ALOHA, обсудим отдельный узел. Мы будем использовать те же допущения, что и в случае дискретного протокола ALOHA, и примем за единицу времени интервал (квант), требующийся для передачи одного кадра. В любой момент времени вероятность того, что узел передает кадр, равна p. Предположим, передача этого кадра началась в момент времени t_0. Как видно из рис. 5.11, чтобы этот кадр был передан успешно, никакой другой узел не должен начать свою передачу во временном интервале $[t_0-1,t_0]$, так как иначе она совпадет по времени с началом отправки нашего узла i.

Вероятность того, что остальные узлы не начнут передачу в течение этого интервала времени, равна $(1-p)^{N-1}$. Аналогично никакой другой узел не должен начать свою передачу, пока передает наш узел *i*, так как такая передача также приведет к коллизии, но уже с концом нашего кадра *i*. Вероятность этого события также равна $(1-p)^{N-1}$. Таким образом, вероятность успешной передачи кадра данным узлом становится равной $p(1-p)^{2(N-1)}$. При стремлении количества узлов к бесконечности максимальная эффективность чистого протокола ALOHA будет равна всего лишь $1/(2e)$, то есть половине от максимальной эффективности дискретного протокола ALOHA. Такова плата за полную децентрализацию.

ИСТОРИЯ

Норм Абрамсон и ALOHAnet

Доктор философии Норм Абрамсон любил серфинг и интересовался коммутацией пакетов. Это сочетание увлечений привело его в 1969 году в Гавайский университет. Гавайи представляют собой множество гористых островков, на которых трудно установить традиционную локальную сеть. В свободное от серфинга время Абрамсон размышлял о том, как разработать сеть с коммутацией пакетов, передаваемых по радио. У спроектированной им сети был один центральный хост и несколько второстепенных узлов, разбросанных по Гавайским островам. У сети было два канала, для каждого из которых использовался свой частотный диапазон. По нисходящему широковещательному каналу пакеты рассылались от центрального хоста остальным узлам. По восходящему каналу остальные узлы посылали пакеты центральному хосту. Помимо информационных пакетов центральный хост также посылал подтверждения для каждого успешно принятого пакета.

Поскольку второстепенные узлы передавали пакеты децентрализованно, в восходящем канале неизбежно возникали коллизии. Это наблюдение натолкнуло Абрамсона на идею протокола ALOHA (чистого), описанного в этой главе. В 1970 году Абрамсон при финансовой поддержке управления ARPA соединил сеть ALOHAnet с сетью ARPAnet. Эта работа не только привела к рождению первой беспроводной сети с коммутацией пакетов, но еще и вдохновила Боба Меткалфа на создание на основе ALOHA протокола CSMA/CD и локальной Ethernet-сети.

Протокол множественного доступа с контролем несущей (CSMA)

В обоих вариантах протокола ALOHA, дискретном и чистом, узел принимает решение о передаче кадра независимо от активности осталь-

ных узлов, присоединенных к широковещательному каналу. В частности, узел не обращает внимания на то, ведется ли в данный момент передача другими узлами, и не прекращает ее в случае коллизий. Если вспомнить нашу аналогию с вечеринкой, протоколы ALOHA подобны невоспитанным людям, прерывающим чужую беседу и продолжающим говорить, несмотря на то что в нее вступили другие участники вечеринки. У людей также есть свои протоколы, позволяющие им не только вести себя более цивилизованно, но и тратить меньше времени на «коллизии» друг с другом и, таким образом, повышать «производительность» беседы. В частности, существуют два важных правила вежливого разговора.

- *Слушайте, прежде чем говорить.* Если кто-то уже говорит, подождите, пока он не закончит. В мире компьютерных сетей это правило называется **контролем несущей** и заключается в том, что узел прослушивает канал перед тем, как начать передачу. Если по каналу передается кадр, узел выжидает («отступает») в течение случайного периода времени, а затем начинает передачу.
- *Если кто-то начал говорить, прекращайте разговор.* В мире компьютерных сетей это правило называется **обнаружением коллизий**. Оно заключается в том, что во время передачи узел прослушивает канал. Если он обнаруживает, что другой узел в этот момент времени тоже ведет передачу, он прекращает свою и выжидает в течение случайного периода времени, после чего начинает новый цикл проверки канала и передачи, если канал оказывается не занят.

Эти два правила реализованы в семействе протоколов CSMA (Carrier Sense Multiple Access — **множественный доступ с контролем несущей**) и CSMA/CD (CSMA with Collision Detection — **множественный доступ с контролем несущей и обнаружением коллизий**). Было разработано множество вариантов протоколов CSMA и CSMA/ CD[569, 289, 342, 309]. Далее будут рассмотрены наиболее важные и фундаментальные свойства CSMA и CSMA/CD.

Первый вопрос о протоколе CSMA, который может возникнуть, состоит в том, как вообще могут возникать коллизии, если все узлы прослушивают несущую? В самом деле, ведь узел воздерживается от передачи, если он замечает, что канал занят. Ответ на этот вопрос лучше всего проиллюстрировать с помощью пространственно-временной диаграммы[350]. На рис. 5.12 показана пространственно-временная диаграмма работы четырех узлов (А, Б, В, Г), присоединенных к линейной

широковещательной шине. На горизонтальной оси фиксируется положение каждого узла в пространстве, вдоль вертикальной оси изменяется время.

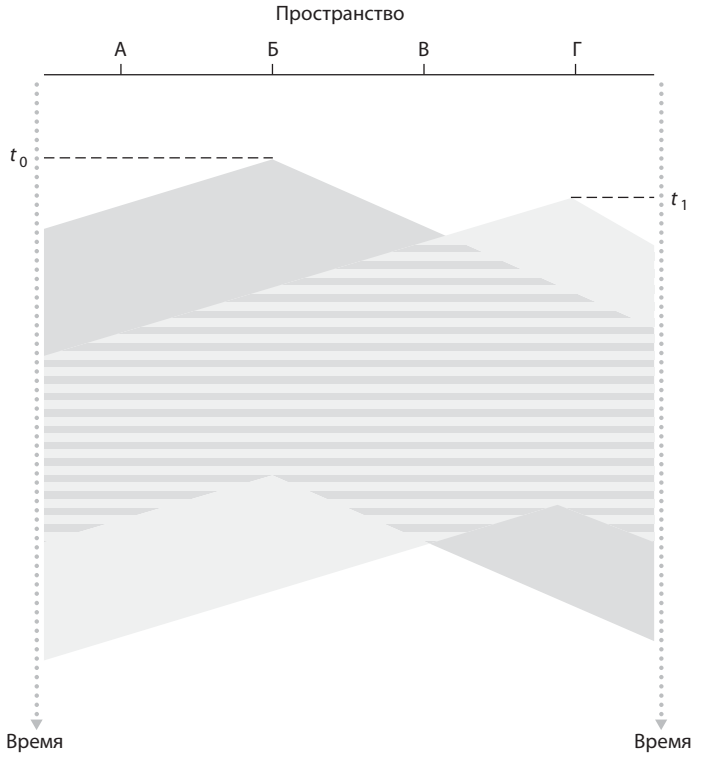


Рис. 5.12. Пространственно-временная диаграмма с двумя узлами CSMA, между которыми возникает коллизия при передаче

В момент времени t_0 узел Б опрашивает канал и приходит к выводу, что канал свободен, так как никакой другой узел в этот момент не ведет передачу. Поэтому узел Б начинает передачу, и передаваемый им сигнал распространяется по широковещательному носителю в обоих направлениях со скоростью, близкой к скорости света, но конечной. В момент времени t_1(t_1>t_0) у узла Г появляется кадр для передачи. Хотя узел Б в этот момент t_1 уже отправляет данные, передаваемый им сигнал еще не достиг узла Г, таким образом, узел Г полагает, что канал в момент t_1 свободен. Поэтому в соответствии с протоколом CSMA узел Г начинает передачу своего кадра. Немного позднее сигнал, отправляемый узлом Б, смешивается с сигналом узла Г — возникает коллизия. Из рисунка видно, что производительность широковещательного канала в значи-

тельной степени зависит от **времени прохождения сигнала** по каналу от одного узла до другого. Чем больше это время, тем выше вероятность коллизий, вызванных тем, что сигнал уже начавшего передачу узла не успел достичь другого узла, готового к передаче.

Множественный доступ с контролем несущей и обнаружением коллизий (CSMA/CD)

Изображенные на рис. 5.12 узлы не обнаруживают коллизию. Оба узла (Б и Г) продолжают передавать свои кадры целиком и после коллизии. При использовании протокола с обнаружением коллизий узел прекращает передачу, как только обнаруживает коллизию. На рис. 5.13 показан тот же сценарий, что и на рис. 5.12, но в этом случае узлы прекращают передачу, обнаружив коллизию. Разумеется, добавление функции обнаружения коллизий к протоколу множественного доступа положительно повлияет на производительность протокола, так как поврежденный (из-за наложения на кадр другого узла), а значит — бесполезный кадр не будет передаваться по нему целиком.

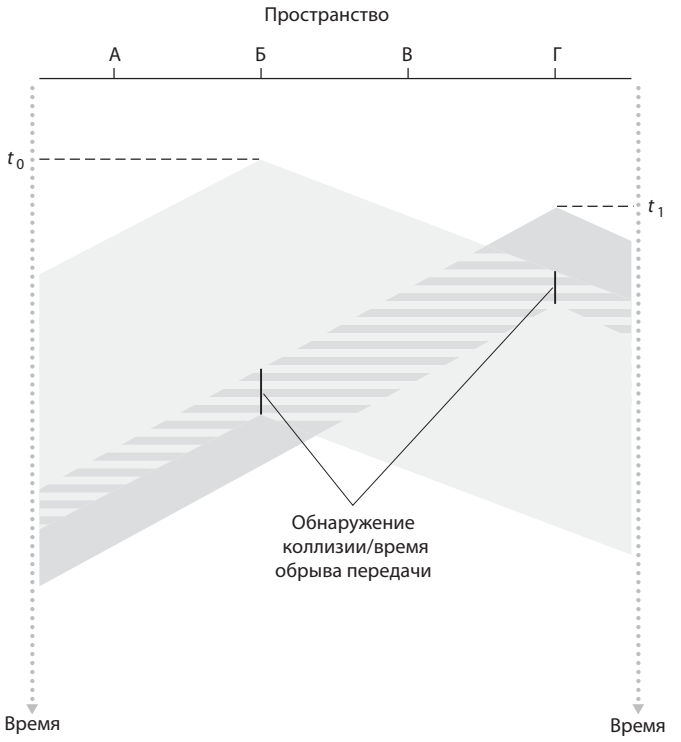


Рис. 5.13. Протокол CSMA с обнаружением коллизий

Прежде чем приступать к анализу протокола CSMA/CD, давайте рассмотрим, как он работает с точки зрения адаптера (на узле), подключенного к широковещательному каналу.

1. Адаптер получает дейтаграмму с сетевого уровня, готовит кадр канального уровня и заполняет буфер кадра в адаптере.

2. Если адаптер обнаруживает, что канал не занят (то есть на адаптер из этого канала не поступает никаких сигналов), он начинает передачу кадра. Иначе, если адаптер узнает, что канал занят, то он дожидается исчезновения сигналов в этом канале, после чего начинает передачу.

3. При передаче адаптер следит за тем, не пойдет ли какая-нибудь сигнальная энергия от других адаптеров, использующих этот широковещательный канал.

4. Если адаптер передает полный кадр, не зафиксировав встречных сигналов от других адаптеров, то обработка (передача) этого кадра завершена. В противном случае передача кадра обрывается (то есть не оканчивается).

5. После такого обрыва передачи адаптер выжидает в течение случайного интервала времени, а затем возвращается к шагу 2.

Длительность ожидания должна быть именно случайной (а не фиксированной) по понятным причинам: если два узла сначала одновременно пытались передать по кадру, а когда это не удалось — перешли в режим ожидания на фиксированное количество времени — то и выйдут из ожидания они одновременно, в результате чего такие коллизии будут бесконечными. Но каков разумный промежуток, от которого стоит отталкиваться при выборе **случайного отложенного времени**? Если интервал большой, а количество узлов, подверженных коллизиям, не столь велико, то узлам, вероятно, потребуется ждать слишком долго до того, как они снова смогут проверить, свободен ли канал (и на протяжении всего этого времени канал будет простаивать). С другой стороны, если интервал невелик, а узлов, между которыми возникают коллизии, очень много, то вполне вероятно, что многие значения, выбранные случайным образом, будут практически идентичны, и между передающими узлами вновь возникнет коллизия. Нам нужно, чтобы интервал был кратким при малом количестве узлов, но большим, если в сети окажется много узлов, между которыми могут возникать коллизии.

Экспоненциальный двоичный алгоритм выдержки, используемый в Ethernet, а также в протоколах множественного доступа кабельных сетей DOCSIS[140], красиво решает эту проблему. Так, при передаче кадра, с которым уже произошло n коллизий, узел случайным образом выбирает значение K из интервала $\{0,1,2,...2^n-1\}$. Соответственно, чем больше коллизий испытал кадр, тем больше интервал, из которого выбирается K. В случае с Ethernet точная длительность ожидания для узла равна $K\times512$ бит раз (то есть K — кратное истечение того промежутка времени, который требуется для отправки 512 байт по сети Ethernet). Максимальное значение, которое может принимать K, жестко ограничено 10.

Рассмотрим пример. Предположим, что узел впервые пытается передать кадр и при попытке передачи возникает коллизия. Тогда узел выбирает $K = 0$ с вероятностью 0,5 или $K = 1$, также с вероятностью 0,5. При выборе $K = 0$ узел сразу же начинает опрашивать канал. При выборе $K = 1$ узел выжидает, пока пройдет 512 бит (в Ethernet 100 Мбит/с это происходит примерно за 5,12 мс), после чего начинает новый цикл опроса и передает информацию, если обнаруживает, что канал простаивает. После второй коллизии значение K с равной вероятностью выбирается из интервала {0,1,2,3}. После трех коллизий этот интервал увеличивается до {0,1,2,3,4,5,6,7}. После 10 и более коллизий K с равной вероятностью выбирается из диапазона {0,1,2,...,1023}. Соответственно, размер множества, из которого выбирается значение K, возрастает по экспоненте при увеличении количества коллизий. Именно поэтому данный алгоритм называется «экспоненциальным двоичным алгоритмом выдержки».

Также отметим, что всякий раз, когда узел подготавливает кадр для передачи, он запускает алгоритм CSMA/CD. При этом не учитывается, сколько коллизий могло произойти в недавнем прошлом. Итак, сохраняется вероятность, что новый кадр сможет проскользнуть сразу же (попасть в успешную передачу), в то время как другие кадры по-прежнему будут оставаться в состоянии экспоненциальной двоичной выдержки.

Эффективность алгоритма CSMA/CD

Когда кадр для отправки имеется только у одного узла, узел может задействовать для его передачи всю пропускную способность канала. Типичные значения пропускной способности в сетях Ethernet равны 10 Мбит/с, 100 Мбит/с или 1 Гбит/с. Однако если таких узлов много, то фактическая скорость передачи данных в канале может быть гораздо

ниже. **Эффективность алгоритма CSMA/CD** на достаточно длительном промежутке времени определяется как количество времени, за которое кадры передаются по каналу без коллизий при наличии большого количества активных узлов, где каждый имеет множество кадров для отправки. Чтобы записать аналитическое выражение для приблизительной оценки эффективности сетей Ethernet, обозначим с помощью $d_{\text{распростр}}$ максимальное время, требуемое сигналу для распространения между двумя адаптерами. Через $d_{\text{передача}}$ обозначим время, требуемое для передачи максимально крупного кадра (для Ethernet со скоростью 10 Мбит/с это значение составляет около 1,2 мкс). Развернутое объяснение вывода эффективности алгоритма CSMA/CD выходит за рамки этой книги (см. работы Лэма[309] и Бертсикеса[50]). Здесь мы просто укажем следующее выражение:

$$\text{Эффективность} = \frac{1}{5d_{\text{распростр}}/d_{\text{передача}}}$$

Из формулы видно, что по мере приближения $d_{\text{распростр}}$ к 0 эффективность приближается к 1. Это понятно и на интуитивном уровне: если на распространение тратится нулевое время, то конфликтующие узлы будут мгновенно отменять передачу, не тратя пространство канала впустую. Кроме того, когда значение $d_{\text{передача}}$ чрезмерно увеличивается, эффективность стремится к 1. Также понятно, что, когда кадр занимает канал, он задерживается там на достаточно долгое время, но на всем его протяжении канал будет осуществлять полезную работу.

5.3.3. Протоколы поочередного доступа

Как уже упоминалось, двумя желательными свойствами протокола множественного доступа являются, во-первых, возможность единственного активного узла передавать свои данные с максимальной пропускной способностью канала R бит/с, во-вторых, возможность для каждого из M активных узлов передавать свои данные со скоростью R/M бит/с. Протоколы ALOHA и CSMA удовлетворяют первому требованию, но не удовлетворяют второму. Это подвигло исследователей на создание нового класса протоколов — **протоколов поочередного доступа**. Как и в случае с протоколами произвольного доступа, их существуют десятки, и у каждого есть множество вариантов. Здесь мы рассмотрим два наиболее важных протокола поочередного доступа. Первый из них — **протокол опроса**. При использовании протокола опроса один из узлов должен быть назначен главным (управляющим). Главный узел поочередно **опрашивает** все узлы. Например, сначала главный узел посылает

сообщение узлу 1, сообщая ему, что он может передать некоторое максимальное количество кадров. После того как узел 1 передает несколько кадров, главный узел разрешает передать некоторое количество кадров узлу 2. (Главный узел может определить момент завершения передачи очередным узлом по отсутствию сигнала в канале.) Данная процедура продолжается бесконечно, при этом главный узел в цикле опрашивает все узлы.

Протокол опроса устраняет коллизии и пустые кванты, от которых страдают протоколы произвольного доступа. Таким образом, эффективность протокола опроса значительно выше. Однако у протокола опроса есть несколько недостатков. Первый заключается в том, что определенное время тратится протоколом опроса на саму процедуру опроса, то есть на выдачу узлу разрешения на передачу. Например, если только один узел является активным, тогда он не сможет передавать со средней скоростью, равной полной пропускной способности канала, так как после отправки активным узлом разрешенного количества кадров главный узел будет опрашивать остальные узлы в каждом цикле. Второй недостаток является еще более серьезным. Он заключается в том, что при выходе из строя главного узла вся деятельность канала прекращается. В качестве примеров протоколов опроса можно привести 802.15 и Bluetooth, о которых мы поговорим в разделе 6.3.

Второй протокол поочередного доступа — **протокол с передачей маркера**. В этом протоколе главного узла не существует. Все узлы, присоединенные к широковещательному каналу, обмениваются небольшим специальным кадром, называемым **маркером** (токеном). Порядок обмена маркера фиксирован. Например, узел 1 всегда посылает маркер узлу 2, а узел 2 всегда посылает маркер узлу 3 и т. д.; а узел *N* всегда посылает маркер узлу 1. Получив маркер, узел удерживает его, только если у него есть данные для передачи; в противном случае он немедленно передает маркер следующему узлу. Если к моменту получения маркера у узла есть кадры для передачи, он отправляет некое максимальное количество кадров, после чего пересылает маркер следующему узлу. Передача маркера осуществляется децентрализованно и обладает высокой эффективностью. Но проблемы могут возникнуть и в данной схеме. Например, выход из строя одного узла может вывести из строя весь канал, а если какой-либо узел забудет передать маркер, потребуется специальная процедура вывода канала из тупиковой ситуации. За многие годы было разработано множество протоколов с передачей маркера, в частности, FDDI (протокол волоконно-оптического интерфейса передачи

данных)[263] и протокол IEEE 802.5 для передачи маркера по сети с кольцевой конфигурацией[237], в каждом из которых приходилось решать эти и другие неприятные вопросы.

5.3.4. DOCSIS: протокол канального уровня для кабельного доступа в Интернет

В предыдущих трех разделах мы познакомились с тремя большими классами протоколов множественного доступа: протоколы с разделением канала, протоколы произвольного доступа и протоколы поочередного доступа. Очень удобно показать приемы работы с ними на примере кабельной сети доступа, поскольку здесь мы найдем различные аспекты использования *каждого* из этих трех классов протоколов множественного доступа!

Как вы помните из раздела 1.2.1, кабельная сеть доступа обычно соединяет несколько тысяч клиентских кабельных модемов с терминальной станцией кабельных модемов (CMTS), расположенной в головном узле компьютерной сети. Стандарт передачи данных по коаксиальному кабелю (DOCSIS)[140] описывает архитектуру кабельной сети доступа и применяемые в ней протоколы. Стандарт DOCSIS задействует мультиплексирование с частотным разделением (FDM) для разделения нисходящих (от CMTS к модему) и восходящих (от модема к CMTS) сетевых сегментов на множество частотных каналов. Ширина каждого нисходящего канала составляет 6 МГц, максимальная пропускная способность — около 40 Мбит/с на канал (хотя на практике такая частота передачи данных по кабельному модему достигается редко). Максимальная ширина восходящего канала составляет 6,4 МГц, а его максимальная пропускная способность — примерно 30 Мбит/с. Все восходящие и нисходящие каналы являются широковещательными. Кадры передаются по нисходящему каналу головной станцией и принимаются всеми кабельными модемами, подключенными к этому каналу. Поскольку головная станция всего одна, в этом направлении не возникает проблем с множественным доступом. Управление восходящими потоками — более интересная и технически сложная задача, поскольку один и тот же канал одновременно используется множеством кабельных модемов, отправляющих свои данные на головную станцию. Естественно, в таких случаях могут возникать коллизии.

Как показано на рис. 5.14, все восходящие каналы работают с разделением времени на интервалы (подобно механизму TDM). Каждый ин-

тервал содержит последовательность мини-интервалов, в ходе которых кабельные модемы могут передавать информацию CMTS. CMTS явно выделяет конкретным кабельным модемам право на передачу информации в ходе тех или иных мини-интервалов. Для этого CMTS отправляет по нисходящему каналу специальное управляющее сообщение, называемое MAP, в котором указывает, какой кабельный модем (располагающий данными для отправки) может их передать в течение мини-интервала, приходящегося на период, обозначенный в MAP-сообщении. Поскольку мини-интервалы явно выделяются кабельным модемам, головная станция может гарантировать, что в ходе мини-интервала никаких коллизий не возникнет.

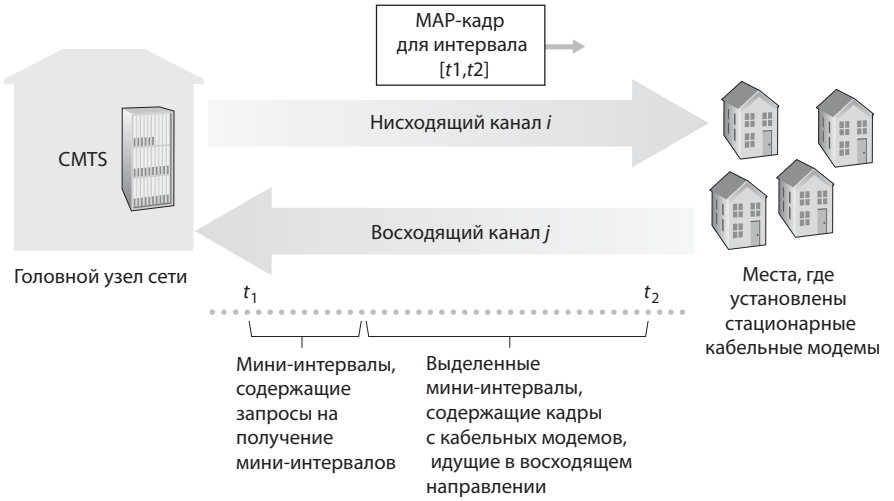


Рис. 5.14. Восходящие и нисходящие каналы между головной узловой станцией и кабельными модемами

Но как же CMTS узнает, какие из кабельных модемов располагают данными для отправки? Для этого модемы отправляют CMTS кадры с запросом мини-интервалов. Для передачи таких запросов выделяются специальные мини-интервалы, предназначенные именно для этой цели, как показано на рис. 5.14. Такие кадры с запросами на получение мини-интервалов отправляются с расчетом на произвольный доступ, поэтому между ними могут возникать коллизии. Кабельный модем не может путем опроса узнать, занят ли восходящий канал, равно как не может обнаружить коллизии. Вместо этого модем логически заключает, что его кадр с запросом мини-интервала претерпел коллизию, если не получает отклика на запрошенное выделение ресурса в следующем нисходящем

управляющем сообщении. Когда таким образом логически установлен факт коллизии, модем применяет экспоненциальный двоичный алгоритм задержки для повторного запроса о мини-интервале на будущий квант времени. Если в восходящем канале не так много трафика, то кабельный модем может передавать кадры с данными в течение тех квантов, которые номинально выделяются для запросов о мини-интервалах (следовательно, модему не приходится ждать получения такого мини-интервала).

Итак, кабельная сеть доступа — превосходный образец практического применения протоколов множественного доступа. Здесь мы наблюдаем и технологии FDM и TDM, и произвольный доступ, и централизованно выделяемые кванты времени — все в одной сети!

5.4. Локальная сеть с коммутируемым доступом

Выше мы рассмотрели широковещательные сети и протоколы множественного доступа. Теперь обратимся к коммутируемым локальным сетям. На рис. 5.15 представлена такая сеть, в которой при помощи четырех коммутаторов соединяются три факультета, два сервера и маршрутизатор.

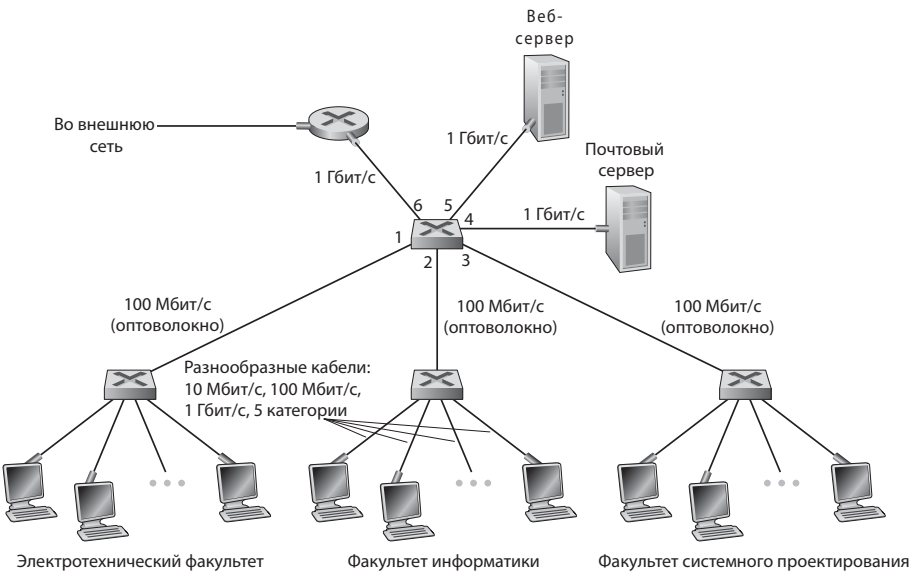


Рис. 5.15. Университетская сеть, соединенная при помощи четырех коммутаторов

Поскольку эти коммутаторы работают на канальном уровне, они переключают кадры канального уровня (а не дейтаграммы сетевого уровня) и не различают сетевых адресов, а также не используют алгоритмов маршрутизации — в частности, RIP или OSPF, которые обеспечивают прокладку путей по сети в коммутаторах уровня 2. Как мы вскоре увидим, для перенаправления пакетов в сети коммутаторов используются не сетевые адреса, а адреса канального уровня. Мы начнем изучение коммутации в локальных сетях с адресации канального уровня (раздел 5.4.1). Затем мы поговорим о широко известном протоколе Ethernet (раздел 5.5.2). После этого мы обратимся к тому, как функционируют коммутаторы канального уровня (раздел 5.4.3), а потом (раздел 5.4.4) изучим, как такие коммутаторы применяются для создания крупномасштабных локальных сетей.

5.4.1. Адресация канального уровня и протокол ARP

У хостов и маршрутизаторов есть адреса канального уровня. Возможно, сейчас вам это кажется удивительным — ведь в главе 4 было сказано, что у хостов и маршрутизаторов есть и адреса сетевого уровня. Может возникнуть вопрос: зачем же нужны адреса как на канальном, так и на сетевом уровне? В этих разделах мы не только опишем синтаксис и функционирование адресации канального уровня, но и постараемся объяснить, почему два уровня адресации не только полезны, но и, в сущности, необходимы. Кроме того, мы рассмотрим протокол разрешения адресов (ARP), который обеспечивает трансляцию (преобразование) IP-адресов в адреса канального уровня.

MAC-адреса

В действительности адрес канального уровня есть не у узла, а у адаптера (то есть у сетевого интерфейса). Следовательно, хост или маршрутизатор с несколькими сетевыми интерфейсами будет иметь несколько адресов канального уровня. При этом важно отметить, что те интерфейсы коммутаторов канального уровня, которые соединяются с хостами и маршрутизаторами, не снабжаются такими адресами. Дело в том, что задача коммутатора канального уровня — передавать дейтаграммы между хостами и маршрутизаторами. Коммутатор прозрачно выполняет эту работу — то есть хосту или маршрутизатору не приходится явно сообщать адрес кадра тому коммутатору, через который этот кадр проходит. Это иллюстрирует рис. 5.16. Адрес в локальной сети, или **LAN-**

адрес, также называют **физическим адресом**, **Ethernet-адресом** или **MAC-адресом** (Media Access Control — управление доступом к среде передачи). В большинстве локальных сетей (включая Ethernet-сети и беспроводные локальные сети стандарта 802.11) MAC-адрес представляет собой 6-байтовое число, что позволяет использовать 2^{48} возможных адресов. Как показано на рис. 5.16, эти 6-байтовые адреса, как правило, изображаются в шестнадцатеричном виде, при этом каждый байт адреса записывается как пара шестнадцатеричных цифр. Хотя адрес адаптера в локальной сети изначально задумывался как постоянный, в настоящее время существует возможность программно изменять MAC-адрес адаптера. В оставшейся части этого раздела мы будем считать, что MAC-адрес адаптера является постоянным.

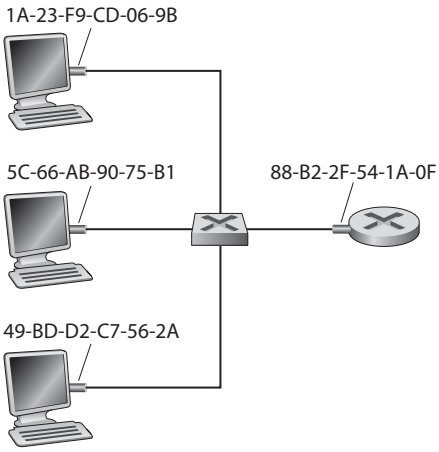


Рис. 5.16. Каждый из интерфейсов, подключенных к локальной сети, имеет уникальный MAC-адрес

Одно интересное свойство MAC-адресов заключается в том, что не может существовать двух адаптеров с одинаковыми адресами. Это может показаться удивительным, так как сетевые адаптеры производятся в разных странах, разными производителями. Как может компания, производящая адаптеры в Тайване, гарантировать, что адреса ее устройств будут отличаться от адресов адаптеров, производимых другой компанией в Бельгии? Дело в том, что физическим адресным пространством управляет институт IEEE (Institute of Electrical and Electronics Engineers — Институт инженеров по электротехнике и электронике). В частности, когда компания хочет выпускать адаптеры, она приобретает блок адресного пространства, состоящий из 2^{24} адресов. IEEE выде-

ляет блок из 2^{24} адресов, фиксируя старшие 24 бита физического адреса и позволяя компании создавать уникальные комбинации из младших 24 разрядов для каждого адаптера.

Таким образом, MAC-адреса адаптеров образуют плоскую (а не иерархическую) структуру и не изменяются при перемещении адаптеров. У ноутбука с Ethernet-картой всегда один и тот же MAC-адрес независимо от того, где находится этот компьютер. Вспомним, что IP-адреса, напротив, образуют иерархическую структуру (то есть делятся на сетевую и хостовую части), и при перемещении хоста IP-адрес узла должен быть изменен. Таким образом, MAC-адрес адаптера аналогичен номеру карточки социального страхования (эти номера также образуют плоскую адресную структуру и не изменяются при перемещении их владельца). IP-адрес можно сравнить с почтовым адресом человека — иерархическим и изменяющимся при смене места жительства владельца. Многие люди активно пользуются как номерами социального страхования, так и почтовыми адресами. Аналогично, на хосте и интерфейсе полезно иметь как адрес сетевого уровня, так и MAC-адрес.

Когда адаптер хочет переслать кадр другому адаптеру в той же локальной сети, передающий адаптер помещает в кадр адрес получателя в локальной сети. Как мы вскоре увидим, коммутатор может передавать входящий кадр на все свои интерфейсы. В главе 6 мы увидим, что в сетях 802.11 также выполняется широковещательная передача кадров. Это значит, что адаптер может получить кадр, который ему не предназначен. Следовательно, получив кадр, адаптер проверяет, совпадает ли целевой MAC-адрес в этом кадре с собственным MAC-адресом этого адаптера. Если зафиксировано совпадение, принимающий адаптер извлекает из кадра дейтаграмму и передает ее вверх по стеку протоколов. Если совпадения нет, то адаптер отбрасывает этот кадр, не передавая дейтаграмму сетевого уровня вверх по стеку протоколов. Таким образом, узел принимает и обрабатывает только те кадры, которые адресованы ему.

Однако иногда передающий узел *бывает* заинтересован в том, чтобы все адаптеры в локальной сети приняли и *обработали* кадр, который он посылает. В этом случае передающий адаптер помещает в поле адреса получателя специальный **широковещательный адрес** MAC. В локальных сетях, использующих 6-байтовые адреса (как, например, Ethernet или 802.11), широковещательный адрес представляет собой строку из 48 двоичных единиц (то есть FF-FF-FF-FF-FF-FF в шестнадцатеричной нотации).

ПРИНЦИПЫ В ДЕЙСТВИИ

Обеспечение независимости уровней

Есть несколько причин, по которым узлам помимо адресов сетевого уровня выделяются MAC-адреса. Во-первых, локальные сети предназначены для работы с произвольными протоколами сетевого уровня, а не только с протоколом IP и Интернетом. Если бы вместо «нейтральных» MAC-адресов адаптерам назначались IP-адреса, адаптерам было бы трудно поддерживать другие протоколы сетевого уровня (например, IPX или DECnet). Во-вторых, если бы адаптеры должны были использовать IP-адреса вместо MAC-адресов, тогда адрес сетевого уровня пришлось бы хранить в оперативной памяти адаптера и перенастраивать его при каждом перемещении устройства или при включении питания. Правда, можно было бы вообще не использовать адреса в адаптерах и передавать данные (как правило, IP-дейтаграммы) каждого полученного адаптером кадра вверх по стеку протоколов. После этого родительский узел мог бы проверять соответствие адреса сетевого уровня. Недостаток такого варианта состоит в том, что в этом случае родительский узел прерывался бы каждым кадром, посланным по локальной сети, включая кадры, предназначающиеся другим узлам той же широковещательной сети. Таким образом, для большей независимости уровней в стеке протоколов уровни должны обладать собственными схемами адресации. Мы уже ознакомились с тремя типами адресов: именами хостов на прикладном уровне, IP-адресами на сетевом уровне и MAC-адресами на канальном уровне.

Протокол разрешения адресов

При передаче дейтаграмм используются и адреса сетевого уровня (например, IP-адреса Интернета), и адреса канального уровня (то есть MAC-адреса), поэтому возникает потребность в преобразовании одних адресов в другие. В Интернете эту работу выполняет **протокол разрешения адресов** (Address Resolution Protocol, **ARP**)[423].

Чтобы понять, зачем нужен протокол ARP, рассмотрим сеть, изображенную на рис. 5.17. В этом простом примере у каждого узла есть IP-адрес, а у адаптера каждого узла есть MAC-адрес. IP-адреса, как обычно, представляются в виде четырех десятичных чисел, а MAC-адреса показаны в шестнадцатеричном виде. В этом разделе мы будем считать, что коммутатор передает все кадры широковещательно. То есть как только коммутатор получает кадр на любом интерфейсе, он перенаправляет

этот кадр на все другие свои интерфейсы. В следующем разделе будет более подробно описано, как именно работают коммутаторы.

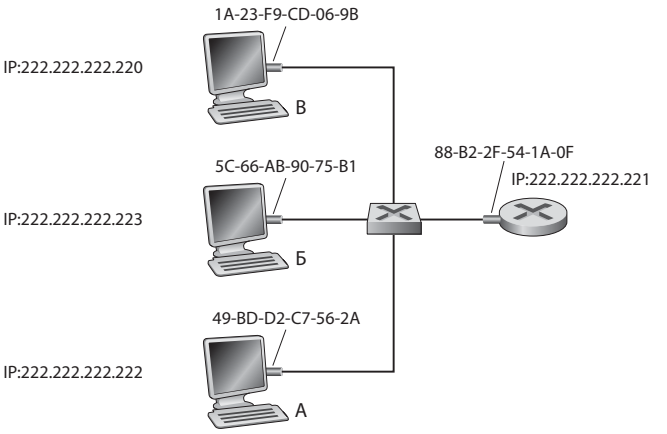


Рис. 5.17. Каждый интерфейс в локальной сети имеет IP-адрес и MAC-адрес

Теперь предположим, что хост с IP-адресом 222.222.222.220 хочет послать IP-дейтаграмму хосту 222.222.222.222. В данном примере и исходный, и целевой хост относятся к одной и той же подсети, их адресация организована так, как описано в разделе 4.2.2. Чтобы выполнить эту задачу, передающий хост должен отправить адаптеру не только IP-дейтаграмму, но также MAC-адрес узла 222.222.222.222. Получив IP-дейтаграмму и MAC-адрес узла, адаптер передающего хоста формирует кадр канального уровня, содержащий MAC-адрес принимающего узла, и передает кадр в локальную сеть.

В этом разделе нам требуется ответить на важный вопрос: каким образом передающий хост определяет MAC-адрес хоста с IP-адресом 222.222.222.222? Как вы уже догадались, он делает это с помощью модуля ARP. В данном случае ARP-модуль исходного хоста принимает в качестве ввода любой IP-адрес из своей локальной сети, а в ответ возвращает соответствующий MAC-адрес. Здесь хост-отправитель с IP-адресом 222.222.222.220 указывает своему ARP-модулю IP-адрес 222.222.222.222. На это ARP-модуль отвечает соответствующим LAN-адресом узла, то есть адресом 49-BD-D2-C7-56-2A.

Итак, ARP-модуль преобразует IP-адрес в MAC-адрес узла. Во многом это аналогично системе DNS (см. раздел 2.5), преобразующей имена хостов в IP-адреса. Однако важное различие между этими двумя схе-

мами преобразования адресов заключается в том, что DNS преобразует имена хостов в IP-адреса во всем Интернете, тогда как протокол ARP занимается только IP-адресами в пределах одной локальной сети. Если бы узел в Калифорнии попытался узнать LAN-адрес для IP-адреса узла в Миссисипи, протокол ARP вернул бы ошибку.

Теперь, выяснив, что делает протокол ARP, посмотрим, как он это делает. У ARP-модуля каждого узла есть оперативная память, в которой хранится **ARP-таблица**. В этой таблице прописаны IP-адреса хостов локальной сети и соответствующие им MAC-адреса. На рис. 5.18 показано, как могла бы выглядеть ARP-таблица для узла 222.222.222.220. Для каждой пары адресов в таблице также содержится поле **предписанного времени жизни** (Time To Live, **TTL**), в котором указывается, когда данная запись будет удалена из таблицы. Обратите внимание, что в таблице не обязательно содержатся записи для всех узлов локальной сети. Например, записи для одних узлов могут быть удалены, так как время их жизни истекло, тогда как записи для других узлов вообще могут никогда не попасть в эту таблицу. Типичное значение времени жизни — 20 мин с момента помещения записи в ARP-таблицу.

IP-адрес	MAC-адрес	TTL
222.222.222.221	88-B2-2F-54-1A-0F	13:45:00
222.222.222.223	5C-66-AB-90-75-B1	13:52:00

Рис. 5.18. Так могла бы выглядеть ARP-таблица для хоста 222.222.222.220

Теперь предположим, что хост 222.222.222.220 хочет отправить дейтаграмму с указанием IP-адреса другому хосту или маршрутизатору той же локальной сети. Для этого передающему узлу нужно по IP-адресу получающего узла узнать его MAC-адрес. Эта задача несложная, если в ARP-таблице передающего узла содержится запись для узла-получателя. Но что делать, если такой записи в ARP-таблице нет? Например, пусть узел 222.222.222.220 желает переслать дейтаграмму узлу 222.222.222.222. В этом случае передающий узел определяет нужный ему адрес при помощи протокола ARP. Сначала передающий узел формирует специальный **ARP-пакет**. В ARP-пакете содержится несколько полей, среди которых есть IP-адреса и MAC-адреса передающего и принимающего узлов. Для обоих ARP-пакетов (запроса и ответа) используется один и тот же формат. Цель ARP-пакета с запросом состоит в том, чтобы опросить все остальные узлы локальной

сети и определить LAN-адрес, соответствующий интересующему нас IP-адресу.

Итак, в нашем примере узел 222.222.222.220 передает ARP-пакет с запросом своему адаптеру вместе с указанием переслать этот пакет по широковещательному MAC-адресу FF-FF-FF-FF-FF-FF. Адаптер инкапсулирует ARP-пакет в кадр канального уровня, указывает широковещательный адрес в поле адреса получателя и передает кадр в подсеть. Если вспомнить нашу аналогию с номерами карточек социального страхования и почтовыми адресами, то можно заметить, что ARP-запрос аналогичен человеку, выкрикивающему в переполненном зале с офисными кабинками некоторой компании (например, «СоцСтрах»): «Какой номер карточки социального страхования у сотрудника из кабины 112 офиса 13 компании «СоцСтрах», Подольск, Московская область?» Кадр с ARP-запросом принимается всеми остальными адаптерами подсети, и (поскольку в запросе использовался широковещательный адрес) каждый адаптер передает содержащийся в кадре ARP-пакет своему узлу. Каждый узел проверяет, совпадает ли его IP-адрес с указанным IP-адресом получателя в ARP-пакете. Узел, обнаруживший совпадение, посылает запрашивающему узлу ответный ARP-пакет с указанным в нем соответствующим MAC-адресом. После этого запрашивающий узел 222.222.222.220 может обновить свою ARP-таблицу и отправить IP-дейтаграмму, заключенную в кадр канального уровня, где MAC-адрес назначения соответствует адресу хоста или маршрутизатора, ответившего на предыдущий ARP-запрос.

Следует сделать два интересных замечания о протоколе ARP. Во-первых, ARP-запрос посылается в широковещательном кадре, а ответ передается в стандартном кадре. Прежде чем продолжить чтение, подумайте, почему. Во-вторых, в протоколе ARP реализован принцип самонастройки (plug-and-play), так как ARP-таблица узла формируется автоматически — ее не нужно настраивать администратору вычислительной сети. И если узел отсоединяется от подсети, соответствующая ему запись по истечении времени жизни удаляется из таблицы.

Зачастую студенты интересуются, к протоколам какого уровня относится ARP — сетевого или канального? Как мы убедились, пакет ARP инкапсулируется в кадр канального уровня и, следовательно, архитектурно располагается выше канального уровня. Однако в ARP-пакете есть поля, содержащие адреса канального уровня, что позволяет считать его протоколом канального уровня. Правда, у него есть и адреса сетево-

го уровня, что позволяет с тем же успехом относить его к протоколам сетевого уровня. Таким образом, правильнее всего трактовать его как граничный протокол, не вписывающийся в тот простой стек протоколов, который мы изучили в главе 1. Да, на практике встречаются и такие сложности!

Передача дейтаграммы узлу за пределы подсети

Теперь должно быть ясно, как работает протокол ARP, когда узел хочет послать дейтаграмму другому узлу *той же самой подсети*. Но теперь мы рассмотрим более сложную ситуацию, в которой хост подсети хочет послать дейтаграмму сетевого уровня хосту, находящемуся *за пределами подсети* (то есть в другую подсеть). Обсудим этот вопрос на примере сети, состоящей из двух подсетей, соединенных маршрутизатором (рис. 5.19).

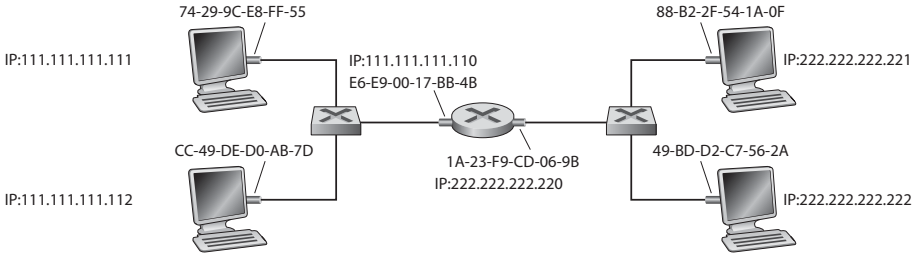


Рис. 5.19. Две подсети, соединенные маршрутизатором

На рис. 5.19 следует отметить ряд интересных вещей. У каждого хоста есть ровно один IP-адрес и один адаптер. Но, как было показано в разделе 4.4 главы 4, у маршрутизатора есть по одному IP-адресу для *каждого* интерфейса. У каждого интерфейса маршрутизатора есть собственный ARP-модуль и собственный адаптер. У изображенного на рис. 5.19 маршрутизатора два интерфейса, поэтому у него два IP-адреса, два ARP-модуля и два адаптера. Разумеется, у каждого адаптера есть свой MAC-адрес.

Обратите также внимание, что адреса всех интерфейсов подсети 1, имеют вид 111.111.111/24, а адреса всех интерфейсов, соединенных с подсетью 2, имеют вид 222.222.222/24. Следовательно, все интерфейсы, подключенные к подсети 1, имеют адреса вида 111.111.111.xxx, а все интерфейсы, подключенные к подсети 2, имеют адреса вида 222.222.222.xxx.

Теперь предположим, что хост 111.111.111.111 из подсети 1 хочет переслать IP-дейтаграмму хосту 222.222.222.222 из подсети 2. Передающий хост, как обычно, отправляет дейтаграмму своему адаптеру. Но при этом передающий хост должен указать адаптеру MAC-адрес. Какой MAC-адрес следует использовать? Соблазнительно предположить, что это должен быть локальный адрес адаптера хоста 222.222.222.222, то есть 49-BD-D2-C7-56-2A. Однако это неверно. Если передающий адаптер будет использовать этот MAC-адрес, тогда ни один из адаптеров подсети 1 не станет передавать такую IP-дейтаграмму своему сетевому уровню, так как адрес получателя в кадре не совпадет с MAC-адресом ни одного адаптера подсети 1. Переданная дейтаграмма просто умрет и попадет в свой дейтаграммный рай.

Если внимательно посмотреть на рис. 5.19, то можно заметить, что для передачи дейтаграммы в подсеть 2 ее нужно направлять интерфейсу маршрутизатора по адресу 111.111.111.110, а этот маршрутизатор располагается на расстоянии одного перехода на пути к конечному узлу. Таким образом, в поле MAC-адреса получателя кадра следует указывать адрес адаптера интерфейса маршрутизатора 111.111.111.110, то есть E6-E9-00-17-BB-4B. Как передающему хосту узнать MAC-адрес узла 111.111.111.110? С помощью протокола ARP, разумеется! Как только передающий адаптер получает этот MAC-адрес, он создает кадр и посылает его в подсеть 1. Адаптер маршрутизатора в подсети 1 видит, что данный кадр канального уровня адресован ему, и поэтому передает кадр сетевому уровню маршрутизатора. Ура! IP-дейтаграмма была успешно перемещена от хоста-отправителя на маршрутизатор! Но мы еще не закончили. Нам нужно еще переслать дейтаграмму от маршрутизатора получателю. Теперь маршрутизатор должен определить, по какому интерфейсу следует переслать дейтаграмму. Как было показано в главе 4, выбор делается при помощи таблицы перенаправления, хранящейся на маршрутизаторе. В этой таблице маршрутизатор находит запись, по которой определяет, что дейтаграмму следует отправить через интерфейс 222.222.222.220. Этот интерфейс передает дейтаграмму своему адаптеру, который помещает ее в новый кадр и посылает этот кадр в подсеть 2. Теперь уже MAC-адрес кадра указывает на его конечного получателя. И как же маршрутизатор узнает его MAC-адрес? При помощи протокола ARP, конечно!

Протокол ARP для Ethernet-сети определен в документе RFC 826[423]. Прекрасное введение в тему ARP имеется в документе RFC 1180. Мы вернемся к протоколу ARP в упражнениях, представленных в конце главы.

5.4.2. Стандарт Ethernet

Технология Ethernet очень популярна на рынке локальных сетей. В 80-е годы и в начале 90-х технологии Ethernet приходилось конкурировать с множеством альтернативных технологий локальных сетей, включая маркерный доступ по сети с кольцевой конфигурацией (token ring), FDDI и ATM. Некоторым из этих технологий удалось на несколько лет захватить часть рынка. Но технология Ethernet, разработанная в середине 70-х, продолжает расти и развиваться до сих пор, а в последние годы она прочно заняла доминирующее положение на рынке. Сегодня Ethernet представляет собой превалирующую технологию локальных сетей, и, похоже, такая ситуация сохранится в обозримом будущем. Значение Ethernet в локальных сетях так же велико, как и значение Интернета в глобальных сетях.

Успеху технологии Ethernet способствовало множество причин. Во-первых, Ethernet была первой локальной сетью, получившей широкое распространение. Благодаря этому администраторы вычислительной сети очень близко познакомились с технологией Ethernet и уже не хотели переходить на другие технологии локальных сетей, когда те появились на рынке. Во-вторых, такие технологии, как сети маркерного доступа, FDDI и ATM, были более сложными и дорогими, чем Ethernet, что еще больше препятствовало переходу на них. В-третьих, наиболее сильным стимулом перехода на другие технологии локальных сетей (например, FDDI или ATM), как правило, являлась более высокая скорость передачи данных. Однако Ethernet всегда отвечала ударом на удар, и каждая новая версия Ethernet не уступала конкурентам, а то и превосходила их по данному параметру. В начале 90-х была представлена коммутируемая Ethernet-сеть, что позволило снова повысить эффективную скорость передачи данных. Наконец, благодаря популярности Ethernet аппаратура Ethernet (в частности, адаптеры и коммутаторы) стала массово производиться и дешеветь.

Первая локальная Ethernet-сеть была придумана Бобом Меткалфом и Дэвидом Боггсом в середине 70-х. В этой сети для соединения узлов использовался коаксиальный кабель. Сами топологии сетей Ethernet в основном не изменились ни в 80-е годы, ни до середины 90-х. Ethernet с шинной топологией представляет собой широковещательную локальную сеть, все проходящие по которой кадры обрабатываются *всеми* адаптерами, подключенными к шине. Здесь можно вспомнить протокол множественного доступа Ethernet CSMA/CD и алгоритм двоичной экспоненциальной выдержки, рассмотренный в разделе 5.3.2.

К концу 1990-х большинство компаний и университетов заменили свои локальные сети Ethernet-конфигурациями, применяя звездообразную топологию сети на основе концентратора. В такой системе хосты (и маршрутизаторы) напрямую подключаются к концентратору при помощи медного кабеля «витая пара». **Концентратор** (хаб) — это устройство физического уровня, которое оперирует отдельными битами, а не кадрами. Когда с интерфейса поступает бит, представляющий собой нуль или единицу, концентратор просто воспроизводит его, а потом передает с более высокой энергией на все другие интерфейсы. Соответственно, сеть Ethernet со звездообразной топологией, использующая концентратор, также является широковещательной локальной сетью. Когда концентратор получает бит с любого из своих интерфейсов, он посылает копии этого бита на все другие интерфейсы. В частности, если концентратор одновременно получает кадры с двух разных интерфейсов, происходит коллизия, и узлы, породившие эти кадры, должны повторить передачу.

В начале 2000-х сеть Ethernet претерпела еще одно коренное эволюционное изменение. В конфигурациях сетей Ethernet по-прежнему использовалась звездообразная топология, но концентратор в ее центре был заменен **коммутатором** (свич). Ниже в этой главе мы подробнее поговорим о коммутируемых сетях Ethernet. Пока достаточно оговориться, что коммутатор не только значительно снижает количество коллизий, но и отлично справляется с промежуточным хранением пакетов. Однако в отличие от маршрутизаторов, работающих на всех уровнях вплоть до третьего, коммутатор работает только до второго уровня.

Рис. 5.20. Структура Ethernet-кадра

Структура Ethernet-кадра

Разобравшись в структуре кадра, которая схематически показана на рис. 5.20, мы довольно много узнаем о технологии Ethernet. Чтобы сделать наше обсуждение более предметным, рассмотрим передачу IP-дейтаграммы от одного хоста другому в ситуации, когда оба хоста находятся в одной и той же локальной Ethernet-сети (например, как на рис. 5.17). (Отметим, однако, что, хотя в нашем примере полезная нагрузка представляет собой IP-дейтаграмму, кадр Ethernet может переносить и дру-

гие пакеты сетевого уровня.) Пусть физический адрес передающего адаптера A равен AA-AA-AA-AA-AA-AA, а физический адрес принимающего адаптера, адаптера B — BB-BB-BB-BB-BB-BB. Передающий адаптер инкапсулирует IP-дейтаграмму в Ethernet-кадр и отправляет кадр физическому уровню. Принимающий адаптер получает кадр от физического уровня, извлекает IP-дейтаграмму и передает ее сетевому уровню. Рассмотрим в этом контексте шесть полей Ethernet-кадра, показанных на рис. 5.20.

- *Поле данных (от 46 до 1500 байт)*. Это поле содержит IP-дейтаграмму. **Максимальный размер передаваемого блока** (Maximal Transfer Unit, **MTU**) в Ethernet-сети составляет 1500 байт. Это означает, что если размер IP-дейтаграммы превышает 1500 байт, то хост должен разбить ее на отдельные фрагменты (см. подраздел 4.4.1). Минимальный размер поля данных равен 46 байт. Другими словами, если размер IP-дейтаграммы меньше 46 байт, то данные, помещаемые в это поле, дополняются байтами-заполнителями. Сетевой уровень получает дейтаграмму от канального уровня с этими дополнительными байтами и отсекает все лишнее сам, ориентируясь на поле длины в заголовке IP-дейтаграммы.
- *Адрес получателя (6 байт)*. Это поле содержит MAC-адрес принимающего адаптера, а именно BB-BB-BB-BB-BB-BB. Получив Ethernet-кадр с адресом получателя BB-BB-BB-BB-BB-BB или широковещательным адресом MAC, адаптер B передает содержимое поля данных сетевому уровню. В противном случае он отбрасывает кадр.
- *Адрес отправителя (6 байт)*. Это поле содержит MAC-адрес адаптера, передающего кадр в локальную сеть, а именно AA-AA-AA-AA-AA-AA.
- *Поле типа (2 байта)*. Поле типа позволяет локальной Ethernet-сети мультиплексировать протоколы сетевого уровня. Чтобы понять, что это означает, вспомним, что хосты могут помимо протокола IP использовать и другие протоколы. В самом деле, любой хост может поддерживать несколько протоколов сетевого уровня — разные протоколы для разных приложений. По этой причине, получив Ethernet-кадр, адаптер B должен знать, какому протоколу сетевого уровня он должен передать (то есть демультиплексировать) содержимое поля данных. Каждому сетевому протоколу (например, IP, Novell IPX или AppleTalk) присвоен зафиксированный в стандарте номер типа. Обратите внимание, что поле типа аналогично полю протокола в дейта-

грамме сетевого уровня и полю номера порта сегмента транспортного уровня. Все эти поля служат для связи протокола одного уровня с протоколом уровнем выше.

- *CRC (4 байта)*. Как было показано в подразделе 5.2.3, назначение поля CRC заключается в том, чтобы получающий адаптер (адаптер Б) мог определить, не исказился ли кадр при передаче, то есть обнаружить ошибки в кадре.
- *Преамбула (8 байт)*. Ethernet-кадр начинается с 8-байтового поля преамбулы. В каждый из первых 7 байт преамбулы записывается значение 10101010, а в последний байт — значение 10101011. Первые 7 байт должны «разбудить» принимающие адаптеры и помочь им синхронизировать свои таймеры с часами отправителя. Как уже отмечалось, адаптер А должен передать кадр со скоростью 10 Мбит/с, 100 Мбит/с или 1 Гбит/с в зависимости от типа локальной Ethernet-сети. Однако поскольку в реальном мире ничто не совершенно, скорость передачи всегда будет несколько *отличаться* от номинала. Величина этого отклонения скорости другим адаптерам локальной сети *заранее* не известна. Таким образом, первые 62 бита преамбулы, представляющие собой чередующиеся нули и единицы, позволяют приемнику с достаточной точностью настроиться на скорость передатчика, а последние два разряда (две единицы подряд) сообщают адаптеру Б, что преамбула закончилась и следом идет уже первый информационный байт поля кадра. Адаптер Б понимает, что следующие 6 байт содержат адрес получателя. Конец кадра адаптер может распознать просто по отсутствию сигнала в линии.

ИСТОРИЯ

Боб Меткалф и Ethernet

Готовясь к получению степени доктора философии в Гарварде в начале 70-х годов, Боб Меткалф работал над проектом ARPAnet в Массачусетском технологическом институте (MIT). Во время обучения он познакомился с работами Абрамсона над сетью ALOHA и протоколами произвольного доступа. Получив степень незадолго до начала работы в исследовательском центре Xerox PARC корпорации Xerox в Пало-Альто, он посетил Абрамсона и его коллег в Гавайском университете, где в течение трех месяцев изучал сеть ALOHAnet. В центре Xerox PARC Меткалф занялся компьютерами Alto, во многом послужившими прототипами первых персональных компьютеров 80-х годов. Меткалф осознал необходимость разработки недо-

рогого способа соединения подобных компьютеров в сеть. Таким образом, вооружившись знаниями о сетях ARPAnet и ALOHAnet, а также о протоколах произвольного доступа, Меткалф вместе со своим коллегой Дэвидом Боггсом спроектировал Ethernet-сеть.

Оригинальная Ethernet-сеть, разработанная Меткалфом и Боггсом, работала на скорости 2,94 Мбит/с и соединяла до 256 хостов на расстоянии до одной мили. Меткалфу и Боггсу удалось уговорить большую часть исследователей Xerox PARC принять участие в тестировании этой сети. Затем Меткалф сумел объединить корпорации Xerox, Digital и Intel в целях разработки стандарта Ethernet со скоростью 10 Мбит/с, ратифицированного институтом IEEE. Однако компания Xerox не проявила большого интереса к коммерциализации технологии Ethernet, поэтому в 1979 году Меткалф основал собственную компанию, 3Com, которая занялась разработкой и коммерциализацией сетевых технологий, включая технологию Ethernet. В частности, в начале 80-х годов компания 3Com подготовила и выпустила на рынок Ethernet-карты для очень популярных в то время персональных компьютеров IBM PC. В 1990 году Меткалф покинул компанию 3Com. В это время в ней работали 2000 сотрудников, а доходы составляли 400 млн долларов.

Все технологии Ethernet предоставляют сетевому уровню службу без установления соединения. Таким образом, когда адаптер А хочет послать дейтаграмму адаптеру Б, он инкапсулирует ее в Ethernet-кадр и передает этот кадр в локальную сеть, даже не выполнив процедуры рукопожатия с адаптером Б. Подобная не требующая соединения служба на канальном уровне (уровень 2) аналогична дейтаграммной службе протокола IP (уровень 3) и службе протокола UDP (уровень 4).

Все разновидности технологий Ethernet предоставляют сетевому уровню ненадежную службу. В частности, когда адаптер Б получает кадр от адаптера А, он выполняет CRC-контроль кадра, но не передает в ответ подтверждения, что кадр успешно прошел проверку (также не передается и отрицательная квитанция, если контрольная сумма не совпала со значением поля CRC). Когда кадр не проходит такой проверки, принимающий адаптер просто отбрасывает его. То есть у адаптера А нет ни малейшего представления о том, прошел переданный им кадр CRC-контроль или нет. Благодаря отсутствию на канальном уровне службы надежной доставки технология Ethernet остается простой и дешевой. Но это также означает, что в потоке дейтаграмм, переданных сетевому уровню, могут быть пропуски.

Если в потоке данных возникают пропуски, потому что протокол Ethernet теряет кадры, видит ли эти пропуски приложение на хосте Б? Как отмечалось в главе 3, это полностью зависит от используемого приложением протокола (UDP или TCP). Если это протокол UDP, тогда приложению самому придется заниматься проблемой потерянных кадров. Если же это протокол TCP, то проблему пропущенных кадров возьмет на себя протокол TCP, с помощью механизма подтверждений и интервалов ожидания заставляя хост А передавать пропущенные данные повторно. Обратите внимание, когда протокол TCP передает данные повторно, они снова проходят через тот же Ethernet-адаптер. Так что в этом смысле Ethernet может повторять передачу данных. Однако следует помнить, что Ethernet-сеть не догадывается о том, что передает данные повторно, оставаясь в неведении, получает ли она новенькую дейтаграмму с новыми данными или дейтаграмму с данными, которые уже были переданы как минимум один раз.

Технологии Ethernet

Выше мы говорили об Ethernet как о едином стандарте протоколов. На самом же деле существует *множество* разновидностей Ethernet, которые обозначаются довольно сложными аббревиатурами, например, 10BASE-T, 10BASE-2, 100BASE-T, 1000BASE-LX и 10GBASE-T. Эти и многие другие технологии Ethernet были стандартизированы за прошедшие годы рабочей группой IEEE 802.3 CSMA/CD (Ethernet)[236]. Сокращения на первый взгляд могут показаться устрашающими, но на самом деле в них прослеживается определенный порядок. Первая часть аббревиатуры указывает на скорость, применяемую в стандарте: 10, 100, 1000 или 10G означает, соответственно, 10 Мбит/с, 100 Мбит/с, 1 Гбит/с и 10 Гбит/с. BASE означает немодулированный Ethernet — то есть физический носитель передает только трафик Ethernet; практически все стандарты 802.3 касаются именно немодулированного Ethernet. Последняя часть аббревиатуры описывает сам физический носитель. Для Ethernet существуют спецификации как канального, так *и* физического уровня, а передача данных осуществляется по разнообразным физическим носителям, включая коаксиальный кабель, медный кабель и оптоволокно. Вообще буква T означает медный кабель типа «витая пара».

Исторически технология Ethernet задумывалась для передачи информации по коаксиальному кабелю. Ранние стандарты 10BASE-2 и 10BASE-5 описывают Ethernet-передачу со скоростью 10 Мбит/с по

2 типам коаксиального кабеля, в обоих случаях длина кабеля не должна превышать 500 м. Передача на более длинные дистанции осуществляется при помощи повторителя. Это устройство физического уровня, которое принимает на входе сигнал и воспроизводит его на выходе. Коаксиальный кабель, показанный на рис. 5.20, красноречиво свидетельствует, что Ethernet является средством широковещательной передачи данных. Все кадры, передаваемые интерфейсом, принимаются на всех остальных интерфейсах, а протокол Ethernet CDMA/CD красиво решает проблему множественного доступа. Узлы просто подключаются к кабелю и — *вуаля*! — у нас есть локальная сеть.

За прошедшие годы технология Ethernet претерпела ряд эволюционных изменений, и нынешний Ethernet весьма отличается от оригинальной шинной топологии, работавшей с применением коаксиального кабеля. Сегодня в большинстве конфигураций узлы подключаются к коммутатору при помощи двухточечных сегментов, изготовленных из медного кабеля «витая пара» или оптоволоконного кабеля, как показано на рис. 5.15–5.17.

В середине 90-х появился стандарт Ethernet со скоростью передачи данных 100 Мбит/с — то есть в 10 раз быстрее, чем 10 Мбит/с. В нем сохранились оригинальный протокол Ethernet MAC и формат кадра, но были описаны более высокоскоростные физические уровни для медного кабеля (100BASE-T) и для оптоволоконного кабеля (100BASE-FX, 100BASE-SX, 100BASE-BX). На рис. 5.21 представлены эти различные стандарты, а также обычный протокол Ethernet MAC и формат кадра.

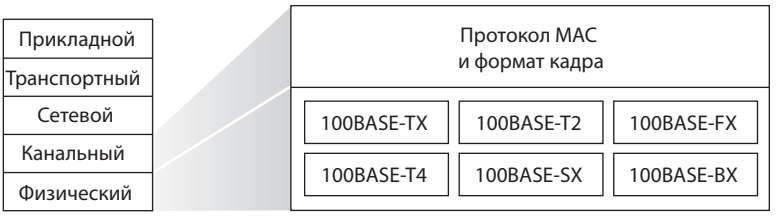


Рис. 5.21. Стандарты Ethernet 100 Мбит/с: канальный уровень общий, физические уровни отличаются

Максимальная длина кабеля для Ethernet 100 Мбит/с составляет 100 метров при передаче по витой паре и до нескольких километров, если используется оптоволокно. Таким образом удается соединять коммутаторы Ethernet в разных зданиях.

Гигабитный Ethernet является усовершенствованием исключительно успешных стандартов 10 Мбит/с и 100 Мбит/с. Гигабитный Ethernet обеспечивает максимальную физическую передачу данных со скоростью 1000 Мбит/с, а также полную совместимость со всей огромной базой уже установленного Ethernet-оборудования. Согласно документу IEEE 802.3z, стандарт Gigabit Ethernet реализует следующий функционал:

- Использует стандартный формат кадра Ethernet (рис. 5.20) и обеспечивает обратную совместимость с технологиями 10BASE-T и 100BASE-T. Таким образом, обеспечивается удобная интеграция Gigabit Ethernet в уже развернутую техническую базу Ethernet-оборудования.
- Допускает использование двухточечных каналов, а также разделяемых широковещательных каналов. При работе с двухточечными каналами используются коммутаторы, а при работе с широковещательными — концентраторы, как было описано выше. В терминологии Gigabit Ethernet концентратор называется *буферизованным распределителем*.
- Использует протокол CSMA/CD при работе с широковещательными каналами. Чтобы обеспечивать при этом приемлемую эффективность работы, максимальное расстояние между узлами строго ограничивается.
- Обеспечивает полнодуплексную эксплуатацию двухточечных каналов на скорости 1000 Мбит/с в обоих направлениях.

Изначально технология Gigabit Ethernet работала по оптоволоконному кабелю, но теперь также допускает эксплуатацию кабеля категории 5 UTP. Технология Ethernet 10 Гбит/с была стандартизирована в 2007 году, после чего локальные Ethernet-сети стали еще мощнее.

Завершая обсуждение технологии Ethernet, постараемся ответить на вопрос, который вы уже наверняка перед собой ставили. Во времена шинных топологий и звездообразных топологий с применением концентратора в Ethernet, разумеется, использовалась широковещательная связь (о которой мы говорили в разделе 5.3). Когда применялся такой механизм связи, между кадрами могли возникать коллизии, если узлы передавали данные одновременно. Для того чтобы справляться с коллизиями, стандарт Ethernet предусматривал использование протокола CSMA/CD, который действительно очень эффективен при прокладке кабельных локальных сетей на небольших территориях. Но в настоящее

время в сетях Ethernet преобладают звездообразные топологии с применением коммутаторов, где реализуется коммутация пакетов с промежуточным хранением и передачей. В таком случае, сохраняется ли сегодня необходимость в использовании MAC-протокола? Как мы вскоре увидим, коммутатор координирует свои операции по передаче данных и никогда не передает более одного кадра на конкретный интерфейс в определенный момент времени. Более того, современные коммутаторы являются полнодуплексными — то есть обеспечивают передачу кадров в обоих направлениях без интерференции. Иными словами, в коммутируемых локальных Ethernet-сетях коллизии отсутствуют, а значит, не нужен и протокол MAC!

Как мы могли убедиться, современный Ethernet *очень сильно* отличается от исходной технологии, изобретенной Меткалфом и Боггсом около 30 лет назад. Скорости передачи информации возросли на три порядка, кадры Ethernet передаются по самым разным носителям, преобладает коммутируемая организация сетей Ethernet, и даже протокол MAC зачастую оказывается не нужен! Можно ли все это *по-прежнему* называть словом Ethernet? Разумеется, «да, по определению». Интересно отметить, что, несмотря на все перечисленное, в этой технологии сохранилась одна постоянная, которая ничуть не изменилась за 30 лет. Это формат кадра Ethernet. Пожалуй, именно формат кадра является истинным и нестареющим ядром стандарта Ethernet.

5.4.3. Коммутаторы канального уровня

До сих пор мы специально не уточняли, что именно делает коммутатор и как он работает. Задача коммутатора — получать входящие кадры канального уровня и передавать их по исходящим каналам. В данном подразделе мы подробно изучим эту функцию перенаправления. Мы убедимся, что коммутатор как таковой является **прозрачным** для хостов и маршрутизаторов, работающих в подсети. Это означает, что хост или маршрутизатор адресуют кадр для другого хоста или маршрутизатора (а не для коммутатора) и спокойно отсылают этот кадр по локальной сети, не учитывая этап работы коммутатора, который получит кадр и переадресует его. Скорость, с которой кадры поступают на любой из выходных интерфейсов коммутатора, иногда может превышать производительность канала, подключенного к этому интерфейсу. Чтобы справиться с этой проблемой, выходные интерфейсы коммутатора снабжены буферами — точно так же, как выходные интерфейсы маршрутизатора

имеют буферы для хранения дейтаграмм. Итак, давайте подробнее ознакомимся с функционированием коммутатора.

Перенаправление данных и фильтрация

Фильтрацией называют способность коммутатора определять, следует ли передать кадр определенному интерфейсу или его можно просто отбросить. **Перенаправлением данных** называют способность коммутаторов определять, какому из интерфейсов следует направить кадр. Фильтрация и перенаправление кадров осуществляются при помощи **таблицы коммутации**. В таблице коммутации содержатся записи для некоторых (не обязательно всех) узлов локальной сети. Каждая запись таблицы коммутации содержит, во-первых, MAC-адрес узла, во-вторых, номер интерфейса коммутатора, ведущего к узлу, и, в-третьих, время включения этой информации в таблицу. Пример таблицы для самого верхнего коммутатора, представленного на рис. 5.15, приведен на рис. 5.22.

Адрес	Интерфейс	Время
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
…	…	…

Рис. 5.22. Часть таблицы для самого верхнего коммутатора, изображенного на рис. 5.15

Хотя представленное здесь описание механизмов перенаправления кадра может напомнить описание перенаправления дейтаграмм в главе 4, мы вскоре обнаружим существенные различия. Следует отметить, что коммутаторы используют MAC-адреса, а не адреса сетевого уровня (например, IP-адреса). Мы также вскоре увидим, что процесс формирования таблицы коммутации значительно отличается от построения таблицы маршрутизации.

Чтобы понять механизмы фильтрации и перенаправления, предположим, что на коммутатор поступает кадр с адресом получателя DD-DD-DD-DD-DD-DD по интерфейсу *x*. Коммутатор делает выборку из своей таблицы по MAC-адресу. Далее возможны три случая:

- В таблице нет записи, соответствующей DD-DD-DD-DD-DD-DD. В таком случае коммутатор переправляет копии кадра во *все* выходные буферы, расположенные на всех интерфейсах, кроме *x*. Иными

словами, если запись для адреса назначения отсутствует, то коммутатор передает кадр широковещательным образом.

- В таблице есть запись, ассоциирующая DD-DD-DD-DD-DD-DD с интерфейсом x. В данном случае сегмент приходит из сегмента ЛВС, содержащего адаптер DD-DD-DD-DD-DD-DD. Не требуется перенаправлять кадр на какие-либо другие интерфейсы, поэтому коммутатор выполняет фильтрацию, просто отбрасывая этот кадр.
- В таблице есть запись, ассоциирующая DD-DD-DD-DD-DD-DD с интерфейсом $y \neq x$. В таком случае кадр должен быть переправлен в тот сегмент локальной сети, который ассоциирован с интерфейсом y. Коммутатор выполняет функцию перенаправления, помещая кадр в выходной буфер, предшествующий интерфейсу y.

Рассмотрим работу этих правил на примере самого верхнего коммутатора с рис. 5.15 и его таблицы коммутации, показанной на рис. 5.22. Предположим, что кадр с адресом получателя 62-FE-F7-11-89-A3 прибывает на коммутатор через интерфейс 1. Коммутатор сверяется с таблицей и определяет, что получатель находится в сегменте локальной сети, соединенном с интерфейсом 1 (то есть в локальной сети электротехнического факультета). Это означает, что данный кадр уже получили все хосты этой локальной сети, поэтому коммутатор отфильтровывает (то есть отбрасывает) его. Теперь предположим, что кадр с тем же адресом получателя прибывает на коммутатор по интерфейсу 2. Коммутатор снова заглядывает в таблицу и видит, что адресат доступен через интерфейс 1. Поэтому коммутатор переправляет кадр в выходной буфер интерфейса 1. Из этого примера должно быть ясно, что, пока таблица коммутатора содержит полную и точную информацию, коммутатор доставляет кадры хостам-получателям, не прибегая к широковещанию.

В этом отношении коммутатор гораздо «интеллектуальнее» концентратора. Но каким же образом конфигурируется таблица коммутатора? Существуют ли на канальном уровне аналоги протоколов маршрутизации сетевого уровня? Либо несчастный менеджер должен выкраивать время и конфигурировать таблицу коммутации вручную?

Самообучение

Коммутатор обладает замечательным свойством (особенно для перетрудившегося администратора), состоящим в том, что его таблица формируется автоматически, динамически и автономно — без вмеша-

тельства администратора вычислительной сети или протокола конфигурирования. Другими словами, коммутаторы обладают способностью **самообучения**, которая реализуется следующим образом.

1. Изначально таблица коммутатора пуста.
2. Для каждого полученного кадра коммутатор сохраняет в своей таблице, во-первых, MAC-адрес, содержащийся в *поле адреса отправителя* кадра, во-вторых, номер интерфейса, по которому прибыл кадр, в-третьих, время получения кадра. Таким образом, коммутатор сохраняет в своей таблице сведения о том, в каком сегменте локальной сети располагается узел, отправивший кадр. Если каждый узел локальной сети передаст по кадру, тогда в таблице коммутации окажутся MAC-адреса всех узлов.
3. Если за определенный период времени (**время старения**) коммутатор не получает кадров с некоторым адресом, этот адрес удаляется из таблицы. Таким образом, если один персональный компьютер в сети заменяется другим (с другим адаптером), локальный адрес первого персонального компьютера в конце концов будет удален из таблицы коммутации.

Рассмотрим способность к самообучению на примере самого верхнего коммутатора с рис. 5.15 и его таблицы коммутации, показанной на рис. 5.22. Пусть в момент времени 9:39 кадр с адресом источника 01-12-23-34-45-56 прибывает по интерфейсу 2. Предположим, что этот адрес отсутствует в таблице. В таком случае коммутатор добавляет к таблице новую запись (рис. 5.23).

Адрес	Интерфейс	Время
01-12-23-34-45-56	2	9:39
62-FE-F7-11-89-A3	1	9:32
7C-BA-B2-B4-91-10	3	9:36
...	...	...

Рис. 5.23. Коммутатор узнает о местоположении адаптера с адресом 01-12-23-34-45-56

Продолжая данный пример, предположим, что время старения для коммутатора составляет 60 минут и между 9:32 и 10:32 на коммутатор не поступают кадры с адресом источника 62-FE-F7-11-89-A3. В таком случае, в 10:32 коммутатор удаляет этот адрес из своей таблицы.

Коммутаторы являются **самонастраивающимися устройствами,** и для их запуска не требуется вмешательство администратора вычислительной сети или пользователя. Администратор, желающий установить коммутатор, должен просто подключить сегменты локальной сети к интерфейсам коммутатора. Специалисту не приходится конфигурировать таблицы коммутатора на этапе установки или при удалении хоста из одного из сегментов локальной сети. Кроме того, коммутаторы являются полнодуплексными: это означает, что любой интерфейс коммутатора может одновременно как отправлять, так и получать информацию.

Свойства коммутации на канальном уровне

Описав общие принципы работы коммутатора канального уровня, рассмотрим его особенности и свойства. Можно перечислить ряд преимуществ, связанных с использованием коммутаторов, а не устройств с широковещательными каналами — таких как шины или звездообразные топологии с применением концентратора:

- *Устранение коллизий*. Если локальная сеть построена с применением коммутаторов (а не концентраторов), полоса передачи данных совершенно не расходуется впустую, так как в ней не возникает коллизий! Коммутаторы буферизуют кадры и никогда не передают более одного кадра в определенный сегмент сети в конкретный момент времени. Как и при работе с маршрутизатором, максимальная совокупная пропускная способность коммутатора равна суммарной частоте всех интерфейсов коммутатора. Следовательно, коммутаторы позволяют существенно увеличить производительность локальной сети по сравнению с использованием широковещательных каналов.
- *Гетерогенные каналы*. Поскольку коммутатор изолирует каналы друг от друга, разные каналы в локальной сети могут работать с разной скоростью и находиться на разных носителях. Например, у самого верхнего коммутатора на рис. 5.22 может быть три канала 1000BASE-T с медными кабелями, скорость передачи каждого 1 Гбит/с, два канала 100BASE-FX с оптоволоконными кабелями, скорость передачи — по 100 Мбит/с, а также один канал с медным кабелем, стандарт 100BASE-T. Такой коммутатор идеально подходит для одновременного использования в сети как устаревающего, так и современного оборудования.
- *Управление*. Наряду с обеспечением повышенной безопасности (см. врезку «О безопасности» далее), коммутатор также упрощает

управление сетью. Например, если адаптер выходит из строя и начинает непрерывно посылать Ethernet-кадры (такой адаптер иногда называют «болтливым»), коммутатор может выявить эту проблему и на внутрисистемном уровне отключить данный адаптер. Располагая такой возможностью, администратор вычислительной сети может спать спокойно, а не вскакивать с постели среди ночи и не лететь на работу, чтобы исправить ситуацию. Аналогично, если обрезать кабель, то отключится лишь тот хост, который был подсоединен этим кабелем к коммутатору. Во времена коаксиальных кабелей многим администраторам вычислительной сети приходилось тратить целые часы, «проходя по линии» (точнее говоря, «проползая по полу»), чтобы отыскать повреждение кабеля, из-за которого легла вся сеть. В главе 9 мы поговорим и о том, что коммутаторы собирают статистику об использовании полосы передачи данных, частоте возникновения коллизий и типах трафика, предоставляя всю эту информацию администратору вычислительной сети. Все эти сведения могут использоваться для отладки и устранения проблем, а также для планирования развития локальной сети. Существуют исследования, рассматривающие возможности дальнейшего расширения управляющих функций в локальных сетях Ethernet, которые уже реализуются в пилотных конфигурациях[76, 294].

О БЕЗОПАСНОСТИ

Анализ коммутируемой локальной сети: отравление коммутатора

Когда хост подключается к коммутатору, он, как правило, получает только те кадры, которые предназначены именно ему. Вернемся, например, к коммутируемой локальной сети, изображенной на рис. 5.17. Когда хост А отправляет кадр хосту Б, а в таблице коммутации есть запись для хоста Б, коммутатор направит этот кадр только хосту Б. Если на хосте В это время будет работать анализатор, то хост В сможет просмотреть содержимое пакета, идущего от А к Б. Соответственно, в среде коммутируемой локальной сети (в отличие от широковещательной локальной сети, такой, как 802.11 LAN или Ethernet-сети на основе концентратора) злоумышленнику сложнее анализировать кадры таким образом. Однако, поскольку коммутатор широковещательно передает кадры с адресами назначения, отсутствующими в таблице коммутации, анализатор на хосте В все-таки может просмотреть некоторые кадры, которые не предназначены самому В. Более того, анализатор сможет про-

сматривать и широковещательные кадры Ethernet, имеющие широковещательный адрес назначения FF–FF–FF–FF–FF–FF. Широко известная разновидность атак в коммутируемых сетях называется **отравлением коммутатора**. При такой тактике на коммутатор направляется масса пакетов со множеством разных фиктивных MAC-адресов. В результате таблица коммутатора быстро заполняется такими фиктивными адресами до отказа, и в ней не остается места для MAC-адресов настоящих хостов. В результате коммутатор начинает передавать большую часть кадров широковещательным образом, и анализатор легко их просматривает[603]. Однако, поскольку такая атака является достаточно сложной даже для опытного злоумышленника, коммутаторы значительно менее уязвимы для несанкционированного сетевого анализа, нежели концентраторы и беспроводные локальные сети.

Сравнение коммутаторов и маршрутизаторов

Как мы узнали в главе 4, маршрутизатор — это устройство для коммутации пакетов, которое работает с промежуточным хранением и перемещает пакеты, опираясь на адресацию сетевого уровня. Хотя коммутатор также обеспечивает передачу с промежуточным хранением, он принципиально отличается от маршрутизатора тем, что при перенаправлении пакетов использует их MAC-адреса. Итак, маршрутизатор осуществляет коммутацию пакетов на третьем уровне, а коммутатор — на втором.

Пусть между маршрутизаторами и коммутаторами и существуют фундаментальные отличия, администратору вычислительной сети зачастую приходится выбирать одно из устройств, которое должно служить в сети соединительным узлом. Например, в сети с рис. 5.15 администратор вычислительной сети вполне мог бы установить маршрутизатор, а не коммутатор, соединив с его помощью факультетские локальные сети, серверы и шлюз, стоящий на границе локальной сети и Интернета. Действительно, маршрутизатор мог бы обеспечивать обмен информацией между факультетами без возникновения коллизий. Итак, если сетевые взаимодействия можно обеспечивать как при помощи коммутаторов, так и с применением маршрутизаторов, то каковы достоинства и недостатки каждого из этих подходов?

Сначала поговорим о коммутаторах. Как было указано выше, коммутаторы являются самонастраивающимися устройствами — за что их так обожают вечно занятые администраторы вычислительной сети во

всем мире. Кроме того, коммутаторы могут достаточно быстро выполнять фильтрацию и перенаправление данных. Как показано на рис. 5.24, коммутатору приходится обрабатывать кадры лишь до второго уровня, а маршрутизатор обрабатывает дейтаграммы до третьего уровня. С другой стороны, для предотвращения зацикливания широковещательных кадров активная топология коммутируемой сети ограничена связующим деревом. Кроме того, в больших коммутируемых сетях нужны объемные ARP-таблицы на хостах и маршрутизаторах, из-за чего будет генерироваться существенный ARP-трафик, нуждающийся в обработке. Вдобавок коммутаторы очень чувствительны к широковещательным штормам. Если какой-нибудь хост «спятит» и начнет генерировать бесконечный поток Ethernet-кадров, то коммутаторы будут транслировать все эти кадры, из-за чего сеть просто рухнет.

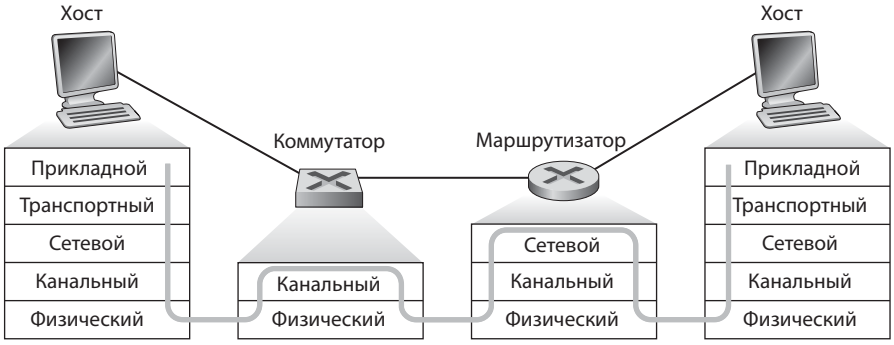


Рис. 5.24. Обработка пакетов на коммутаторах, маршрутизаторах и хостах

Теперь рассмотрим достоинства и недостатки маршрутизаторов. Поскольку адресация на сетевом уровне зачастую является иерархической (а не плоской, как в случае с MAC-адресами), пакеты обычно не движутся через маршрутизатор циклически, даже если в сети есть дублирующие пути. Конечно, такое зацикливание пакетов может происходить, если таблицы маршрутизаторов сконфигурированы неправильно — эта проблема была рассмотрена в главе 4. Однако протокол IP применяет специальное поле заголовка дейтаграммы, которое позволяет ограничить такую цикличность. Следовательно, возможные пути пакетов не ограничиваются связующим деревом, пакет выбирает оптимальную траекторию между исходным и конечным хостом. Поскольку маршрутизаторы не имеют ограничений, обусловленных связующим деревом, они позволяют выстраивать в Интернете очень насыщенную топологию, которая включает в себя, в частности, множество активных

каналов между Европой и Северной Америкой. Еще одна особенность маршрутизаторов заключается в том, что они обеспечивают защиту от широковещательных штормов второго уровня, применяя для этого брандмауэр. Вероятно, самый серьезный недостаток маршрутизаторов заключается в том, что их приходится настраивать вручную — необходимо конфигурировать IP-адреса как для маршрутизаторов, так и для подключенных к ним хостов. Кроме того, маршрутизатор обычно тратит больше времени на обработку одного пакета, чем коммутатор, поскольку ему приходится оперировать полями вплоть до третьего уровня[390].

Учитывая, что и коммутаторов, и у маршрутизаторов есть свои достоинства и недостатки (см. табл. 5.1), попробуем разобраться: в каких случаях сеть организации (например, университетская или корпоративная) должна использовать коммутаторы, а в каких — маршрутизаторы? Как правило, если сеть невелика и включает в себя несколько сотен хостов, то и LAN-сегментов в ней немного. Такая сеть вполне может обслуживаться лишь коммутаторами, поскольку они локализуют трафик и повышают суммарную пропускную способность, не требуя никакой дополнительной конфигурации IP-адресов. Но если мы имеем дело с большой сетью, количество хостов в которой исчисляется тысячами, то в ней, как правило, есть и маршрутизаторы (наряду с коммутаторами). Маршрутизаторы обеспечивают более надежное разграничение трафика, справляются с широковещательными штормами и более интеллектуально прокладывают маршруты между хостам в сети.

Табл. 5.1. Сравнение важнейших возможностей распространенных устройств, обеспечивающих сетевые взаимодействия

	Концентраторы	**Маршрутизаторы**	**Коммутаторы**
Изоляция трафика	Нет	Да	Да
Самонастройка	Да	Нет	Да
Оптимальная маршрутизация	Нет	Да	Нет

Более подробно достоинства и недостатки сетей с коммутаторами и маршрутизаторами рассмотрены в работах Мейерса[343] и Кима[285]. Здесь также затрагивается тема того, как можно усовершенствовать технологию LAN, чтобы количество хостов в локальных сетях удалось увеличить на два порядка по сравнению с сегодняшним уровнем.

5.4.4. Виртуальные локальные сети

Выше, обсуждая рис. 5.15, мы уже отмечали, что современные компьютерные сети организаций зачастую имеют иерархическую конфигурацию. В каждой рабочей группе (отделе или факультете) есть собственная коммутируемая локальная сеть, соединенная с коммутируемыми локальными сетями других отделов при помощи иерархической структуры из коммутаторов. В идеальном мире такая конфигурация должна функционировать превосходно, но на практике все зачастую складывается иначе. В конфигурации сети с рис. 5.15 можно отметить три недостатка:

- *Отсутствует изоляция трафика.* Хотя иерархия и позволяет ограничить групповой трафик в пределах одного коммутатора, широковещательный трафик (то есть, кадры с ARP- или DHCP-сообщениями, либо кадры, чье место назначения еще не известно самообучающемуся коммутатору) по-прежнему должны обходить всю сеть организации. Ограничив область действия такого широковещательного трафика, мы сможем улучшить производительность локальной сети. Что еще важнее, иногда желательно ограничивать широковещательный трафик локальной сети по соображениям безопасности/конфиденциальности. Так, если к одной группе относятся менеджеры высшего звена данной компании, а к другой — недовольные сотрудники, применяющие анализаторы пакетов Wireshark, то администратору вычислительной сети стоит принять меры, чтобы трафик руководителей ни в коем случае не попадал на компьютеры таких сомнительных подчиненных. Чтобы обеспечить изоляцию такого типа, можно заменить центральный коммутатор на рис. 5.15 маршрутизатором. Чуть ниже мы убедимся, что такое ограничительное решение реализуемо и при помощи одних лишь коммутаторов (на уровне 2).

- *Неэффективное использование коммутаторов.* Если бы в организации было не 3 группы, а 10, то потребовалось бы 10 коммутаторов первого уровня. Если бы каждая из этих групп была невелика — скажем, включала бы по 10 человек, — то хватило бы всего одного коммутатора на 96 портов, чтобы удовлетворить потребности каждого, однако такой коммутатор не обеспечивал бы изоляцию каналов.

- *Управление пользователями.* Если тот или иной сотрудник может переходить из одной группы в другую, то на рис. 5.15 придется менять физическую прокладку кабелей, чтобы подключить сотрудника к другому коммутатору. Если сотрудник может одновременно относиться к двум группам, проблема тем более усложняется.

К счастью, все эти проблемы решаются при помощи коммутатора, поддерживающего **виртуальные локальные сети** (virtual local area networks, **VLAN**). Как понятно из названия, такой коммутатор позволяет определять множество *виртуальных* частных сетей в пределах одной *физической* инфраструктуры. Хосты в рамках виртуальной локальной сети взаимодействуют так, как если бы они (и только они) были подключены к одному коммутатору. В виртуальной локальной сети на основе портов порты (интерфейсы) коммутатора делятся на группы администратором вычислительной сети. Каждая такая группа представляет собой виртуальную локальную сеть, причем порты каждой образуют широковещательный домен (это означает, что широковещательный трафик с любого порта из группы будет попадать только на другие порты этой же группы). На рис. 5.25 показан один коммутатор с 16 портами.

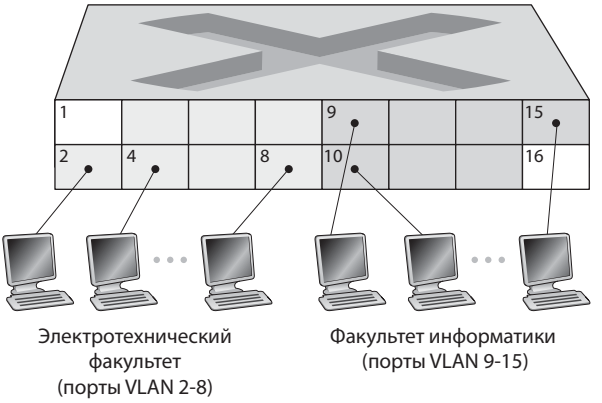


Рис. 5.25. Один коммутатор, на котором сконфигурированы две виртуальные локальные сети

Порты 2–8 относятся к виртуальной локальной сети электротехнического факультета, порты 9–15 относятся к виртуальной локальной сети факультета информатики (порты 1 и 16 к локальным сетям не относятся). Виртуальная локальная сеть решает все проблемы, обозначенные выше: кадры из локальных сетей двух факультетов будут изолированы друг от друга, два коммутатора с рис. 5.15 заменяются одним, а если пользователь с порта 8 переходит на факультет информатики, то администратор вычислительной сети просто переконфигурирует программное обеспечение VLAN так, чтобы порт 8 относился к виртуальной локальной сети факультета информатики. Не составляет труда представить себе, как конфигурируется и работает коммутатор VLAN. Администратор вычислительной сети объявляет, что порт относится к той

или иной виртуальной локальной сети (причем те порты, которые не присвоены той или иной сети явно, считаются портами виртуальной локальной сети, заданной по умолчанию). Это делается при помощи программы для управления коммутатором. В коммутаторе ведется таблица, в которой записываются соответствия между портами и виртуальными локальными сетями. Аппаратная часть коммутатора просто доставляет кадры между портами, относящимися к одной и той же виртуальной локальной сети.

Но добившись полного разграничения двух виртуальных локальных сетей, мы столкнулись с новой сложностью. Как теперь передать трафик с электротехнического факультета на факультет информатики? Возможное решение — подключить один из портов коммутатора, обслуживающего виртуальную локальную сеть (например, порт 1 на рис. 5.25) к внешнему маршрутизатору и сконфигурировать этот порт как относящийся к виртуальным локальным сетям сразу двух факультетов. В данном случае, хотя электротехнический факультет и факультет информатики и станут физически использовать один и тот же коммутатор, логическая конфигурация будет такова, как если бы у этих факультетов имелись отдельные коммутаторы, соединенные через маршрутизатор. IP-дейтаграмма, идущая с электротехнического факультета на факультет информатики, сначала пройдет через VLAN электротехнического факультета, чтобы попасть на маршрутизатор, а затем маршрутизатор перешлет ее в виртуальную локальную сеть факультета информатики, на нужный хост с этого факультета. К счастью, производители коммутаторов облегчают администратору вычислительной сети создание таких конфигураций, конструируя единое устройство, в котором есть *и* коммутатор VLAN, *и* маршрутизатор, поэтому отдельный внешний маршрутизатор не требуется. В одной из задач для самостоятельной работы к этой главе мы исследуем подобный сценарий более детально.

Возвращаясь к рис. 5.15, предположим, что у нас нет отдельного факультета системного проектирования, зато отдельные кафедры электротехнического факультета и факультета информатики расположены в разных корпусах, где им (конечно!) требуется доступ в сеть, причем кафедры одного факультета должны относиться к его общей виртуальной локальной сети. На рис. 5.26 показан еще один коммутатор на 8 портов, и его порты по мере необходимости назначаются как элементы разных виртуальных локальных сетей — электротехнического факультета или факультета информатики. Но как же подключить друг к другу два этих коммутатора? Есть следующее простое решение: на каждом из

коммутаторов можно определить по одному порту, относящемуся к сети электротехнического факультета (и по одному — для факультета информатики), а потом соединить эти порты друг с другом, как показано на рис. 5.26(а). Однако такое решение не масштабируется, поскольку для *N* виртуальных локальных сетей потребуется задействовать по *N* портов на каждом коммутаторе, просто чтобы объединить два устройства.

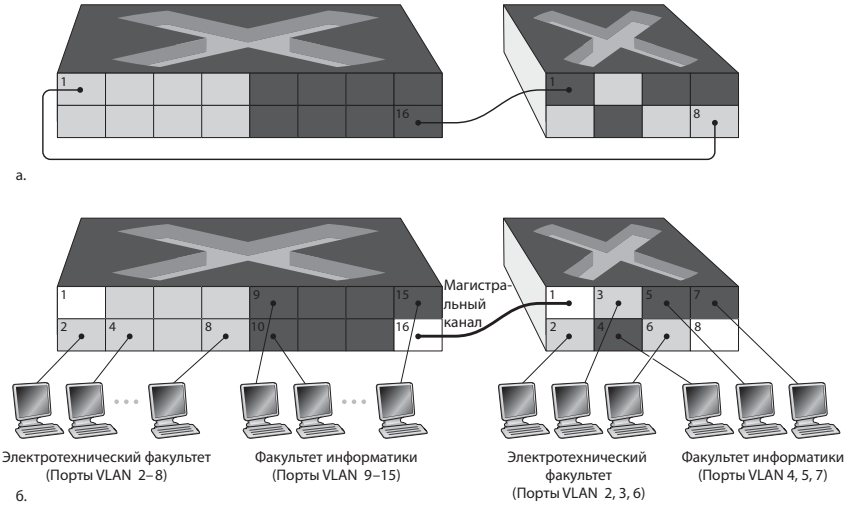


Рис. 5.26. Соединение двух коммутаторов с двумя виртуальными локальными сетями: (а) два кабеля (б) магистральный канал

Более удобный для масштабирования метод подключения коммутаторов из виртуальных локальных сетей называется **VLAN-транкингом**. При использовании такого метода, проиллюстрированного на рис. 5.26(б), на каждом коммутаторе выделяется специальный порт (на коммутаторе слева это порт 16, на коммутаторе справа — порт 1), который конфигурируется как магистральный (trunk port) и предназначается именно для соединения двух VLAN-коммутаторов. Магистральный порт относится ко всем виртуальным локальным сетям, и кадры, направляемые в любую сеть VLAN, проходят по магистральному каналу на другой коммутатор. Но здесь возникает еще один вопрос: как коммутатор узнает, что кадр, прибывший на магистральный порт, относится к конкретной виртуальной локальной сети? Институт IEEE описал расширенный формат кадров Ethernet 802.1Q, специально для работы с кадрами, проходящими по магистральному каналу. Как показано на рис. 5.27, кадр 802.1Q состоит из стандартного Ethernet-кадра и 4-байтной **метки VLAN**. Эта метка содержит идентификационную информа-

цию той виртуальной локальной сети, к которой относится данный кадр. Метка VLAN записывается в кадр коммутатором на стороне отправителя, а на стороне получателя считывается и удаляется другим коммутатором. Сама VLAN-метка состоит из двухбайтного поля TPID (Tag Protocol Identifier — идентификатор протокола метки) с фиксированным шестнадцатеричным значением 81-00, из двухбайтного поля управляющей информации метки (Tag Control Information), содержащего 12-битное поле идентификатора VLAN, и трехбитного поля приоритета, которое функционально напоминает поле TOS из заголовка IP-дейтаграммы.

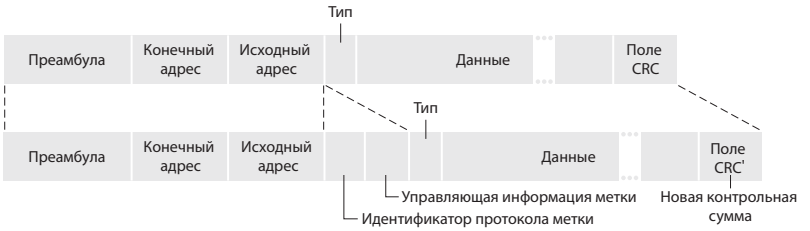


Рис. 5.27. Исходный кадр Ethernet (вверху) и VLAN-кадр Ethernet, снабженный меткой стандарта 802.1Q (внизу)

Здесь мы лишь кратко коснулись темы VLAN и сосредоточились на обсуждении таких виртуальных локальных сетей, которые работают на основе портов. Также необходимо отметить, что существует и ряд других способов определения сетей VLAN. В виртуальных локальных сетях на основе MAC-адресов администратор вычислительной сети задает множество MAC-адресов, относящихся к конкретной виртуальной сети. Когда устройство подключается к порту, мы относим этот порт к соответствующей сети VLAN в зависимости от MAC-адреса устройства. Виртуальные локальные сети также могут определяться на основе протоколов сетевого уровня (например, Ipv4, Ipv6 или Appletalk) и по другим критериям. Подробнее эти возможности описаны в стандарте 802.1Q[234].

5.5. Виртуализация каналов: сеть как канальный уровень

Поскольку эта глава посвящена протоколам канального уровня, а мы приближаемся к ее концу, давайте поговорим о том, как развивались трактовка и значение термина *канал*. В начале этой главы мы рассматривали канал как физическую сущность (провод). Этот провод связывает два хоста, обменивающиеся информацией. При изучении протоко-

лов множественного доступа мы убедились, что несколько хостов могут быть соединены совместно используемым (разделяемым) кабелем и что в роли такого «кабеля» также могут выступать радиоволны или другие носители. Таким образом, мы рассматривали канал как все более абстрактную сущность. В разделе о локальных Ethernet-сетях (рис. 5.15) мы узнали, что среда для соединения хостов может представлять собой довольно сложную инфраструктуру, в которой применяются коммутаторы. Но в ходе всего этого развития понятий сами хосты «все так же считали», что соединительная среда представляет собой обычный провод, работающий на канальном уровне, к которому эти хосты подключены. Так, мы могли убедиться, что Ethernet-хост может совершенно не представлять, как именно он подключен к другим хостам локальной сети: одним коротким LAN-сегментом (рис. 5.17), территориально рассредоточенной коммутируемой локальной сетью (рис. 5.15) или виртуальной локальной сетью (рис. 5.26).

Если соединение между двумя хостами устанавливается с помощью модема коммутируемого доступа, то канал фактически представляет собой линию из телефонной сети. Это логически самостоятельная глобальная телекоммуникационная сеть с собственными коммутаторами, каналами и стеками протоколов для передачи данных и обмена сигналами. Однако на канальном уровне Интернета коммутируемое соединение по телефонной линии воспринимается как связь по обычному «кабелю». Можно сказать, что Интернет виртуализует телефонную сеть, трактуя ее как технологию канального уровня и обеспечивая соединяемость канального уровня между двумя хостами. Возможно, вы помните из главы 2, что оверлейная (наложенная) сеть трактует Интернет просто как носитель для обеспечения связи между двумя наложенными узлами. Она выстраивается на базе Интернета, точно как Интернет выстраивается на телефонной сети.

В этом разделе мы рассмотрим сети MPLS, в которых осуществляется многопротокольная коммутация по меткам. Сети MPLS, в отличие от телефонных сетей с коммутацией каналов, работают с коммутацией пакетов и с установлением виртуальных соединений. В сетях MPLS используются собственные форматы пакетов и методы пересылки данных. Поэтому с учебной точки зрения обсуждение сетей MPLS органично вписывается в материал как о сетевом, так и о канальном уровне. Но в контексте Интернета можно рассматривать MPLS как технологию, сопоставимую с телефонными сетями и коммутируемым Ethernet, которая служит для соединения IP-устройств. Именно поэтому мы обсудим

MPLS в главе о канальном уровне. Сети с ретрансляцией кадров и архитектура ATM также могут применяться для соединения IP-устройств, хотя две эти технологии и являются уже сравнительно старыми (впрочем, их по-прежнему активно внедряют) и здесь рассматриваться не будут. Об этих технологиях подробно рассказано в очень интересной книге Горальски[187]. Здесь мы можем позволить себе лишь краткое введение в эту технологию, поскольку по ней можно написать целые тома (и такие тома уже написаны). Подробное изложение технологии MPLS дается в книге Дэви[124]. Мы поговорим, в основном, о том, как MPLS-серверы соединяют IP-устройства, но также позволим себе несколько более глубокое изучение технологий, лежащих в основе MPLS.

5.5.1. Многопротокольная коммутация по меткам

Многопротокольная коммутация по меткам (MPLS) — это технология, появившаяся в результате ряда промышленных разработок, которые начались еще в середине и конце 90-х. Цель этих разработок заключалась в повышении скорости пересылки данных на IP-маршрутизаторах путем внедрения в них одной из основных концепций, принятых на вооружение в сетях с виртуальными каналами. Эта сущность — метка с фиксированной длиной. Цель разработчиков этих технологий заключалась не в отказе от инфраструктуры передачи IP-дейтаграмм, основанной на маршрутизации, в пользу альтернативной архитектуры на основе меток фиксированной длины и виртуальных каналов, а в дополнении имеющейся архитектуры путем выборочной пометки дейтаграмм и далее маршрутизации дейтаграмм с применением меток фиксированной длины, когда это осуществимо. Важно отметить, что такие функции выполняются в тесном взаимодействии с протоколом IP, используют IP-адресацию и маршрутизацию. Организация IETF унифицировала все эти разработки в протоколе MPLS[492, 493], фактически интегрировав техники работы с виртуальными каналами в сеть с маршрутизацией дейтаграмм.

Начиная наше изучение MPLS, рассмотрим формат кадра канального уровня, который обрабатывается маршрутизатором, поддерживающим MPLS. На рис. 5.28 видно, что кадр канального уровня, передаваемый между устройствами с поддержкой MPLS, содержит небольшой MPLS-заголовок, добавляемый между заголовком уровня 2 (например, Ethernet) и уровня 3 (то есть, IP). В стандарте RFC 3032[493] определяется формат MPLS-заголовка для таких каналов. В других стандартах RFC также определяются заголовки для сетей ATM и сетей с ретрансляцией

кадров. Среди полей MPLS-заголовка есть, в частности, метка (выполняющая роль такого идентификатора виртуального канала, о котором мы говорили еще в разделе 4.2.1). 3 бита зарезервированы для экспериментального использования. Еще в заголовке есть один бит S, который обозначает конец серии расположенных друг над другом MPLS-заголовков (это отдельная более сложная тема, которую мы не будем здесь рассматривать), а также поле предписанного времени жизни.

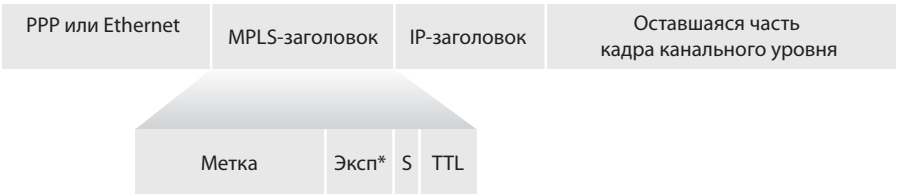


Рис. 5.28. MPLS-заголовок размещен между заголовками канального и сетевого уровней

Из рис. 5.28 сразу становится понятно, что усовершенствованный MPLS-кадр можно пересылать лишь между устройствами, поддерживающими протокол MPLS (ведь обычный маршрутизатор будет изрядно озадачен, если найдет MPLS-заголовок там, где ожидает встретить IP-заголовок!) Маршрутизаторы, поддерживающие MPLS, зачастую именуются **маршрутизаторами с коммутацией по меткам**. Такие устройства пересылают MPLS-кадр, сверяясь с MPLS-меткой в своей таблице маршрутизации, а затем сразу же передают кадр на соответствующий выходной интерфейс. Соответственно, маршрутизатор с поддержкой MPLS *не* извлекает IP-адрес назначения и не ищет совпадения самого длинного префикса в таблице маршрутизации. Но как маршрутизатор узнает, поддерживает ли его маршрутизатор-сосед MPLS, а также какая метка должна быть ассоциирована с заданным IP-адресом? Чтобы ответить на эти вопросы, давайте рассмотрим, как взаимодействует группа маршрутизаторов с поддержкой MPLS.

В примере на рис. 5.29 маршрутизаторы от M1 до M4 поддерживают MPLS. M5 и M6 — стандартные IP-маршрутизаторы. M1 объявляет M2 и M3, что он (M1) может переправить информацию в точку A и что полученный кадр с MPLS-меткой 6 будет передан в точку A. Маршрутизатор M3 объявляет M4, что он может направлять информацию к точ-

* Экспериментальное использование, 3 бита; в текущий момент используется в качестве поля Class of Service (CoS).

кам назначения А и Г и что туда отправятся соответственно входящие кадры с MPLS-метками 10 и 12. Наконец, маршрутизатор М2 объявляет маршрутизатору М4, что он (М2) может отправить информацию в точку А и что полученный кадр с MPLS-меткой 8 будет направлен к А. Итак, маршрутизатор М4 оказывается в интересной ситуации: он знает два MPLS-пути, по которым может достичь А: через интерфейс 0 с исходящей MPLS-меткой 10 и через интерфейс 1 с исходящей MPLS-меткой 8. Общая картина, проиллюстрированная на рис. 5.29, такова: IP-устройства М5, М6, А и Г соединены друг с другом через MPLS-инфраструктуру (маршрутизаторы М1, М2, М3 и М4 поддерживают MPLS). Это соединение выполнено примерно таким же образом, как коммутируемая локальная сеть или ATM-сеть соединяется при помощи IP-устройств. Кроме того, как и в коммутируемых локальных сетях или ATM, маршрутизаторы М1 — М4 с поддержкой MPLS взаимодействуют, *даже не притрагиваясь к IP-заголовку пакета*.

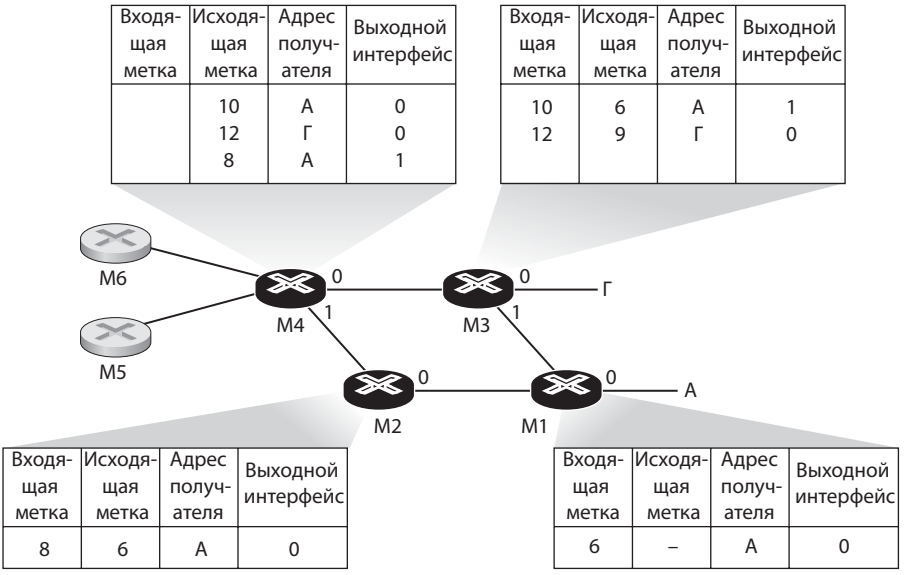


Рис. 5.29. Перенаправление информации с поддержкой MPLS

Выше мы не указали конкретный протокол, который применяется для распространения меток между маршрутизаторами, поддерживающими технологию MPLS. Дело в том, что подробное обсуждение обмена сигналами в MPLS выходит за рамки этой книги. Правда, мы отметим, что рабочая группа IETF по технологии MPLS указала в стандарте RFC 3468[515], что для обеспечения обмена сигналами в MPLS будет использо-

ваться протокол RSVP-TE[497], являющийся расширением над протоколом RSVP. Кроме того, мы не обсудили, как именно вычисляются пути для передачи пакетов между маршрутизаторами, поддерживающими MPLS, ни как MPLS собирает информацию о состоянии канала (например, ширину полосы передачи данных, не зарезервированную MPLS), применяемую в таких расчетах путей. Для подачи такой информации в маршрутизаторы с поддержкой MPLS применяются усовершенствованные варианты алгоритмов с учетом состояния канала (например, OSPF). Интересно отметить, что сами алгоритмы расчета пути не стандартизированы и в настоящее время определяются производителями устройств.

До сих пор в нашем обсуждении MPLS мы делали акцент на том факте, что этот протокол выполняет коммутацию на основе меток, соответственно, ему не нужно учитывать при работе IP-адрес пакета. Но наиболее важные преимущества MPLS и причины огромного интереса, который в настоящее время проявляется к этому протоколу, заключаются не в потенциальном увеличении скорости коммутации, а в новых возможностях управления трафиком, которые дает MPLS. Как было указано выше, у маршрутизатора M4 есть *два* MPLS-пути к точке A. Если бы пересылка выполнялась на IP-уровне, на основании IP-адреса, то протоколы IP-маршрутизации, изученные нами в главе 4, дали бы всего один путь к узлу A путь с наименьшей стоимостью. Следовательно, MPLS позволяет вести пакеты по таким маршрутам, которые недоступны при использовании стандартных IP-протоколов маршрутизации. Здесь мы имеем дело с одним из простых вариантов **управления трафиком** при помощи MPLS[505, 503, 485, 676]. В данном случае оператор сети может переопределять стандартную IP-маршрутизацию и принудительно направлять часть трафика к месту назначения по заданному пути. Одновременно к тому же месту назначения уже по другому пути может направляться другой трафик (выбор иного пути может быть связан с соображениями политики, безопасности или с какими-то другими причинами).

MPLS также можно использовать и в других целях. Например, для быстрого восстановления путей MPLS-маршрутизации, то есть для перенаправления трафика по заранее рассчитанному резервному пути в случае отказа основного[275, 208, 516]. Наконец, отметим, что MPLS может применяться (и действительно применяется) при создании так называемых **виртуальных частных сетей** (virtual private networks, **VPN**). Организуя виртуальные частные сети для заказчика, Интернет-провайдер, чтобы объединить их, использует свою сеть с поддержкой MPLS. MPLS

может применяться для отграничения как ресурсов, так и адресации, применяемой в частной виртуальной сети заказчика, от адресации и ресурсов других пользователей, работающих в сети данного Интернет-провайдера. Подробнее см. в работе Деклерка[127].

Мы вынуждены ограничиться столь кратким обсуждением MPLS. Рекомендуем вам ознакомиться с источниками, упомянутыми в этом разделе. Учитывая, как разнообразны варианты применения MPLS, эта технология вскоре может превратиться в настоящий швейцарский армейский нож для управления Интернет-трафиком! Для несведущих следует пояснить, что швейцарский армейский нож известен своей универсальностью и годится «на все случаи жизни».

5.6. Организация сетей для дата-центров

В последние годы крупные Интернет-корпорации, такие, как Google, Microsoft, Facebook и Amazon (а также другие подобные компании из Азии и Европы) строят огромные дата-центры, где располагаются десятки тысяч, а то и сотни тысяч хостов. Эти хосты параллельно поддерживают функционирование различных облачных приложений (поисковых систем, почтовых сервисов, социальных сетей, электронных магазинов, т.д.). В каждом *дата-центре есть собственная сеть*, соединяющая его хосты друг с другом, а также сам дата-центр — с Интернетом. В этом разделе мы сделаем краткое введение в функционирование сетей дата-центров для облачных приложений.

Стоимость содержания крупного дата-центра огромна. Например, если в дата-центре находится 100 000 хостов, то его обслуживание обходится более чем в 12 млн долларов ежемесячно[189]. Около 45% этих средств уходит на поддержку самих хостов (которые нужно менять раз в 3-4 года), 25% — на инфраструктуру, в частности, на трансформаторы, источники бесперебойного питания (системы UPS), электрогенераторы на случай длительных перебоев с электричеством и системы охлаждения. 15% — на потребляемую электроэнергию. Еще 15% — на обслуживание сетевого оборудования (в частности, маршрутизаторов, коммутаторов и выравнивателей нагрузки), внешних каналов и транзитного трафика. В данных процентных показателях учтена стоимость амортизации оборудования; таким образом, общая метрика стоимости применяется для расчета как единовременной покупки оборудования, так и текущих затрат на его обслуживание — например на электроснаб-

жение. И хотя сетевое оборудование не является самой большой статьей расходов, усовершенствование сетей имеет определяющее значение для снижения затрат на их обслуживание и обеспечения максимальной производительности[189].

Если сравнить дата-центр с ульем, то рабочими пчелами в нем будут хосты. Они хранят и отдают контент (например, веб-страницы и видеоролики), на них хранится электронная почта и документы. Все хосты вместе выполняют масштабные распределенные вычисления (например, они могут осуществлять распределенное вычисление индекса страниц для поисковых систем). В дата-центрах используются особые хосты, называемые **блейдами**. Внешне они немного напоминают коробки для пиццы. Как правило, они имеют обычную для хостов архитектуру, включающую процессор, память и дисковый накопитель. Хосты вертикально располагаются друг над другом в специальных стойках; как правило, на одной стойке находится 20–40 «блейдов». На верхушке каждой стойки располагается коммутатор, метко именуемый **надстоечным** (top-of-rack, TOR). Этот коммутатор соединяет друг с другом хосты, находящиеся в стойке, а также подключает их к другим коммутаторам дата-центра. В частности, у каждого хоста в стойке есть сетевая интерфейсная плата, соединяющая его с надстоечным коммутатором, а у каждого надстоечного коммутатора — дополнительные порты, через которые его можно подключать к другим коммутаторам. Хотя современные хосты обычно подключены к своим надстоечным коммутаторам при помощи Ethernet-соединения со скоростью 1 Гбит/с, скоро могут стать нормой 10 Гбит/с. Кроме того, каждому хосту присваивается внутренний IP-адрес из адресного пространства конкретного дата-центра.

В сети дата-центра поддерживаются два типа трафика. Во-первых, это трафик, идущий между внешними клиентами и внутренними хостами, во-вторых — трафик, идущий только между внутренними хостами. Для работы с трафиком между внутренними хостами и клиентами в сети дата-центра применяются один или несколько **граничных маршрутизаторов**, которые соединяют сеть дата-центра с глобальным Интернетом. Итак, сеть дата-центра не только соединяет его стойки друг с другом, но также подключает эти стойки к граничным маршрутизаторам. На рис. 5.30 изображен пример сети дата-центра. **Проектирование сетей для дата-центров** — это искусство создания внутренних сетей и протоколов, соединяющих стойки как одну с другой, так и с граничными маршрутизаторами. В последнее время появилось множество научных исследований, посвященных такому проектированию[189, 190, 19, 360, 195, 90, 8, 20, 659, 156, 200, 668, 357, 38, 119, 404].

Балансировка нагрузки

Облачный дата-центр — как у Google или Microsoft — параллельно предоставляет множество приложений. В частности, это поисковые, почтовые и видео-приложения. Для поддержки запросов от внешних клиентов каждое приложение ассоциируется с общедоступным IP-адресом, по которому клиенты отправляют запросы и с которого получают отклики. Внутри дата-центра приходящие извне запросы сначала направляются на **распределитель (балансировщик) нагрузки**. Задача этого механизма — распределять запросы по хостам, выравнивая испытываемую ими нагрузку в зависимости от того, насколько загружен задачами конкретный хост в настоящий момент. В крупном дата-центре зачастую имеется несколько распределителей нагрузки, каждый из которых обслуживает конкретные облачные приложения. Такой распределитель нагрузки иногда именуется «коммутатором четвертого уровня», поскольку принимает решения в зависимости от номера порта (уровень 4), а также от IP-адреса получателя, содержащегося в пакете. Получив запрос, относящийся к тому или иному приложению, распределитель нагрузки передает его одному из хостов, работающих с этим приложением. Хост может вызывать службы других хостов в ходе выполнения этого запроса. Когда хост заканчивает обработку запроса, он отправляет ответ обратно распределителю нагрузки, который, в свою очередь, ретранслирует этот ответ внешнему клиенту. Распределитель не только рассредоточивает нагрузку между хостами, но и выполняет NAT-подобные функции, транслируя общедоступный внешний IP-адрес во внутренний IP-адрес соответствующего хоста. Затем выполняется обратная трансляция — для пакетов, идущих по направлению к клиентам. Таким образом, исключается непосредственный контакт между хостами и клиентами. Это полезно с точки зрения безопасности, поскольку дата-центр скрывает внутреннюю сетевую структуру и не позволяет клиентам взаимодействовать с хостами.

Иерархическая архитектура

Для небольшого дата-центра, где работает всего несколько тысяч хостов, вполне может подойти простая сеть, в которой будет граничный маршрутизатор, распределитель нагрузки и несколько десятков стоек, соединенных общим Ethernet-коммутатором. Но чтобы дата-центр мог масштабироваться, расширяясь до сотен тысяч хостов, в нем часто применяется **иерархия маршрутизаторов и коммутаторов** — такая топология, как показана на рис. 5.30. В верхней части иерархии находится граничный маршрутизатор, подключенный к маршрутизаторам доступа

(на рис. 5.30 всего два маршрутизатора доступа, но их может быть гораздо больше). Под каждым маршрутизатором расположены три уровня коммутаторов. Каждый маршрутизатор доступа подключается к коммутатору верхнего уровня, а каждый из таких коммутаторов соединяется с множеством коммутаторов второго уровня и с распределителем нагрузки. В свою очередь, каждый коммутатор второго уровня подключается к множеству стоек через их надстоечные коммутаторы (которые относятся уже к третьему уровню). Все каналы, как правило, используют для работы с канальными и физическими протоколами связь по Ethernet, со смешанным применением медных и оптоволоконных кабелей. При соблюдении такой иерархической структуры можно масштабировать дата-центр до сотен тысяч хостов.

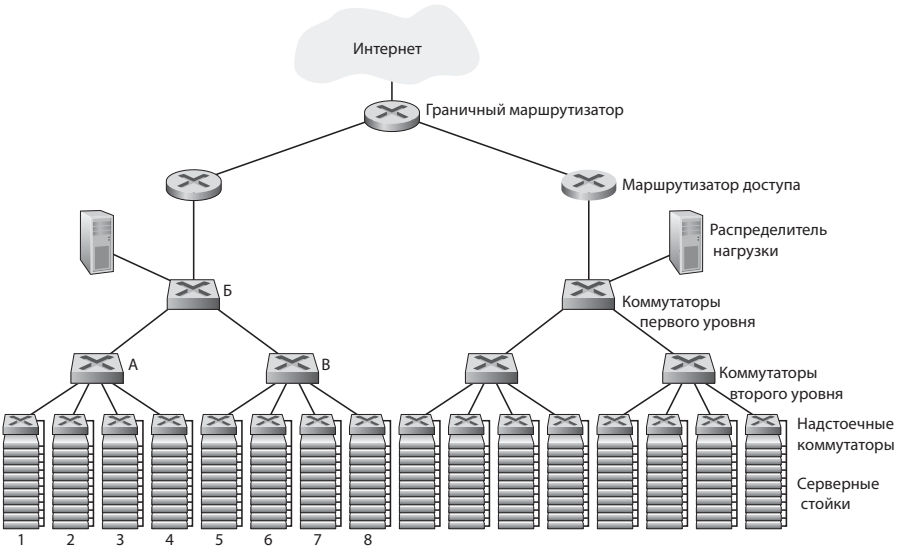


Рис. 5.30. Сеть дата-центра с иерархической топологией

Поскольку для провайдера облачных приложений критически важно непрерывно обеспечивать высокую их доступность, в дата-центрах также присутствует избыточное оборудование и каналы (на рис. 5.30 они не показаны). Например, каждый надстоечный коммутатор может подключаться к двум коммутаторам второго уровня, а каждый маршрутизатор доступа, коммутатор первого или второго уровня можно продублировать и также интегрировать этот дубль в общую систему[190, 102]. В иерархической схеме на рис. 5.30 мы видим, что под каждым маршрутизатором доступа образуется своя подсеть хостов. Для локализации широковеща-

тельного ARP-трафика каждая из этих подсетей разбивается на более мелкие VLAN-подсети, содержащие по несколько сотен хостов[189].

Хотя традиционная иерархическая архитектура, описанная выше, решает проблему масштабирования, она страдает от *ограниченной межхостовой производительности*[190]. Чтобы понять суть этого ограничения, вновь обратимся к рис. 5.30 и предположим, что каждый хост соединяется со своим надстоечным коммутатором при помощи канала с частотой 1 Гбит/с, а между коммутаторами работают каналы с частотой 10 Гбит/с. Два хоста в одной стойке в любой момент могут обмениваться информацией на скорости 1 Гбит/с, если это допускают интерфейсные карты хостов. Однако если в сети дата-центра существует множество потоков данных, идущих одновременно, то максимальная скорость обмена данными между двумя хостами из *разных* стоек может быть гораздо ниже. Чтобы подробнее разобраться в этой проблеме, давайте рассмотрим модель трафика, состоящего из 40 потоков информации, одновременно идущих между 40 парами хостов в разных стойках. Точнее, давайте предположим, что каждый из 10 хостов стойки 1 на рис. 5.30 посылает поток данных на парный хост, расположенный в стойке 5. Аналогично существует десять потоков данных, одновременно идущих между хостами из стоек 2 и 6, и между хостами из стоек 3 и 7, и между хостами из стоек 4 и 8. Если любой из таких потоков будет разделять пропускную способность канала с другими потоками, проходящими по этой линии, то 40 потоков, попадающие в канал от А к Б шириной 10 Гбит/с, равно как и потоки из аналогичного канала от Б к В, будут иметь в распоряжении всего 10 Гбит/с / 40 = 250 Мбит/с, что существенно ниже возможностей интерфейсной карты. Эта проблема еще сильнее обостряется для тех потоков между хостами, которые должны переходить на более высокие уровни в иерархии. Возможное решение, позволяющее справиться с таким ограничением — внедрение высокоскоростных коммутаторов и маршрутизаторов. Но в таком случае сам дата-центр станет гораздо дороже, поскольку такое оборудование очень недешево.

Важно поддерживать достаточно широкую полосу передачи данных для обмена информацией между хостами, поскольку основное требование к дата-центрам — гибкое размещение вычислений и сервисов[190, 156]. Например, крупный Интернет-поисковик может работать на тысячах хостов, распределенных по множеству стоек. Полоса передачи данных для обмена информацией между такими парами хостов должна оставаться стабильно широкой. Аналогично, сервису облачных вычислений, например EC2, может потребоваться эксплуатировать на разных хостах множество виртуальных машин, обслуживающих клиентов. Эти машины должны выда-

вать максимальную производительность, независимо от конкретного расположения хостов в дата-центре. Если хосты физически распределены по множеству стоек, то в сети вполне могут возникать узкие места, подобные описанным выше, и производительность будет довольно низкой.

Тенденции в развитии сетей для дата-центров

Чтобы удешевить дата-центры и в то же время повысить их производительность и сократить среднее время задержки, облачные гиганты Интернета — например Google, Facebook, Amazon и Microsoft — постоянно развертывают новые проекты сетей для дата-центров. Хотя эти проекты и являются проприетарными, можно описать ряд важных тенденций.

Одна из таких тенденций — применение новых архитектур взаимных соединений и сетевых протоколов, которые позволяют преодолевать недостатки традиционных иерархических систем. Например, активно используется подход с заменой иерархии коммутаторов и маршрутизаторов на **полносвязную топологию**[190, 19, 195]. Такая топология показана на рис. 5.31. В подобной системе каждый коммутатор первого уровня подключается ко всем коммутаторам второго уровня, так, чтобы (1) межхостовому трафику никогда не приходилось подниматься над уровнями коммутаторов и (2) при наличии n коммутаторов первого уровня мы увидим между любыми двумя коммутаторами второго уровня n непересекающихся путей. Такой дизайн позволяет значительно оптимизировать межхостовую передачу данных. Чтобы убедиться в этом, вновь вернемся к нашему примеру с 40 потоками. Топология с рис. 5.31 позволяет справляться с такой совокупностью потоков, поскольку существует четыре непересекающихся пути между первым и вторым коммутаторами второго уровня. Вместе они дают общую пропускную способность в 40 Гбит/с между вышеупомянутыми двумя коммутаторами. Подобный вариант дизайна не только позволяет значительно сгладить ограничение, связанное с межхостовой пропускной способностью, но также создает более гибкую вычислительную и служебную среду, где связи между любыми двумя стойками, не подключенными к одному и тому же коммутатору, являются логически эквивалентными, независимо от конкретного расположения этих стоек в дата-центре.

Другая важная тенденция связана с применением контейнерных модульных дата-центров (MDC)[680, 654]. Заводская сборка MDC представляет собой «мини дата-центр», заключенный в стандартном 12-метровом транспортируемом контейнере. В каждом таком контейнере насчитывается до

нескольких тысяч хостов, установленных в десятках плотно расположенных стоек. Там, где устраивается большой дата-центр, такие контейнеры подключаются друг к другу и к Интернету. Когда контейнер в заводской комплектации устанавливается в большом дата-центре, техническое обслуживание такого контейнера зачастую осложняется. Поэтому каждый контейнер конструируется с расчетом на постепенное снижение производительности (мягкую деградацию). По мере того, как со временем компоненты контейнера (серверы и коммутаторы) начинают отказывать, контейнер продолжает работать, но его производительность постепенно снижается. После того, как значительная часть компонентов отключится и, таким образом, производительность упадет ниже приемлемого порогового уровня, весь контейнер изымают из эксплуатации и заменяют новым.

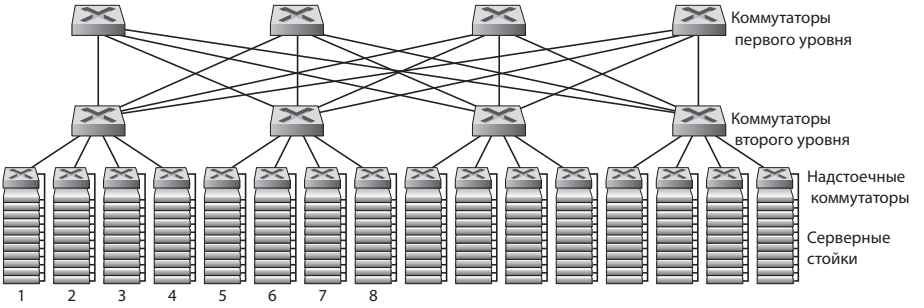


Рис. 5.31. Сильно связанная топология сети данных

При выстраивании дата-центра из контейнеров возникают новые вызовы, связанные с прокладкой сетей. С MDC-контейнерами используются сети двух типов: внутренняя сеть, ограниченная пределами одного контейнера, и опорная (базовая) сеть, соединяющая между собой контейнеры[195, 156]. В каждом контейнере, который содержит порядка нескольких тысяч хостов, можно построить полносвязную сеть, описанную выше, пользуясь недорогими легкодоступными коммутаторами Gigabit Ethernet. Однако создание опорной сети, которая включает в себя сотни и тысячи контейнеров, но при этом должна обеспечивать широкую полосу передачи данных между хостами под типичной нагрузкой — по-прежнему нетривиальная задача. В работе Фаррингтона[156] предложена гибридная электро-оптическая архитектура коммутаторов для взаимного соединения контейнеров.

При применении сильно связанных топологий оказывается очень сложно разрабатывать алгоритмы маршрутизации для взаимодействия

коммутаторов такой сети. Возможное решение — применить случайную маршрутизацию[190]. Другой вариант, описанный в работе Гао[195] — использование на каждом хосте нескольких интерфейсных карт. Это позволяет подключить хост сразу к нескольким дешевым легкодоступным коммутаторам, а сами хосты могут интеллектуально направлять трафик между этими коммутаторами. В современных дата-центрах широко применяются варианты и расширения этих подходов. Вероятно, в скором будущем появятся и многие другие инновации в проектировании дата-центров; заинтересованные читатели могут ознакомиться с научными работами на эту тему.

5.7. Ретроспектива: один день из жизни запроса веб-страницы

Итак, в этой главе мы рассмотрели канальный уровень, а в предыдущих главах — сетевой, транспортный и прикладной уровни. Наше путешествие вниз по стеку протоколов подошло к концу. В самом начале было сказано: «бóльшая часть этой книги посвящена протоколам компьютерных сетей», и в первых пяти главах мы действительно занимались именно этой тематикой. Прежде чем перейти к более узкоспециальным главам из второй части книги, мы хотели бы поставить точку в нашем изучении стека протоколов, сделав интегрированный целостный обзор тех из них, которые были изучены выше. Такая общая картина, в частности, поможет нам оценить, какое (огромное!) множество протоколов применяется при удовлетворении даже самого простого сетевого запроса, связанного с загрузкой веб-страницы.

На рис. 5.32 проиллюстрирована наша модель: студент Боб работает с ноутбуком, подключается к Ethernet-коммутатору университетской сети и загружает веб-страницу (скажем, google.com). Как мы знаем, для выполнения такой, казалось бы, простой задачи на внутрисистемном уровне происходит *масса* всего интересного. В лабораторной работе Wireshark в конце этой главы более подробно исследуются файлы трассировки, в которых содержится информация о пакетах, передаваемых при реализации таких сценариев.

5.7.1. Начало: DHCP, UDP, IP и Ethernet

Предположим, Боб загружает свой ноутбук, а потом подключает его к Ethernet-кабелю. Этот кабель подсоединен к университетскому

Ethernet-коммутатору, который, в свою очередь, подключен к маршрутизатору, как показано на рис. 5.32. Университетский маршрутизатор обслуживается Интернет-провайдером, в данном случае — comcast.net. В нашем примере провайдер Comcast предоставляет сервис DNS всему университету. Это означает, что DNS-сервер находится в сети Comcast, а не в университетской сети. Мы будем исходить из того, что DHCP-сервер работает прямо на маршрутизаторе — действительно, зачастую так и бывает.

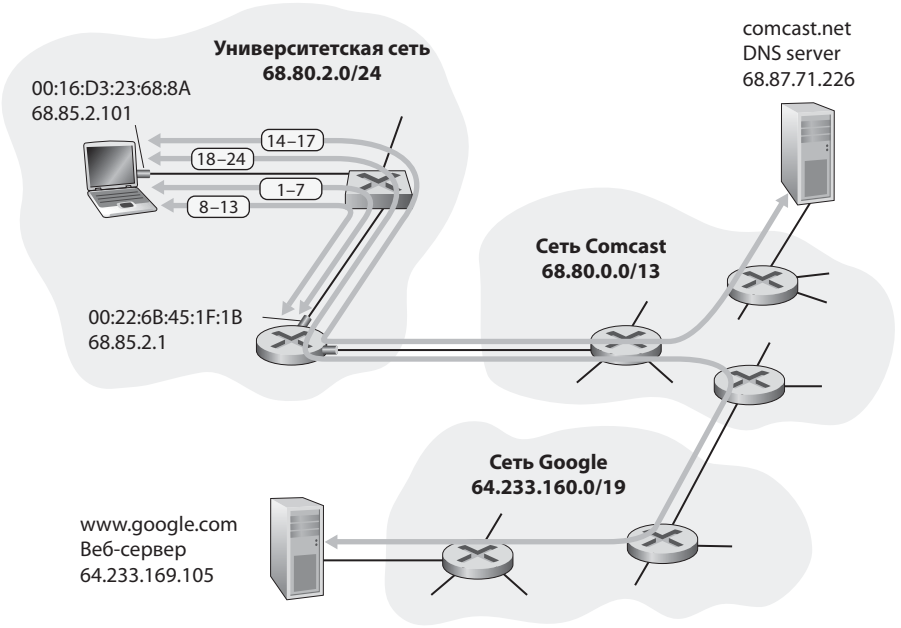


Рис. 5.32. Один день из жизни запроса веб-страницы. Схема сети и действия в ней

Когда Боб только подключает ноутбук к сети, он еще ничего не может сделать (даже загрузить веб-страницу), пока у него нет IP-адреса. Соответственно, первым делом ноутбук Боба должен запустить протокол DHCP для получения с локального DHCP-сервера не только IP-адреса, но и другой информации:

1. Операционная система на ноутбуке Боба создает **сообщение с запросом DHCP** (см. раздел 4.4.2) и записывает это сообщение в **UDP-сегмент** (см. раздел 3.3). В качестве порта получателя указывается порт 67 (DHCP-сервер), а порт отправителя имеет номер 68 (DHCP-клиент). После этого UDP-сегмент записывается в **IP-**

дейтаграмму (раздел 4.4.1) с широковещательным IP-адресом получателя (255.255.255.255). IP-адрес отправителя будет равен 0.0.0.0, так как у ноутбука Боба еще нет IP-адреса.

2. IP-дейтаграмма, содержащая сообщение с запросом DHCP, затем помещается в **Ethernet-кадр** (раздел 5.4.2). Ethernet-кадр обладает MAC-адресами назначения FF:FF:FF:FF:FF:FF, поэтому кадр будет широковещательно передаваться всем устройствам, подключенным к коммутатору (остается надеяться, что среди этих устройств окажется и DHCP-сервер). Исходный MAC-адрес кадра совпадает с адресом ноутбука Боба, 00:16:D3:23:68:8A.
3. Широковещательный Ethernet-кадр, содержащий DHCP-запрос — это первый кадр, отправленный ноутбуком Боба на Ethernet-коммутатор. Коммутатор широковещательно передает этот входящий кадр на все свои выходные порты, в том числе на порт, подключенный к маршрутизатору.
4. Маршрутизатор получает широковещательный Ethernet-кадр с DHCP-запросом на интерфейс с MAC-адресом 00:22:6B:45:1F:1B, после чего IP-дейтаграмма извлекается из Ethernet-кадра. Широковещательный IP-адрес назначения дейтаграммы указывает, что IP-дейтаграмма должна быть обработана на данном узле более высокоуровневыми протоколами, поэтому полезная нагрузка дейтаграммы (UDP-сегмент) демультиплексируется (раздел 3.2) вплоть до UDP, а сообщение с запросом DHCP извлекается из UDP-сегмента. Итак, теперь у DHCP-сервера есть сообщение с запросом DHCP.
5. Предположим, что DHCP-сервер, работающий на маршрутизаторе, может выделять адреса из блока **CIDR** (раздел 4.4.2) 68.85.2.0/24. Таким образом, в данном примере все IP-адреса университета будут относиться к блоку адресов провайдера Comcast. Далее допустим, что DHCP-сервер выделяет ноутбуку Боба адрес 65.85.2.101. DHCP-сервер создает **сообщение DHCP ACK** (раздел 4.4.2), в котором содержится этот IP-адрес, а также следующая информация: IP-адрес DNS-сервера (68.87.71.226), IP-адрес шлюзового маршрутизатора, задаваемого по умолчанию (68.85.2.1) и, наконец, блок подсети (65.85.2.0/24), он же — «маска сети». DHCP-сообщение записывается в UDP-сегмент, который заключается в IP-дейтаграмму, а она, в свою очередь — в Ethernet-кадр. Ethernet-кадр располагает исходным MAC-адресом (это адрес интерфейса маршрутизатора, соединяющего маршрутизатор с домашней сетью — 00:22:6B:45:1F:1B) и конечным MAC-адресом (это адрес ноутбука Боба 00:16:D3:23:68:8A).

6. Ethernet-кадр, содержащий сообщение DHCP ACK (одноадресно), передается маршрутизатором на коммутатор. Поскольку коммутатор является **самообучающимся** (раздел 5.4.3), а ранее получил Ethernet-кадр (с DHCP-запросом) с ноутбука Боба, коммутатору уже известно, что кадр, идущий на адрес 00:16:D3:23:68:8A, нужно просто передать на выходной порт, ведущий к ноутбуку Боба.

7. Ноутбук Боба получает сообщение DHCP ACK, извлекает IP-дейтаграмму из Ethernet-кадра, затем UDP-сегмент из IP-дейтаграммы, после чего — сообщение DHCP ACK из UDP-сегмента. Затем DHCP-клиент с ноутбука Боба записывает его IP-адрес, а также IP-адрес его DNS-сервера. Кроме того, он заносит адрес в свою **IP-таблицу маршрутизации** (раздел 4.1). Ноутбук Боба будет отсылать на заданный по умолчанию шлюз все дейтаграммы, чей адрес назначения находится вне его подсети 68.85.2.0/24. На данном этапе ноутбук Боба уже инициализировал свои сетевые компоненты и готов обрабатывать операции выборки веб-страниц. Обратите внимание: из четырех этапов работы с DHCP, описанных в главе 4, необходимыми являются только первые два.

5.7.2. Начало продолжается: DNS и ARP

Когда Боб вводит в браузер адрес www.google.com, начинается длинная цепочка событий. Ее конечный результат — отображение главной страницы Google в браузере. Браузер с ноутбука Боба начинает этот процесс, создав **TCP-сокет** (раздел 2.7), который будет использоваться для отправки **HTTP-запроса** (раздел 2.2) на сайт www.google.com. Чтобы создать этот сокет, ноутбуку Боба потребуется узнать IP-адрес www.google.com. В разделе 2.5 мы изучили, что для такого преобразования доменного имени в IP-адрес используется **протокол DNS**.

8. Для этого операционная система на ноутбуке Боба создает **сообщение с запросом DNS** (раздел 2.5.3), помещая строку «www.google.com» в тот раздел DNS-сообщения, где содержится запрос. Затем это DNS-сообщение записывается в UDP-сегмент с портом назначения 53 (это DNS-сервер). После этого данный UDP-сегмент помещается в IP-дейтаграмму, имеющую IP-адрес получателя 68.87.71.226 (это адрес DNS-сервера, который мы приняли в возвращенном сообщении DHCP ACK на этапе 5) и IP-адрес отправителя 68.85.2.101.

9. После этого ноутбук Боба помещает дейтаграмму, содержащую сообщение с запросом DNS, в Ethernet-кадр. Этот кадр будет отправлен

(адресован на сетевом уровне) на шлюзовой маршрутизатор сети того университета, где учится Боб. Однако, хотя ноутбуку Боба и известен IP-адрес шлюзового маршрутизатора университета (65.85.2.1) — эта информация была получена в сообщении DHCP ACK на этапе 5, описанном выше, ему не известен MAC-адрес этого шлюза. Чтобы получить MAC-адрес шлюзового маршрутизатора, на ноутбуке Боба понадобится задействовать **протокол ARP** (раздел 5.4.1).

10. Ноутбук Боба создает **сообщение с запросом ARP**, где указывается целевой IP-адрес 68.85.2.1 (адрес шлюзового маршрутизатора). ARP-сообщение помещается в Ethernet-кадр с широковещательным адресом получателя (FF:FF:FF:FF:FF:FF), после чего ноутбук отправляет Ethernet-кадр на коммутатор, который, в свою очередь, доставляет этот кадр на все подключенные к нему устройства, в том числе на шлюзовой маршрутизатор.
11. Шлюзовой маршрутизатор получает кадр с сообщением с ARP-запросом на интерфейс, ведущий в него из университетской сети, и определяет, что целевой IP-адрес 68.85.2.1 в ARP-сообщении совпадает с IP-адресом его интерфейса. После этого шлюзовой маршрутизатор подготавливает **ARP-ответ**, указывая, что его MAC-адрес 00:22:6B:45:1F:1B соответствует IP-адресу 68.85.2.1. Сообщение с ARP-ответом записывается в Ethernet-кадр с адресом получателя (это ноутбук Боба). После этого кадр отправляется на коммутатор, который доставляет его на ноутбук Боба.
12. Ноутбук Боба получает кадр с ответным ARP-сообщением и извлекает MAC-адрес шлюзового маршрутизатора (00:22:6B:45:1F:1B) из этого сообщения.
13. Теперь ноутбук Боба (*наконец-то*!) может отправить Ethernet-кадр с DNS-запросом на MAC-адрес шлюзового маршрутизатора. Обратите внимание: IP-дейтаграмма в этом кадре имеет IP-адрес получателя 68.87.71.226 (это адрес DNS-сервера), тогда как адрес получателя кадра — 00:22:6B:45:1F:1B (это шлюзовой маршрутизатор). Ноутбук Боба отправляет кадр на коммутатор, который, в свою очередь, доставляет этот кадр на шлюзовой маршрутизатор.

5.7.3. Начало продолжается: внутридоменная маршрутизация на DNS-сервер

14. Шлюзовой маршрутизатор получает кадр и извлекает из него IP-дейтаграмму, содержащую DNS-запрос. Маршрутизатор уточняет адрес

назначения этой дейтаграммы (68.87.71.226) и определяет по своей таблице маршрутизации, что дейтаграмма должна быть отправлена на маршрутизатор сети Comcast, который изображен на рис. 5.32 левее всех. IP-дейтаграмма помещается в кадре канального уровня для перемещения по каналу, который связывает университетский маршрутизатор с маршрутизатором Comcast, изображенным на рис. 5.32 левее всех. Кадр отправляется по этому каналу.

15. Вышеупомянутый маршрутизатор из сети Comcast получает кадр, извлекает IP-дейтаграмму, проверяет ее адрес получателя (68.87.71.226), а потом по своей таблице маршрутизации определяет выходной интерфейс, через который этот кадр должен быть отправлен на DNS-сервер. Таблица маршрутизации была заполнена при помощи протокола внутридоменной маршрутизации сети Comcast (это может быть протокол **RIP, OSPF** или **IS-IS**, см. раздел 4.6), а также с применением протокола **BGP**, обеспечивающего **междоменную маршрутизацию в Интернете**.

16. Наконец, IP-дейтаграмма, содержащая DNS-запрос, прибывает на DNS-сервер. DNS-сервер извлекает сообщение с DNS-запросом, ищет в своей базе данных DNS адрес www.google.com (раздел 2.5), после чего находит запись DNS-ресурса, содержащую IP-адрес (64.233.169.105) сайта www.google.com (предполагается, что эта запись уже кэширована на DNS-сервере). Как вы помните, эти кэшированные данные берутся с **авторитетного DNS-сервера** (раздел 2.5.2), отвечающего за сайт google.com. Сервер формирует **ответное DNS-сообщение**, в котором содержится отображение данного хост-имени на соответствующий IP-адрес, и помещает это ответное сообщение в UDP-сегмент. Сегмент, находящийся в IP-дейтаграмме, адресуется на ноутбук Боба (68.85.2.101). Эта дейтаграмма будет отправлена обратно в сеть Comcast, а оттуда далее, через Ethernet-коммутатор на ноутбук Боба.

17. Ноутбук Боба извлекает IP-адрес сервера www.google.com из DNS-сообщения. Наконец, *проделав всю эту огромную работу*, ноутбук Боба готов подключиться к серверу www.google.com!

5.7.4. Клиент-серверное взаимодействие: TCP и HTTP

18. Теперь, когда ноутбук Боба обладает IP-адресом www.google.com, он может создать **TCP-сокет** (раздел 2.7), который будет использоваться для отправки сообщения **HTTP GET** (раздел 2.2.3) на сайт www.google.com. Когда Боб создает TCP-сокет, протокол TCP в ноутбуке

Боба сперва должен выполнить **тройное рукопожатие** (раздел 3.5.6) с протоколом TCP на www.google.com. Поэтому для начала ноутбук Боба создает сегмент **TCP SYN** с портом назначения 80 (для HTTP), помещает TCP-сегмент в IP-дейтаграмме, задавая IP-адрес получателя 64.233.169.105 (www.google.com). Эта дейтаграмма помещается в кадр с MAC-адресом получателя 00:22:6B:45:1F:1B (шлюзовой маршрутизатор). После всех этих операций кадр отправляется на коммутатор.

19. Маршрутизаторы из университетской сети, сети Comcast и сети Google шлют дейтаграмму с пакетом TCP SYN на адрес www.google.com. Для этого используются таблицы маршрутизации каждого из них, как описано выше в шагах 14-16. Как вы помните, записи в таблицах маршрутизации, управляющие перемещением пакетов по междоменным связям между сетями Comcast и Google, формируются протоколом **BGP** (раздел 4.6.3).
20. Наконец, дейтаграмма с пакетом TCP SYN прибывает на www.google.com. Там сообщение TCP SYN извлекается из дейтаграммы и демультиплексируется на нужный сокет, ассоциированный с портом 80. Специальный сокет соединения (раздел. 2.7) создается для установления соединения между HTTP-сервером Google и ноутбуком Боба по протоколу TCP. Генерируется сегмент TCP SYNACK (раздел 3.5.6), затем этот сегмент помещается в дейтаграмме, адресованной на ноутбук Боба, и, наконец, заключается в кадре канального уровня, подходящем для передачи по каналу, соединяющему www.google.com и его первый транзитный маршрутизатор.
21. Дейтаграмма, содержащая сегмент TCP SYNACK, направляется через сети Google, Comcast и университета, в итоге оказываясь на сетевой Ethernet-карте ноутбука Боба. Дейтаграмма демультиплексируется в операционной системе на TCP-сокет, созданный в шаге 18, и этот сокет переходит в соединенное состояние.
22. Когда сокет на ноутбуке Боба (*наконец-то*!) готов отправлять байты на www.google.com, браузер на ноутбуке создает сообщение с запросом HTTP GET (раздел 2.2.3), содержащее адрес URL, откуда требуется взять данные. Затем сообщение HTTP GET записывается на сокет, причем запрос GET становится полезным содержимым TCP-сегмента. TCP-сегмент помещается в дейтаграмму, отправляется и доставляется на сайт www.google.com, как описано в шагах 18–20 выше.
23. HTTP-сервер по адресу www.google.com считывает сообщение HTTP GET с TCP-сокета, создает **HTTP-ответ** (раздел 2.2), помещает кон-

тент запрошенной веб-страницы в тело ответного HTTP-сообщения, после чего отправляет это сообщение на TCP-сокет.

24. Дейтаграмма, содержащая HTTP-ответ, направляется через сети Google, Comcast и университетскую сеть, после чего прибывает на ноутбук Боба. Браузер на ноутбуке считывает HTTP-сообщение с сокета, извлекает html-код веб-страницы из тела HTTP-ответа и (*наконец-то*!) отображает веб-страницу.

В вышеописанном сценарии было затронуто множество базовых аспектов работы с сетями! Если вы в основном понимаете все шаги в вышеприведенном примере, то и вы усвоили массу фундаментальных тем с тех пор, как прочли в разделе 1.1: «бóльшая часть этой книги посвящена протоколам компьютерных сетей». На том этапе вы, возможно, еще не вполне представляли себе, что такое «протокол»! Вышеприведенный пример может показаться очень детализированным, однако на самом деле мы не упомянули в нем еще ряд дополнительных протоколов (например, NAT — службу трансляции сетевых адресов, работающую на университетском шлюзовом маршрутизаторе, протоколы безопасности для доступа к университетской сети, шифрования сегментов и дейтаграмм, протоколы для управления сетью), а также аспектов (веб-кэширование, иерархия DNS), которые играют немаловажную роль в глобальном Интернете. Ряд этих тем, а также многие другие будут рассмотрены далее, во второй части этой книги.

Наконец, необходимо отметить, что пример, приведенный нами выше, был интегрированным и целостным, но в то же время мы смогли достаточно подробно показать в нем «внутренности» многих из тех протоколов, которые были изучены в первой части книги. Пример, в основном, был призван ответить на вопросы «как», а не «почему». Более обширное и вдумчивое рассмотрение организации сетевых протоколов в целом дается в работе Кларка[109] и документе RFC 5218[553].

5.8. Заключение

В данной главе мы обсудили канальный уровень: его службы, основополагающие принципы функционирования, ряд важных узкоспециальных протоколов, которые используют данные принципы при реализации канальных сервисов.

Мы узнали, что основное назначение канального уровня заключается в перенаправлении дейтаграммы сетевого уровня с одного узла (это

может быть хост, коммутатор, маршрутизатор, точка доступа Wi-Fi) на другой, смежный с ним узел. Мы убедились, что все протоколы канального уровня работают путем инкапсуляции дейтаграммы сетевого уровня в кадре сетевого уровня, после чего этот кадр передается по каналу на смежный узел. Однако мы узнали, что кроме формирования кадров протоколы канального уровня предоставляют разнообразные службы по доступу к каналу, передаче и доставке данных. Это разнообразие отчасти обусловлено широким разнообразием типов каналов, с которыми должны работать такие протоколы. В обычном двухточечном канале есть всего один отправитель и один получатель, которые обмениваются информацией по одному «кабелю». Канал множественного доступа совместно используется сразу многими отправителями и получателями. Соответственно, протокол для обслуживания такого канала с множественным доступом располагает специальным протоколом множественного доступа для координации работы с каналом. При работе с MPLS «канал», соединяющий два смежных узла (например, в контексте протокола IP два IP-маршрутизатора являются смежными, то есть, располагаются на расстоянии одного перехода друг от друга по пути к тому или иному узлу назначения), на практике может представлять собой самодостаточную *сеть*. В сущности, феномен сети, которую можно представить как всего один канал, не должен казаться странным. Например, телефонная линия, соединяющая домашний компьютер/модем с удаленным модемом/маршрутизатором, в сущности, представляет собой извилистый путь, проложенный через достаточно сложную телефонную *сеть*.

Среди базовых принципов обмена информацией на канальном уровне мы затронули темы обнаружения и исправления ошибок, протоколы множественного доступа, адресацию канального уровня, виртуализацию (виртуальные локальные сети), а также создание обширных коммутируемых локальных сетей и сетей для дата-центров. В настоящее время такие коммутируемые сети привлекают значительное внимание исследователей. Что касается обнаружения и исправления ошибок, мы рассмотрели, как можно добавлять дополнительные биты к заголовку кадра, чтобы обнаруживать, а в некоторых случаях — и исправлять ошибки, связанные с инвертированием разрядов. Такие ошибки могут возникать в ходе передачи кадра по каналу. Мы исследовали простые схемы контроля четности и проверки контрольных сумм, а также поговорили о более надежном методе, связанном с применением кодов циклического контроля. Затем мы поговорили о протоколах множественного доступа. Мы описали и изучили три основных подхода к обеспечению и координации доступа к широковещательному каналу: способы сегментирования канала (TDM,

FDM), методы произвольного доступа (протоколы ALOHA и CSMA) и методы поочередного доступа (опрос и передача маркера). Мы изучили сеть для кабельного доступа и обнаружили, что в ней действительно применяются многие из вышеописанных методов множественного доступа. Мы выяснили, что если несколько узлов совместно используют широковещательный канал, то требуется специальная адресация на канальном уровне. Мы узнали, что адреса на канальном и на сетевом уровнях значительно отличаются. Именно поэтому в Интернете применяется специальный протокол разрешения адресов (ARP), используемый для преобразования этих адресов из одной формы в другую. Далее мы подробно исследовали исключительно успешный сетевой протокол Ethernet. Далее мы поговорили о том, как узлы, совместно использующие широковещательный канал, образуют локальную сеть и как множество таких сетей можно объединить в более крупную локальную сеть — и все это *без* привлечения маршрутизации сетевого уровня. Кроме того, мы выяснили, как можно создавать множество виртуальных локальных сетей на базе одной физической инфраструктуры подобной сети.

Мы завершили изучение канального уровня, рассмотрев, как сети MPLS предоставляют сервисы канального уровня при соединении IP-маршрутизаторов и сделали обзор конструкций сетей, которые задействуются в современных огромных дата-центрах. Мы резюмировали эту главу (а в сущности — все первые пять глав нашей книги), описав, как много протоколов требуется для загрузки одной-единственной веб-страницы. Закончив изучение канального уровня, *мы завершили наше путешествие вниз по стеку протоколов*! Разумеется, под канальным уровнем есть еще и физический, но его детали лучше изучать в другом курсе (например, по теории телекоммуникаций, а не по компьютерным сетям). Правда, мы затронули некоторые аспекты физического уровня в главе 1 (раздел 1.2) и в этой главе. В следующей главе мы вновь затронем физический уровень, когда займемся изучением характеристик беспроводных каналов.

Хотя наше путешествие по стеку протоколов и окончено, изучение компьютерных сетей далеко от завершения. В следующих четырех главах мы обсудим беспроводные сети, мультимедийные сети, сетевую безопасность и администрирование сетей. Эти темы не укладываются в тот или иной конкретный уровень, а охватывают сразу несколько. Понимание этих тем (которые трактуются в некоторых работах по сетям как продвинутые) требуют глубокого понимания всех уровней стека протоколов. Именно это понимание вы должны были приобрести, прочитав главу о протоколах канального уровня.

Глава 6
БЕСПРОВОДНЫЕ И МОБИЛЬНЫЕ СЕТИ

В мире телефонии последние 15 лет, бесспорно, стали «золотой эпохой» сотовой связи. Количество абонентов сотовых сетей по всему миру увеличилось с 34 млн в 1993 году до почти 5,5 млрд в 2011-м. Более того, количество абонентов сотовой связи превышает количество абонентов проводных телефонных сетей. Многочисленные преимущества сотовой связи очевидны для всех: безграничный доступ к глобальной телефонной сети в любом месте и в любое время с помощью легкого ультрапортативного мобильного устройства. Ждет ли нас вскоре подобный взрыв для мобильных Интернет-устройств, всех этих ноутбуков, КПК и смартфонов, обещающих безграничный доступ к глобальной сети в любое время и в любом месте?

Вне зависимости от темпов роста отрасли беспроводных Интернет-устройств в будущем, сегодня уже ясно, что сети и службы, связанные с мобильной связью, останутся с нами и точно уже никуда не денутся. С точки зрения сетевых технологий, задачи поставленные такими сетями, а именно на сетевом и канальном уровнях, настолько сильно отличаются от того, что мы видим в проводных компьютерных сетях, что для изучения особенностей беспроводных и мобильных сетей нам потребуется отдельная (например, *эта*) глава.

Мы начнем ее с обсуждения мобильных пользователей, беспроводных каналов связи и сетей, а также их взаимоотношений с более крупными (как правило, проводными) сетями, к которым они подключены. Мы проведем различие между задачами, поставленными *беспроводной* природой каналов связи в таких сетях, и *мобильностью*, предоставляемой этими сетями. Это поможет нам лучше выделить, идентифицировать и разобраться с ключевыми концептами в данной сфере. Обратите внимание, что во многих сетевых средах сетевые узлы являются беспроводными, но не мобильными (например, домашние или офисные беспроводные сети с подключенными стационарными рабочими станциями с большими дисплеями). Кроме того, существуют ограниченные формы мобильности, которым не требуются беспроводные сети (например, работник, использующий проводной ноутбук дома, выключает его, приез-

жает на работу на машине и подключает этот ноутбук к проводной сети организации, в которой работает). Конечно, наибольший восторг вызывают те сетевые среды, в которых пользователи являются как беспроводными, *так и* мобильными, например, если мобильный пользователь (скажем, находящийся на заднем сиденье автомобиля) делает голосовой вызов посредством IP-телефонии (VoIP) и большое количество непрерывных TCP-подключений при быстрой езде по автостраде со скоростью 160 км/ч. Именно здесь, на пересечении беспроводного и мобильного, мы столкнемся с самыми интересными техническими задачами!

Мы начнем с примера, сочетающего в себе беспроводную связь и мобильность — сеть, к которой пользователи подключаются посредством беспроводной связи (и, возможно, мобильной телефонии) и далее, снова по беспроводным каналам, подключаются к еще большей сетевой инфраструктуре. После этого в разделе 6.2 мы рассмотрим характеристики такого беспроводного канала связи. Кроме того, в текст главы мы также включили краткое описание технологии множественного доступа с кодовым разделением (CDMA), протокола разделенного доступа к среде передачи данных, часто используемого в беспроводных сетях. В разделе 6.3 мы более подробно изучим некоторые аспекты канального уровня стандарта беспроводной локальной сети IEEE 802.11 (Wi-Fi), кроме того, мы вкратце рассмотрим технологию Bluetooth и прочие персональные сети. В разделе 6.4 представлен обзор доступа в Интернет посредством сетей сотовой связи, в том числе технологии третьего (3G) и четвертого (4G) поколений, предоставляющей как голосовую связь, так и высокоскоростной доступ к Интернету. В разделе 6.5 мы обратим свое внимание на мобильность, сосредоточившись на проблемах определения местонахождения пользователя мобильной сети, его маршрутизации и «передачи» пользователей, активно перемещающихся по сети от одной точки доступа к другой. Мы изучим, каким образом эти мобильные службы применяются в стандарте мобильного IP и GSM в разделах 6.6 и 6.7 соответственно. Наконец, в разделе 6.8, мы поговорим о влиянии беспроводных каналов связи и мобильности на протоколы транспортного уровня и сетевые приложения.

6.1. Введение

На рис. 6.1 изображена конфигурация сети, на примере которой мы будем рассматривать особенности беспроводной передачи данных и мобильности. Для начала, наш рассказ будет носить достаточно общий характер. Это сделано для того, чтобы включить в обсуждение как беспро-

водные локальные сети, например IEEE 802.11, так и сети сотовой связи, такие как сеть 3G. По ходу обсуждения мы сузим наш рассказ до более детального повествования о конкретных беспроводных структурах.

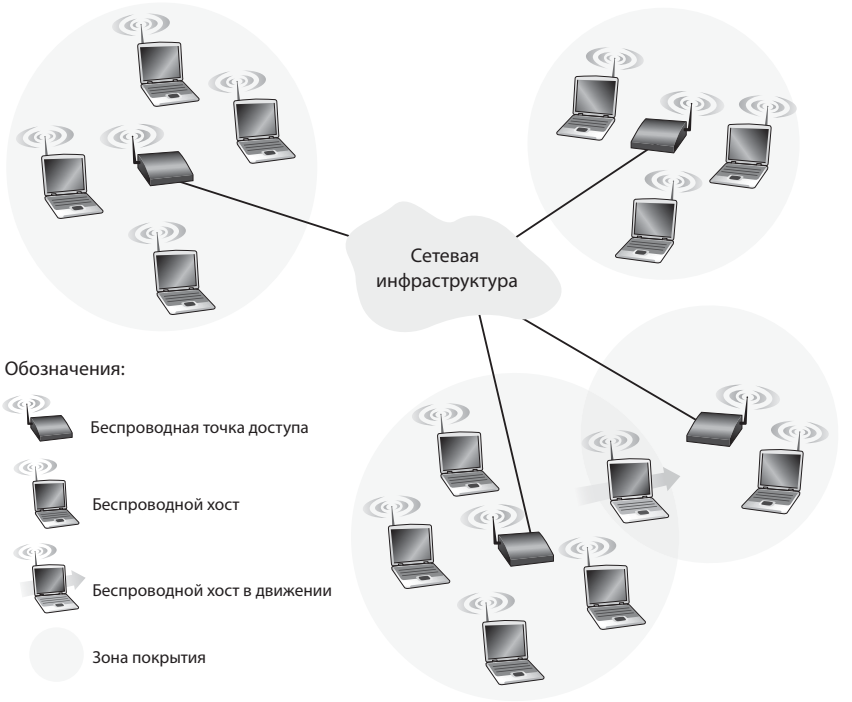


Рис. 6.1. Элементы беспроводной сети

В беспроводной сети можно выделить следующие элементы:

- *Беспроводной хост.* Как и в случае с проводными сетями, хосты — это конечные устройства, на которых осуществляется выполнение приложений. **Беспроводным хостом** может быть ноутбук, КПК, смартфон или настольный компьютер. Сами по себе хосты могут быть или не быть мобильными.

ИСТОРИЯ

Публичный доступ к Wi-Fi: в ближайшем будущем на ближайшем фонарном столбе?

Точки доступа к Wi-Fi в местах общего пользования, где пользователи могут получить доступ к беспроводной сети 802.11, все быстрее и быстрее становятся обыденным явлением в гостиницах, аэропор-

тах и кафе по всему миру. В большинстве студенческих городков предоставляется неограниченный доступ к беспроводным сетям, кроме того в наши дни сложно найти отель, который бы не располагал беспроводным доступом к Интернету.

За прошедшее десятилетие во многих городах были развернуты и эксплуатируются муниципальные сети Wi-Fi. Идея обеспечить широким массам неограниченного доступа к Wi-Fi в качестве еще одной услуги муниципалитета (наподобие уличных фонарей), преодолев цифровой барьер, предоставив всем горожанам неограниченный доступ в Интернет и ускорив темпы экономического развития, является очень привлекательной. Большое количество городов по всему миру, в том числе Филадельфия, Торонто, Гонконг, Миннеаполис, Лондон и Окленд, либо вынашивают планы предоставить неограниченный беспроводной доступ к Интернету на своей территории, либо уже достигли определенного прогресса в реализации этой идеи.

Целью осуществления такого проекта в Филадельфии было «превратить Филадельфию в крупнейшую в США точку доступа к Wi-Fi и оказать поддержку в улучшении образования, уменьшении цифрового барьера, развитии городских микрорайонов, а также сократить расходы на правительство». Результатом амбициозной программы — соглашения между городом, некоммерческой организацией «Беспроводная Филадельфия» и провайдером услуг доступа к Интернету «EarthLink» — стало создание действующей сети, состоящей из точек доступа 802.11b, монтируемых на столбах световых опор и устройствах контроля дорожного движения, которая покрыла 80% территории города. Однако по финансовым и эксплуатационным причинам эта сеть была продана группе частных инвесторов в 2008 году, а те позже перепродали сеть обратно городу в 2010-м. Другие города, например Миннеаполис, Торонто и Гонконг, добились успеха при гораздо меньших усилиях.

Сам факт, что сети 802.11 функционируют в не лицензируемом диапазоне (и поэтому могут быть развернуты без необходимости приобретать дорогостоящие разрешения на использование частотного диапазона), казалось бы, мог сделать их привлекательными в финансовом плане. Однако точки доступа 802.11 (см. раздел 6.3) обладают гораздо меньшей территорией охвата, чем базовые станции сотовой связи 3G (см. раздел 6.4), из-за чего требуется установка большего количества точек доступа для покрытия одинаковых по площади территорий. Однако с другой стороны, сотовые сети передачи данных, предоставляющие доступ в Интернет, работают в диапазоне частот, подлежащем лицензированию. Операторы сотовой связи платят за разрешение на использование частотного диапазона миллиарды долларов, что, конечно, превращает такие сети в бизнес, а не услугу муниципальных властей.

- *Беспроводные каналы связи*. Хост подключается к базовой станции (см. ниже) или к другому беспроводному хосту посредством **беспроводного канала связи**. Различные технологии беспроводной связи характеризуются различной скоростью передачи данных и могут обеспечивать связь на различных расстояниях. На рис. 6.2 показаны две ключевые характеристики (территория покрытия и скорость передачи данных) наиболее популярных стандартов беспроводных сетей. (Впрочем, цель этого рисунка — предоставить только общую информацию о вышеупомянутых характеристиках. Так, некоторые из этих типов сетей лишь начинают внедряться, а скорость передачи данных в некоторых случаях может не достигать указанных значений или превышать их, в зависимости от расстояния, состояния канала связи и количества пользователей, подключенных к беспроводной сети.) Мы рассмотрим эти стандарты в первой части данной главы. В разделе 6.2 мы также поговорим о других характеристиках беспроводных каналов связи, таких как интенсивность и причины возникновения битовых ошибок.

  На рис. 6.1 показано, что беспроводные хосты, расположенные на периферии сети подключены с помощью беспроводных каналов связи к большей сетевой инфраструктуре. Забегая вперед, добавим, что беспроводные каналы связи иногда также используются *внутри* самой сети для соединения маршрутизаторов, коммутаторов и прочего сетевого оборудования. Однако в этой главе мы сфокусируемся на использовании беспроводной связи на периферии сети, так как именно здесь приходится сталкиваться с наиболее захватывающими техническими задачами и именно здесь происходит основной рост.

- *Базовая станция*. **Базовая станция** — это ключевой компонент инфраструктуры беспроводной сети. В отличие от беспроводного хоста и беспроводного канала связи, у базовой станции нет очевидных аналогов в проводных сетях. Базовая станция отвечает за отправку и получение данных (например, пакетов) от беспроводного хоста, связанного с этой базовой станцией. Базовая станция также часто ответственна за координацию передачи данных большого количества беспроводных хостов, подключенных к ней. Когда мы говорим, что беспроводной хост «подключен» к базовой станции мы имеем в виду, что (1) хост находится в зоне досягаемости базовой станции и (2), что хост использует базовую станцию для передачи данных в более крупную сеть. Примерами базовых станций в сотовой телефонии служат **вышки сотовой связи**, а в беспроводных локальных сетях 802.11 — **точки доступа**.

На рис. 6.1 базовая станция подключена к большей сети (например, к Интернету, к корпоративной, домашней, либо телефонной сети). Поэтому она функционирует как посредник канального уровня между беспроводным хостом и остальным миром, с которым этот хост общается.

Часто хосты, подключенные к базовым станциям, называются работающими в **инфраструктурном режиме**, так как все традиционные сетевые службы (например, назначение адресов или маршрутизация) предоставляются самой сетью, к которой хост подключается посредством базовой станции. В сетях с **прямым подключением** отсутствует подобная инфраструктура, к которой подключались бы беспроводные хосты. В отсутствие такой инфраструктуры хосты должны самостоятельно предоставлять такие службы, как маршрутизация, назначение адресов, DNS-подобная трансляция имен и прочее.

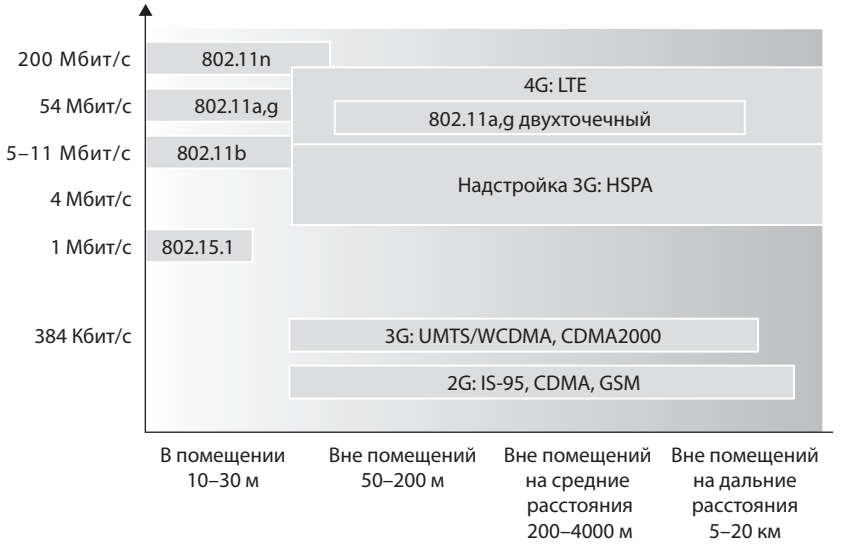


Рис. 6.2. Характеристика каналов связи отдельных стандартов беспроводных сетей

Когда мобильный хост выходит из зоны покрытия одной базовой станции и попадает в зону покрытия другой, он меняет точку подключения к большей сети (то есть, иными словами, подключается к другой базовой станции). Этот процесс называют **эстафетной передачей**. Такая мобильность поднимает большое количество непростых вопросов. Если хост может перемещаться, то каким образом необходимо определять его текущее местонахождение в сети для переадресации направляемых это-

му мобильному хосту данных? Каким образом осуществляется адресация, ведь хост может находиться в любом из многих доступных мест? Если хост перемещается *во время* активности TCP-соединения или телефонного вызова, каким образом должна происходить маршрутизация данных, чтобы соединение при этом не прерывалось? Эти и большое (большое!) количество других вопросов превратили беспроводные и мобильные сети в поле очень захватывающих исследований.

- *Сетевая инфраструктура*. Это та самая большая сеть, с которой может связаться хост.

  Обсудив все «кусочки» беспроводной сети, мы обращаем ваше внимание на то, что эти кусочки могут быть скомбинированы различными способами для создания различных типов беспроводных сетей. Классификация типов беспроводных сетей может оказаться вам полезной при дальнейшем чтении данной главы, а также, если вы будете читать или изучать какие-либо материалы о беспроводных сетях помимо нашей книги. На самом верхнем уровне мы можем классифицировать беспроводные сети по двум критериям: во-первых, проходит ли пакет данных в беспроводной сети только через *один или несколько беспроводных переходов* и во-вторых, — присутствует ли в сети какая-либо *инфраструктура*, как, например, базовая станция.

- *Однопереходная сеть с инфраструктурой*. В таких сетях присутствует базовая станция, подключаемая к большей проводной сети (например, Интернет). Более того, все сообщение между этой базовой станцией и беспроводным хостом осуществляется за один беспроводный переход. Сетями 802.11 вы чаще всего пользуетесь в учебных аудиториях, кафе или библиотеках. Сотовые сети передачи данных 3G, о которых мы поговорим чуть позже, также попадают под эту категорию.

- *Однопереходная сеть без инфраструктуры*. В таких сетях отсутствует базовая станция, подключаемая к беспроводной сети. Однако, как мы увидим в дальнейшем, один из узлов такой однопереходной сети может выступать в качестве координатора передачи данных между остальными узлами. Сети Bluetooth (которые мы изучим в разделе 6.3.6) и сети 802.11 в режиме прямого подключения являются однопереходными сетями без инфраструктуры.

- *Многопереходная сеть с инфраструктурой*. В таких сетях базовая станция присутствует и подключается по проводам к большей про-

водной сети. Однако некоторые беспроводные узлы должны осуществлять связь через другие беспроводные узлы для установления связи через базовую станцию. Под эту категорию подпадают некоторые беспроводные сенсорные сети и так называемые **беспроводные смешанные сети**.

- *Многопереходная сеть без инфраструктуры*. В сетях такого типа отсутствует базовая станция и сообщения, отправляемые одним узлом, могут передаваться посредством еще нескольких узлов, прежде чем достигнут места назначения. Узлы сети также могут быть мобильными, при этом соединения между узлами будут изменяться. Сети такого класса называют **мобильными децентрализованными сетями** (mobile ad-hoc network**, MANET**). Если узлами такой сети являются какие-либо транспортные средства, то сеть называется **децентрализованной транспортной сетью** (vehicular ad-hoc network, **VANET**). Как вы можете представить себе, разработка протоколов для таких сетей — это серьезный вызов и объект непрекращающихся исследований.

В этой главе мы ограничимся, по большей части, повествованием об однопереходных сетях, а затем в основном о сетях с инфраструктурой.

А сейчас давайте глубже изучим технические задачи, встающие перед нами при работе с беспроводными и мобильными сетями. Мы начнем с изучения отдельного беспроводного канала связи, откладывая обсуждение вопросов мобильности до более поздних разделов главы.

6.2. Беспроводные каналы связи и характеристики сети

Давайте начнем с рассмотрения простой проводной, например, домашней, сети с проводным коммутатором Ethernet (см. раздел 5.4), служащим для соединения хостов. Если мы заменим проводную сеть Ethernet беспроводной сетью 802.11, то интерфейс беспроводной сети заменит интерфейс Ethernet хоста, а точка доступа — коммутатор Ethernet. Однако на уровне сети и выше не потребуется практически никаких изменений. Поэтому для рассмотрения важных различий между проводными и беспроводными сетями, мы, по-видимому, должны обратить внимание на канальный уровень. На самом деле мы можем найти некоторое количество различий между проводным и беспроводным каналом связи:

- *Ослабление сигнала.* Электромагнитное излучение ослабляется при прохождении через вещество (например, радиосигнал, проходящий через стену). Сигнал рассеивается даже в свободном пространстве, что приведет к ослаблению силы сигнала (иногда это явление называют **потерями при распространении**) при увеличении расстояния между отправителем и получателем.
- *Помехи от других источников.* Радиоисточники, ведущие передачу на одинаковой частоте, будут создавать друг для друга помехи. Например, беспроводные телефоны работают на частоте 2,4 ГГц, и ту же частоту имеют беспроводные локальные сети 802.11b. Поэтому пользователь беспроводной локальной сети 802.11b, говорящий по беспроводному телефону, работающему на частоте 2,4 ГГц, может столкнуться с тем, что ни телефон, ни сеть не будут нормально функционировать. В дополнение к помехам от источников радиосигналов, помехи может также вызвать электромагнитный шум из окружающей среды (например, мотор, работающий неподалеку, или микроволновая печь).
- *Эффект многолучевого распространения волн.* **Эффект многолучевого распространения волн** возникает при отражении частей электромагнитных волн от различных объектов и земли, в результате физическая длина пути, проходимого сигналом между отправителем и получателем, может варьироваться. Это приводит к «размыванию» сигнала, получаемого приемником. Объекты, движущиеся между получателем и отправителем, могут время от времени изменять интенсивность эффекта многолучевого распространения волн.

Для более подробного изучения характеристик беспроводных каналов связи, моделей и измерений см. публикацию Андерсона[22].

Все вышесказанное делает очевидным факт, что количество битовых ошибок в беспроводных каналах будет большим, чем в проводных. Поэтому неудивительно, что в протоколах беспроводных каналов связи (таких как протокол 802.11, который мы изучим в следующем разделе) используются не только мощные алгоритмы выявления ошибок с помощью кода циклического контроля (CRC), но также надежные протоколы передачи данных канального уровня, позволяющие пересылать поврежденные кадры данных.

Обсудив все проблемы, которые могут возникнуть в беспроводном канале связи, давайте обратим наше внимание на хост, принимающий сигнал. Хост получает электромагнитный сигнал, являющийся сочета-

нием ослабленной формы исходного сигнала, переданного отправителем (ослабленного за счет потерь при распространении и многолучевого распространения), и фонового шума в окружающей среде. **Соотношение сигнал-шум** (signal-to-noise ratio, **SNR**) — это относительный показатель силы принимаемого сигнала (то есть переданных данных) и фонового шума. Как правило, показатель SNR измеряется в децибелах (дБ), единице, используемой инженерами-электриками, по мнению некоторых, в основном для того, чтобы запутать ученых-компьютерщиков. Показатель SNR в децибелах является двадцатикратным отношением десятичного логарифма амплитуды принимаемого сигнала к амплитуде фонового шума. Для наших целей нам достаточно знать лишь то, что большое значение показателя SNR облегчает приемнику работу по отделению передаваемого сигнала от фонового шума.

На рис. 6.3 (адаптирован из работы Холланда[106]) приводится **соотношение интенсивности (коэффициента) битовых ошибок** (bit error rate, **BER**), что, грубо говоря, есть вероятность получения ресивером ошибочного бита и показателя соотношения сигнал-шум (SNR) для трех техник модуляции сигнала при кодировании передаваемого сигнала по идеальному беспроводному каналу связи. Теория модуляции и кодирования сигнала, а также отделение сигнала от фонового шума и интенсивность битовых ошибок находятся далеко вне поля зрения нашей книги[593]. Тем не менее на рис. 6.3 приводится ряд характеристик физического уровня, важных для понимания протоколов беспроводной связи более высоких уровней:

- *Для приведенной схемы модуляции верно, что чем выше значение SNR, тем ниже показатель BER.* Так как отправитель может увеличить показатель SNR, увеличив силу передаваемого сигнала, отправитель может уменьшить возможность получения битового кадра с ошибкой, увеличив мощность передачи. Заметим, однако, что увеличение силы за определенные граничные значения не дает сколько-нибудь ощутимого практического преимущества, например, для уменьшения значения BER с 10^{-12} до 10^{-13}. Кроме того, увеличение мощности передачи имеет определенные *недостатки*: увеличение энергозатрат со стороны отправителя (важный аспект для пользователей мобильных устройств с аккумуляторным питанием), кроме того, увеличивается вероятность интерференции сигналов отправителя и других участников (см. рис. 6.4(б)).
- *Для заданного значения показателя SNR, чем выше скорость передачи данных (как верных, так и ошибочных), тем выше значение*

BER. Например, на рис. 6.3 видно, что для показателя SNR в 10 дБ, скорость передачи данных при модуляции BPSK равна 1 Мбит/с, при этом значение BER меньше 10^{-7}, в то время как для модуляции QAM16 со скоростью передачи данных 4 Мбит/с, значение BER уже 10^{-1}, что слишком высоко для практического применения. Однако, при показателе SNR 20 дБ скорость передачи данных при модуляции QAM16 равняется 4 Мбит/с, а значение BER — 10^{-7}, при том что скорость передачи данных при модуляции BSPK составляет всего 1 Мбит/с, но значение BER этого типа модуляции сигнала настолько низко, что его невозможно отобразить на нашем графике. В данной ситуации, если значение BER, равное 10^{-7}, является терпимым, то предпочтительней было бы использование модуляции QAM16. Эти размышления приводят нас к еще одной, последней характеристике, описанной далее.

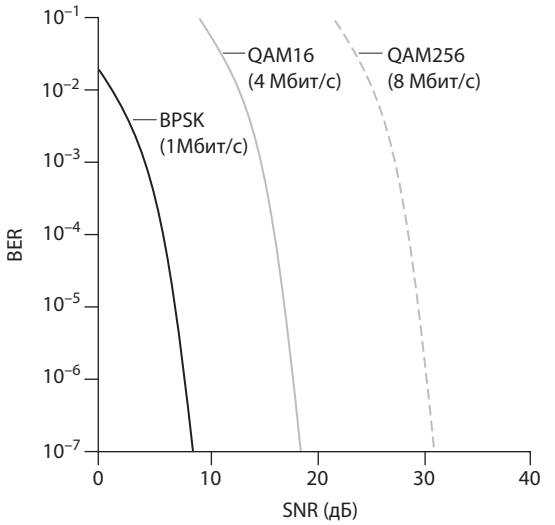


Рис. 6.3. Интенсивность битовых ошибок, скорость передачи данных и SNR

- *Динамический выбор метода модуляции сигнала на физическом уровне может быть использован, чтобы адаптировать технологию модуляции сигнала к условиям канала связи*. Из-за мобильности значение SNR (а значит, и BER) может измениться ввиду изменений в окружающей среде. Адаптивная модуляция и кодирование сигнала используются в системах сотовой связи, а также в сетях Wi-Fi 802.11 и сотовых сетях передачи данных 3G, которые мы изучим в разделах 6.3 и 6.4. Благодаря этому возможно, например, осуществить выбор

технологии модуляции, позволяющей обеспечить максимальную скорость передачи данных, когда значение показателя BER ограничено в заданных характеристиках канала связи.

Более высокий и меняющийся с течением времени коэффициент битовых ошибок является не единственным отличием беспроводного от проводного канала связи. Вспомним, что в случае с проводными каналами, все узлы сети могут получать передачи данных от всех прочих узлов. В случае с беспроводной сетью не все так просто, как показано на рис. 6.4. Предположим, что станция А передает сигнал станции Б, представим также, что и станция В передает сигнал станции А. Из-за существования так называемой **проблемы скрытых передатчиков** физические препятствия в окружающей среде (например, гора или здание) могут помешать станциям А и В слышать друг друга, несмотря на это сигналы обеих станций создают помехи на станции назначения, Б. Это отображено на рис. 6.4(а). Второй вариант развития событий, при котором на ресивере также возникают не обнаружимые конфликты, является результатом **затухания** сигнала в виду его прохождения по беспроводной среде. Рисунок 6.4(б) является иллюстрацией ситуации, в которой станции А и В расположены таким образом, что их сигналы не обладают достаточной силой для обнаружения друг друга, при этом их сигналы *достаточно* сильны, чтобы создавать друг для друга помехи на станции Б. Как мы увидим в разделе 6.3 из-за существования скрытой проблемы передачи и эффекта ослабления организация множественного доступа в случае с беспроводной сетью является гораздо более сложной задачей, нежели в случае с проводной.

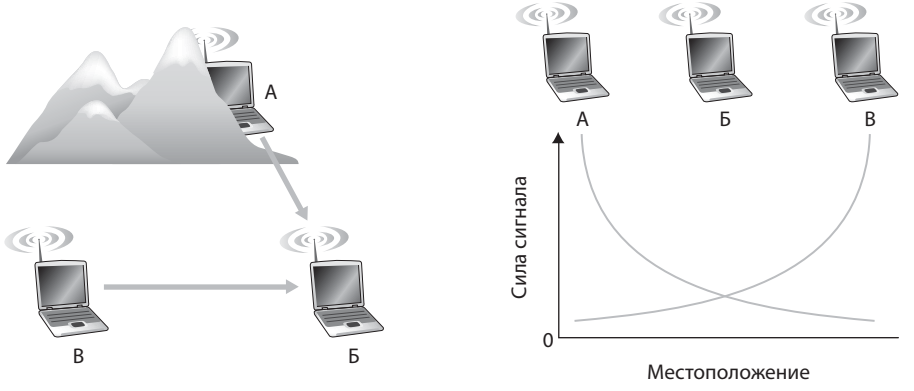


Рис. 6.4. Проблема скрытых передатчиков, вызванная препятствием (а) и ослаблением сигнала (б)

6.2.1. CDMA

Вспомним из текста главы 5, что, когда хосты связываются с использованием множественного доступа к среде передачи, чтобы предотвратить смешивание на ресиверах сигналов, посылаемых различными отправителями, требуется использование соответствующих протоколов. В главе 5 мы описывали три класса протоколов доступа к среде передачи данных: разделение каналов связи, произвольный доступ и очередность. Технология **множественного доступа с кодовым разделением** (code division multiple access, **CDMA**) относится к семейству протоколов разделения каналов связи. Так как технология CDMA очень важна в мире беспроводной связи, мы вкратце обсудим ее сейчас, прежде чем начнем изучать конкретные технологии беспроводного доступа в последующих главах.

Протокол CDMA подразумевает, что каждый бит данных отправляется и кодируется с помощью умножения этого бита на сигнал (код), изменяющий его при гораздо большей скорости (известной как **скорость передачи элементов сигнала**), чем исходная последовательность битов данных.

На рис. 6.5 показан простой, идеализированный сценарий кодировки/декодирования с использованием CDMA. Предположим, что скорость, с которой исходные биты данных достигают кодировщика CDMA определяет расчетную единицу времени, то есть для передачи каждого бита исходных данных требуется однобитный временной промежуток (слот). Допустим, что d_i — это значение бита данных для i-того битового слота. Для удобства математических расчетов, мы представим бит данных со значением 0 в качестве –1. Каждый битовый слот затем подразделяется на M мини-слотов. На рис. 6.5 $M = 8$, хотя на практике число M гораздо больше. Код CDMA, используемый отправителем, состоит из последовательности значений M, c_m, $m = 1,..., M$, каждое из которых принимает значение, либо +1, либо –1. M-битный CDMA код, используемый отправителем в примере на рис. 6.5: (1, 1, 1, –1, 1, –1, –1, –1).

Для иллюстрации того, как работает CDMA, давайте обратим внимание на i-й бит данных, d_i. Для m-того мини-слота промежутка времени битовой передачи бита d_i вывод кодировщика CDMA, $Z_{i,m}$ равняется значению d_i, помноженному на m-тый бит, назначенный к коду CDMA, c_m:

$$Z_{i,m} = d_i \times c_m \tag{6.1}$$

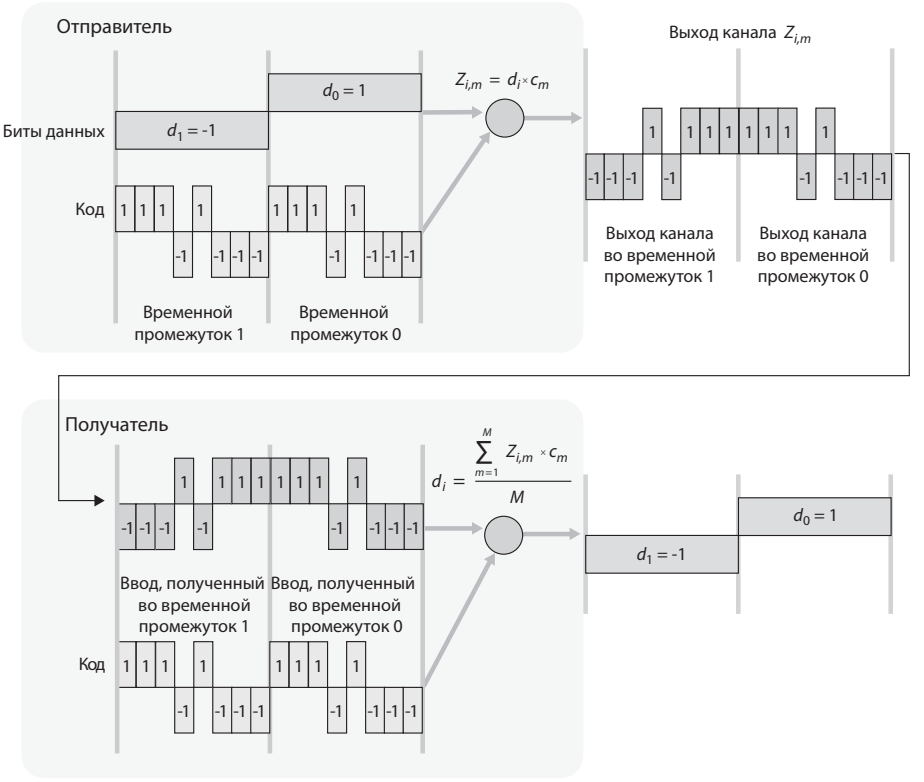


Рис. 6.5. Простой пример CDMA: отправитель кодирует, получатель декодирует

В простом мире, где не существует отправителей, создающих помехи друг для друга, ресивер получит кодированные биты $Z_{i,m}$ и восстановит исходный бит данных d_i, произведя расчет по формуле:

$$d_i = \frac{1}{M}\sum_{m=1}^{M} Z_{i,m} \times c_m \qquad (6.2)$$

Читатель может проработать пример на рис. 6.5, дабы убедиться в том, что исходные биты данных на самом деле корректно восстанавливаются с помощью (6.2).

Однако наш мир далек от идеала, а поэтому, как мы уже замечали ранее, протоколу CDMA приходится работать в условиях наличия других отправителей, создающих помехи, кодирующих и передающих свои данные, используя другие коды. Но, в таком случае, каким же образом может CDMA-ресивер восстановить исходные биты данных отправителя, если они смешиваются с теми, что посылают другие отправители? Протокол CDMA работает уже исходя из того, что к сигналу добавляют-

ся посторонние битовые сигналы. Это значит, что если в течение одного и того же временного мини-слота три отправителя посылают значение 1, а четвертый –1, то значение сигнала, полученного всеми ресиверами в этот мини-промежуток времени равняется 2 (так как 1 + 1 + 1 — 1 = 2). При наличии нескольких отправителей, отправитель *s* вычисляет кодирование передаваемых $Z^s_{i,m}$ точно так же, как в (6.1). Значение, полученное ресивером в течение *m*-го временного мини-промежутка *i*-го битового слота, однако, теперь, *сумма* всех переданных битов данных всех *N* отправителей в течение этого мини-промежутка равняется:

$$Z^*_{i,m} = \sum_{S=1}^{N} Z^s_{i,m}$$

Что удивительно, если коды отправителей подобраны тщательно, то ресивер может восстановить все данные, посланные отправителем, отделив их от агрегированного сигнала, просто воспользовавшись кодом отправителя, как в (6.2):

$$d_i = \frac{1}{M}\sum_{m=1}^{M} Z^*_{i,m} \times c_m \qquad (6.3)$$

как показано на рис. 6.6 для примера с двумя CDMA-отправителями. *M*-битовый CDMA код, использованный верхним отправителем (1, 1 , 1, –1, 1, –1, –1, –1), тогда как код, используемый нижним отправителем — (1, –1, 1, 1, 1, –1, 1, 1). На рис. 6.6 изображено, как ресивер восстанавливает исходные биты данных от верхнего отправителя. Обратите внимание, что ресивер может вычленить данные, посылаемые отправителем 1, несмотря на помехи от передачи данных отправителя 2.

Вспомните нашу аналогию с коктейлями из главы 5. Работа протокола CDMA напоминает присутствие завсегдатаев вечеринок, говорящих на разных языках. В таких ситуациях люди очень хорошо умеют образовывать «группы по интересам», состоящие из гостей, говорящих на одном языке, отфильтровывая при этом группы, в которых беседы ведутся на иностранных языках. Здесь мы видим, что CDMA — это разделяющий протокол в том смысле, что он разделяет на части кодовое пространство (codespace), а не время или диапазоны частот и выделяет каждому узлу сети часть этого пространства.

Краткость нашего обсуждения технологии CDMA в данном разделе вызвана необходимостью. На практике же существует еще большее количество сложностей, с которыми необходимо справляться. Во-первых, чтобы ресиверы сигналов CDMA могли правильно вычленить сигнал, посылаемый отправителем, необходимо очень тщательно подходить к выбору кодов CDMA. Во-вторых, в нашем обсуждении мы допусти-

ли, что силы сигналов, получаемых от различных отправителей, равны, в реальности этого крайне сложно достичь. Существует большое количество литературы по вопросам решения вышеуказанных и прочих проблем, связанных с протоколом CDMA, см. более подробную информацию в работах Пикхольтца[395] и Витерби[650].

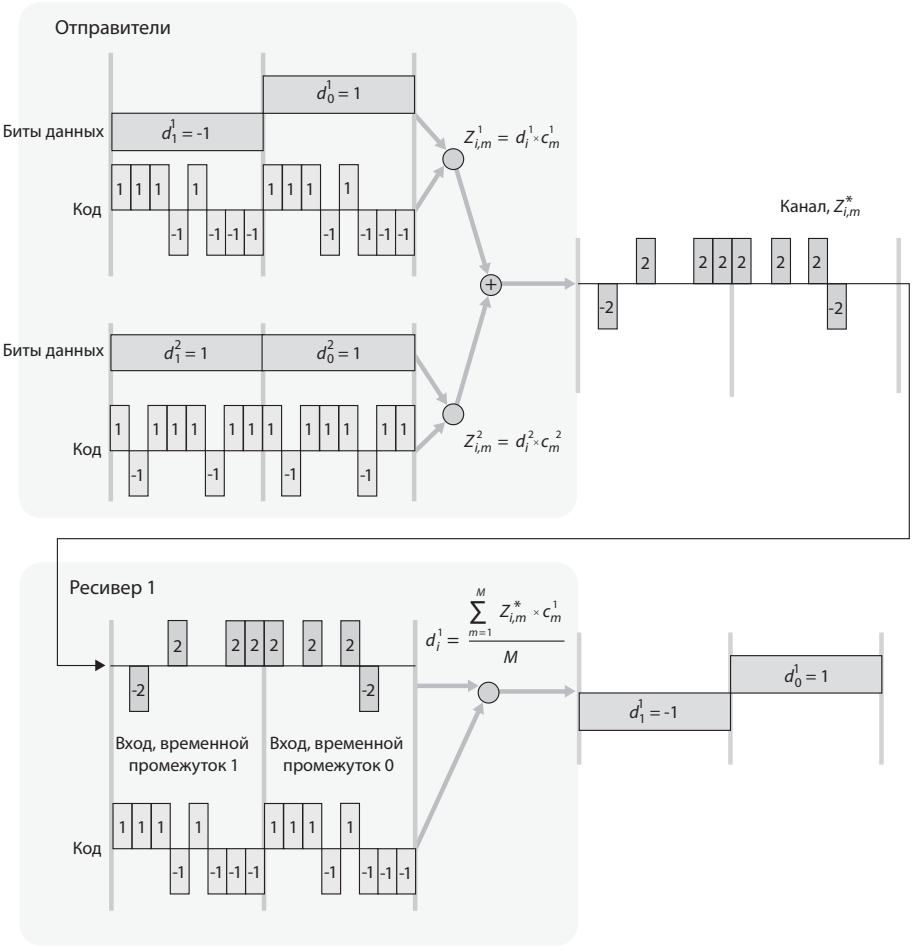


Рис. 6.6. Пример с двумя отправителями CDMA

6.3. Wi-Fi: Беспроводные локальные сети 802.11

Прочно обосновавшиеся на рабочих местах, в наших домах, образовательных учреждениях, кафе, аэропортах и уличных перекрестках,

беспроводные локальные сети превратились в одну из самых важных технологий доступа к Интернету. Несмотря на то, что в 1990-х годах велись разработки большого количества технологий и стандартов для беспроводных локальных сетей, победителем этого состязания стал лишь один класс стандартов беспроводных сетей: беспроводная локальная сеть **IEEE 802.11 wireless LAN**, или просто **Wi-Fi**. В этом разделе мы более подробно познакомимся с беспроводными локальными сетями 802.11, изучив структуру кадров, протокол доступа к среде передачи данных, а также организацию межсетевого взаимодействия локальных сетей 802.11 и проводных локальных сетей Ethernet.

Существует несколько стандартов беспроводных локальных сетей 802.11, в том числе 802.11b, 802.11a и 802.11g. В табл. 6.1 приводится краткое обобщение характеристик вышеперечисленных стандартов, впрочем, стандарт 802.11g по популярности далеко опережает своих собратьев. Кроме того, сегодня доступны устройства, работающие в двойном (802.11a/g) и даже тройном (802.11a/b/g) режимах.

Между всеми тремя стандартами семейства 802.11 есть очень много общего. Так, например, они используют одинаковый протокол доступа к среде передачи данных, CSMA/CA, о котором мы также поговорим уже в ближайшее время. Структуры кадров канального уровня всех трех стандартов также идентичны. Все три стандарта обладают возможностью уменьшать скорость передачи данных с целью достижения более дальних расстояний. Все три стандарта могут использоваться как в режиме «инфраструктуры», так и в режиме «децентрализованная одноранговая сеть». Однако, как отражено в табл. 6.1, между тремя стандартами существует несколько значительных отличий на физическом уровне.

Стандарт 802.11b обладает скоростью передачи данных 11 Мбит/с и работает в не лицензируемом диапазоне частот 2400,0–2483,5 МГц, разделяя его вместе с телефонами, работающими на частоте 2,4 ГГц, и микроволновыми печами. Скорость передачи данных беспроводной сети 802.11a значительно выше, однако, как и рабочий частотный диапазон. Работая в более высоком частотном диапазоне, локальные сети 802.11a имеют меньшую площадь зоны покрытия и больше страдают от эффекта многолучевого распространения волн. Сети типа 802.11g работают в том же низком диапазоне, что и сети 802.11b, и имеют с ними обратную совместимость, поэтому многие пользователи обновляют свои клиентские устройства 802.11b до стандарта 802.11g. Кроме того, стандарт 802.11g предоставляет более высокую скорость передачи данных, сопоставимую со скоростью стандарта 802.11a.

Табл. 6.1. Обзор характеристик стандартов семейства IEEE 802.11

Стандарт	Диапазон частот (Россия), МГц	Скорость передачи данных, Мбит/с
802.11b	2400–2483,5	до 11
802.11a	5150–5350	до 54
802.11g	2400–2483,5	до 54

Сравнительно новый стандарт сетей Wi-Fi, 802.11n[229], предполагает использование антенн множественного ввода и множественного вывода (multiple-input & multiple-output — MIMO). Это означает, что со стороны отправителя и получателя присутствует по две или более антенн, передающих или принимающих сигналы[133]. В зависимости от используемого типа модуляции сигнала, стандарт 802.11n позволяет достигнуть скорости передачи данных до нескольких сотен мегабит в секунду.

6.3.1. Архитектура сетей 802.11

На рис. 6.7 изображены основные компоненты архитектуры беспроводной локальной сети 802.11. Фундаментальным блоком архитектуры сетей 802.11 является **основной набор служб** (basic service set, **ОНС**). ОНС содержит одну или несколько беспроводных станций и центральную **базовую станцию**, называемую в данном случае **точкой доступа** (access point, **ТД**). На рис. 6.7 изображено, что две точки доступа ТД в каждом из основных наборов служб ОНС подключаются к некоему устройству сетевого взаимодействия, например коммутатор или маршрутизатор. В типичной домашней сети присутствует одна точка доступа и один маршрутизатор (обычно объединенные в одно устройство), которые подключают ОНС к Интернету.

Точно так же, как и в случае с устройствами Ethernet, каждой беспроводной станции 802.11 присваивается свой собственный 6-байтный адрес MAC, сохраняемый в прошивке адаптера станции (иными словами, в сетевую интерфейсную карту 802.11). Каждая точка доступа ТД обладает собственным MAC адресом беспроводного интерфейса. Как и в случае с технологией Ethernet, эти MAC адреса управляются организацией IEEE (Институт инженеров по электронике и электротехнике, США) и (в теории) являются уникальными.

Как было отмечено в разделе 6.1, беспроводные локальные сети, в которых используются точки доступа, часто называются **инфраструктурными беспроводными локальными сетями**, при этом под «инфраструк-

турой» понимаются как сами точки доступа ТД, так и инфраструктура Ethernet, осуществляющая их подключение к маршрутизатору.

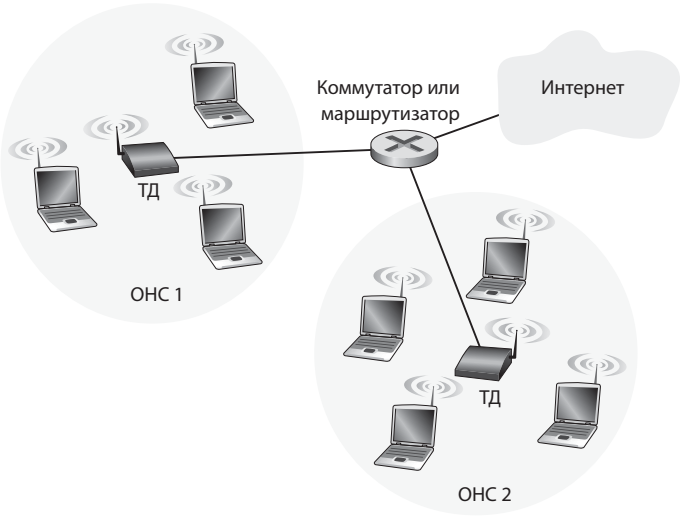


Рис. 6.7. Архитектура локальной сети IEEE 802.11

Как показано на рис. 8.1, станции IEEE 802.11 также могут организовываться в одноранговую сеть — сеть, в которой отсутствуют какое-либо централизованное управление и связи с «внешним миром». В данном случае сеть формируется «на лету» из мобильных устройств, находящихся в непосредственной близости, которым необходимо связаться друг с другом и которые не находятся в зоне действия уже существующей инфраструктурной сети.

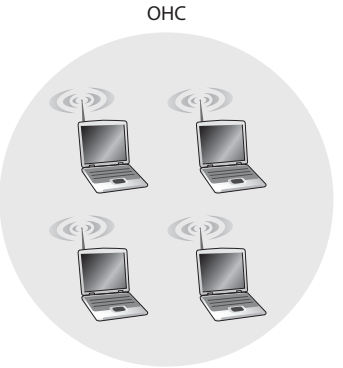


Рис. 6.8. Одноранговая сеть IEEE 802.11

Одноранговую сеть можно создать, если несколько владельцев ноутбуков собираются вместе (например, в конференц-зале, поезде или в машине) и желают обменяться данными в отсутствие централизованной точки доступа. К одноранговым сетям в настоящее время в мире проявляется огромнейший интерес в связи с постоянно увеличивающимся количеством портативных устройств, способных к сетевому взаимодействию. Однако в этом разделе мы сосредоточимся на изучении инфраструктурных беспроводных локальных сетей.

Каналы и ассоциации

В случае с сетью 802.11, прежде чем посылать или отправлять данные уровня сети, каждая из беспроводных сетей должна быть ассоциирована, или подключена, к какой-либо точке доступа. Несмотря на то что ассоциация требуется при использовании всех стандартов семейства 802.11, мы обсудим эту тему конкретно в контексте сети типа IEEE 802.11b/g.

При установке точки доступа администратор вычислительной сети присваивает ей **идентификатор набора служб** (service set identifier, **SSID**), состоящий, как правило, из одного-двух слов. (Так, например, при просмотре «доступных сетей» в операционной системе Microsoft Windows XP, вы увидите список идентификаторов SSID точек доступа, находящихся в зоне досягаемости). Администратор также должен присвоить точке доступа номер канала. Для того чтобы понять, что такое номер канала, вспомним, что сети 802.11 работают в диапазоне от 2400,0–2483,5 МГц. Так, в диапазоне 85 МГц, сеть 802.11 определяет 11 частично пересекающихся каналов. Любые два канала не пересекаются тогда и только тогда, когда они разделены четырьмя и более каналами. В частности набор трех каналов с номерами 1, 6 и 11 является единственным набором непересекающихся каналов. Это значит, что администратор мог бы создать беспроводную локальную сеть с общей максимальной скоростью передачи данных 33 Мбит/с, установив в одном и том же физическом местоположении три точки доступа 802.11, присвоив им каналы номер 1, 6 и 11 и соединив каждую из точек доступа с помощью коммутатора.

Теперь, имея общее представление о каналах сети 802.11, давайте опишем интересную (и вовсе не редкую) ситуацию, получившую название «джунгли Wi-Fi». **Джунглями Wi-Fi** называется любое физическое местоположение, где какая-либо беспроводная станция принимает достаточно сильный сигнал от двух и более точек доступа. Например, такая

ситуация сложилась во многих кафе города Нью-Йорк, где беспроводная станция может поймать сигнал от большого количества точек доступа, расположенных по соседству. Одна из точек может принадлежать кафе, в то время как другая — находиться в квартире, расположенной неподалеку от этого кафе. Скорее всего, все эти точки доступа находятся в разных IP-подсетях и каждой из них был присвоен сетевой канал вне зависимости друг от друга.

Теперь предположим, что вы заходите в такие «джунгли Wi-Fi» со своим портативным компьютером в поисках беспроводного Интернета и черничных кексов. Предположим также, что в этих джунглях находятся пять точек доступа. Чтобы получить доступ к Интернету, ваша беспроводная станция должна присоединиться к какой-то одной и только одной подсети и поэтому должна быть **ассоциирована** с одной и только одной из точек доступа. Ассоциация означает, что беспроводная станция создает как бы виртуальный провод, соединяющий ее с точкой доступа. Говоря конкретнее, только одна ассоциированная точка доступа будет отправлять вашей беспроводной станции кадры данных (то есть кадры, содержащие какую-то информацию, такие как дейтаграммы), а ваша беспроводная станция, в свою очередь, сможет отправлять кадры данных в Интернет только через ассоциированную с ней точку доступа. Но каким образом ваша беспроводная станция ассоциируется с какой-либо конкретной точкой доступа? И более фундаментальный вопрос: каким образом ваша беспроводная станция узнает, какие точки доступа присутствуют (если вообще таковые есть) в этих «джунглях»?

Стандарт 802.11 требует, чтобы точка доступа периодически посылала **сигнальные кадры данных**, каждый из которых должен содержать идентификатор SSID и MAC-адрес точки доступа. Ваша беспроводная станция, зная о том, что точки доступа передают сигнальные кадры, просто сканирует все 11 каналов в поисках сигнальных кадров от какой-либо точки доступа, которая может находиться в зоне досягаемости (некоторые из точек доступа могут передавать свои сигналы по одному и тому же каналу данных — не забывайте, вы в джунглях!). Получив с помощью сигнальных кадров данных информацию о ближайших точках доступа, вы (или ваш беспроводной хост) выбираете одну из них и производите ассоциацию.

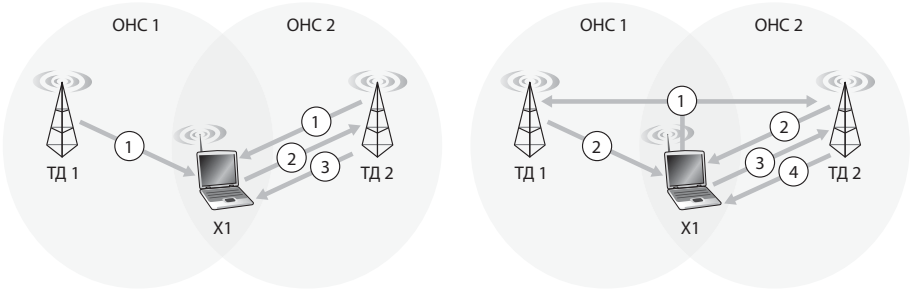
В стандарте 802.11 не определяется алгоритм, по которому производится выбор одной из доступных точек доступа. Такой алгоритм оставлен на усмотрение разработчиков программного обеспечения и прошивок для беспроводных хостов. Обычно хост выбирает точку доступа

с наибольшей силой сигнала. Однако несмотря на то, что сила сигнала может быть хорошей (см. пример из раздела 6.3), это не единственная характеристика, определяющая насколько успешно хост будет принимать сигнал. В частности, точка доступа может иметь сильный сигнал, но она оказаться перегруженной подключенными хостами (которым потребуется разделить между собой пропускную способность канала точки доступа), в то время как незагруженная точка доступа останется непопулярной из-за более слабого сигнала. Поэтому было предложено большое количество альтернативных способов выбирать точку доступа[645, 367, 627]. Для ознакомления с интересным и приближенным к практике обсуждением измерения силы сигнала см. работу Барвелла[40].

Процесс сканирования и прослушивания каналов с целью поймать сигнальные кадры данных известен как **пассивное сканирование** (см. рис. 6.9(а)). Беспроводной хост (на рисунке X1) также может осуществить **активное сканирование**, передавая пробный кадр данных, который будет получен всеми точками доступа, находящимися в зоне досягаемости хоста, как показано на рис. 6.9(б). Точка доступа отвечает на пробный запрос с помощью кадра данных пробного ответа. После этого беспроводной доступ уже может выбрать из ответивших на пробный запрос точку доступа, к которой подключится.

После выбора точки доступа беспроводной хост посылает ей запрос на ассоциацию, а та, в свою очередь, направляет кадр данных с ответом на запрос. Обратите внимание, что это второе «рукопожатие» запрос/ответ при активном сканировании является необходимостью, так как точка доступа, ответившая на первый пробный запрос не знает, которую из (возможно, многих) ответивших точек доступа выберет хост для последующей ассоциации. Этот процесс похож на то, как клиент DHCP осуществляет выбор среди множества серверов DHCP (см. рис. 4.21). После завершения процесса ассоциации с точкой доступа, хост попробует подключиться к подсети (с точки зрения IP адресации, которую мы обсуждали в разделе 4.4.2), к которой уже подключена точка доступа. Поэтому хост, как правило, будет с помощью точки доступа посылать в подсеть сообщение обнаружения DHCP (см. рис. 4.21), чтобы получить от подсети IP-адреса. По получении адреса хост превращается в еще один из хостов с IP-адресом подсети.

Чтобы ассоциироваться с конкретной точкой доступа беспроводной станции, возможно, потребуется произвести аутентификацию. Беспроводные локальные сети 802.11 предлагают большое количество альтернатив для аутентификации и получения доступа.



а. Пассивное сканирование
1. Сигнальные кадры, отправляемые точками доступа
2. Кадр Запроса ассоциации отправлен: X1 к выбранной точке доступа
3. Кадр Ответа об ассоциации отправлен: выбранная точка доступа к X1

б. Активное сканирование
1. Кадр Пробного Запроса передан от X1 (широковещательно)
2. Кадры Пробного Ответа отправлен точками доступа
3. Кадр Запроса ассоциации отправлен: X1 к выбранной точке доступа
4. Кадр Ответа об ассоциации отправлен: выбранная точка доступа к X1

Рис. 6.9. Активное и пассивное сканирование с целью поиска точек доступа

Один из подходов, используемых многими компаниями, состоит в передаче прав доступа беспроводной базовой станции в зависимости от ее MAC адреса. Второй подход, используемый многими Интернет-кафе, предполагает использование имен пользователей и паролей. В обоих случаях точка доступа чаще всего связывается с сервером аутентификации, передавая информацию между беспроводной станцией конечного пользователя и сервером аутентификации с помощью особого протокола, например RADIUS[487] или DIAMETER[520]. Отделение сервера аутентификации от точки доступа позволяет использовать один сервер для обслуживания сразу нескольких точек доступа, а также централизовать зачастую очень важные решения об аутентификации и предоставлении доступа на одном сервере, что позволяет снизить затраты на содержание точек доступа, а также упростить их организацию. В разделе 8.8 мы увидим, что в новом протоколе IEEE 802.11i, определяющем аспекты безопасности протокола 802.11, используется именно этот подход.

6.3.2. Протокол 802.11. MAC

После того как базовая станция ассоциирована с точкой доступа, она может наладить отправку и прием кадров данных базовой станции. Однако, так как несколько станций могут одновременно попытаться отправить кадры данных по одному каналу, чтобы упорядочить передачу данных, потребуется использование протокола множественного доступа. В данном

контексте под **станцией** понимается либо беспроводная станция, либо точка доступа. Как мы уже обсуждали ранее в главе 5 и разделе 6.2.1, если говорить в общем, то существует три класса протоколов множественного доступа, предполагающие либо разделение каналов связи (в том числе, CDMA), произвольный доступ, либо очередность. Вдохновленные огромным успехом технологии Ethernet, использующей протокол произвольного доступа, разработчики технологии 802.11 решили также использовать протокол произвольного доступа для беспроводных локальных сетей 802.11. Протокол произвольного доступа часто называют **CSMA с исключением столкновений**, или более кратко, **CSMA/CA***. Как и в случае с протоколом CSMA/CD, использующемся в технологии Ethernet, аббревиатура CSMA расшифровывается как «carrier sense multiple access», по-русски это звучит как «множественный доступ с контролем несущей» — то есть каждая станция анализирует состояние канала перед началом передачи данных, если канал занят, то станция воздерживается от передачи данных. Несмотря на то, что в обеих технологиях, Ethernet и 802.11, используется произвольный доступ с контролем несущей, у двух протоколов MAC есть существенные различия.

Во-первых, вместо обнаружения столкновений (collision detection), в технологии 802.11 применяется исключение столкновений (collision avoidance). Во-вторых, из-за относительно большего количества битовых ошибок в беспроводных каналах связи, в технологии 802.11 (в отличие от Ethernet) используется схема подтверждения/повторной передачи (ARQ) канального уровня. Далее мы опишем исключение столкновений и схемы подтверждения канального уровня технологии 802.11.

Давайте вспомним из разделов 5.3.2 и 5.4.2, что при использовании технологии Ethernet и алгоритма обнаружения столкновений станция Ethernet анализирует параметры передачи канала связи. Если в ходе передачи данных такая станция обнаружит, что в тот же самый момент другая станция отправляет данные, то наша станция прервет передачу и попробует произвести ее вновь через некоторый случайный промежуток времени. В отличие от протокола Ethernet 802.3, протокол MAC 802.11 *не* использует алгоритмы обнаружения столкновений. И на это есть две важные причины:

- Способность обнаруживать столкновения требует возможности одновременно посылать (собственные сигналы станции) и принимать (с целью определения того, передает ли другая станция сигналы).

* CA от англ. *collision avoidance* — исключение столкновений. — *Примеч. пер.*

Так как принимаемый сигнал, как правило, значительно слабее передаваемого, создание аппаратного обеспечения, которое могло бы обнаруживать столкновения, слишком затратно.

- Однако более важным является то, что даже если бы адаптер мог одновременно передавать сигналы и прослушивать канал связи (и, соответственно, отменять передачу данных, если прослушиваемый канал оказывался бы занятым), то такой адаптер все равно не смог бы обнаружить все столкновения из-за проблемы скрытых передатчиков данных и эффекта ослабления сигнала, которые мы обсуждали в разделе 6.2.

Так как в беспроводных локальных сетях 802.11 не используется технология обнаружения столкновений, станция всегда *передает кадр данных полностью*. Иными словами, после запуска станции обратного пути уже нет. Как многие могут подумать, передача кадров данных целиком (в особенности, длинных кадров) при большой вероятности возникновения столкновений может значительно ухудшить производительность протокола множественного доступа. Для уменьшения вероятности столкновения, в сетях 802.11 используется несколько технологий, которые мы обсудим в ближайшее время.

Однако прежде чем обсуждать избежание столкновений, мы изучим **схему подтверждения канального уровня** сетей 802.11. Давайте вспомним из содержания раздела 6.2, что, когда станция, подключенная к беспроводной локальной сети, посылает кадр данных, этот кадр может не достичь места назначения по нескольким причинам. Для того чтобы справиться с неуменьшающейся вероятностью неудачной отправки кадра данных, протокол MAC 802.11 использует подтверждение канального уровня. Как показано на рис. 6.10, когда станция назначения принимает сигнал, проходящий контроль с использованием кода циклического контроля (CRC), она ожидает небольшой период времени, известный как **короткий межкадровый промежуток** (Short Inter-Frame Spacing, **SIFS**), после чего посылает обратно кадр данных с подтверждением. Если передающая станция не получает подтверждение в течение определенного времени, делается предположение, что произошла какая-то ошибка, и станция повторно посылает кадр данных, используя для доступа к каналу связи протокол CSMA/CA. Если после нескольких попыток повторной отправки данных станция не получает подтверждения, то попытки прекращаются и кадр данных уничтожается.

Обсудив, как в сетях 802.11 используется подтверждение канального уровня, мы теперь готовы к описанию протокола 802.11 CSMA/CA.

Предположим, что станции (беспроводной станции или точке доступа) требуется передать кадр данных.

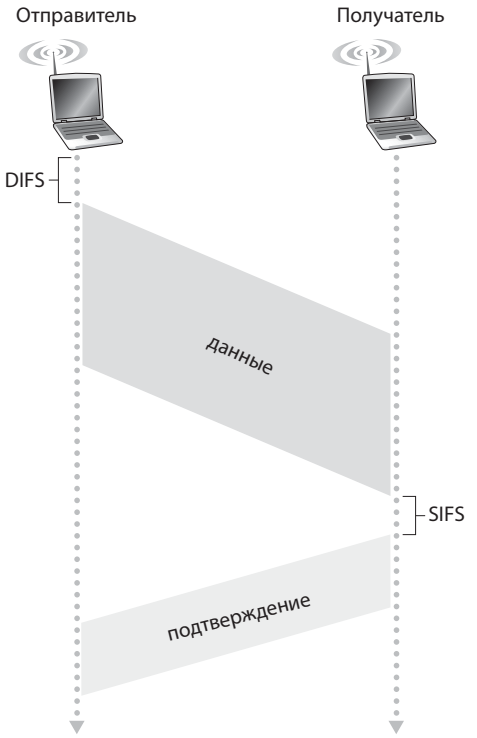


Рис. 6.10. Использование подтверждения канального уровня в сетях 802.11

1. Если в результате первоначального прослушивания канала станция определит, что он является свободным, то отправка кадра данных осуществляется через короткий промежуток времени, известный как **распределенный межкадровый промежуток** (Distributed Interframe Space, **DIFS**).

2. В противном случае станция случайным образом устанавливает временной интервал для отсрочки передачи кадра данных, используя бинарный экспоненциальный алгоритм (который мы уже обсуждали в разделе 5.3.2), и начинает отсчитывать полученное значение, когда канал определяется свободным. Пока канал остается занятым, отсчет не ведется.

3. Когда отсчет достигает нуля (обратите внимание, что это может случиться, только если канал будет определяться как свободный),

станция передает кадр данных целиком — и переходит в режим ожидания подтверждения.

4. Если подтверждение получено, то передающая станция «знает», что переданный ею кадр данных был корректно получен станцией-получателем. Если нужно послать еще один кадр данных, то его отправка осуществляется при помощи протокола CSMA/CA, начиная с шага 2. Если подтверждение получено не было, то станция возвращается в режим отсрочки передачи данных (шаг 2) и вновь выбирает случайное значение временного промежутка, но уже из большего интервала.

Вспомним, что при использовании протокола множественного доступа CSMA/CD технологии Ethernet (раздел 5.3.2) станция начинает передачу сигнала, как только канал будет найден свободным. Однако при использовании протокола CSMA/CA станция воздерживается от отправки данных, пока таймер отсчета не достигнет нуля, даже если канал был определен как свободный. Почему в данном случае в протоколах CSMA/CD и CMSA/CA используются такие разные подходы?

Для того чтобы ответить на данный вопрос, давайте представим ситуацию, когда две станции должны передать кадр данных, но ни она не приступает к передаче, так как они определяют, что ее уже осуществляет третья станция. При использовании протокола CSMA/CD технологии Ethernet обе станции начали бы передачу данных сразу же после того, как обнаружили бы, что третья ее завершила. Это привело бы к столкновению, что для протокола CSMA/CD не является серьезной проблемой, так как обе станции во избежание бесполезных усилий по передаче остатков кадров данных тут же прервали бы отправку. Однако в случае с сетями 802.11 ситуация несколько отличается. Так как технология 802.11 не предполагает обнаружение столкновений и прекращение отправки данных, кадр данных, пострадавший от столкновения, передается полностью. Поэтому целью технологии 802.11 является избежание столкновений во всех случаях, когда это является возможным. Если обе станции, работающие в сети 802.11, обнаруживают, что канал связи занят, они входят в режим отсрочки отправки данных, выбирая при этом в идеале разные значения временного интервала. Если значения на самом деле различаются, то одна из станций начнет передачу данных в освободившийся канал раньше другой. Кроме того, если станции не скрыты друг от друга, то «проигравшая» поймает сигнал «станции-победительницы», заморозит счетчик и будет воздерживаться от передачи данных до тех пор, пока ее не завершит станция-победительница.

Таким образом удается избежать дорогостоящих столкновений. Однако столкновения в сетях 802.11 все же иногда происходят. Так, две станции могут быть скрыты друг от друга, либо обе станции могут выбрать случайные значения временного интервала отправки данных, которые находятся слишком близко, и сигнал первой станции не успевает достичь места назначения. Вспомните, что ранее, при обсуждении алгоритмов случайного доступа, мы уже сталкивались с этой проблемой в контексте рис. 5.12.

Работа со скрытыми передатчиками: RTS и CTS

Протокол MAC сетей 802.11 также включает изящную (хотя и необязательную) схему резервирования, позволяющую избежать столкновений даже при наличии в сети скрытых передатчиков. Давайте исследуем эту схему в контексте рис. 6.11, изображающего две беспроводные станции и одну точку доступа. Обе беспроводные станции находятся в зоне покрытия (обозначена на рис. более темным кругом) точки доступа, обе станции также ассоциированы с данной точкой доступа. Однако из-за эффекта ослабления сигнала, территория покрытия беспроводных станций ограничена площадью более светлых кругов на рис. 6.11, вследствие чего обе станции остаются скрытыми друг от друга, несмотря на то, что не скрыты для точки доступа.

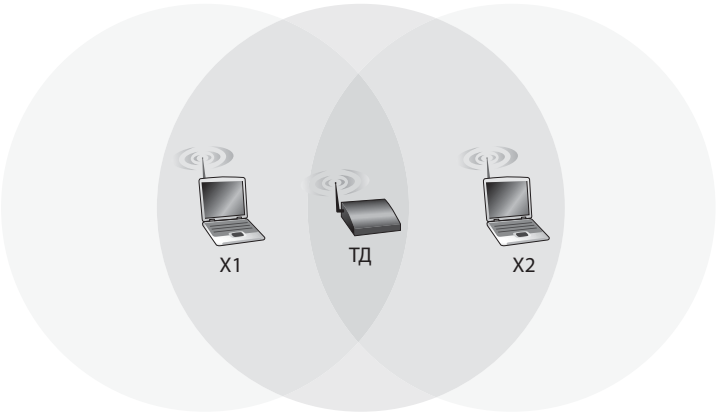


Рис. 6.11. Пример скрытых передатчиков: станция X1 скрыта от X2 и наоборот

Теперь попытаемся понять, почему скрытые передатчики могут представлять собой проблему. Представьте, что станция X1 передает кадр данных, но в это же самое время станция X2 также готова начать переда-

чу кадра данных точке доступа ТД. Станция X2, не улавливая сигналов станции X1, сначала выждет промежуток времени DIFS, а затем начнет передачу кадра данных, что приведет к столкновению. В результате применение канала связи окажется бесполезным на протяжении передачи кадров данных, как от станции X1, так и от X2.

Во избежание возникновения этой проблемы протокол IEEE 802.11 позволяет станциям воспользоваться коротким управляющим кадром **запроса отправки (RTS)** и коротким управляющим кадром **разрешения на отправку (CTS)** для *резервирования* доступа к каналу связи. Когда передатчик хочет отправить кадр DATA, он сначала должен переслать точке доступа кадр RTS, обозначающий общее количество времени, необходимое для отправки кадра DATA и кадра подтверждения (ACK). Точка доступа, получив кадр RTS, отвечает передачей кадра CTS. Кадры CTS имеют две цели: они дают отправителю исключительное разрешение на отправку данных, а также запрещают другим базовым станциям начинать передачу в течение зарезервированного промежутка времени.

Поэтому, как показано на рис. 6.12, перед передачей кадра DATA, станция X1 пересылает кадр RTS, который улавливается всеми станциями, находящимися в круге, в том числе точкой доступа ТД. Поле чего точка доступа отвечает отправкой кадра CTS, который также улавливается всеми базовыми станциями, находящимися в зоне покрытия точки доступа, в том числе станциями X1 и X2. Станция X2, поймав кадр CTS, воздерживается от отправки данных в течение промежутка времени, указанного в кадре CTS. Кадры RTS, CTS, ACK и DATA изображены на рис. 6.12.

Использование кадров RTS и CTS позволяет улучшить производительность двумя путями:

- Сглаживание проблемы скрытых передатчиков, так как длинные кадры DATA передаются только после резервирования канала связи.
- Так как кадры RTS и CTS являются короткими, их столкновение продлится только на протяжении промежутка времени, занимаемого этими короткими кадрами. После успешной передачи кадров RTS и CTS проблем с передачей кадров DATA и ACK не возникает.

Мы рекомендуем вам протестировать апплеты на веб-сайтах **tinyurl.com/25nyjp9** и **tinyurl.com/48mkj**. Эти апплеты иллюстрируют работу протокола CSMA/CA, в том числе порядок обмена кадрами RTS/CTS.

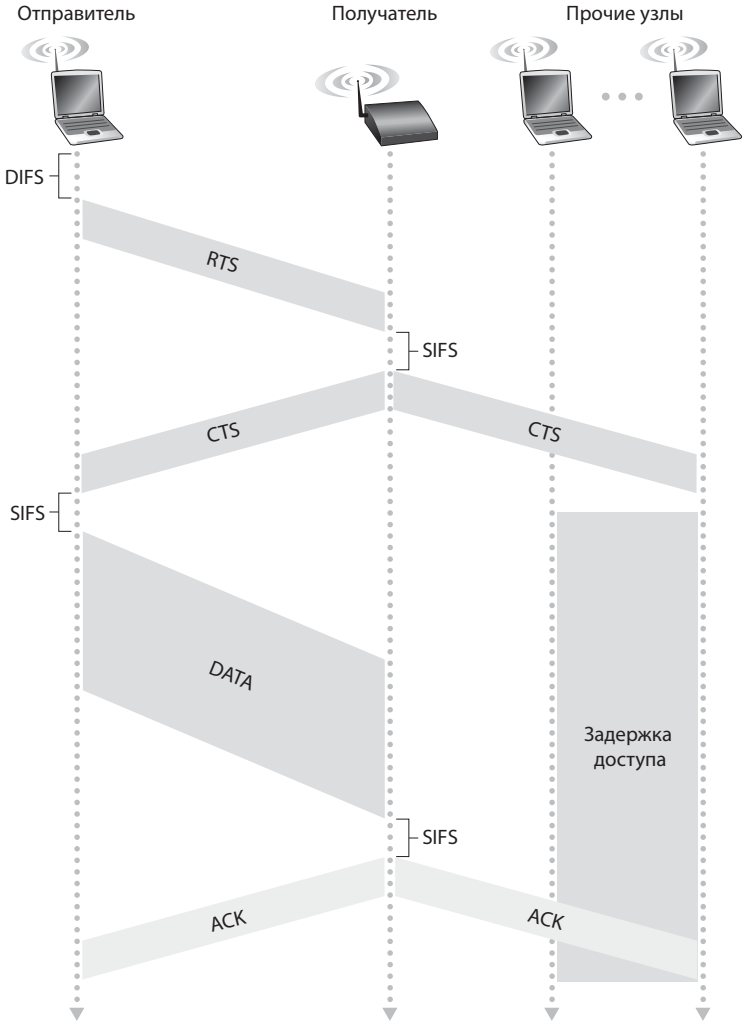


Рис. 6.12. Избежание столкновений с помощью кадров RTS и CTS

Несмотря на то что обмен кадрами RTS/CTS позволяет уменьшить количество столкновений, этот обмен также привносит определенные задержки и потребляет ресурсы канала связи. По этой причине обмен кадрами RTS/CTS используется (если используется), только чтобы зарезервировать канал связи для передачи длинного кадра DATA. На практике беспроводные станции часто устанавливают лимиты длительности (пороговые значения) RTS, с тем чтобы использовать технологию обмена кадрами RTS/CTS, только если длина кадра превышает установленный лимит. В настройках большого числа беспроводных

станций значение RTS по умолчанию превышает максимальную длину кадра, таким образом, обмен кадрами RTS/CTS пропускается для всех отправляемых кадров DATA.

Использование сетей 802.11 для организации двухточечной связи

До настоящего момента мы сосредоточились на обсуждении использования сети 802.11 для множественного доступа. Следует отметить, что если два узла оборудованы направленными антеннами, то эти антенны можно использовать для работы с протоколом 802.11 чтобы создать, по сути, двухточечное соединение. Учитывая низкую стоимость аппаратного обеспечения сетей 802.11 общего назначения, использование направленных антенн и увеличенная мощность передачи позволяют использовать сети 802.11 в качестве недорогого способа обеспечить двухточечное соединение на расстоянии в десятки километров. В работе Рамана[407] описывается такая многопереходная беспроводная сеть, использующаяся на сельскохозяйственных равнинах бассейна реки Ганг в Индии и состоящая из двухточечных соединений по технологии 802.11.

6.3.3. Кадр IEEE 802.11

Несмотря на то, что между кадрами сетей 802.11 и Ethernet очень много общего, кадры сетей 802.11 также содержат определенный набор полей, специфичных для применяемых в этой технологии беспроводных каналов связи. Кадр сети 802.11 изображен на рис. 6.13. Числа над каждым из полей кадра представляют собой длину этих полей в *байтах*, тогда как числа над каждым подполем контрольного поля кадра представляют длину соответствующих подполей в *битах*. Давайте теперь перейдем к изучению полей кадра и нескольких наиболее важных подполей контрольного поля кадра.

Рис. 6.13. Кадр 802.11

Полезное содержимое и код циклического контроля

Сердцем кадра данных является полезное содержимое, которое, как правило, состоит из дейтаграммы IP или пакета ARP. Несмотря на то что максимально допустимая длина поля «Полезное содержимое» составляет 2312 байт, как правило, при передаче дейтаграммы IP или пакета ARP она не превышает 1500 байт. Как и кадр сетевой технологии Ethernet, кадр сети 802.11 также содержит 32-битный алгоритм проверки с использованием кода циклического контроля (CRC), благодаря которому получатель сможет обнаружить битовые ошибки в переданном кадре. Как мы уже убедились, битовые ошибки в беспроводных локальных сетях являются гораздо более частым явлением, чем в проводных сетях, поэтому алгоритм CRC здесь очень полезен.

Поля адреса

Возможно, самое потрясающее отличие кадра 802.11 заключается в том, что он содержит *четыре* поля адреса, в каждом из которых, в свою очередь, содержатся 6-байтные MAC адреса. Но зачем нужно четыре поля адреса? Неужели поля MAC адреса отправителя и поля MAC адреса получателя недостаточно? Ведь при использовании технологии Ethernet этого вполне хватает? Оказывается, что три поля адреса необходимы для целей межсетевой совместимости, а именно для передачи дейтаграммы уровня сети от беспроводной станции через точку доступа на интерфейс маршрутизатора. Четвертое поле необходимо, когда точки доступа пересылают кадры данных в режиме одноранговогo соединения. Так как мы фокусируем свое внимание только на сетях с инфраструктурой, давайте обсудим три первых поля адреса. В стандарте 802.11 они определены следующим образом:

- Адрес 2 — это MAC адрес станции, передающей кадр данных. Поэтому, если какая-либо беспроводная станция инициирует передачу данных, то ее MAC адрес заносится в поле адреса 2. Аналогично, если точка доступа выступает передатчиком данных, то в поле адреса 2 записывается MAC адрес этой точки доступа
- Адрес 1 — это MAC адрес станции, которой предназначается передаваемый кадр данных. Поэтому, если беспроводная станция передает кадр, то поле адреса 1 будет содержать MAC адрес точки доступа-получателя. Аналогично, если точка доступа выступает передатчиком данных, то в поле адреса 1 записывается MAC адрес беспроводной станции-получателя.

- Для понимания содержания адреса 3, давайте вспомним такое понятие, как основной набор служб (ОНС), состоящий из точек доступа и беспроводных станций и являющийся частью подсети, а также и то, что данная подсеть соединена с другими подсетями посредством некоторого маршрутизатора. В поле адреса 3 содержится MAC адрес этого маршрутизатора.

Чтобы лучше понять, зачем нужно поле адреса 3, давайте изучим пример межсетевого взаимодействия, изображенного на рис. 6.14. На этом рисунке вы можете видеть две точки доступа ТД, каждая из которых отвечает за несколько беспроводных станций. Каждая из точек доступа имеет прямое подключение к маршрутизатору, который, в свою очередь, соединяется с Интернетом. Мы должны понимать, что точка доступа является устройством сетевого уровня, и поэтому не может ни пользоваться IP, ни работать с IP-адресами. Теперь давайте предположим, что нам необходимо переслать дейтаграмму от интерфейса маршрутизатора M1 к беспроводной станции X1. Маршрутизатору неизвестно, что между ним и станцией X1 присутствует также Точка доступа ТД. Если смотреть с точки зрения маршрутизатора, X1 — это всего лишь хост в одной из подсетей, подключенный к нему (маршрутизатору).

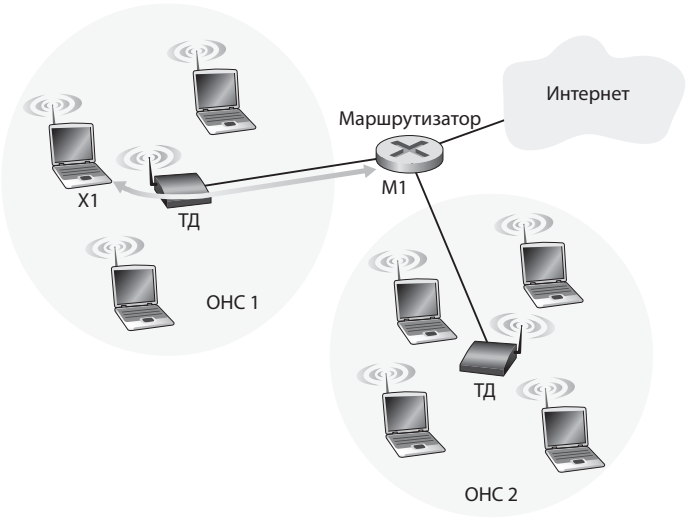


Рис. 6.14. Использование полей адреса в кадрах данных 802.11: пересылка кадров между X1 и M1

- Маршрутизатор, которому известен IP-адрес станции X1 (благодаря адресу места назначения дейтаграммы), использует протокол ARP

для определения MAC-адреса X1 также, как если бы мы имели дело с обычной локальной сетью Ethernet. После получения MAC-адреса X1 интерфейс маршрутизатора M1 инкапсулирует дейтаграмму в кадре Ethernet. Поле адреса отправителя данного кадра содержит MAC-адрес интерфейса M1, а поле адреса получателя — MAC-адрес станции X1.

- Когда кадр данных Ethernet 802.3 поступает на точку доступа ТД, эта точка доступа преобразует его в кадр 802.11 перед передачей по беспроводному каналу связи. Точка доступа заполняет поля адреса 1 и 2 MAC-адресом станции X1 и собственным MAC адресом соответственно, как было описано выше. В поле адреса 3 точка доступа записывает адрес интерфейса маршрутизатора M1. Таким же образом, с помощью поля адреса 3, станция X1 может определить MAC-адрес маршрутизатора, направившего дейтаграмму в подсеть.

Теперь давайте представим, что происходит, когда беспроводная станция X1 отвечает, перемещая дейтаграмму от X1 к интерфейсу M1.

- Станция X1 создает кадр 802.11, заполняя поля адреса 1 и 2 MAC адресом точки доступа и собственным MAC адресом соответственно, как описано выше. В поле адреса 3 записывается адрес интерфейса маршрутизатора M1.
- Когда точка доступа получает кадр 802.11, она преобразует его в кадр Ethernet. Поле адреса отправителя содержит MAC адрес станции X1, а поле адреса получателя — MAC адрес интерфейса M1. Таким образом, поле адреса 3 позволяет точке доступа определить MAC адрес места назначения при создании кадра Ethernet.

В заключение следует сказать, что поле адреса 3 играет очень важную роль в организации межсетевого взаимодействия основного набора служб ОНС и проводной локальной сети.

Поля порядкового номера, длительности и контроля кадра

Давайте вспомним, что в случае с технологией 802.11, при успешном получении кадра данных, станция всегда посылает отправителю подтверждение. Из-за того, что подтверждение способно затеряться, станция может отправить несколько копий кадра подтверждения. Как мы уже наблюдали по ходу обсуждения протокола `rdt2.1` (раздел 3.4.1), использование порядковых номеров позволяет получателю отличить только что переданный кадр данных от повторно отправляемых копий

подтверждения. Поле порядкового номера кадра 802.11, таким образом, служит на канальном уровне тем же целям, что на транспортном уровне, речь о котором шла в главе 3.

Вспомним также, что протокол 802.11 позволяет передающим станциям зарезервировать канал связи на определенный промежуток, включающий в себя время для передачи кадра данных и время для передачи подтверждения. Значение длительности этого промежутка записывается в соответствующее поле кадра (как данных, так и кадров RTS и CTS).

Как показано на рис. 6.13, поле контроля кадра состоит из большого числа подполей. Мы лишь вкратце обсудим несколько самых важных из них, для получения более подробной информации, мы рекомендуем вам обратиться к спецификации 802.11[204, 116, 228]. Поля *тип* и *подтип* используются для различия кадров ассоциации, RTS, CTS, ACK и данных. Поля *к* и *от* определяют значения различных полей адреса (значения полей изменяются, в зависимости от того, используется ли режим инфраструктуры или одноранговой сети, а также в случае с режимом инфраструктуры, в зависимости от того, какой элемент сети, беспроводная станция или точка доступа посылает кадр). Наконец, поле WEP служит для указания того, применяется ли шифрование. (Шифрование WEP обсуждается в главе 8.)

6.3.4. Мобильность в рамках единой IP-подсети

С целью увеличения физической площади покрытия беспроводной локальной сети, организации и университеты зачастую развертывают несколько основных наборов служб ОНС внутри единой IP-подсети. Из-за чего естественным образом возникает вопрос мобильности между ОНС: каким образом беспроводной станции осуществить переход от одного ОНС к другому, не прекращая текущего сеанса TCP? Как мы увидим в одном из следующих подразделов, вопрос мобильности решается достаточно просто, если ОНС является частью подсети. При перемещении станций между подсетями потребуется более сложный протокол управления мобильностью, например, один из тех, речь о которых пойдет в разделах 6.5 и 6.6.

Сейчас давайте разберем конкретный пример мобильности между ОНС в рамках одной подсети. На рис. 6.15 изображено два соединенных между собой основных набора служб ОНС, а также хост X1, совершающий перемещение от ОНС1 к ОНС2. Так как в данном примере устрой-

ство, соединяющее два набора служб, *не является* маршрутизатором, все станции в обоих ОНС принадлежат к единой IP-подсети. Поэтому, когда станция X1 перемещается от ОНС1 к ОНС2, она может сохранять присвоенный ей IP-адрес и, соответственно, все установленные соединения TCP. Если бы соединяющим устройством был маршрутизатор, то станции X1 приходилось бы получать новый IP-адрес всякий раз при перемещении в новую подсеть. Такое изменение адреса нарушит (и, естественно, прекратит) все установленные станцией X1 соединения TCP. В разделе 6.6 мы увидим, как для избегания подобной проблемы можно воспользоваться протоколом уровня сети, таким как мобильный IP-протокол.

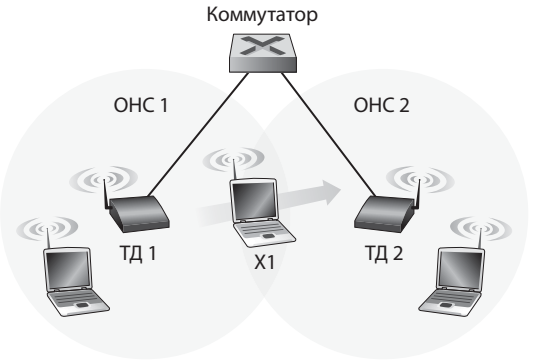


Рис. 6.15. Мобильность в рамках единой подсети

Но что же на самом деле происходит, когда станция X1 перемещается от ОНС1 к ОНС2? Как только X1 удаляется от точки доступа ТД1 и определяет ослабление сигнала ТД1, она начинает поиск более сильного сигнала. X1 получает сигнальные кадры от ТД2 (которая, в большинстве корпоративных и университетских сетях, будет иметь тот же сетевой идентификатор SSID, что и ТД1). После этого X1 прекращает ассоциацию с ТД1 и ассоциируется с ТД2, при этом сохраняя присвоенный ей IP-адрес и установленные подключения TCP.

Вышеописанное позволяет справиться с проблемой перехода, с точки зрения как хоста, так и точки доступа. Но что насчет коммутатора на рис. 6.15? Каким образом он «узнает», что хост переместился от одной точки доступа к другой? Как вы наверняка помните из главы 5, коммутаторы обладают способностью к самообучению — и самостоятельного создания таблиц переадресации. Функция самообучения позволяет справляться со случайными перемещениями (например, если какого-

либо работника переводят из одного отдела в другой). Однако коммутаторы не создавались для поддержки высокомобильных пользователей, желающих сохранять подключения TCP, перемещаясь между основными наборами служб ОНС. Для решения этой задачи вспомним, что перед осуществлением перемещения в таблице переадресации коммутатора есть запись, позволяющая соотнести MAC-адрес станции X1 с исходящим интерфейсом коммутатора, с помощью которого можно получить доступ к X1. Если станция X1 изначально находилась в ОНС1, то дейтаграмма, предназначенная этой станции, будет направлена ей через точку доступа ТД1. Однако как только станция X1 ассоциируется с основным набором служб ОНС2, все предназначенные ей кадры данных должны направляться через ТД2. Одним из решений (впрочем, оно немного хакерское) является отправка точкой доступа ТД2 широковещательного кадра Ethernet с адресом станции X1 для коммутатора сразу же по завершении ассоциации. Когда коммутатор получает этот кадр, происходит обновление таблицы переадресации, позволяющее получить доступ к станции X1 посредством точки доступа ТД2. Группа по разработке стандартов 802.11f занимается созданием протокола взаимодействия точек доступа для решения вышеописанной и связанной с ней задач.

6.3.5. Дополнительные функции 802.11

Мы закончим наше описание технологии 802.11 кратким обсуждением двух дополнительных функций сетей типа 802.11. Как мы вскоре убедимся, обе эти функции *не полностью* описаны в спецификации стандарта 802.11, но существуют благодаря другим механизмам, включенным в спецификацию. Это позволяет различным производителям реализовывать эти функции, используя собственные (патентованные) подходы, придавая им, в идеале, некоторое конкурентное преимущество.

Подстройка скорости передачи данных

Ранее, на рис. 6.3, мы видели, что для различных условий соотношения сигнал-шум (SNR) применяются различные технологии модуляции сигнала. Теперь, для примера, давайте представим мобильного пользователя сети 802.11, находящегося в 20 метрах от базовой станции в условиях высокого соотношения сигнал-шум. Учитывая большое значение показателя SNR, пользователь может связываться с базовой станцией с помощью технологии модуляции физического уровня, позволяющей сохранить вы-

сокие скорости передачи данных при низком показателе количества битовых ошибок (BER). Счастливчик, не правда ли?! Теперь давайте представим, что пользователь пришел в движение и начал отходить от базовой станции, при этом по мере увеличения расстояния началось падение значения показателя SNR. В этом случае техника модуляции сигнала в протоколе 802.11, используемом для связи базовой станции с пользователем, не изменится, количество битовых ошибок станет неприемлемо высоким при снижении соотношения сигнал-шум, в результате чего передаваемые кадры данных будут приниматься некорректно.

По этой причине некоторые решения 802.11 обладают функцией подстройки скорости передачи данных, осуществляющей выбор используемой техники модуляции сигнала физического уровня в зависимости от текущих характеристик канала связи. Если узел посылает два кадра подряд и не получает подтверждения (имплицитный индикатор битовых ошибок на канале), скорость передачи данных падает до ближайшего нижнего показателя. Если получение 10 кадров подряд подтверждено, либо если время, отсчитываемое таймером с последнего понижения скорости, истекает, то скорость передачи данных увеличивается на показатель. Технология адаптации скорости передачи данных основана на той же философии «апробации», что и механизм контроля перегрузки технологии TCP. В некотором роде технология подстройки скорости похожа на ребенка, просящего своих родителей дать ему еще больше и больше сладостей, если ребенок мал, или возможности вернуться домой поздней и поздней, если это подросток, до тех пор, пока родители не скажут ему «Хватит!» и ребенок не отступит (чтобы попробовать позже и разузнать, не улучшились ли условия!). Для улучшения схемы автоматической подстройки скорости передачи данных было предложено также несколько других идей[206, 273, 305].

Управление питанием

Для мобильных устройств электропитание — это очень ценный ресурс. Именно поэтому протокол стандарта 802.11 предлагает возможности по управлению питанием, позволяющие узлам сети 802.11 минимизировать время, необходимое функциям прослушивания канала, передачи и приема данных, а также оставаться включенным другому электрооборудованию. Управление питанием в сетях 802.11 работает следующим образом. Узел может явным образом переходить как в режим сна, так и в рабочее состояние (не то что сонные студенты в аудиториях!). Узел сообщает точке доступа о предстоящем переходе в режим

сна, устанавливая значение 1 для соответствующего бита заголовка кадра 802.11. После этого таймер узла автоматически настраивается таким образом, чтобы «разбудить» узел прежде, чем точка доступа отправит следующий запланированный сигнальный кадр (вспомним, что, как правило, точка доступа отправляет сигнальные кадры каждые 100 мкс). Так как благодаря установленному значению бита управления питанием точка доступа «знает», что узел готовится отойти ко сну, она (точка доступа) не будет направлять ему никаких кадров данных, вместо этого все кадры данных, предназначенные спящему хосту, будут буферизированы и отправлены позднее.

Узел пробуждается и очень быстро переходит в полноценное рабочее состояние (в отличие от сонного студента, всего лишь за 250 мкс![273]) незадолго до того, как точка доступа пошлет сигнальный кадр. Отправляемый точкой доступа сигнальный кадр содержит список узлов сети, чьи кадры данных были буферизированы в блоке памяти точки доступа. Если для узла нет буферизированных кадров данных, он может вновь вернуться в режим сна. В противном случае узел может явно затребовать буферизированные кадры данных с помощью отправки точке доступа специального опросного сообщения (polling message). Учитывая время между отправкой сигнальных кадров, равное 100 мкс, время на пробуждение — 250 мкс, и столь же незначительное время, требующееся на получение сигнального кадра и проверку на наличие буферизированных кадров, узел, не отправляющий или не получающий кадров данных, может оставаться в режиме сна до 99% рабочего времени, что позволяет существенно сократить энергозатраты.

6.3.6. Персональные сети: Bluetooth и Zigbee

Как показано на рис. 6.2, стандарт IEEE 802.11 WI-FI предназначен для соединения по беспроводной сети устройств, находящихся до 100 м друг от друга (за исключением случаев, когда сеть 802.11 применяется для двухточечной связи с использованием направительных антенн). Два других протокола IEEE 802.11 — Bluetooth и Zigbee (определяемые в стандартах IEEE 802.15.1 и IEEE 802.15.4[230]) и WiMAX (определяемый в стандарте IEEE 802.16[232, 233]) — являются стандартами для связи на более коротких длинных расстояниях соответственно. Мы еще вкратце поговорим о стандарте WiMAX, когда будем обсуждать мобильные сети для передачи данных в разделе 6.4, сейчас же обратим свое внимание на сети с более короткой дистанцией.

Bluetooth

Сеть IEEE 802.15.1 имеет небольшую зону покрытия, потребляет небольшое количество энергии и требует небольших финансовых затрат. По сути это энергосберегающая, низкоскоростная технология, «замещающая кабель» и работающая на коротких дистанциях, используемая для связи между собой ноутбуков, периферийных устройств, сотовых телефонов и смартфонов. В то время как 802.11 — это более энергозатратная, скоростная технология «доступа» в Интернет, работающая на средних дистанциях. Именно поэтому сети 802.15.1 часто называют беспроводными персональными сетями WPAN (от англ. Wireless Personal Area Networks). Физический и канальный уровни сетей 802.15.1 базируются на более ранних спецификациях технологии **Bluetooth** для персональных сетей[204, 54]. Сети 802.15.1 работают в нелицензируемом радиодиапазоне 2,4 ГГц по шаблону TDM (Time-Division Multiplexing — мультиплексирование с разделением времени). Величина временного промежутка составляет 625 мкс. На протяжении каждого временного промежутка отправитель выполняет передачу на одном из 79 каналов, при этом канал меняется при смене временного промежутка по известной псевдослучайной схеме. Таким образом осуществляется переключение каналов, также известное как **расширение спектра со скачкообразной перестройкой частоты FHSS** (Frequency-Hopping Spread Spectrum), распределяющее передачу по полосам частотного диапазона. Скорость передачи данных в сетях 802.15.1 может достигать 4 Мбит/с.

Все сети 802.15.1 являются одноуровневыми, так как для связи устройств 802.15.1 не требуется никакой сетевой инфраструктуры (например, точек доступа). Поэтому устройства 802.15.1 должны заниматься самоорганизацией и упорядочиванием. В первую очередь, устройства 802.15.1 организуются в так называемые **пикосети**, содержащие до восьми активных устройств, как показано на рис. 6.16. Одно из таких устройств называется «ведущим» (master), а остальные устройства — «подчиненными» (slaves). Ведущие устройства воистину являются «хозяевами» пикосети — часы такого устройства определяют время сети, оно может осуществлять отправку данных во все нечетные промежутки времени, в то время как подчиненные устройства могут вести передачу только после того, как с ними связалось ведущее устройство (в предыдущий временной интервал), при этом подчиненное устройство может передавать данные только ведущему устройству. В дополнение к подчиненным, в сети может также находиться до 255 зарегистрированных устройств. Такие устройства не могут осуществлять связь до тех пор,

пока их статус не будет изменен ведущим узлом с «зарегистрированных» на «активные».

Рис. 6.16. Пикосеть Bluetooth

Для получения более подробной информации о сетях WPAN 802.15.1, интересующиеся читатели могут обратиться к справочниками по технологии Bluetooth[204, 54] или официальному веб-сайту IEEE 802.15[230].

Zigbee

Следующий вид персональной сети, описываемый в стандарте IEEE, — это сеть Zigbee стандарта 802.14.5[230]. В то время как сети Bluetooth, являющиеся «кабелезаменителями», предоставляют скорость передачи данных более 1 Мбит в секунду, сети Zigbee предназначены для работы менее энергоемких, менее требовательных к скорости передачи данных приложений с меньшим циклом активности, нежели сети Bluetooth. Хотя большинство из нас думает, что «больше значит лучше», не для всех сетевых приложений требуются более высокая скорость передачи данных и, соответственно, более высокие затраты (как экономические, так и энергетические). Например, домашние датчики температуры и освещенности, устройства обеспечения безопасности, а также настенные переключатели являются очень простыми, не энергозатратными, дешевыми устройствами с малыми интервалами активности. Таким образом, сеть Zigbee идеально подходит для таких устройств. Скорость передачи данных в каналах сети Zigbee зависит от рабочей частоты канала и может составлять 20, 40, 100 и 250 Кбит/с.

Узлы сетей Zigbee делятся на два вида. Так называемые ограниченно-функциональные устройства работают под контролем единственного «полнофункционального устройства», что напоминает ведомые устройства сетей Bluetooth. Полнофункциональное устройство может играть роль, аналогичную роли «ведущего» устройства сети Bluetooth, заключающуюся в управлении множеством подчиненных устройств. Кроме того, несколько полнофункциональных устройств могут быть организованы в смешанную сеть, в которой они занимаются маршрутизацией кадров данных между собой. В технологии Zigbee также применяется большое количество механизмов, с которыми мы уже сталкивались при рассмотрении других протоколов канального уровня: сигнальные кадры и подтверждения канального уровня (аналогичны сетям 802.11), протоколы множественного случайного доступа с анализом состояния канала и двоичной экспоненциальной задержкой (как в сетях 802.11 и Ethernet), а также фиксированное, гарантированное резервирование временных интервалов (как в технологии DOCSIS).

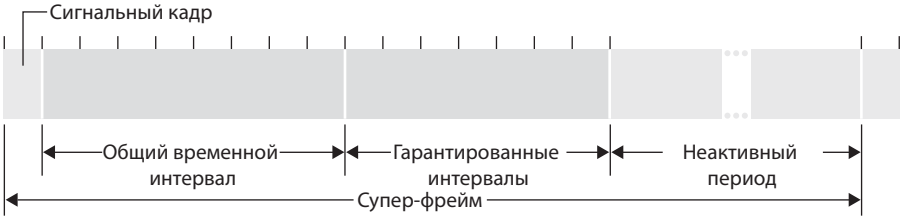


Рис. 6.17. Структура супер-фрейма сетей Zigbee 802.14.4

Существует множество вариантов конфигурации сетей Zigbee. Давайте рассмотрим вариант с единым полнофункциональным устройством, управляющим множеством ограниченно-функциональных устройств с использованием механизма разделения по временным интервалам и сигнальных кадров. На рис. 6.17 изображен случай, когда сеть Zigbee делит время на рекуррентные суперфреймы, каждый из которых начинается с сигнального кадра. Каждый сигнальный кадр подразделяет суперфрейм на активный период (на протяжении которого устройства могут осуществлять передачу данных) и неактивный период (на протяжении которого все устройства, в том числе, контроллер, могут переходить в режим сна и экономить тем самым электроэнергию). Активный период состоит из 16 временных интервалов, некоторые из них используются устройствами для осуществления случайного доступа на манер технологии CDMA/CA. Некоторые из временных интервалов ре-

зервируются контроллером для каких-либо конкретных устройств, что, таким образом, гарантирует этим устройствам доступ к каналу связи. Более подробную информацию о сетях Zigbee вы можете найти в публикациях Баронти[42] и спецификации IEEE 802.15.4 2012[231].

6.4. Доступ в Интернет посредством сетей сотовой радиосвязи

В предыдущем разделе мы рассмотрели механизмы, с помощью которых хосты, подключенные к точке Wi-Fi — иными словами, точке доступа типа 802.11, — могут получить доступ в Интернет. Однако площадь покрытия большинства точек доступа Wi-Fi сравнительно мала: от 10 до 100 м в диаметре. Что же мы делаем, когда нам срочно необходимо получить доступ в Интернет, но мы не можем подключиться к точке Wi-Fi?

Принимая во внимание тот факт, что сотовая телефония очень широко распространена во всех уголках мира, вполне естественным является желание дополнить сотовые сети так, чтобы они обеспечивали не только голосовую телефонную связь, но также и беспроводной доступ к Интернету. В идеале, такой вариант доступа в Интернет будет достаточно высокоскоростным и предоставит пользователям безусловную мобильность, позволяя им сохранять активные TCP-подключения во время перемещений в пространстве, например, поездки на автобусе. Обладая высокой скоростью передачи данных в восходящем и нисходящем направлениях, такие сети могут даже поддерживать видеоконференцию при постоянном перемещении пользователей. Сценарий тут не такой уж и изысканный. По данным на 2012 год, большое количество операторов сотовой связи в США предоставляли своим абонентам услугу доступа в Интернет посредством сотовой сети на скорости в несколько сотен килобит в секунду по цене около 50 долларов или даже ниже. В сотовых сетях с услугами широкополосного доступа в Интернет, которые мы рассмотрим в этом разделе, становится возможным достижение скоростей в несколько мегабит в секунду, кроме того, такие сети становятся все более и более распространенными.

В этом разделе мы вкратце рассмотрим уже используемые и только зарождающиеся технологии доступа в Интернет посредством сотовой связи. Мы обратим внимание не только на первичный, беспроводной, транзитный участок сети, но также и на сеть, соединяющую беспроводной транзитный участок в большую телефонную сеть и/или подключа-

ющую его к Интернету. В разделе 6.7 мы порассуждаем над вопросами маршрутизации телефонных вызовов в условиях перемещения пользователя от одной базовой станции к другой. Наше краткое обсуждение будет обязательно включать только упрощенное высокоуровневое описание технологий сотовой связи. Конечно же, современная сотовая связь имеет большую ширину и глубину, при этом во многих университетах она изучается в рамках сразу нескольких дисциплин. Читателям, желающим более детально разобраться в вопросе, мы бы порекомендовали обратиться к работам Гудмена[185], Кааранена[270], Лина[321], Корхонена[295], Шиллера[586], Скауриаса[595], Тернера[641] и Акилдиза[17]. Книга Моули[355] особенно замечательна и заслуживает наших отдельных рекомендаций.

6.4.1. Обзор архитектуры сотовых сетей

При описании архитектуры сетей сотовой связи мы будем пользоваться терминологией, употребляемой в стандартах *глобальной системы мобильной связи* (Global System for Mobile Communication, **GSM**). В целях исторической справедливости следует отметить, что аббревиатура GSM изначально расшифровывалась как Groupe Spécial Mobile, прежде чем в западном мире стали применять более англофицированный термин, позволивший сохранить буквы изначальной аббревиатуры. В 1980-х годах европейцы осознали потребность в создании единой общеевропейской системы цифровой сотовой связи, которая заменила бы большое количество несовместимых друг с другом аналоговых систем, что, собственно и привело к появлению стандарта GSM[355]. В начале 1990-х годов европейцам удалось успешно развернуть первые сети GSM, и с тех пор этот стандарт уже превратился в огромного многотонного Кинг-Конга мира сотовой связи: ему принадлежит более 80% всех абонентов сотовой связи мира.

Когда говорят о технологиях сотовой связи, их зачастую распределяют по принадлежности к тому или иному «поколению». Первые поколения систем сотовой связи создавались в основном для обеспечения голосового трафика. Системами первого поколения (1G) были сети FDMA, разработанные только для голосовой связи. Такие системы поколения 1G сейчас практически не встречаются: их заменяют цифровые системы поколения 2G. Первоначально системы сотовой связи второго поколения также создавались только для голосовой связи, однако позже они были расширены (2,5G) для предоставления услуг обмена данными (например, доступа в Интернет) при сохранении доступа к услугам передачи голоса.

В настоящее время в мире развертываются сети поколения 3G, они предоставляют пользователям услуги по передаче голоса и данных, однако роль передачи данных в таких сетях постоянно растет, при этом каналы радиосвязи предоставляют все большие и большие скорости.

ИСТОРИЯ

Мобильные сети поколения 3G против беспроводных локальных сетей

Большое количество операторов сотовой связи разворачивают сети поколения 3G, предоставляющие скорости передачи данных до 2 Мбит/с в помещении и до 384 кбит/с или выше на улице. Сети третьего поколения работают в лицензируемых радиочастотных диапазонах, при этом некоторые операторы сотовой связи вынуждены платить правительствам стран огромные суммы денег за лицензии на использование частотного диапазона. Сети третьего поколения позволяют пользователям, находясь на улице, в достаточно удаленной точке, получить доступ в Интернет, при этом не прекращая движения, наподобие современных возможностей доступа к мобильной сети. Например, сеть 3G позволяет абоненту получить доступ к дорожной карте, находясь в движущемся автомобиле, или к афише кино и театров, когда абонент загорает на пляже. Однако у кого-то могут появиться сомнения по поводу того, до какой степени будут расти такие сети, принимая во внимание их себестоимость, а также тот факт, что зачастую абоненты могут одновременно иметь доступ и к 3G и к беспроводной локальной сети.

- Зарождающаяся инфраструктура беспроводных локальных сетей в скором времени может разрастись до очень больших размеров и стать всеобъемлющей. Беспроводные локальные сети 802.11, работающие на скорости 54 Мбит/с, получают очень широкое применение. Практически все портативные компьютеры и смартфоны оснащены интерфейсами для подключения к локальным сетям 802.11. Более того, только появляющиеся Интернет-устройства, такие как беспроводные камеры и цифровые фоторамки, также комплектуются экономичными интерфейсами для подключения к беспроводным локальным сетям.
- Базовые станции беспроводных локальных сетей также могут взять на себя выполнение задач мобильной телефонии. Большое количество мобильных телефонов уже в наши дни могут подключаться не только к сотовым сетям, но и к сетям IP-телефонии благодаря встроенному программному обеспечению или сервисам VoIP, наподобие Skype. Таким образом обеспечивается «обход» сервисов оператора сотовой связи 3G по передаче голоса и данных.

Впрочем, другие специалисты, наоборот, уверены, что сети поколения 3G не только станут очень успешными, но также совершат революцию в стиле нашей работы и жизни. Наиболее вероятно, что и 3G и Wi-Fi станут основными стандартами беспроводной связи, при этом перемещающиеся по сети беспроводные устройства будут самостоятельно осуществлять выбор технологии доступа, наиболее подходящей для использования, в зависимости от фактического местонахождения.

Архитектура сети сотовой связи, 2G: Голосовая связь

Сам термин «сотовая» означает, что область, покрываемая сетью сотовой связи, разделена еще на несколько зон, называемых **сотами**, — они изображены в виде шестиугольников в левой части рис. 6.18. Как и стандарт 802.11 Wi-Fi, у сетей GSM существует собственная, присущая только сетям такого вида номенклатура. В каждой соте сети есть **базовая трансиверная** (приемопередающая) **станция (БТС)**, принимающая от мобильных станций в соте и посылающая этим мобильным станциям собственные сигналы. Площадь покрытия соты зависит от множества факторов, в том числе и от мощности передатчика БТС, мощности передатчиков абонентских устройств, наличия в соте создающих помехи зданий, а также высоты антенн базовой станции. Хотя на рис. 6.11 изображено, что базовая трансиверная станция находится в центре каждой соты, в настоящее время во многих сетях БТС располагается на пересечении трех сот таким образом, что одна БТС с направительными антеннами может обслуживать одновременно три соты.

Стандарт GSM для сетей поколения 2G предполагает сочетание технологий частотного и временного мультиплексирования FDM/TDM (радио) для воздушного интерфейса. Вспомните из текста главы 1, что при использовании только технологии FDM (частотное мультиплексирование) канал связи разделяется на сегменты, представляющие собой разные частотные диапазоны, причем для осуществления одного голосового вызова выделяется одна частота. Также вспомним из главы 1, что при использовании только технологии TDM (временное мультиплексирование) время разделяется на фреймы, а каждый из фреймов дополнительно разбит на промежутки (слоты), при этом каждому осуществляемому вызову назначается конкретный временной промежуток в цикличном фрейме. При сочетании технологий частотного и временного мультиплексирования канал связи разбивается на несколько частот, при этом время каждой из выделенных частот канала разбивается

на отдельные фреймы, а затем промежутки (слоты). Поэтому в случае с системой, сочетающей технологии FDM и TDM, если канал разбит на F частот, а время — на T промежутков, то канал связи сможет одновременно поддерживать количество вызовов, равное $F \times T$. Вспомните также из раздела 5.3.4, что в кабельных сетях также применяется подход сочетания частотного и временного мультиплексирования FDM/TDM. В системах GSM выделяются частоты по 200 кГц, при этом каждая из частот поддерживает восемь вызовов с временным мультиплексированием. Речь, передаваемая по сетям GSM, кодируется на скорости 13 и 12,2 кбит/с.

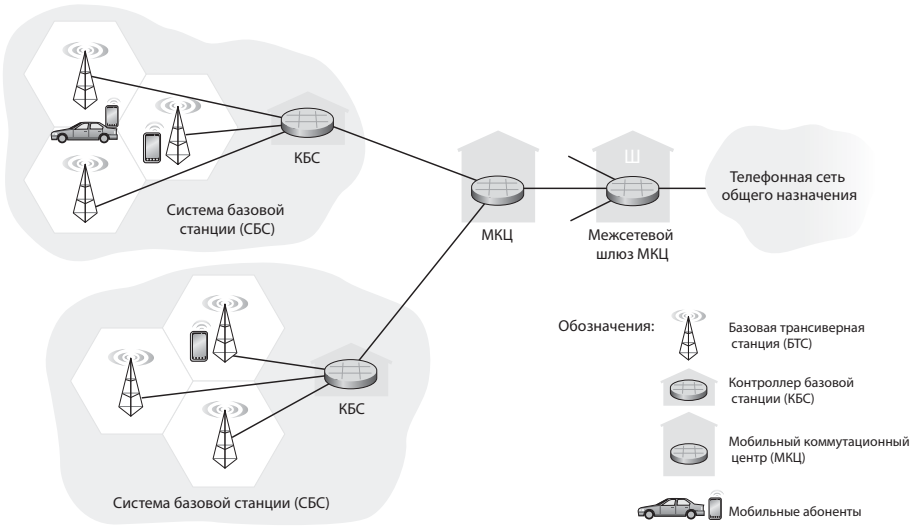


Рис. 6.18. Компоненты архитектуры сети GSM поколения 2G

Контроллер базовой станции (КБС) сетей GSM, как правило, обслуживает сразу несколько десятков базовых трансиверных станций. Роль КБС состоит в том, чтобы выделить мобильным абонентам радиоканал БТС, выполнить **пеленг** (например, определить, в какой из сот сети находится нужный абонент), а также осуществить передачу мобильного абонента (эту тему мы вскоре обсудим в разделе 6.7.2). Множество контроллеров базовой станции и управляемых ими базовых трансиверных станций представляет собой **систему базовой станции (СБС)**.

В разделе 6.7 мы увидим, что **мобильный коммуникационный центр (МКЦ)** играет важную роль в авторизации пользователя и обслуживании его абонентского счета (например, определяет, имеет ли право

мобильное устройство подключаться к сети), установлении и прекращении вызова, а также передачи абонента. Единый МКЦ может содержать до 5 КБС, в результате чего количество абонентов на один МКЦ равняется примерно 200 тысячам. Сеть оператора сотовой связи, как правило, включает несколько МКЦ, в том числе и специальные МКЦ, называемые межсетевыми шлюзами, которые соединяют сотовую сеть оператора связи с более крупной телефонной сетью общего назначения.

6.4.2. Сотовая сеть передачи данных поколения 3G: Интернет для абонентов сотовых сетей

Главной темой нашей дискуссии в разделе 6.4.1 было подключение абонентов сотовой сети, осуществляющих голосовой вызов, к телефонной сети общего назначения. Однако, конечно, находясь в пути, нам также хотелось бы иметь возможность просмотреть электронную почту, подключиться ко Всемирной паутине, воспользоваться сервисами, связанными с местоположением (например, картами или советами по выбору ресторана) и, возможно, даже посмотреть видео в режиме онлайн. Чтобы осуществить все вышеописанное, нашему смартфону потребуется запустить полный пакет протокола TCP/IP (в том числе канальный, сетевой, транспортный и прикладной уровни) и подключиться к Интернету через сотовую сеть передачи данных. Сама тема сотовых сетей передачи данных представляет собой весьма запутанное собрание конкурирующих и постоянно развивающихся стандартов, появляющихся как только одно поколение (или полупоколение) приходит на смену предыдущему и приносит с собой новые технологии с новыми сервисами и аббревиатурами. Однако проблема состоит еще и в том, что на сегодня в мире не существует единой организации, которая бы задавала требования к технологиям поколений 2,5G, 3G, 3,5G и 4G. Это усложняет процесс выявления различий между соревнующимися стандартами. В нашем следующем обсуждении мы обратим наше внимание на стандарты третьего поколения UMTS (Universal Mobile Telecommunications Service — Универсальная система мобильной связи), созданные организацией 3GPP (3rd Generation Partnership project — Проект партнерства третьего поколения)[2]. Сегодня технология UMTS очень популярна.

Давайте теперь пройдемся сверху вниз по архитектуре сотовой сети передачи данных поколения 3G, представленной на рис. 6.19.

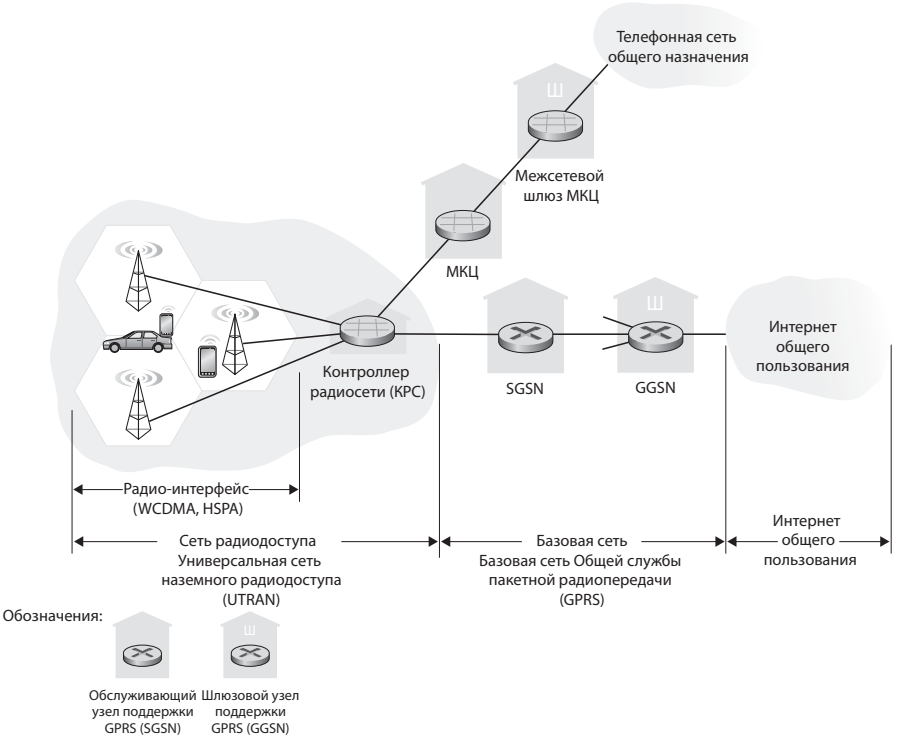


Рис. 6.19. Архитектура системы 3G

Базовая сеть 3G

Базовая мобильная сеть передачи данных 3G ответственна за подключение сетей радиодоступа к Интернету общего пользования. Базовая сеть взаимодействует с компонентами имеющейся сотовой сети для передачи голоса (в частности, с МКЦ), с которыми мы уже встречались ранее на рис. 6.18. Принимая во внимание внушительное количество уже имеющихся инфраструктурных компонентов (и выгодных сервисов!) в существующих мобильных сетях для передачи голоса, подход разработчиков к сервисам передачи данных поколения 3G становится предельно ясен: *оставить без изменений существующие мобильные сети GSM для передачи голоса, добавить возможности по передаче данных в дополнение к уже имеющейся сотовой сети.* Альтернативный подход — попытка интеграции новых сервисов по передаче данных напрямую в уже имеющуюся базовую сеть для передачи голоса — породила бы трудности аналогичные тем, с которыми мы познакомились в разделе 4.4.4, когда обсуждали интеграцию новых (IPv6) и старых, доставшихся нам в наследство (IPv4) технологий в Интернете.

В базовой сети 3G есть два типа узлов: **обслуживающий узел поддержки GPRS (SGSN)** и **шлюзовой узел поддержки GPRS (GGSN)**. Аббревиатура GPRS расшифровывается как «общая служба пакетной радиопередачи (данных)» — ранний вариант услуги передачи данных в сетях 2G. В данном разделе мы обсудим преобразованную версию технологии GPRS, используемую в сетях 3G. Узел SGSN ответственен за доставку дейтаграмм от/на мобильные узлы сети радиодоступа, с которой этот узел связан. Узел SGSN взаимодействует с МКЦ, находящимся в нужной зоне сети, предоставляющим возможности по авторизации абонента и его передачи с сохранением информации местоположения (соты) об активных мобильных узлах в сети радиодоступа, и выполняющим переадресацию дейтаграмм между мобильными узлами в сети радиодоступа и узлом GGSN. Узел GGSN играет роль шлюза, подключающего несколько узлов SGSN к более крупной сети — Интернету. Таким образом, узел GGSN — это последний компонент инфраструктуры сети 3G, с которым встречается дейтаграмма, созданная мобильным узлом перед попаданием в более крупную сеть Интернет. Для внешнего мира узел GGSN выглядит как еще один шлюзовой маршрутизатор, коих много, так как мобильность узлов сети 3G скрыта от внешнего мира за узлом GGSN.

Сеть радиодоступа поколения 3G: беспроводная периферия

Сеть радиодоступа 3G — это сеть первого скачка, непосредственно с которой мы взаимодействуем как пользователи сервисов 3G. **Контроллер радиосети (КРС)**, как правило, управляет сразу несколькими базовыми трансиверными станциями сот сети, как это делают базовые станции, встречающиеся в системах второго поколения (в словаре сетей 3G UMTS из называют «Узлами Б» — крайне неочевидное название!). Беспроводной канал связи каждой соты соединяет мобильные узлы с трансиверной станцией, точно так же, как и в случае с сетью 2G. КРС подключается и к голосовой сотовой сети с коммутацией каналов через МКЦ, а также к Интернету с коммутацией пакетов данных через узел SGSN. Таким образом, несмотря на то, что для предоставления услуг по передаче голоса и данных в сетях 3G используются разные базовые сети, эти сети подключены к единой сети радиодоступа первого/последнего скачка.

Существенное отличие технологии 3G UMTS от сетей второго поколения заключается в том, что вместо схемы FDMA/ стандарта GSM, в сетях UMTS в рамках временных промежутков TDMA используется механизм CDMA под названием Direct Sequence Wideband CDMA

(DS-WCDMA — широкополосный множественный доступ с кодовым разделением каналов и прямой последовательностью)[121]. Временные промежутки TDMA поочередно становятся доступными для множества частот, что позволяет с еще большей эффективностью использовать все три подхода разделения канала связи, которые мы обозначили ранее в главе 5. Такой механизм, опять же, аналогичен механизму, использующемуся в кабельных сетях (см. раздел 5.3.4). Такое концептуальное изменение требует создания новой сети беспроводного доступа поколения 3G, работающей параллельно с сетью 2G и ее системами базовых станций, как показано на рис. 6.19. Сервис по передаче данных, связанный с технологией WCDMA, получил название HSPA (High Speed Packet Access — Высокоскоростная пакетная передача данных), заявленная максимальная скорость передачи данных по нисходящему каналу связи может достигать 14 Мбит/с. Более подробную информацию о сетях третьего поколения вы можете найти на официальном сайте Проекта партнерства третьего поколения (3GPP)[2].

6.4.3. Переход к 4G: LTE

Сейчас, когда по всему миру разворачиваются сети поколения 3G, следует ли ожидать, что появление сетей 4G будет очень нескорым? Конечно же нет! Более того, разработка, тестирование и первичное развертывание систем 4G осуществлюется уже сегодня! Стандарт 4G LTE (Long-Term Evolution — Долгосрочное развитие), изданный организацией 3GPP несет в себе два инновационных изменения по сравнению с сетями 3G:

- **Усовершенствование пакетное ядро** (Evolved Packet Core, EPC)[3]. EPC представляет собой упрощенную базовую сеть, полностью построенную на IP, которая объединяет прежде раздельные сеть передачи голоса с коммутацией по каналам связи и сеть для передачи данных с коммутацией по пакетам данных, изображенные на рис. 6.19. Таким образом, в сети, полностью построенной на IP (сеть AIPN), голос и данные передаются в виде IP-дейтаграмм. Как мы уже видели в главе 4 и еще обсудим более детально в главе 7, модель сервиса IP под названием «лучшее из возможного» изначально не очень хорошо отвечает строжайшим требованиям по производительности, выдвигаемым к трафику данных технологии VoIP (Voice-over-IP — Голос по IP), если только сетевые ресурсы не подвергаются тщательному управлению с целью избежать перенасыщения, а не реакции на

него. Поэтому ключевая задача EPC — управлять ресурсами сети таким образом, чтобы сохранялась возможность предоставлять услуги высокого качества. Кроме того, технология EPC позволяет отчетливо разделять плоскости управления сетью и передачи пользовательских данных, поддерживая при этом большое количество функций мобильности, применение которых мы изучим в разделе 6.7 на контрольной плоскости. Технология EPC допускает использование сетей радиодоступа разных типов, в том числе Motorola 2007 устаревших сетей 2G и сетей радиодоступа поколения 3G, которые могут быть подключены к базовой сети. Мы рекомендуем вам обратиться к двум очень полезным источникам на сайтах **www.motorola.com**[354] и **lightreading.com**[18].

- **Сеть радиодоступа LTE**. В сетях LTE применяется сочетание частотного и временного мультиплексирования нисходящего канала, называемое также мультиплексированием с ортогональным разделением частот (Orthogonal Frequency Division Multiplexing — OFDM)[568, 151]. В данном случае термин «ортогональный» подчеркивает то, что сигналы, посылаемые в разные частотные каналы, практически не накладываются один на другой, даже если разрыв между частотами каналов очень невелик. В сети LTE каждому мобильному узлу предоставляется один или несколько временных промежутков длиной 0,5 мс в одной или нескольких частотах канала. На рис. 6.20 изображено выделение восьми временных промежутков на четырех частотах. Предоставление мобильному узлу значительно большего количества временных промежутков (на одной или нескольких частотах) позволяет достичь значительно больших скоростей передачи данных. Выделение и повторное выделение временных промежутков для мобильных узлов может производиться, внимание, один раз в миллисекунду! Кроме того, для изменения скорости передачи данных могут использоваться различные схемы модуляции сигнала (см. обсуждение рис. 6.3 выше и схем модуляции в сетях Wi-Fi). Еще одной инновацией в сетях радиодоступа LTE является использование сложных антенн многоканального ввода и многоканального вывода (MIMO). Максимальная скорость передачи данных пользовательского устройства в сетях LTE по нисходящему каналу составляет 100 Мбит/с и Мбит/с — по восходящему при полезном использовании 20 МГц беспроводного спектра.

Стандарт LTE не обязывает к выделению конкретных временных промежутков для мобильных узлов. Вместо этого, решение о том, какие

мобильные узлы смогут осуществить передачу данных в заданный временной промежуток на заданной частоте, принимается алгоритмами-планировщиками использования времени производителя оборудования сети LTE и/или оператора данной сети. Планирование «на ходу» (или «оппортунистическое планирование»)[47, 293, 300], подбирающее протокол физического уровня в зависимости от состояния канала связи между отправителем и получателем, а также позволяющее осуществить выбор получателей, которым будут отправлены пакеты данных в зависимости от состояния канала связи, способствует наиболее эффективному использованию беспроводной среды контроллером радиосети. Кроме того, становится возможным применение приоритетов пользователей и договорных уровней обслуживания (например, серебряный, золотой или платиновый) при передаче пакетов данных по нисходящему каналу. В дополнении к вышеописанным возможностям сетей LTE, технология LTE-Advanced (Расширенный LTE) позволяет мобильным узлам развивать скорости передачи данных в несколько сотен мегабит в секунду в нисходящем направлении с помощью выделения таким узлам агрегированных каналов связи[17].

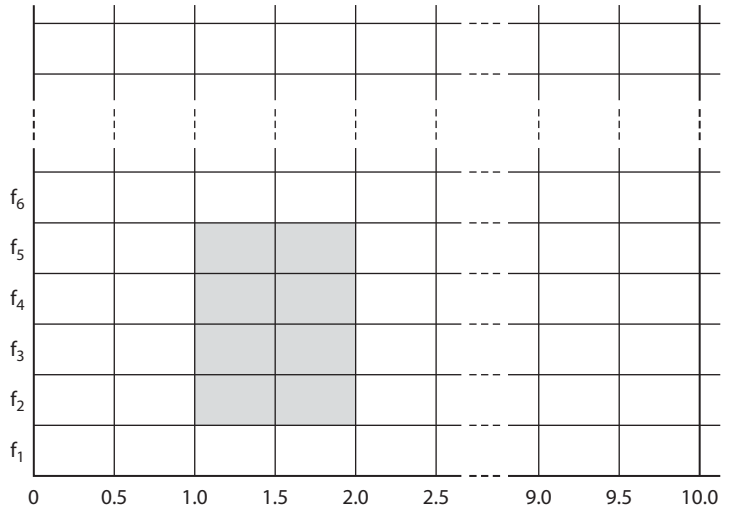


Рис. 6.20. Двадцать временных промежутков по 0,5 мс организованы в 10 мс фреймы на всех частотах канала. Затенены восемь выделенных временных промежутков

Еще одна беспроводная технология четвертого поколения — WiMAX (World Interoperability for Microwave Access — Глобальная совместимость для микроволнового доступа) — семейство стандартов

IEEE 802.16, значительно отличающихся от LTE. Пока еще не известно, в пользу какой из технологий четвертого поколения — LTE или WiMAX — будет сделан выбор, но в момент написания данной книги (весна 2012-го) технология LTE имела значительное превосходство. С подробным описанием технологии WiMAX вы можете познакомиться на сайте **www.wimaxforum.org**.

6.5. Управление мобильностью: Принципы

Теперь, когда мы уже обсудили *беспроводную* природу каналов связи беспроводных сетей, настало время переключить наше внимание на *мобильность*, открываемую беспроводными каналами связи. В самом широком смысле, мобильный узел — это узел, меняющий точку подключения к сети с течением времени. По причине того, что сам термин «мобильность» приобрел большое количество значений, как в мире компьютерных, так и сетевых технологий, сначала нам следует обсудить более подробно несколько категорий мобильности.

- *С точки зрения оборудования сетевого уровня насколько мобилен тот или иной пользователь?* Физически мобильный пользователь, в зависимости от того, как он или она перемещается между точками подключения, представляет собой целый набор различных задач для сетевого уровня. На одном конце спектра на рис. 6.21 пользователь может перемещаться по зданию с ноутбуком, оборудованным интерфейсом для подключения к беспроводной сети. Как мы уже видели в разделе 6.3.4, с точки зрения сетевого уровня этот пользователь *не* является мобильным. Более того, если пользователь остается ассоциированным с одной и той же точкой доступа вне зависимости от местоположения, такой пользователь не является мобильным даже на канальном уровне.

  Представим, что на противоположном конце спектра находится пользователь, несущийся на скорости 150 километров час по автомагистрали на своем спортивном BMW, при этом он проезжает мимо большого количества беспроводных точек доступа и хочет сохранить установленное TCP подключение к удаленному приложению на протяжении всей поездки. Этот пользователь *однозначно* является мобильным! Однако между этими крайностями есть пользователь, переносящий портативный компьютер с одного местоположения (например, рабочего кабинета или общежития) в другое (например, кофейня или учебная аудитория) и желающий подключиться

к сети в новом местоположении. Этот пользователь также является мобильным (хотя, конечно, гораздо менее мобильный, чем водитель BMW!), однако он не нуждается в сохранении текущих подключений во время перемещения между точками подключения к сети. На рис. 6.21 изображен спектр категорий мобильности пользователей с точки зрения сетевого уровня.

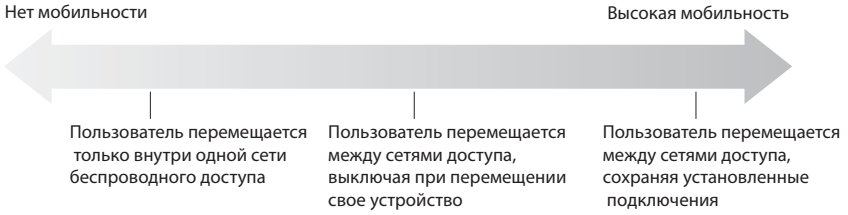


Рис. 6.21. Различные категории мобильности с точки зрения уровня сети

- *Насколько важно для мобильного узла сохранять свой адрес неизменным?* Когда вы имеете дело с мобильной телефонией, ваш номер телефона, а по сути это является адресом сетевого уровня вашего мобильного телефона, остается неизменным при перемещении из сети одного оператора в сеть другого. Должен ли ноутбук сохранять тот же IP-адрес при перемещении между IP-сетями?

  Ответ на этот вопрос во многом зависит от запущенных в момент времени приложений. Так, для водителя спортивного BMW, желающего сохранять текущее TCP-подключение к удаленному приложению во время скоростной поездки по автобану, было бы гораздо удобнее иметь неизменный IP-адрес. Вспомним из текста главы 3, что Интернет-приложению требуется знать IP-адрес и номер порта устройства, с которым установлено подключение. Если мобильному устройству удается при перемещении сохранять IP-адрес, то мобильность становится невидимой с точки зрения удаленного приложения. Такая невидимость несет большую пользу, ведь приложению не нужно более заботиться о том, что IP-адрес может поменяться в любой момент, таким образом, единый программный код приложения может быть использован для обслуживания как мобильных, так и немобильных подключений. В следующем разделе мы убедимся, что мобильный IP предоставляет необходимую прозрачность, что позволяет мобильному узлу сохранять фиксированный IP-адрес при перемещении между сетями.

- *Какая доступна вспомогательная проводная инфраструктура?* Во всех вышеописанных ситуациях мы всегда отталкивались от того,

что существует некая постоянная инфраструктура, к которой может подключиться мобильный пользователь, например, это может быть домашняя ISP сеть, сеть беспроводного доступа в офисе, а также сети беспроводного доступа, развернутые вдоль автобана. Но что если нет ни одной из перечисленных инфраструктур? Если два пользователя находятся в достаточной близости друг к другу для установления соединения, могут ли они установить сетевое подключение в отсутствие всякой инфраструктуры сетевого уровня? Децентрализованные (ad hoc) сети обеспечивают как раз нужные возможности. Эта быстроразвивающаяся сфера в данный момент представляет собой передовой край исследования мобильных сетей, но находится вне поля зрения нашей книги. Книга Перкинса[389] и веб-сайт рабочей группы IETF Mobile Ad Hoc Network[333] содержат наиболее детальное описание данной проблемы.

Для иллюстрации проблем, возникающих в связи с разрешением мобильным пользователям сохранять установленные подключения при перемещении между сетями, давайте проведем аналогию с примером из человеческий жизни. Молодой человек двадцати с чем-то лет, покидая родительский дом, становится «мобильным», проживая в общежитиях и на съемных квартирах, часто меняя при этом свой адрес. Если давняя подруга желает пообщаться с ним, каким образом она может получить новый адрес своего мобильного друга? Одним из наиболее распространенным способов является связь с его семьей, так как мобильный молодой человек, скорее всего, «зарегистрирует» свой новый адрес среди членов семьи (как минимум, для того, чтобы родители смогли при необходимости оказать материальную поддержку в выплате квартплаты). Таким образом, родительский дом с неизменившемся адресом становится единственным местом, куда могут в первую очередь обратиться другие люди, желающие наладить контакт с мобильным молодым человеком. Впоследствии общение между друзьями может быть либо опосредованное (например, если корреспонденция сначала отправляется на адрес родительского дома, а затем пересылается по адресу мобильного молодого человека), либо прямое (например, если подруга будет пользоваться адресом, полученным от родителей своего друга для непосредственной отправки писем).

В сетевых технологиях постоянный дом мобильного узла (например, ноутбука или смартфона) называется **домашней сетью**, а мобильное устройство, входящее в состав домашней сети и выполняющее задачи по управлению мобильностью (о нем речь пойдет ниже), называется **до-**

машним агентом. Сеть, в которой находится мобильный узел в настоящий момент времени, носит название **чужой** (или **посещенной**) **сети**, а устройство-элемент чужой сети, помогающее мобильному узлу с задачами управления мобильностью, называется **агентом чужой сети**. Так для мобильных специалистов домашней, скорее всего, будет являться сеть компании, в которой они работают, тогда как посещенной — сеть на рабочем месте коллеги, которого они посещают. **Корреспондент** — это устройство, желающее установить связь с мобильным узлом. Вышеперечисленные концепции отражены на рис. 6.22 вместе с понятиями адресации, обсуждаемыми ниже. Обратите внимание, что на рис. 6.22 агенты изображены в сочетании с маршрутизаторами (т. к. процессы обрабатываются маршрутизаторами), однако в качестве альтернативы, процессы могут также выполняться на других хостах или серверах, присутствующих в сети.

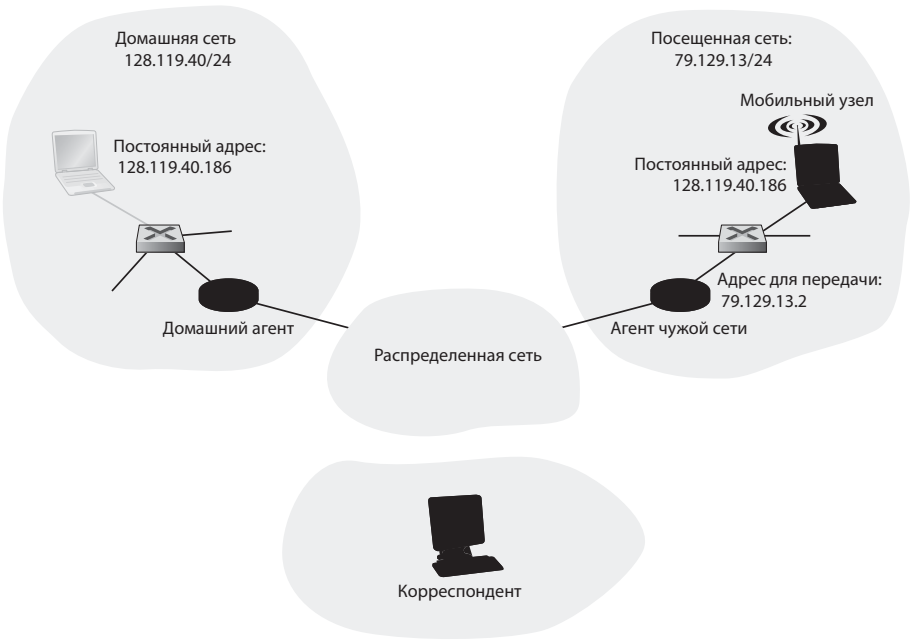


Рис. 6.22. Первичные элементы архитектуры мобильной сети

6.5.1. Адресация

Как уже отмечалось выше, чтобы мобильность была прозрачной для сетевых приложений, мобильному узлу желательно сохранять свой

адрес в ходе перемещения из одной сети в другую. Когда мобильный узел находится в чужой сети, в нее должен быть маршрутизирован весь поток данных, посылаемый на постоянный адрес узла. Каким образом это осуществить? Первый вариант — это когда чужая сеть должна сообщать всем остальным сетям, что к ней подключился данный узел. Это можно осуществить путем обычного обмена внутридоменной и междоменной маршрутной информацией, что не потребует значительных изменений в имеющейся инфраструктуре маршрутизации. Чужая сеть может просто оповестить соседние сети о наличии у нее высокоспецифичного маршрута к постоянному адресу мобильного узла (что, по сути, является оповещением других сетей о наличии корректного пути для маршрутизации дейтаграмм на постоянный адрес мобильного узла, см. раздел 4.4). После этого соседние сети распространят эту маршрутную информацию как часть обычной процедуры обновления маршрутной информации и таблиц переадресации. Когда мобильный узел покинет одну чужую сеть и подключится к другой, эта новая чужая сеть распространит новый высокоспецифичный маршрут к мобильному узлу, а старая чужая сеть удалит маршрутную информацию, касающуюся данного узла.

Такой подход позволяет решить сразу две проблемы, при этом отсутствует какая-либо необходимость во внесении сколько-нибудь значительных изменений в инфраструктуру сетевого уровня. Другие сети знают местоположение мобильного узла и, благодаря тому, что таблицы перенаправления отсылают дейтаграммы в чужую сеть, маршрутизация дейтаграмм на мобильный узел значительно облегчается. Однако значительным недостатком этого подхода является ограниченная масштабируемость. Если бы ответственность за управление мобильностью лежала на сетевых маршрутизаторах, то им пришлось бы обрабатывать таблицы перенаправления, содержащие записи о, возможно, миллионах мобильных узлов и обновлять эти записи при перемещении этих узлов. Еще несколько других недостатков данного подхода будут изучены в упражнениях, доступных среди примеров для данной главы к книге.

Альтернативный подход (применяемый на практике) состоит в выталкивании обработки мобильности из ядра сети на периферию — сквозная тема в нашем изучении архитектуры Интернета. Вполне естественно было бы осуществить это посредством домашней сети мобильного узла. Аналогично тому, как родители отслеживают местоположение своего двадцатилетнего чада, домашний агент из домашней сети мобильного узла может заниматься отслеживанием того, к какой чужой сети под-

ключается мобильный узел. Для обновления местоположения мобильного узла потребуется некий протокол взаимодействия мобильного узла (или агента чужой сети, представляющего интересы мобильного узла) и домашнего агента.

Сейчас давайте более подробно обсудим агент чужой сети. Концептуально наиболее простой подход, изображенный на рис. 6.22, заключается в расположении агентов сети на пограничных маршрутизаторах чужой сети. Одной из задач агента чужой сети является создание для мобильного узла так называемого **адреса для передачи** (Care-of address, COA). Таким образом, один мобильный узел ассоциируется сразу с двумя адресами: **постоянным адресом** (аналогичен адресу родительского дома мобильного молодого человека из нашего примера) и адресом для передачи, иногда называемым **адресом в чужой сети** (аналогичен адресу дома, в котором временно проживает наш мобильный молодой человек). В примере на рис. 6.22 постоянный адрес мобильного узла 128.119.40.186. При посещении сети 79.129.12/24 мобильному узлу присваивается адрес для передачи 79.129.13.2. Вторая задача агента чужой сети заключается в информировании домашнего агента о том, что данный мобильный узел находится в его (агента чужой сети) сети и ему присвоен адрес для передачи. Вскоре мы увидим, что адрес для передачи будет использован для перенаправления дейтаграмм на мобильный узел через агент чужой сети.

Несмотря на то что мы разделили функциональность мобильного узла и агента чужой сети, мобильному узлу ничего не будет стоить перенять ответственность последнего. Например, при регистрации в чужой сети мобильный узел может самостоятельно получить адрес для передачи (например, с помощью такого протокола как DHCP) и самостоятельно проинформировать домашний агент о полученном адресе для передачи.

6.5.2. Перенаправление на мобильный узел

К настоящему моменту мы уже изучили механизмы получения мобильным узлом адреса для передачи (COA), а также способы информирования домашнего агента о полученном мобильным узлом адресе. Однако оповещение домашнего агента об адресе для передачи решает только часть проблемы. Каким образом необходимо производить адресацию и пересылку дейтаграмм на этот мобильный узел? Так как только домашний агент (но не маршрутизаторы, подключенные к сети) знает местоположение мобильного узла, простой адресации дейтаграммы по

постоянному адресу мобильного узла с последующей отправкой в инфраструктуру уровня сети будет недостаточно. Нужно сделать что-то еще. Для решения этой проблемы можно выделить два подхода, которые мы будем называть прямой и непрямой маршрутизацией.

Непрямая маршрутизация на мобильный узел

Сначала давайте поговорим о корреспонденте, желающем отправить дейтаграмму на мобильный узел. При применении **непрямой маршрутизации** корреспондент просто отправляет дейтаграмму на постоянный адрес мобильного узла, будучи при этом в счастливом неведении о том, находится ли мобильный узел в домашней сети или посещает чужую сеть. Мобильность остается абсолютно невидимой (прозрачной) для корреспондента. В этом случае дейтаграммы, как правило, сначала направляются в домашнюю сеть мобильного узла (см. шаг 1 на рис. 6.23).

Теперь давайте переключим свое внимание на домашний агент. В добавок к ответственности за взаимодействие с агентом чужой сети, позволяющее отслеживать адреса для передачи мобильных узлов, домашний агент имеет еще одну очень важную функцию. Его вторая работа заключается в том, что домашний агент должен быть «начеку» и перехватывать дейтаграммы, адресованные мобильным узлам-элементам домашней сети, находящимся в данный момент времени в других сетях. Домашний агент получает эти дейтаграммы, а затем пересылает их агенту чужой сети в два этапа. Сначала дейтаграмма пересылается агенту чужой сети с помощью адреса для передачи (COA) мобильного узла (шаг 2 на рис. 6.23), а затем переадресуется агентом чужой сети мобильному узлу (шаг 3 на рис. 6.23).

Будет полезно рассмотреть такое перенаправление более детально. Домашнему агенту понадобится отправить дейтаграмму с помощью адреса для передачи, полученного мобильным узлом, таким образом, оборудование сетевого уровня перенаправит дейтаграмму в чужую сеть. С другой стороны, желательно, чтобы отправляемая дейтаграмма осталась неизменной, так как приложению-получателю нет необходимости знать о том, что она была переадресована домашним агентом. Обе цели достигаются в случае, если исходная дейтаграмма, отправляемая корреспондентом, подвергается полной **инкапсуляции** со стороны домашнего агента, то есть передается как часть новой (более крупной) дейтаграммы. Эта большая дейтаграмма адресуется и доставляется на адрес для передачи мобильного узла. Агент чужой сети, которому «принадлежит» адрес для передачи, полученный мобильным узлом, получит и декапсулирует дейтаграмму,

иначе говоря, извлечет исходную дейтаграмму, отправленную корреспондентом из более крупной инкапсулирующей и переадресует извлеченную исходную дейтаграмму (шаг 3 на рис. 6.23) мобильному узлу.

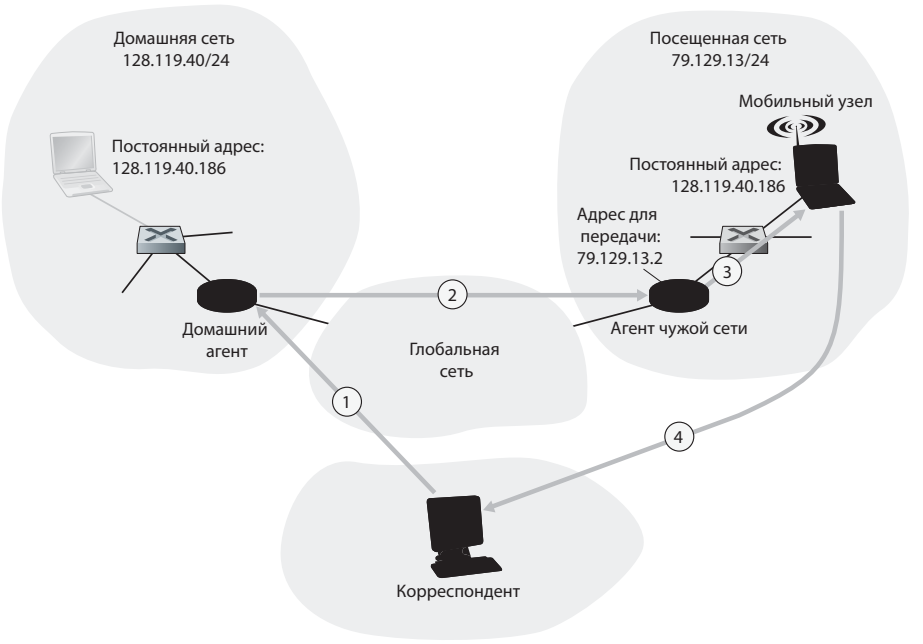


Рис. 6.23. Непрямая маршрутизация на мобильный узел

На рис. 6.24 изображена отправка корреспондентом исходной дейтаграммы в домашнюю сеть, передача инкапсулированной дейтаграммы чужому агенту и доставка исходной дейтаграммы на мобильный узел. Внимательный читатель заметит, что процессы инкапсуляции и декапсуляции, описываемые здесь, идентичны понятию туннелирования, обсуждаемого в главе 4 в контексте многоадресной передачи данных по IP и протокола IPv6.

Теперь давайте обсудим то, каким образом мобильный узел осуществляет отправку дейтаграмм корреспонденту. На самом деле все очень просто, так как мобильный узел может адресовать дейтаграммы корреспонденту *напрямую* (используя свой собственный постоянный адрес как адрес отправителя и адрес корреспондента — в качестве адреса получателя). Так как мобильный узел знает адрес корреспондента, в обратной маршрутизации дейтаграмм через домашнего агента нет необходимости. Это также отражено на рис. 6.24.

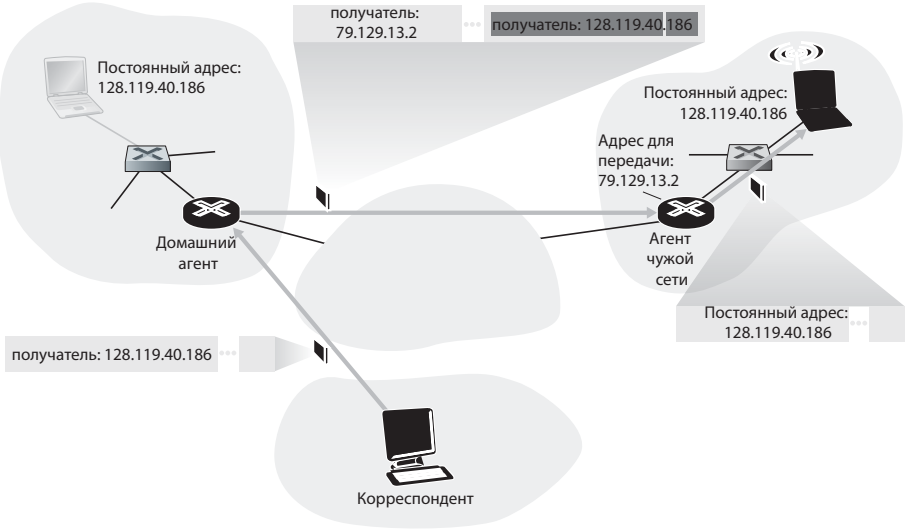


Рис. 6.24. Инкапсуляция и декапсуляция

Давайте подведем итог нашей беседе о непрямой маршрутизации, перечислив функциональность сетевого уровня, необходимую для поддержки мобильности.

- *Протокол связи мобильный узел — агент чужой сети.* Мобильный узел будет зарегистрирован агентом чужой сети при подключении к ней. Точно так же регистрация мобильного узла будет отменена, когда он покинет чужую сеть.
- *Протокол регистрации агент чужой сети — домашний агент.* Агент чужой сети произведет регистрацию адреса для передачи мобильного узла у домашнего агента. Агенту чужой сети не нужно явно отменять регистрацию у домашнего агента, когда мобильный узел покидает сеть, так как регистрация нового адреса для передачи при регистрации мобильного узла в новой сети выполнит эту задачу.
- *Протокол инкапсуляции дейтаграммы домашним агентом.* Инкапсуляция и переадресация исходной дейтаграммы корреспондента вместе с дейтаграммой, отправляемой по адресу для передачи мобильного узла.
- *Протокол декапсуляции дейтаграммы чужим агентом.* Извлечение исходной дейтаграммы корреспондента из инкапсулирующей и пересылка извлеченной дейтаграммы мобильному узлу.

Предыдущее обсуждение позволяет получить представление обо всех элементах — агентах чужой сети, домашнем агенте и непрямой переадресации, — необходимых для того, чтобы мобильный узел мог сохранять установленные соединения при перемещении из одной сети в другую. Для примера совместного использования вышеупомянутых элементов давайте представим, что мобильный узел подключен к чужой сети А, произведена регистрация у домашнего агента полученного узлом адреса для передачи, и поступающие дейтаграммы подвергаются непрямой маршрутизации через агент домашней сети. Теперь мобильный узел перемещается в чужую сеть Б и регистрируется в ней с помощью агента этой сети, сообщающего агенту домашней сети полученный узлом адрес для передачи (COA). Начиная с этого момента домашний агент будет перенаправлять получаемые дейтаграммы в чужую сеть Б. С точки зрения корреспондента мобильность является прозрачной, или невидимой, так как дейтаграммы подвергаются маршрутизации с помощью одного и того же домашнего агента: как до перемещения в другую сеть, так и после. С точки зрения домашнего агента поток дейтаграмм остается непрерывным: изначально входящие дейтаграммы пересылались в чужую сеть А, после изменения адреса для передачи — стали переадресовываться в чужую сеть Б. Однако будет ли поток дейтаграмм оставаться непрерывным с точки зрения мобильного узла при перемещении из одной сети в другую? Так как отрезок времени между отключением мобильного узла от сети А (с этого момента он больше не сможет получать дейтаграммы по сети А) и подключением в сети Б (момент регистрации мобильным узлом нового COA у домашнего агента) невелик, будет потеряно только небольшое количество дейтаграмм. Вспомните из главы 3, что одной из проблем сквозных соединений является потеря большого количества дейтаграмм в связи с перегрузкой сети. Таким образом, случайная потеря нескольких дейтаграмм при перемещении мобильного узла из одной сети в другую ни коим образом не влечет катастрофических последствий. Если есть необходимость в подключении без потерь, то целостность данных будет восстановлена с помощью механизмов верхнего уровня, независимо от того, в чем причина потери данных, будь то перегрузка сети или же мобильность пользователя. Подход с непрямой маршрутизацией применяется в стандарте мобильного IP[559], речь о котором пойдет в разделе 6.6.

Прямая маршрутизация на мобильный узел

Подход с применением непрямой маршрутизации имеет один серьезный недостаток, называемый **проблемой треугольной маршрутизации**: дейтаграммы, адресованные мобильному узлу, должны сначала быть от-

правлены (маршрутизированы) на домашний агент, а затем на чужой агент притом, что между корреспондентом и мобильным узлом существует более эффективный маршрут. В худшей из ситуаций представьте себе мобильного пользователя, решившего подключиться к сети своего коллеги. Оба сидят за одним столом напротив друг друга и обмениваются данными по сети. Дейтаграммы от корреспондента (в данном случае — коллеги посетителя) маршрутизируются на домашний агент мобильного пользователя, а затем обратно в сеть, принадлежащую его коллеге!

Прямая маршрутизация позволяет побороть неэффективность «треугольной» маршрутизации, однако делает это за счет увеличения сложности структуры. При применении подхода с прямой маршрутизацией **агент-корреспондент**, находящийся в той же сети, что и корреспондент, сначала узнает адрес для передачи мобильного узла. Это можно осуществить, если агент-корреспондент опросит домашний агент, при этом предполагается, что (как и при непрямой маршрутизации) была произведена своевременная регистрация и домашний агент обладает самой последней информацией об адресе для передачи мобильного узла. Корреспондент также может выполнять функции агента-корреспондента точно также как мобильный узел может выполнять функции агента чужой сети. Вышеуказанное отражено в шагах 1 и 2 на рис. 6.25. В этом случае агент-корреспондент отсылает дейтаграммы напрямую по адресу для передачи мобильного узла аналогично тому, как направляет данные домашний агент (шаги 3 и 4 на рис. 6.25).

Несмотря на то, что прямая маршрутизация позволяет справиться с проблемой «треугольной» маршрутизации, эта технология ставит перед нами две новые важные задачи:

- **Протокол локализации мобильного пользователя** необходим агенту-корреспонденту для опроса домашнего агента с целью получения адреса для передачи мобильного узла (шаги 1 и 2 на рис. 6.25).
- Каким образом будет осуществляться пересылка данных в случае перемещения мобильного узла из одной чужой сети в другую? В случае непрямой маршрутизации эта проблема легко решалась обновлением адреса для передачи, регистрируемого домашним агентом. Однако при прямой маршрутизации агент-корреспондент опрашивает домашний агент только один раз: в самом начале сеанса передачи данных. Поэтому обновление адреса для передачи, хранящегося в памяти домашнего агента по мере необходимости будет недостаточно для решения проблемы маршрутизации данных на мобильный узел, перешедший в новую чужую сеть.

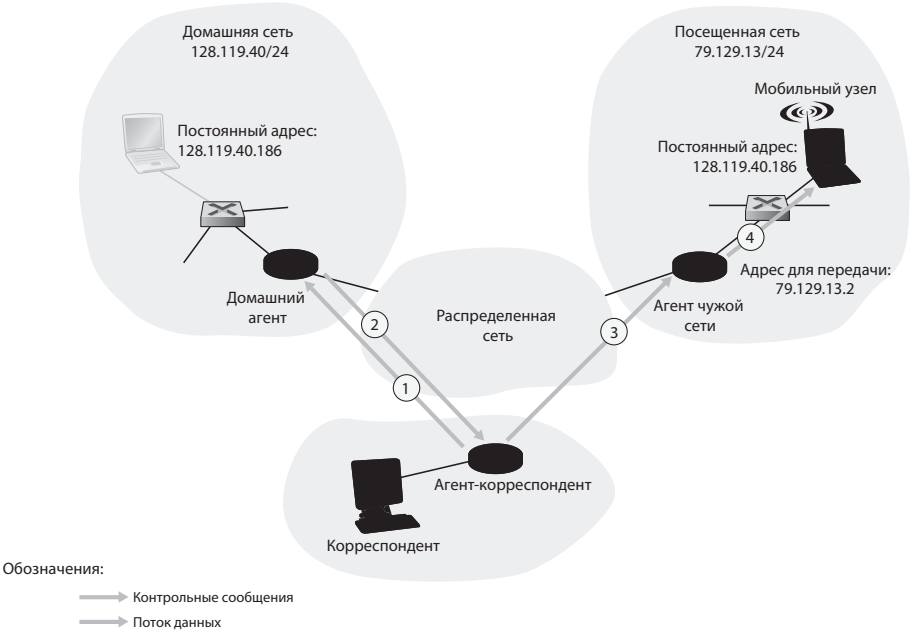


Рис. 6.25. Прямая маршрутизация данных для мобильного пользователя

Одним из возможных решений задачи было бы создание нового протокола, информирующего агент-корреспондент об изменившемся адресе для передачи. Альтернативное решение, применяемое, как вы убедитесь, на практике в сетях стандарта GSM, работает следующим образом. Предположим, что в данный момент происходит передача данных мобильному узлу через чужую сеть, в которой он был локализован в начале сеанса передачи данных (шаг 1 на рис. 6.26).

Мы будем называть агент чужой сети, в которой был впервые найден мобильный узел, **узловым агентом чужой сети**. Когда мобильный узел переходит в новую чужую сеть (шаг 2 на рис. 6.26), он регистрируется с помощью нового агента чужой сети (шаг 3), после чего тот предоставляет узловому агенту чужой сети новый адрес для передачи мобильного узла (шаг 4). Узловой агент чужой сети по получении инкапсулированной дейтаграммы для уже вышедшего из его сети мобильного узла, может совершить ре-инкапсуляцию этой дейтаграммы и переадресовать ее мобильному узлу (шаг 5) с помощью нового адреса для передачи.

Если мобильный узел с течением времени перейдет уже в третью чужую сеть, ее агент свяжется с узловым агентом чужой сети для настройки переадресации данных в новую сеть.

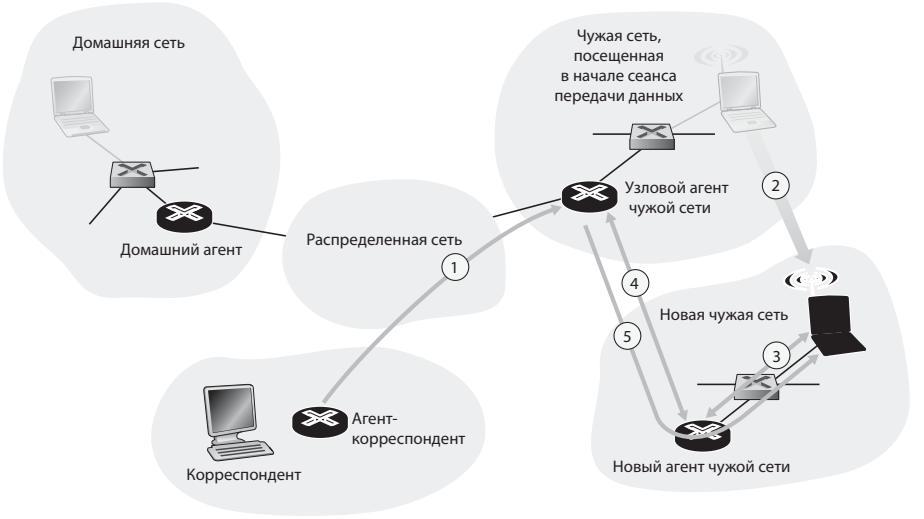


Рис. 6.26. Мобильный трансфер между сетями с помощью прямой маршрутизации

6.6. Мобильный протокол Интернета

Интернет-архитектура и протоколы для поддержки мобильности, известные как мобильный IP, определены, в первую очередь в RFC 5944[559] для IPv4. Мобильный IP представляет собой гибкий стандарт, поддерживающий несколько режимов работы (например, с агентом чужой сети или без него), несколько способов, используя которые, агенты и мобильные узлы могут обнаруживать друг друга, использование одного или нескольких адресов для передачи (COA), а также несколько форм инкапсуляции. По своей сути мобильный IP — достаточно сложный стандарт и его детальное описание потребовало бы отдельной книги, кстати, такая книга существует — ее написал Чарльз Перкинс[388]. Наша скромная цель здесь — дать краткое описание наиболее важных аспектов мобильного IP и проиллюстрировать его применение на нескольких общих примерах.

Архитектура мобильного IP включает большое количество элементов, которые мы уже обсудили ранее, в том числе концепции домашнего агента, агентов чужих сетей, адресов для передачи, а также инкапсуляции и декапсуляции. Использующийся в настоящий момент стандарт RFC 5944[559] предписывает применение непрямой маршрутизации на мобильный узел.

Стандарт мобильного IP содержит три важных составляющих:

- *Обнаружение агента.* Мобильный IP определяет протоколы, используемые домашним агентом и агентом чужой сети для информирования мобильных узлов о предоставляемых сервисах, а также протоколы для востребования мобильными узлами сервисов агентов домашней и чужой сети.
- *Регистрация у домашнего агента.* Мобильный IP определяет протоколы, используемые мобильным узлом и/или агентом чужой сети с целью регистрации или отмены регистрации адреса для передачи у агента домашней сети мобильного узла.
- *Непрямая маршрутизация дейтаграмм.* Стандарт также определяет то, каким образом домашний агент осуществляет пересылку дейтаграмм на мобильные узлы, в том числе правила переадресации дейтаграмм, правила обработки ошибок и несколько форм инкапсуляции[461, 462].

Для стандарта мобильного IP вопросы обеспечения безопасности особенно важны. Например, аутентификация мобильного узла абсолютно необходима для гарантии того, что злоумышленник не зарегистрирует у домашнего агента подложный адрес для передачи, что вызвало бы переадресацию всех дейтаграмм, предназначенных для данного IP-адреса, на устройство злоумышленника. Безопасность мобильного IP достигается с использованием большого количества механизмов, которые мы изучим в главе 8, поэтому мы не будем отдельно обращать внимание на вопросы безопасности в тексте нашего обсуждения в этой главе.

Обнаружение агента

По переходе в новую сеть, будь то подключение к чужой сети или возвращение в домашнюю, узел с мобильным IP должен «идентифицировать личность» соответствующего агента чужой или домашней сети. На самом деле это есть обнаружение нового агента чужой сети, обладающего новым сетевым адресом, позволяющем оборудованию уровня сети мобильного узла понять, что узел переместился в новую чужую сеть. Этот процесс называется **обнаружением агента**. Обнаружение агента происходит одним из двух способов: с помощью информационных сообщений агента, либо через затребование у него информации.

В случае с **информационными сообщениями** агент чужой или домашней сети распространяет («рекламирует») информацию о предо-

ставляемых им услугах, расширяя соответствующий протокол обнаружения маршрутизатора[442]. Агент периодически транслирует сообщение ICMP со значением поля типа 9 (обнаружение маршрутизатора) по всем подключенным к нему каналам связи. Это сообщение обнаружения маршрутизатора содержит IP-адрес маршрутизатора (то есть агента), позволяя таким образом мобильному узлу узнать IP-адрес агента сети. Сообщение обнаружения маршрутизатора также содержит информационное расширение мобильного агента, включающее в себя дополнительную информацию, необходимую мобильному узлу. Среди наиболее важных полей этого расширения можно выделить следующие:

- *Бит домашнего агента (H)*. Означает, что данный агент является домашним агентом сети, в которой он в данный момент находится.
- *Бит агента чужой сети (F)*. Означает, что данный агент является агентом чужой сети для сети, в которой он в данный момент находится.
- *Бит необходимости регистрации (R)*. Означает, что при подключении к данной сети мобильный пользователь *обязан* пройти регистрацию у агента этой сети. В частности это означает, что мобильный пользователь не может получить адрес для передачи в чужой сети (например, с помощью протокола DHCP) и перенять на себя функции агента чужой сети без прохождения процедуры регистрации у ее агента.
- *Биты инкапсуляции M, G*. Обозначают, будет ли использована форма инкапсуляции, отличная от инкапсуляции IP-в-IP.
- *Поля адреса для передачи (COA)*. Перечень из одного или нескольких адресов для передачи, предоставляемый агентом чужой сети. В нашем примере, приводимом ниже, адрес для передачи будет ассоциирован с агентом чужой сети, который станет получать дейтаграммы, отправляемые по этому адресу для передачи и пересылать их нужному мобильному узлу. При регистрации у домашнего агента мобильный пользователь выберет один из этих адресов в качестве своего адреса для передачи.

На рис. 6.27 приведены некоторые ключевые поля информационного сообщения агента.

В случае с **запросом информации об агенте** мобильный узел, желающий получить сведения об агенте, не дожидаясь информационного сообщения, может передать сообщение с требованием информации об

агенте, представляющее собой обычное сообщение ICMP со значением поля типа 10. Агент, получивший сообщение с требованием передачи информации о себе, направит иего напрямую мобильному узлу, который сможет продолжить выполнять необходимые действия так, будто бы получил информационное сообщение без дополнительных требований.

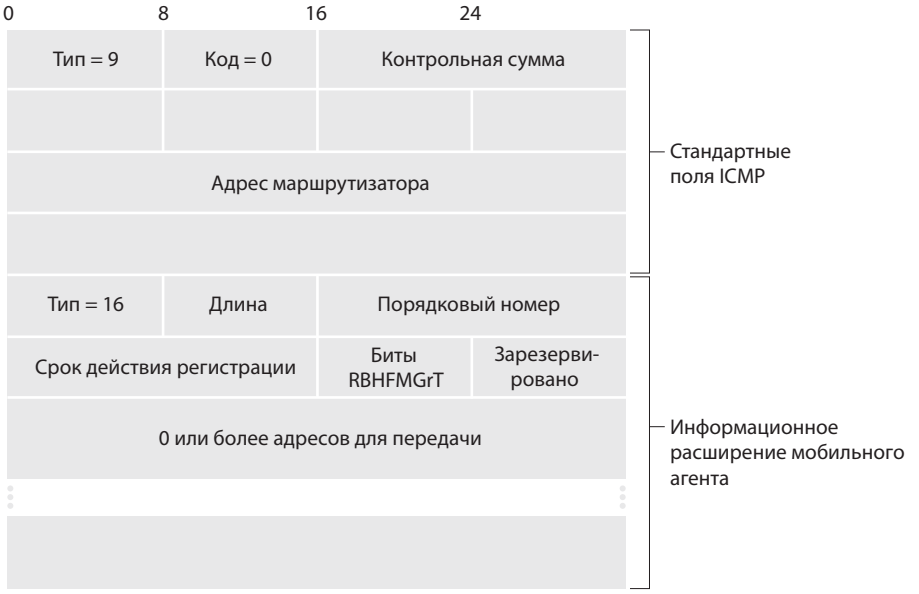


Рис. 6.27. Сообщение обнаружения маршрутизатора ICMP с расширением мобильного агента

Регистрация у домашнего агента

Как только мобильный узел получил адрес для передачи (COA), он должен зарегистрировать этот адрес у агента домашней сети. Это может быть осуществлено либо через агента чужой сети (который и произведет регистрацию COA у домашнего агента), либо непосредственно узлом с мобильным IP. Далее мы рассмотрим первый из двух вариантов. Для его осуществления потребуется пройти четыре шага.

1. Следуя инструкциям в информационном сообщении агента чужой сети, мобильный узел направляет ему регистрационное сообщение IP-адреса. Регистрационное сообщение передается внутри дейтаграммы UDP, направляемой на порт 434. Оно содержит адрес для передачи, предложенный агентом чужой сети, адрес домашнего агента (HA), постоянный адрес мобильного узла (MA) желаемый срок дей-

ствия регистрации, 64-битный регистрационный идентификатор. Желаемый срок регистрации — это значение времени в секундах, на протяжении которой регистрация в сети будет оставаться действительной. Если регистрация не обновляется на домашнем агенте в указанный период времени, она перестает быть действительной. Регистрационный идентификатор играет роль порядкового номера, используемого для соотнесения полученного регистрационного ответа с регистрационным требованием (см. ниже).

2. Агент чужой сети получает регистрационное сообщение и записывает постоянный IP-адрес мобильного узла. Теперь агент этой сети знает, что он должен выполнять поиск дейтаграмм, содержащих инкапсулированные дейтаграммы, адрес назначения которых совпадает с постоянным адресом мобильного узла. После этого агент чужой сети направляет регистрационное сообщение мобильного IP-адреса (опять же внутри дейтаграммы UDP) на порт 434 домашнего агента. Это сообщение содержит адрес для передачи, HA, MA, требуемый формат инкапсуляции, требуемый срок действия регистрации и регистрационный идентификатор.

3. Домашний агент получает регистрационное требование и проверяет его аутентичность и правильность. После этого домашний агент связывает постоянный IP-адрес мобильного узла с адресом для передачи. В будущем дейтаграммы, поступающие на домашний агент и адресованные мобильному узлу, будут инкапсулированы и отправлены посредством туннелирования на адрес для передачи. Домашний агент также отсылает ответ о регистрации мобильного IP-адреса, содержащий HA, MA, реальный срок действия регистрации, регистрационный идентификатор запроса, который был подтвержден данным ответом.

4. Агент чужой сети получает регистрационный ответ и пересылает его мобильному узлу.

На данном этапе регистрация завершена и мобильный узел может получать дейтаграммы, отправленные на его постоянный адрес. Вышеописанные шаги проиллюстрированы на рис. 6.28. Обратите внимание, что домашний агент указывает срок действия регистрации меньший, чем был запрошен мобильным узлом.

Агент чужой сети не должен явно отменять регистрацию адреса для передачи, когда мобильный узел покидает его сеть. Отмена регистрации произойдет автоматически, когда мобильный узел перейдет в новую

сеть (независимо от того, будет ли это еще одна чужая сеть либо домашняя) и зарегистрирует новый адрес для передачи.

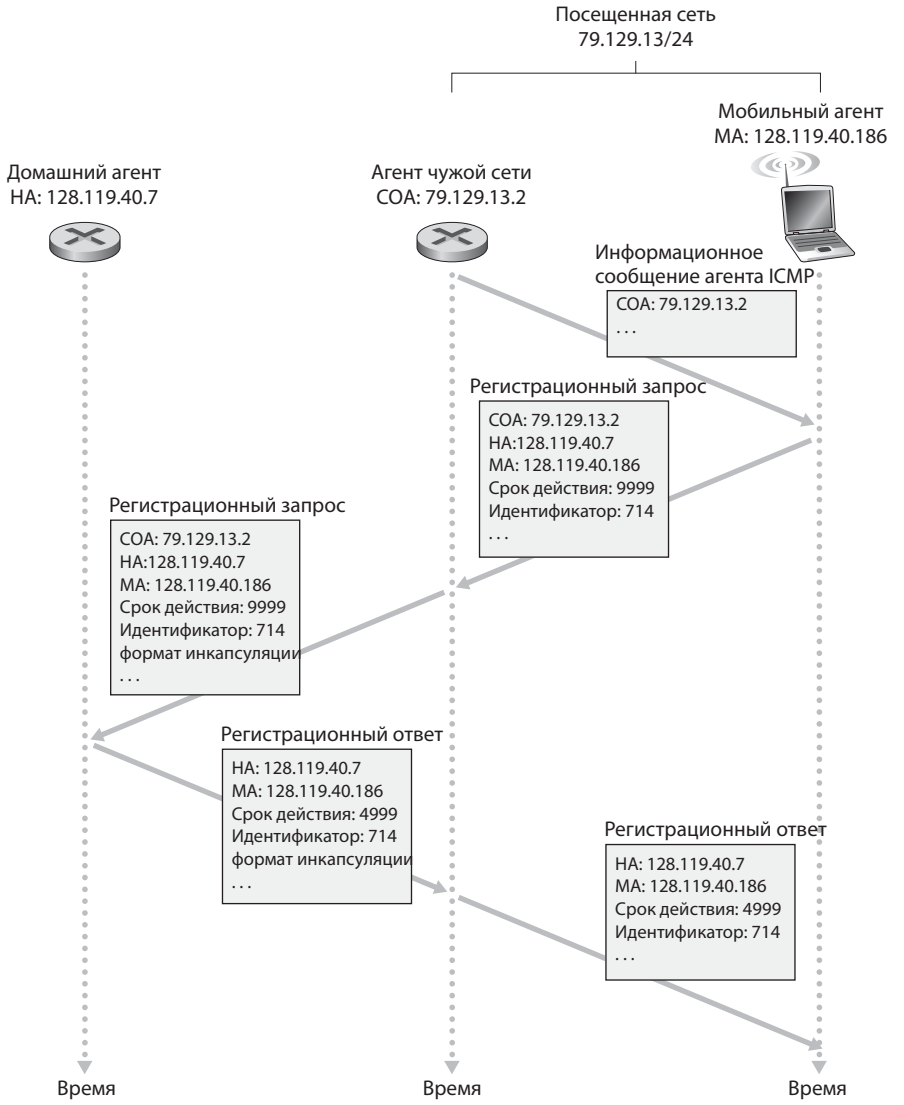


Рис. 6.28. Информационное сообщение агента и регистрация мобильного IP

Стандарт мобильного IP допускает наличие множества дополнительных алгоритмов и возможностей вдобавок к описанным выше. Заинтересованный читатель может получить информацию в книге Перкинса[388] спецификации RFC 5944[559].

6.7. Управление мобильными коммуникациями в сетях сотовой связи

Изучив то, каким образом осуществляется управление мобильными коммуникациями в IP-сетях, давайте теперь переключим свое внимание на сети с еще более давней историей поддержки мобильности — сети сотовой радиотелефонной связи. Так как в разделе 6.4 мы сосредоточились на беспроводных каналах связи «первого перехода» сотовых сетей, здесь мы обратимся к обеспечению собственно мобильности, используя для примера архитектуру мобильных сетей стандарта GSM[185, 270, 295, 595, 641, 355], так сети этого типа представляют собой зрелую и повсеместно применяемую технологию. Как и в случае с протоколом мобильного IP, мы увидим, что большое количество фундаментальных принципов, выделенных нами в разделе 6.5, нашли свое воплощение и в архитектуре сетей стандарта GSM.

Как и в стандарте мобильного IP, в технологии GSM используется подход с непрямой маршрутизацией (см. раздел 6.5.2), при этом вызов корреспондента сначала адресуется на домашнюю сеть мобильного абонента, а затем, оттуда, — на посещенную сеть. В терминологическом аппарате технологии GSM домашнюю сеть мобильного пользователя принято называть **домашней мобильной сетью общего пользования** (иначе, **домашняя PLMN**). Так как аббревиатура «PLMN» довольно труднопроизносима и, памятуя о нашей задаче избежать каши из различных аббревиатур и акронимов, далее по тексту домашнюю сеть PLMN мы будем называть просто **домашней сетью**. Домашняя сеть — это оператор сотовой связи, к которым у абонента заключен договор на обслуживание (то есть оператор, выставляющий абоненту ежемесячный счет на оплату услуг сотовой связи). Посещенная сеть PLMN, которую мы будем называть просто **посещенной сетью**, — это сеть, где абонент находится в данный момент времени.

Как и в случае с технологией мобильного IP, сферы ответственности домашней и посещенной сетей значительно отличаются.

- Домашняя сеть обладает базой данных, называемой **регистром собственных абонентов** (home location register, **HLR**) и содержащей постоянный номер сотового телефона и информацию о каждом из абонентов. Очень важно и то, что в регистре присутствует также информация о местоположении абонентов. Таким образом, если какой-либо из абонентов находится в роуминге в сети другого оператора, регистр содержит достаточное количество информации для получения

(этот процесс мы также вскоре опишем) адреса посещенной сети, на которую должна быть произведена маршрутизация входящего вызова мобильному абоненту. Как мы увидим, при вызове мобильного абонента корреспондент связывается со специальным коммутатором, называемым **шлюзовым коммутационным центром мобильной связи** (Gateway Mobile services Switching Centre, **GMSC**). Опять во избежание путаницы с аббревиатурами, мы будем называть GMSC более описательно: **домашний КЦМС**.

- Посещенная сеть обладает базой данных, называемой **регистром роуминговых абонентов** (visitor location register, **VLR**). Этот регистр содержит запись о каждом абоненте, находящемся *в данный момент времени* в одном из участков сети, обслуживаемом VLR. Таким образом, записи регистра роуминговых абонентов появляются и исчезают по мере подключения и отключения таких абонентов от сети. Обычно регистр роуминговых абонентов совмещается с коммутационным центром мобильной связи (КЦМС), управляющим установлением входящего и исходящего вызова через посещенную сеть.

На самом деле, сеть оператора сотовой связи одновременно является домашней сетью для абонентов, с которыми заключен договор на обслуживание и посещенной сетью для абонентов прочих операторов сотовой связи.

6.7.1. Маршрутизация вызовов мобильного абонента

Сейчас мы дошли до того момента, когда пора объяснить, каким образом совершается вызов мобильного абонента сети стандарта GSM в посещенной сети. Далее мы рассмотрим простой пример, более сложные алгоритмы описаны в работе Моули[355]. На рис. 6.29 отражены следующие шаги:

1. Корреспондент набирает номер мобильного абонента. Этот номер, сам по себе, не имеет никакого отношения к какой-то конкретной телефонной линии или местоположению (в конце концов, телефонный номер фиксирован, а пользователь — мобилен!). Первых цифр номера достаточно для глобальной идентификации домашней сети мобильного абонента. Маршрутизация вызова производится от корреспондента через коммутируемую телефонную сеть общего пользования на домашний КЦМС в домашней сети мобильного абонента. Это лишь первый этап установления вызова.

2. Домашний КЦМС получает вызов и связывается с регистром собственных абонентов сети, чтобы определить местоположение нужного мобильного абонента. В самом простом случае, регистр возвращает **номер мобильной станции в роуминге** (Mobile Station Roaming Number, **MSRN**), который мы будем называть просто **номером в роуминге**. Обратите внимание, что этот номер отличается от постоянного номера, ассоциированного с домашней сетью абонента. Номер в роуминге эфемерен: он присваивается абоненту при подключении к посещенной сети. Роль номера в роуминге подобна роли адреса для передачи в технологии мобильного IP, и, также как и COA, он является невидимым и для корреспондента, и для мобильного абонента. Если в регистре собственных абонентов нет номера в роуминге, то система возвращает адрес регистра роуминговых абонентов посещенной сети. В этом случае (не отражено на рис. 6.29) домашнему КЦМС потребуется направить запрос в регистр роуминговых абонентов (VLR) для получения номера в роуминге мобильного узла. Но каким образом регистр собственных абонентов (HLR) получает номер в роуминге или адрес VLR? Что происходит с этими значениями, когда абонент перемещается в другую посещенную сеть? Вскоре мы дадим ответы на эти важные вопросы.

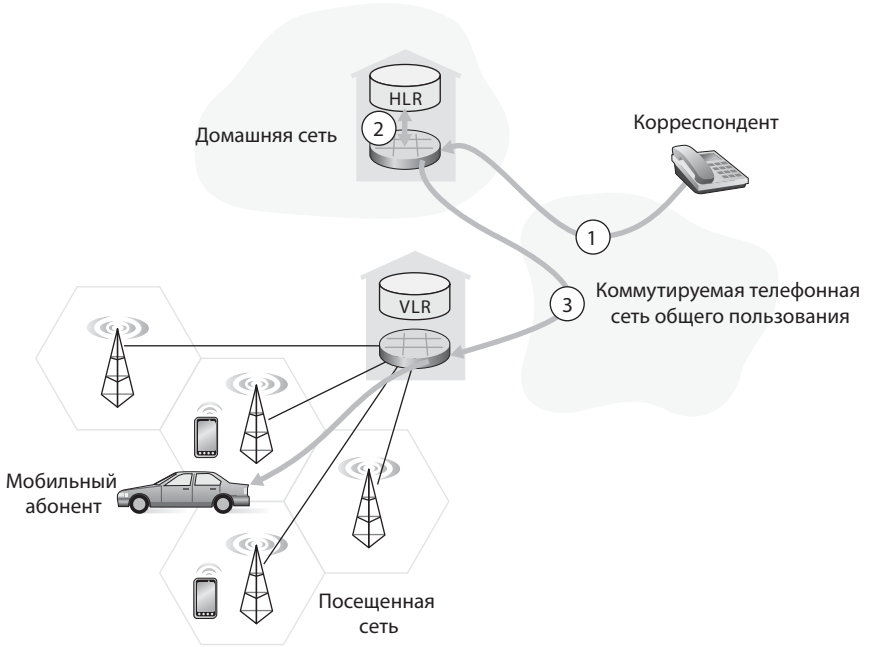


Рис. 6.29. Вызов мобильного абонента: непрямая маршрутизация

3. Получив номер в роуминге, домашний КЦМС устанавливает второй этап вызова через сеть на КЦМС посещенной сети. Вызов устанавливается, будучи маршрутизирован от корреспондента на домашний КЦМС и оттуда на КЦМС посещенной сети, и уже оттуда — на базовую станцию, обслуживающую нужного мобильного пользователя.

В шаге 2 остался без ответа вопрос о том, как регистр собственных абонентов получает информацию о местоположении мобильного абонента. Мобильный абонент должен произвести регистрацию всякий раз при включении телефона или попадании в участок посещенной сети, обслуживаемый новым регистром роуминговых абонентов. Эта операция производится с помощью обмена сигнальными сообщениями между мобильным устройством и регистром роуминговых абонентов. В свою очередь регистр посещенной сети посылает сообщение-запрос обновления местоположения в регистр собственных абонентов домашней сети. Это сообщение передает в регистр домашней сети либо номер абонента в роуминге, по которому с ним может быть установлена связь, либо адрес VLR (в который также может быть позднее направлен запрос на получение номера абонента). Частью такого обмена является получение регистром роуминговых абонентов (VLR) информации об абоненте из регистра собственных абонентов (HLR) с целью определения того, какие услуги должны быть предоставлены абоненту сетью (если вообще обслуживание абонента должно производиться).

6.7.2. Эстафетные передачи в сетях GSM

Процесс **эстафетной передачи** происходит, когда мобильная станция отключается от одной станции и подключается к другой. Как показано на рис. 6.30, вызов абонента изначально (до передачи) маршрутизируется через одну базовую станцию (которую мы будем называть «старой»), а после передачи маршрутизация происходит через другую базовую станцию (которую мы будем называть «новой»). Обратите внимание, что результат передачи абонента одной базовой станции другой состоит не только в переключении приема и передачи данных на новую базовую станцию, но также и в переключении на нее маршрутизации текущих соединений. Сначала давайте предположим, что старая и новая базовые станции обслуживаются одним и тем же КЦМС, и перенаправление осуществляется на нем.

Существует несколько причин для возникновения процесса передачи, в том числе (1) если уровень сигнала между текущей базовой стан-

цией и мобильным устройством снизился настолько, что установленный вызов оказался под угрозой обрыва, и (2) если сота могла оказаться перегруженной в результате обработки большого количества вызовов. Влияние перегрузки может быть сглажено с помощью передачи абонентских мобильных станций в менее загруженные соседние соты.

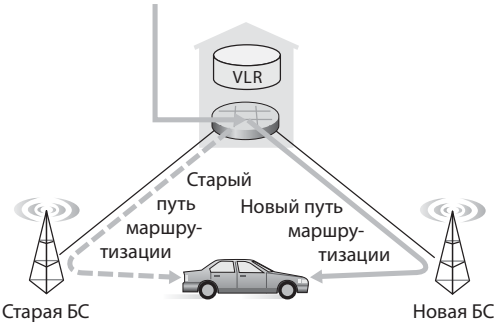


Рис. 6.30. Алгоритм эстафетной передачи между двумя базовыми станциями, обслуживающимися единым КЦМС

Будучи ассоциировано с базовой станцией, мобильное устройство периодически измеряет силу сигнала-маяка от текущей базовой станции, а также прослушивающихся сигналов-маяков соседних базовых станций. Результаты измерений передаются текущей базовой станции один или два раза в секунду. В сетях GSM процесс передачи инициируется старой базовой станцией в зависимости от полученных результатов измерений, загрузки близлежащих сот и других факторов[355]. Стандарт GSM не определяет каких-либо конкретных алгоритмов, используемых базовыми станциями для определения того, нужно ли производить передачу абонента.

На рис. 6.31 изображены шаги алгоритма, если базовая станция все-таки решит произвести передачу мобильного абонента:

1. Старая базовая станция (БС) информирует коммутационный центр посещенной сети в том, что начата процедура передачи, а также отправляет информацию о базовой станции (или, по возможности, набора БС), которой (или которым) будет передан абонент.
2. КЦМС посещенной сети инициирует установку пути к новой БС, выделяя ресурсы, необходимые для поддержания перенаправленного вызова и оповещая новую БС о начале процедуры передачи.
3. Новая базовая станция выделяет и активирует радиоканал для использования с мобильным устройством.

4. Новая базовая станция информирует КЦМС и старую БС о том, что путь КЦМС-новая-БС был успешно установлен и мобильный узел должен быть оповещен о предстоящей передаче. Новая БС предоставляет мобильному узлу всю информацию, необходимую для подключения к новой БС.
5. Мобильный узел получает информацию о необходимости переподключения. Обратите внимание, что до этого момента мобильный узел был в полном и счастливом неведении о работе, произведенной сетью (в т.ч. выделение канала связи новой БС и выделение пути от КЦМС посещенной сети к новой БС) для осуществления передачи.
6. Мобильный узел и новая БС обмениваются одним или несколькими сообщениями для полной активации канала связи с новой БС.
7. Мобильный узел направляет новой БС сообщение с информацией об успешном завершении процедуры передачи, которое в последствие пересылается КЦМС посещенной сети. КЦМС посещенной сети затем перенаправляет установленный вызов мобильному узлу через новую БС.
8. Ресурсы, выделенные по пути к старой БС, освобождаются.

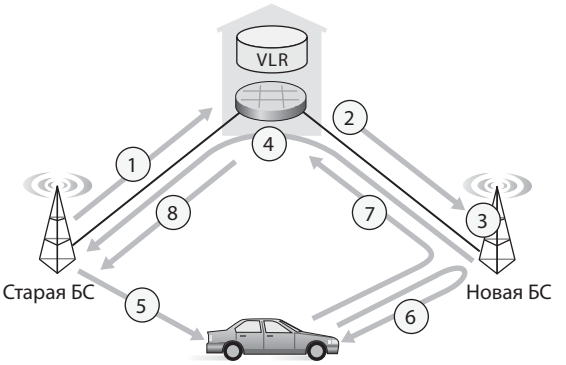
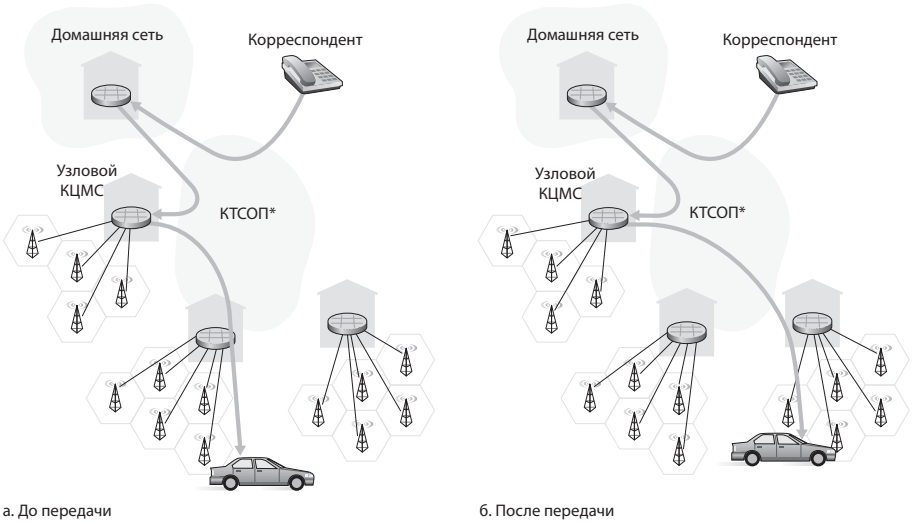


Рис. 6.31. Шаги процедуры эстафетной передачи между базовыми станциями, обслуживающимися единым КЦМС

Давайте завершим наше описание процесса передачи, обсудив, что происходит, когда мобильный узел подключается к базовой станции, ассоциированной с КЦМС, *отличным* от КЦМС старой базовой станции, а также если такая меж-коммутационная передача осуществляется более одного раза. Как показано на рис. 6.32, в стандарте GSM определяется понятие **узлового КЦМС**. Узловым КЦМС называется комму-

тационный центр, посещаемый мобильным узлом при первичном установлении вызова, таким образом, узловой КЦМС остается неизменным до момента завершения вызова. На протяжении всего времени вызова и вне зависимости от количества осуществленных межкоммутационных передач, маршрутизация вызова осуществляется от домашнего КЦМС на узловой, а затем — на КЦМС посещенной сети, к которой подключен абонент в данный момент времени. Когда мобильный абонент переходит из зоны покрытия одного КЦМС в зону покрытия другого, установленный вызов перенаправляется от узлового КЦМС на новый, обслуживающий новую базовую станцию. Таким образом, максимально возможное число коммутационных центров, расположенных на пути между корреспондентом и мобильным узлом, равно трем (домашний, узловой и КЦМС посещенной сети). На рис. 6.32 изображена маршрутизация вызова между КЦМС, посещенными мобильным абонентом.



а. До передачи

б. После передачи

Рис. 6.32. Перенаправление через узловой КЦМС

Существует альтернативный подход к обслуживанию одного скачка КЦМС от узлового до текущего, состоящий в простом связывании в цепочку всех КЦМС, посещенных абонентом, при этом старый КЦМС переводит установленный вызов на новый КЦМС каждый раз, когда абонент покидает зону ответственности текущего КЦМС. На самом деле такое цепное связывание может происходить в сотовых сетях IS-41, где

* КТСОП — коммутируемая телефонная сеть общего пользования. — *Примеч. пер.*

также бывает предусмотрен дополнительный шаг упрощения, позволяющий убрать все КЦМС, расположенные между узловым и текущим КЦМС[321].

Давайте свернем наш разговор про управление мобильностью в сетях GSM, сравнив эту функцию в сетях стандартов GSM и Мобильный IP. Сравнение, приведенное в табл. 6.2, показывает, что несмотря на ряд фундаментальных различий между сотовыми сетями и сетями IP, в них применяется удивительно большое количество сходных функциональных элементов и общих подходов управления мобильностью.

Табл. 6.2. Сходства между мобильным IP и мобильностью в GSM

Элемент сети стандарта GSM	**Комментарий к элементу GSM**	**Элемент сети стандарта Мобильный IP**
Домашняя система	Сеть, которой принадлежит постоянный абонентский номер мобильного пользователя	Домашняя сеть
Шлюзовой коммутационный центр мобильной сети, или просто домашний КЦМС, Регистр собственных абонентов (HLR)	Домашний КЦМС: точка контакта, позволяющая получить маршрутизируемый адрес мобильного пользователя. HLR: база данных в домашней системе, содержащая постоянный абонентский номер, личную информацию, текущее местоположение абонента и информацию о подписке	Домашний агент
Посещенная система	Сеть, к которой подключен абонент в настоящее время, отличная от его домашней сети	Посещенная сеть
Коммутационный центр посещенной мобильной сети, Регистр роуминговых абонентов (VLR)	Посещенный КЦМС: отвечает за установление входящих и исходящих вызовов мобильных узлов, находящихся в сотах, ассоциированных с КЦМС. VLR: временная запись в базе данных посещенной системы, содержащая информацию о подписке каждого абонента в роуминге	
Номер мобильной станции в роуминге (MSRN), или просто номер в роуминге	Маршрутизируемый адрес сегмента сети между домашним и посещенным КЦМС, необходимый для установления вызова, но не видимый ни мобильному узлу, ни корреспонденту	Адрес для передачи

6.8. Беспроводная связь и мобильность: влияние на протоколы верхних уровней

В этой главе мы увидели, что беспроводные сети значительно отличаются от своих проводных аналогов, как на канальном уровне (в ре-

зультате некоторых характеристик беспроводного канала связи, таких как ослабление сигнала, многолучевое распространение и проблема «скрытого передатчика»), так и на сетевом (в результате того, что мобильные пользователи меняют точки подключения к сети). Но есть ли существенные отличия на транспортном и прикладном уровне? Очень хочется верить, что различия на этих уровнях будут минимальны, так как сетевой уровень предоставляет такую же модель сервиса верхних уровней с доставкой данных при минимизации потерь. Аналогично если такие протоколы как TCP и UDP предоставляют приложениям сервисы транспортного уровня, то прикладной уровень также должен остаться неизменным. С одной интуитивной точки зрения это верно: протоколы TCP и UDP способны работать (и работают) в сетях с беспроводными каналами связи. С другой стороны, транспортные протоколы в общем и протокол TCP в частности могут иметь различную производительность в проводных и беспроводных сетях, и именно здесь, в производительности мы найдем основное отличие. Давайте посмотрим, почему.

Вспомним, что протокол TCP повторно передает сегмент, который был либо потерян, либо поврежден на пути от отправителя к получателю. В случае с мобильными пользователями потеря может произойти как из-за перегрузки сети (переполнение буфера маршрутизатора), так и в силу особенностей процесса передачи (то есть от задержек в сегментах перенаправления до перемещений точек подключения мобильного узла). Во всех случаях ACK получатель-отправитель протокола TCP показывает, что сегмент не был получен без повреждений. Отправителю не известно, был ли сегмент потерян из-за перегрузки, во время передачи мобильного узла или же из-за обнаруженных битовых ошибок. Во всех случаях ответ отправителя один: повторить отправку сегмента данных. Ответ системы контроля перегрузок протокола ТСП *также* всегда один: уменьшить окно перегрузки (как обсуждалось в разделе 3.7). Безусловно уменьшая окно перегрузки, протокол TCP имплицитно предполагает, что потеря сегментов данных является результатом перегрузки сети, но не повреждения данных или процесса передачи. В разделе 6.2 мы убедились, что битовые ошибки случаются гораздо чаще в беспроводных сетях, чем в проводных. Когда происходят такие битовые ошибки, или потери являются результатом процесса передачи мобильного узла отправителю, TCP на самом деле нет причин уменьшать окно перегрузки (и, соответственно, уменьшать скорость отправки данных). На самом деле, вполне может быть, что буферы маршрутизаторов пусты, и пакеты данных проходят по сети из конца в конец беспрепятственно и без перегрузок сети.

В начале-середине 1990-х годов исследователи поняли, что при высоких уровнях битовых ошибок беспроводных каналов связи и возможности потерь при передаче мобильного узла ответ контроля перегрузок протокола TCP может создавать проблемы для настроек беспроводной связи. Существует три основных класса подходов к решению этой проблемы:

- *Локальное восстановление*. Протоколы локального восстановления восстанавливают данные от битовых ошибок в то время и в том месте (например, в беспроводном канале связи), где они произошли. Например, протокол 802.11 ARQ, который мы изучили в разделе 6.3 или более сложные подходы, использующие одновременно и ARQ и FEC[32].
- *Предупреждение отправителя TCP о наличии беспроводных каналов связи*. В подходах с применением локального восстановления отправитель TCP остается в счастливом неведении относительно того, что отправляемые им сегменты данных проходят по беспроводному каналу связи. Альтернативный подход заключается в информировании и получателя и отправителя TCP о наличии беспроводных каналов связи, чтобы отличать потери-результаты перегрузки сети, случающиеся в проводных сетях от повреждений/потерь данных, случающихся в беспроводных каналах связи, и включать системы контроля перегрузок только в ответ на потери, вызванные перегрузкой проводных сетей. Балакришнан[34] исследует разные типы протокола TCP, предполагая, что конечные системы могут определять отличия. Лью[324] изучает техники различия потерь на проводных и беспроводных участках пути от отправителя к получателю.
- *Подходы раздельного подключения*. При применении раздельного подключения[33] соединение из конца в конец между мобильным пользователем и другим концом сети разделяется на два подключения транспортного уровня: одно от мобильного хоста до беспроводной точки доступа и другое — от беспроводной точки доступа к другому концу сети (который, мы предположим, является проводным хостом). Таким образом, соединение из конца в конец формируется в результате связывания беспроводной и проводной частей. Транспортный уровень беспроводного сегмента может быть представлен стандартным TCP соединением[33], либо специально созданным протоколом восстановления ошибок над протоколом UDP. Яваткар[679] изучает то, как протокол выборочного повторения транспортного уровня используется для беспроводного подключения. Результаты измерений, опубликованные в работе Вея[662] показывают, что разделенные TCP-подключения широко применяются в сотовых сетях передачи

данных и то, что значительные улучшения могут быть на самом деле достигнуты с использованием разделенного TCP-подключения.

Наше обсуждение протокола TCP в беспроводных каналах связи было достаточно кратким, что вызвано определенной необходимостью, однако с более подробными исследованиями задач и решений TCP в беспроводных сетях можно ознакомиться в работах Ханабали[201] и Льенга[317]. Мы рекомендуем вам обратиться к справочникам, чтобы больше узнать об этой сфере непрекращающихся исследований.

Обсудив протоколы транспортного уровня, давайте перейдем к разговору о влиянии беспроводных и мобильных технологий на протоколы прикладного уровня. Здесь важным замечанием будет то, что ширина беспроводных каналов связи относительно невелика, как мы могли видеть на рис. 6.2. Поэтому приложения, работающие с беспроводными каналами связи и в частности с беспроводными каналами сотовой связи, должны относиться к ширине канала как к дефицитному благу. Например, веб-сервер, предоставляющий запущенному на телефоне поколения 3G веб-браузеру определенный контент, вероятней всего не сможет передать ему столь же насыщенное изображениями содержание страницы, как если бы браузер работал с проводным подключением. Несмотря на то, что беспроводные каналы связи на самом деле ставят несколько вызовов на прикладном уровне, благодаря мобильности они также позволяют создавать и эксплуатировать богатый набор приложений, обладающих функциями вывода результатов в зависимости от контекста и местоположения пользователя[88, 36]. Говоря более общими фразами, беспроводные и мобильные сети будут играть ключевую роль в реализации безграничной компьютерной среды будущего[664]. Будет справедливо также отметить, что, разговаривая о влиянии беспроводных и мобильных сетей на сетевые приложения и используемые ими протоколы, мы с вами увидели лишь верхушку айсберга!

6.9. Заключение

Беспроводные и мобильные сети стали революционным явлением в мире телефонной связи, кроме того, они оказывают серьезнейшее влияние также и на компьютерные сети. Обладая инфраструктурой, позволяющей получить неограниченный доступ к глобальной сети в любое время и в любом месте они не только делают сетевой доступ более свободным, но и позволяют использовать большое количество новых восхитительных

сервисов, связанных с геолокацией. Памятуя о постоянно растущей значимости беспроводных и мобильных сетей, в этой главе мы сфокусировались на принципах, общих технологиях связи и сетевых архитектурах, используемых для поддержки беспроводного и мобильного сообщения.

Мы начали эту главу с введения в тему беспроводных и мобильных сетей, обрисовав важнейшие отличия между задачами, которые ставит перед нами *беспроводная* природа каналов связи таких сетей и *мобильность*, ставшая возможной благодаря этим каналам связи. Такой подход позволил нам более точно изолировать, идентифицировать и изучить ключевые концепты данной сферы. Мы сосредоточили свое внимание на беспроводных коммуникациях, обсудив характеристики беспроводных каналов связи в разделе 6.2. В разделах 6.3 и 6.4 рассмотрены аспекты канального уровня стандарта беспроводной локальной сети IEEE 802.11 (Wi-Fi), двух видов персональных сетей (Bluetooth и Zigbee), доступ в Интернет по сотовым сетям третьего (3G) и четвертого (4G) поколений. После этого мы переключили свое внимание на проблемы обеспечения мобильности. В разделе 6.5 мы выделили несколько форм мобильности, а также проблемы и задачи, которые ставят перед нами отличительные особенности этих форм мобильности и ряд путей их решения. Мы обсудили проблемы поиска местоположения пользователя и маршрутизации данных на его мобильное устройство, а также различные подходы к передаче пользователя при его динамичном перемещении от одной точки подключения к сети к другой. Мы изучили то, каким образом решаются эти проблемы в стандарте GSM и Мобильный IP в разделах 6.6 и 6.7 соответственно. Наконец, мы ознакомились с влиянием, оказываемым беспроводными каналами связи и мобильностью на протоколы транспортного уровня и сетевые приложения в разделе 6.8.

Несмотря на то, что исследованию мобильных и беспроводных сетей мы посвятили целую главу, чтобы всесторонне охватить эту восхитительную и быстро расширяющуюся сферу потребовалось бы написать целую книгу (или даже не одну). Для углубленного изучения данной области мы бы рекомендовали обратиться к литературе, перечисленной в этой главе.

Глава 7
МУЛЬТИМЕДИЙНЫЕ СЕТЕВЫЕ ТЕХНОЛОГИИ

В наши дни люди со всех уголков мира используют Интернет для просмотра художественных фильмов и телевизионных сериалов. Компании, которые занимаются распространением мультимедийной продукции (например, Netflix и Hulu в Северной Америке, Youku и Kankan в Китае), практически у всех на слуху. Но потребление видео — это еще не все; многие люди создают собственные ролики, загружая их на такие веб-сайты, как YouTube. Кроме того, сетевые приложения вроде Skype, Google Talk и QQ (невероятно популярное в Китае) не просто позволяют делать «телефонные вызовы» по Интернету, но и дают возможность дополнять их живым видеорядом, а также организовывать видеоконференции с несколькими участниками. С большой долей уверенности можно сказать, что к концу этого десятилетия индустрии видеоразвлечений и голосовой связи полностью переберутся в Интернет и в большинстве случаев будут работать на беспроводных устройствах, подключенных через сети 4G и Wi-Fi.

Мы начнем эту главу с систематизации мультимедийных приложений (см. раздел 7.1). Вы увидите, что у них может быть разное назначение: *потоковое вещание хранимых аудио- и видеоданных*, *IP-телефония и видеосвязь*, а также *аудио- и видеовещание в реальном времени*. Вы убедитесь в том, что у всех задач, перечисленных выше, есть свои уникальные требования, которые значительно отличаются от требований традиционных «эластичных» приложений, таких как электронная почта, веб-сайты и средства удаленного администрирования. В разделе 7.2 мы подробно рассмотрим потоковое вещание видеоданных, изучим принципы, на которых оно основано (включая буферизацию на клиентской стороне, предварительную загрузку и изменение качества видео в зависимости от пропускных возможностей канала). Мы также изучим принцип работы сетей распространения контента (Content Distribution Network или CDN), которые в наши дни широко используются ведущими системами потокового видеовещания. В качестве примеров возьмем такие сервисы, как YouTube, Netflix и Kankan. Раздел 7.3 посвящен го-

лосовой и видеосвязи, которые, в отличие от «эластичных» приложений, очень чувствительны к задержкам в сети, но терпимы/устойчивы к периодическим потерям данных. Здесь вы узнаете, как с помощью таких методик как адаптивное воспроизведение, прямая коррекция и маскирование ошибок, можно смягчить последствия задержек и потери пакетов в сети. Примером нам послужит приложение Skype. В разделе 7.4 мы поговорим о RTP и SIP — популярных сетевых протоколах для голосового и видеообщения в режиме реального времени. В разделе 7.5 будут рассмотрены сетевые механизмы, с помощью которых можно различать разные виды трафика (например, отделять голосовые вызовы, чувствительные к задержкам, от «эластичных» приложений, занимающихся просмотром веб-страниц), обеспечивая дифференцированный подход к каждому из них.

7.1. Мультимедийные сетевые приложения

Мультимедийными мы называем любые сетевые приложения, в которых используются аудио- или видеоданные. В этом разделе мы займемся их систематизацией. Вы увидите, что каждый класс приложений имеет свои уникальные требования и проблемы, связанные с проектированием. Но прежде чем приступать к обсуждению мультимедийных Интернет-сервисов, давайте сначала рассмотрим особенности, присущие аудио- и видеоданным.

7.1.1. Свойства видеоданных

Наверное, самой заметной характеристикой видеоданных является **высокая интенсивность передачи информации (высокий битрейт)**. Видео, распространяемое в Интернете, обычно имеет скорость передачи от 100 кбит/с (низкое качество, используется в видеоконференциях) до 3 Мбит/с (видеопоток высокой четкости). Чтобы понять, насколько высокие требования к пропускным способностям сети предъявляет видео по сравнению с другими приложениями, давайте возьмем для примера трех пользователей, которые работают с тремя разными Интернет-сервисами. Наш первый пользователь, Фрэнк, просматривает фотографии, которые его друзья разместили на страницах сайта Facebook. Предположим, что он открывает новую фотографию раз в 10 с, а средний размер изображений равен 200 кбайт (чтобы вам было легче ориентироваться в цифрах, мы, как обычно, исходим из того, что 1 кбайт

равен 8000 бит). Наш второй пользователь, Марта, слушает «облачную» Интернет-радиостанцию на своем смартфоне. Будем считать, что радиостанция непрерывно транслирует музыку в формате MP3 со скоростью передачи 128 кбит/с. Третий пользователь по имени Виктор смотрит фильм, закодированный в потоке шириной 2 Мбит/с. Давайте предположим, что каждый из них использует Интернет на протяжении 4000 с (это примерно 67 мин). В табл. 7.1 приводится сравнение скорости передачи данных и общее количество переданной информации для всех трех пользователей. Как видно, скорость передачи данных во время просмотра видео более чем в десять раз превышает аналогичные показатели при потоковом вещании музыки и использовании веб-сайта Facebook. Таким образом, проектируя сетевое видеоприложение, вы должны помнить о высоких требованиях к пропускной способности сети. Учитывая это, а также растущую популярность видеосервисов, стоит ли удивляться, что согласно отчету компании Cisco за 2011 год потоковое вещание и загрузка видеоданных должны занимать примерно 90% глобального трафика в Интернете уже к 2015 году?

Табл. 7.1. Сравнение приложений с точки зрения требований к пропускной способности сети

	Скорость передачи	**Данные, переданные за 67 мин**
Фрэнк (Facebook)	160 кбит/с	80 Мбайт
Марта (музыка)	128 кбит/с	64 Мбайт
Виктор (видео)	2 Мбит/с	1 Гбайт

Еще одной важной характеристикой видеоданных является тот факт, что их можно сжимать, и это позволяет добиться разумного компромисса между качеством и размером. Видео — это последовательность изображений, которые обычно выводятся с постоянной частотой (например, 24 или 30 кадров в секунду). До сжатия цифровое изображение представляет собой массив точек, яркость и цвет каждой из которых представлены довольно большим количеством битов. Существует два вида избыточности видеоданных, которые могут быть использованы **при сжатии**. *Пространственная избыточность* относится к отдельному изображению. Картинка, состоящая в основном из белой области, обладает высокой избыточностью и может быть эффективно сжата без заметной потери качества. *Временная избыточность* связана со схожестью соседних изображений. Например, если два соседних кадра идентичны, один из них можно не кодировать; более разумно будет сделать

пометку о том, что изображение повторяется. Современные алгоритмы сжатия позволяют сузить видеопоток практически до любых размеров. Конечно, чем шире поток, тем выше качество изображения, что напрямую связано с впечатлениями пользователя от просмотра.

При помощи сжатия мы можем создавать **несколько версий** одного и того же видеофайла, с несколькими уровнями качества — например, с шириной потока 300 кбит/с, 1 Мбит/с и 3 Мбит/с). Пользователи смогут выбирать подходящую версию, в зависимости от пропускной способности канала. При наличии высокоскоростного сетевого соединения можно выбрать видео с потоком 3 Мбит/с; владельцам смартфонов, подключенных к сети 3G, вероятно, придется довольствоваться версией с самым низким качеством. В приложениях для видеоконференций сжатие может применяться «на лету», чтобы обеспечить оптимальное качество в условиях доступной пропускной способности сети.

7.1.2. Свойства аудиоданных

Цифровые аудиоданные (включая музыку и оцифрованный голос) имеют куда более низкие требования к пропускной способности сети, чем видео. Однако они обладают своими уникальными свойствами, которые стоит учитывать при проектировании сетевых мультимедийных приложений. Чтобы лучше понять природу этих свойств, давайте сначала рассмотрим процесс преобразования аналогового звука (который генерируется во время разговора или игры на музыкальных инструментах) в цифровой сигнал.

- На аналоговый аудиосигнал накладывается определенная частота дискретизации — например, 8000 отсчетов в секунду. Каждый отсчет — некоторое вещественное число.
- Затем каждый отсчет округляется до какого-то значения из конечного диапазона. Эта операция называется *квантованием*. Количество возможных значений уровней квантования обычно является степенью двойки (например, 256).
- Каждое значение квантования представляется определенным количеством битов. Например, если таких значений 256, то каждое из них (и, следовательно, каждый звуковой отсчет) представляется одним байтом. Позже битовое представление всех отсчетов соединяется и формирует цифровой сигнал. Допустим, частота дискретизации аналогового сигнала — 8000 отсчетов в секунду, и каждый отсчет

после квантования имеет размер 8 бит; тогда итоговый цифровой сигнал будет иметь поток 64 000 бит в секунду. Чтобы вывести звук на аудиодинамики, цифровой сигнал нужно преобразовать обратно в аналоговый — то есть декодировать. Но в результате декодирования мы получим только примерную копию оригинального сигнала, и качество может заметно деградировать (к примеру, так можно потерять высокочастотные звуки). Увеличивая частоту дискретизации и количество уровней квантования, можно добиться большей схожести с оригиналом. Таким образом, как и в случае с видео, мы должны искать компромисс между качеством декодированного сигнала и шириной потока (и, как следствие, размером) цифровых аудиоданных.

Базовая методика кодирования, описанная выше, называется **импульсно-кодовой модуляцией** (Pulse Code Modulation, **PCM**). Она часто применяется для кодирования речи; в этом случае частота 8000 отсчетов в секунду и 8 бит на каждый отсчет, что в результате дает поток 64 кбит/с. В звуковых компакт-дисках (CD), где тоже используется PCM, берутся отсчеты размером 16 бит, а частота дискретизации равна 44100 Гц; в итоге получается поток шириной 705,6 кбит/с (для одного звукового канала) или 1,411 Мбит/с (для стерео).

Однако речь и звук, закодированные при помощи алгоритма PCM, редко можно встретить в Интернете. По аналогии с видеоданными, для уменьшения потока используются различные методы сжатия. Человеческая речь остается разборчивой даже при потоке в 10 кбит/с. Технология кодирования под названием **MPEG 1 layer 3** (более известная как **MP3**) применяется для высококачественного сжатия стереомузыки. Кодировщик формата MP3 может варьировать ширину потока; чаще всего используется значение 128 кбит/с, которое дает незначительное ухудшение качества звука. Существует также стандарт **AAC (Advanced Audio Coding)**, который продвигает компания Apple. Как и в случае с видео, можно заранее подготовить несколько версий аудиофайлов с разной шириной потока.

И хотя, по сравнению с видеоданными, звуковые файлы имеют значительно меньший размер, пользователи в целом более чувствительны именно к аудиопомехам. Представьте видеоконференцию, которая проводится по Интернету. Если картинка будет время от времени исчезать, вряд ли пользователи сильно огорчатся. Но при регулярной потере звукового сигнала конференцию сложно будет продолжать.

7.1.3. Виды сетевых мультимедийных приложений

В Интернете можно найти большое количество полезных и развлекательных мультимедийных приложений. В этом разделе мы разобьем их на три общие категории: *потоковое вещание хранимого аудио/видео*, *IP-телефония и видеосвязь*, а также *потоковое вещание аудио/видео в реальном времени*. Как вы вскоре сможете убедиться, каждая из этих категорий имеет собственные требования и структурные особенности.

Потоковое вещание хранимого аудио/видео

Чтобы не отходить от текущих реалий, мы сосредоточимся на потоковом вещании хранимых видеоданных, которые обычно включают в себя звуковую дорожку. Отдельно рассматривать вещание аудиоданных (например, музыки) нет особого смысла, так как помимо скорости передачи, которая значительно ниже, чем у видео, этот процесс не имеет существенных отличий.

Для этого класса приложений типичным случаем является заранее записанное видео — например, фильм, телепрограмма, запись спортивного события или домашний ролик (как те, что обычно можно видеть на сайте YouTube). Готовые видеофайлы размещаются на серверах, после чего пользователи могут просматривать их *по запросу*. В наши дни потоковым видеовещанием занимаются многие компании, среди которых можно выделить YouTube (Google), Netflix и Hulu. По некоторым оценкам, потоковое вещание хранимых видеоданных занимает более половины всего загружаемого трафика в Интернете[101]. Этот процесс имеет несколько ключевых особенностей.

- *Потоковое вещание*. В условиях потокового вещания клиент обычно начинает воспроизводить видео с некоторой задержкой, обусловленной получением данных от сервера. Это означает, что, пока пользователь смотрит видео, клиент загружает следующие фрагменты потока. Методика, которую мы знаем под названием «**потоковое вещание**», позволяет начинать воспроизведение, не дожидаясь загрузки всего видеофайла целиком (что могло бы вылиться в длительное ожидание).
- *Интерактивность*. Так как данные записаны заранее, пользователь может, например, останавливать просмотр или перематывать видео вперед/назад. Между действием пользователя и реакцией клиента проходит определенное время; если оно не превышает нескольких секунд, отзывчивость считается приемлемой.

- *Непрерывность воспроизведения*. После того как видео начнет воспроизводиться, клиент должен соблюдать временные рамки оригинальной записи. Таким образом, данные от сервера должны приходить вовремя, чтобы клиент успевал их воспроизводить; иначе видео, которое выводится пользователю, будет «замораживаться» (из-за ожидания загрузки) или терять отдельные кадры (если клиент пропускает кадры, не успевшие загрузиться).

Получается, что наиболее важным показателем производительности потокового видео является средняя пропускная способность. Чтобы обеспечить непрерывное воспроизведение, приложение должно иметь доступ к сети, чья пропускная способность как минимум равна ширине потока самих видеоданных. Как вы убедитесь в разделе 7.2, с помощью буферизации и предварительной загрузки возможно гарантировать отсутствие разрывов даже при нестабильном сетевом соединении; главное, чтобы средняя пропускная способность (на временных интервалах в 5–10 секунд) превышала ширину видеопотока[658].

Для многих приложений источником хранимого видео является CDN, а не какой-то отдельный центр обработки данных. Также часто используются пиринговые сети (P2P), когда видеофайл хранится на компьютерах пользователей и по кусочкам передается в приложение с разных узлов (пиров), разбросанных по всему земному шару. Учитывая популярность потокового вещания видеоданных, мы посвятим этой теме отдельный раздел (7.2), где будет уделено внимание таким вещам как буферизация на стороне клиента, предварительная загрузка, изменение качества в зависимости от пропускной способности канала и распространение при помощи CDN.

IP-телефония и видеосвязь

Передачу человеческой речи по Интернету в режиме реального времени часто называют **интернет-телефонией**, поскольку с точки зрения пользователя это похоже на традиционную коммутируемую телефонную связь. Также используется название **IP-телефония** (Voice-over-IP (дословно: Голос-по-протоколуIP), **VoIP**). С видеосвязью все то же самое, только она, кроме самого видео, подразумевает наличие звука. Большинство современных систем интернет-телефонии позволяют организовывать конференции с более чем двумя участниками. В наши дни услуги подобного рода имеют большую популярность; сервисы Skype, QQ и Google Talk ежедневно обслуживают сотни миллионов пользователей.

В главе 2 при рассмотрении требований к сетевым приложениям (см. рис. 2.4) мы определили несколько аспектов, с помощью которых можно классифицировать тот или иной сервис. Два из них, синхронность и устойчивость к потере данных, особенно важны для аудио- и видеосвязи. Синхронность имеет большое значение, так как **задержки сильно сказываются** на качестве общения по сети. В конференциях временной интервал между выполнением какого-то действия (например, произнесением слова или движением) и его фиксацией на стороне других участников, не должен превышать несколько сотен миллисекунд. При голосовом общении задержка менее чем в 150 миллисекунд не заметна для слушателя. Интервал в 150–400 мс может считаться приемлемым показателем. Но все, что находится за этими пределами, нарушает (или вовсе делает невозможным) голосовое общение.

С другой стороны, мультимедийные приложения для общения в Интернете обладают **стойкостью к потере** данных. Отсутствие части пакетов вызывает лишь частичные помехи при воспроизведении аудио- и видеоданных, и они, как правило, почти или полностью незаметны со стороны. Чувствительность к задержкам, но умение справляться с потерей данных — эти характеристики определенно не свойственны традиционным «эластичным» приложениям, таким как веб-страницы, электронная почта, социальные сети и средства удаленного администрирования. Для них долгие задержки могут быть раздражающим фактором, но не более того; зато потеря полноты и целостности передаваемых данных имеет огромнейшее значение. Более подробно об аудио- и видеосвязи мы поговорим в разделе 7.3, где речь пойдет о том, как с помощью адаптивного воспроизведения, прямой коррекции и маскировки ошибок можно смягчить последствия задержек и потери пакетов в сети.

Потоковое вещание аудио/видео в реальном времени

Приложения этого класса похожи на традиционное радио или телевидение, с той лишь разницей, что вещание происходит в Интернете. Пользователь может принимать *живую* радио- или телепередачу (например, трансляцию спортивного матча или новостной выпуск в прямом эфире), идущую из любого уголка планеты. В наши дни тысячи радио- и телестанций по всему миру транслируют свои программы по Интернету.

У приложений, которые занимаются вещанием в реальном времени, обычно много пользователей; и все они одновременно принимают одну

и ту же аудио/видео передачу. И хотя трансляция аудио/видео в реальном времени множеству получателей может быть эффективно реализована при помощи методик многоадресной рассылки по протоколу IP, описанных в разделе 4.7, современные приложения все чаще выполняют эту задачу собственными средствами (такими как пиринговые сети или CDN) или, как вариант, с использованием множества одиночных потоков. Как и в случае с вещанием хранимых данных, средняя пропускная способность сети должна быть выше, чем ширина мультимедийного потока. И так как речь идет о событии в реальном времени, задержки тоже могут вызвать проблемы, хотя требования к синхронизации не должны быть такими строгими, как при голосовом общении. Задержка длиной в десять секунд (или около того) между тем, как пользователь решил посмотреть живую трансляцию, и началом воспроизведения является терпимой. В этой книге мы не будем отдельно останавливаться на потоковом вещании в реальном времени, так как методики, которые в нем задействуются (буферизация, адаптивное использование пропускной способности сети, распространение через CDN), применимы и для хранимых мультимедийных данных.

7.2. Потоковое вещание хранимых видеоданных

Видеоданные, которые вещаются по сети, находятся в готовом виде на серверах и передаются пользователям по запросу. Пользователь может просматривать видео непрерывно, от начала и до конца, а может остановить воспроизведение или перемотать вперед/назад к определенной сцене. Системы видеовещания можно разделить на три категории: **UDP-вещание**, **HTTP-вещание** и **адаптивное HTTP-вещание**. Последние две получили наибольшее распространение, хотя использование протокола UDP тоже остается актуальным.

Общей характеристикой для всех трех типов видеовещания является интенсивное использование буфера на стороне клиента, что позволяет уменьшить последствия от постоянно меняющихся задержек и нестабильной пропускной способности сети. Пользователи привыкли, что при потоковом вещании видео (как хранимого, так и в реальном времени) происходит небольшая задержка между запросом и началом воспроизведения. Следовательно, вместо того чтобы сразу воспроизводить полученные данные, клиент должен накопить какой-то их объем в буфере приложения. Забежав на несколько секунд вперед, клиент может начать

непосредственное воспроизведение. У этого подхода есть два несомненных преимущества. Во-первых, **буферизация на стороне клиента** может скрыть задержки, возникающие при передаче данных. Если какая-то определенная часть видеофайла задержится, пользователь не заметит этого до тех пор, пока не иссякнет содержимое буфера. Во-вторых, даже если пропускная способность сети ненадолго станет меньше ширины видеопотока, пользователь продолжит наслаждаться непрерывным воспроизведением, пока не закончатся ранее накопленные данные.

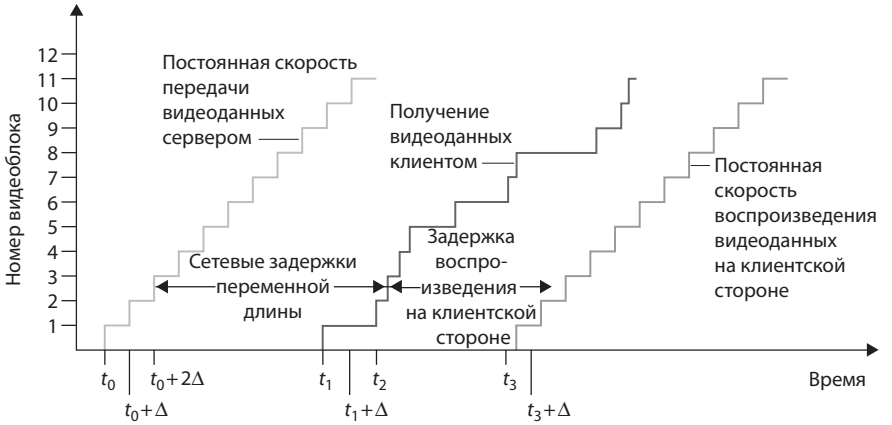


Рис. 7.1. Задержка воспроизведения потокового видео на клиентской стороне

Процесс клиентской буферизации проиллюстрирован на рис. 7.1. Допустим, в нашем случае видеоданные закодированы с потоком фиксированной ширины — то есть все видео-блоки содержат определенное количество кадров, для воспроизведения которых необходимо одно и то же время, Δ. Сервер передает первый блок через t_0 секунд, второй — через $t_0+\Delta$, третий — через $t_0+2\Delta$ и т. д. Когда клиент начнет воспроизведение, каждый блок должен проигрываться на протяжении Δ секунд, чтобы соответствовать хронометражу оригинальной записи. Учитывая постоянно меняющиеся задержки в сети, разные блоки будут передаваться за разные промежутки времени. Первый видеоблок появится на клиентской стороне через t_1 секунд, а второй — через t_2 секунд. Сетевая задержка для i-того блока равна временному отрезку между отправкой блока сервером и моментом его получения на клиентской стороне; как видите, у каждого блока своя сетевая задержка. Если бы клиент начал воспроизведение в момент t_1, при появлении первого блока, второй блок мог бы не успеть дойти к моменту $t_1+\Delta$. В этом случае клиенту пришлось бы либо остановить воспроизведение (и ждать появления второго бло-

ка), либо пропустить часть видеоданных — оба варианта являются нежелательными. Но если клиент подождет до момента t_3, когда будут получены блоки со второго по седьмой, он сможет получать *все* остальные блоки до того, как их нужно будет воспроизводить.

7.2.1. UDP-вещание

Мы не будем слишком углубляться в тему UDP-вещания; по ходу чтения вы получите отсылки на более специализированный материал, посвященный сетевым протоколам. Итак, UDP-вещание позволяет серверу передавать видеоданные со скоростью, необходимой для их потребления на стороне клиента. Для этого видеофрагменты отправляются по протоколу UDP с постоянной частотой. Например, если видео потребляется со скоростью 2 Мбит/с и каждый UDP-пакет хранит 8000 бит, то серверу нужно будет передавать пакеты с периодичностью в (8000 бит)/(2 Мбит/с) = 4 мс. Как вы уже знаете из главы 3, протокол UDP не предусматривает механизм устранения перегрузок, поэтому сервер может отправлять пакеты с частотой потребления видеоданных без ограничений, накладываемых протоколом TCP. При UDP-вещании обычно используется небольшой буфер на клиентской стороне; его размер должен быть достаточным для того, что вместить не более одной секунды видеопотока.

Прежде чем отправлять видеофрагмент, сервер должен оформить его в виде специального транспортного пакета, предназначенного для передачи аудио и видео; для этого используется технология RTP (Real-Time Transport Protocol)[518] или какой-то другой (возможно, закрытый) протокол. Но эту тему мы отложим до раздела 7.3, где технология RTP будет рассмотрена в контексте систем для выполнения звуковых и видеовызовов.

Еще одной отличительной чертой UDP-вещания является наличие дополнительного сетевого соединения, с помощью которого клиент передает серверу команды, сообщая об изменениях в состоянии сеанса (например, об остановке, возобновлении воспроизведения или переходе на другую позицию). Данное соединение во многом похоже на то, которое используется в протоколе FTP (его мы уже рассматривали в главе 2). Для работы с подобного рода управляющими соединениями часто используется открытая технология RTSP (Real-Time Streaming Protocol)[473].

Несмотря на то что UDP-вещание применяется во многих открытых системах и фирменных продуктах, у него есть три значительных недостатка. Во-первых, ввиду непредсказуемой и постоянно меняющейся скорости передачи данных между сервером и клиентом UDP-вещание с фиксированной частотой отправки пакетов может не обеспечить непрерывное воспроизведение. Давайте рассмотрим ситуацию, когда поток потребления видео равен 1 Мбит/с, а пропускная способность сети в целом превышает это значение, но раз в несколько минут она на некоторое время опускается ниже необходимого минимума. В таком случае система UDP-вещания, передающая видеоданные со стабильной частотой 1 Мбит/с по протоколам RTP/UDP, скорее всего, не сможет обеспечить удовлетворительное качество воспроизведения, замораживая картинку или пропуская кадры после каждого падения скорости. Второй недостаток этого подхода заключается в необходимости управляющего медиасервера (например, с использованием протокола RTSP); без него нельзя будет отправлять интерактивные запросы и отслеживать состояние клиента (например, текущая позиция, выполняется ли воспроизведение и т. д.) для *всех* текущих сеансов. Это увеличивает общие издержки и усложняет развертывание крупномасштабных систем распространения видео по запросу. Третий недостаток связан с тем, что многие брандмауэры блокируют обмен пакетами по протоколу UDP, из-за чего пользователи часто не могут получить доступ к UDP-вещанию.

7.2.2. HTTP-вещание

HTTP-вещание подразумевает, что видеоданные хранятся на HTTP-сервере в виде обычного файла с определенным URL-адресом. Когда пользователь хочет начать просмотр, клиентское приложение устанавливает соединение с сервером по протоколу TCP и выполняет по заданному адресу запрос HTTP GET. В ответ сервер сразу же отправляет видеофайл внутри HTTP-сообщения; то есть данные будут передаваться настолько быстро, насколько это позволяют ограничения протокола TCP и система управления потоком. На клиентской стороне полученные байты складываются в буфер. Когда количество байт доходит до какого-то определенного предела, клиентская программа начинает воспроизведение; для этого она периодически захватывает видеокадры из буфера, декодирует их и выводит на экран пользователя.

Из главы 3 вы знаете, что скорость передачи файлов по протоколу TCP может заметно колебаться в результате работы механизма устра-

нения перегрузок. В частности, далеко не редкими являются скачкообразные изменения частоты передачи (см. рис. 3.53). Более того, пакеты могут сильно задерживаться из-за механизма повторной передачи, который тоже является частью протокола TCP. Ввиду этих обстоятельств, а также исходя из здравого смысла, можно с уверенностью сказать, что еще 15–20 лет назад TCP нельзя было бы использовать для качественного потокового видеовещания. Однако со временем архитекторы видео-сервисов поняли, что вышеописанные механизмы вовсе не мешают обеспечивать непрерывное воспроизведение, если при этом на стороне клиента используются буферизация и предварительная загрузка (подробней об этом в следующем разделе).

Протокол HTTP, работающий поверх TCP, во многих случаях позволяет обходить брандмауэры и системы NAT (которые часто блокируют UDP-пакеты, но разрешают передачу данных по HTTP). HTTP-вещание также избавляет нас от необходимости использовать управляющий медиасервер (например, с использованием протокола RTSP), что снижает затраты на широкомасштабное распространение данных по Интернету. Учитывая все эти преимущества, большинство приложений и сервисов для потокового видеовещания (включая YouTube и Netflix) используют в качестве исходного протокола именно HTTP (поверх TCP).

Предварительная загрузка видеоданных

Как вы только что узнали, буферизация на стороне клиента может уменьшить последствия задержек и изменений пропускной способности сети. В примере, который был изображен на рис. 7.1, сервер передает видеоданные со скоростью, на которой они будут воспроизводиться. Но имея дело с *хранимым* видео, клиент может попытаться загружать его на *более высокой* скорости, выполняя тем самым **предварительную загрузку** кадров, которые будут выводиться в будущем. Данные, полученные таким образом, точно так же хранятся в буфере приложения. Предварительная загрузка является естественным следствием использования протокола TCP, механизм устранения перегрузок которого всегда пытается занять весь доступный поток между клиентом и сервером.

Чтобы понять, как работает предварительная загрузка, рассмотрим простой пример. Допустим, видеоданные потребляются со скоростью 1 Мбит/с, а постоянная пропускная способность сети равна 1,5 Мбит/с. В таком случае клиент не только начнет воспроизведение с незначи-

тельной задержкой, но и сможет регулярно пополнять свой буфер (на 500 кбит каждую секунду). Таким образом, если позже скорость передачи видеоданных упадет ниже 1 Мбит/с, клиент сможет какое-то время воспроизводить то, что он успел буферизировать. В исследовании «Мультимедийное вещание посредством TCP: аналитическое исследование производительности»[658] говорится, что если средняя пропускная способность сети примерно в два раза превышает ширину потока мультимедийных данных, передаваемых по протоколу TCP, задержки, связанные с опустошением буфера, будут минимальными.

Буферизация на стороне клиента и TCP-буфер

На рис. 7.2 проиллюстрировано взаимодействие между клиентом и сервером во время HTTP-вещания. Начнем с сервера. Белым цветом представлена часть видеофайла, которая была отправлена в серверный сокет; темная часть — это то, что осталось. Пройдя через сокет, байты попадают в исходящий TCP-буфер и только потом передаются в Интернет (см. главу 3). На рис. 7.2 изображен полный буфер, поэтому на какое-то время сервер прекращает отправлять содержимое видеофайла. На клиентской стороне приложение (медиапроигрыватель) считывает байты из входящего TCP-буфера (посредством клиентского сокета) и помещает их в свой собственный буфер. Вместе с тем клиентское приложение периодически извлекает видеокадры из буфера, декодирует их и выводит на экран пользователя. Стоит отметить, что, если клиентский буфер больше самого видеофайла, весь процесс передачи данных от сервера клиенту ничем отличается от обычной загрузки файлов по HTTP — байты запрашиваются с максимальной допустимой на уровне протокола TCP частотой!

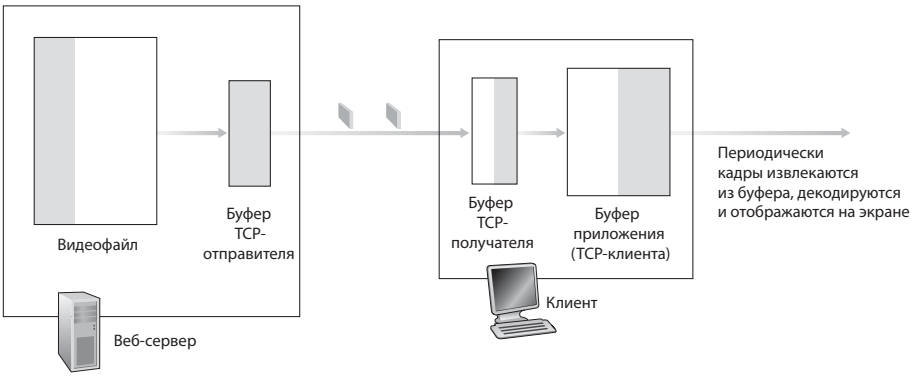


Рис. 7.2. Потоковое вещание статического видео по протоколам HTTP/TCP

Давайте посмотрим, что случится, если пользователь приостановит воспроизведение видео во время вещания. Старые биты не будут изыматься из клиентского буфера, зато новые все так же продолжат поступать. Если размер клиентского буфера ограничен, в какой-то момент он может заполниться, что приведет к «обратному давлению» уже со стороны клиента. Кроме того, байты больше не будут покидать входящий TCP-буфер, поэтому в нем тоже должно закончиться свободное место. В этот момент данные перестают отправляться из исходящего TCP-буфера на стороне клиента. Это приводит к тому, что сервер больше не может отправлять байты в сокет. Таким образом, в случае остановки воспроизведения сервер может быть вынужден отложить передачу данных и заблокироваться, пока воспроизведение не возобновится.

На самом деле даже во время обычного воспроизведения (без остановок) возможна ситуация с заполнением программного и, как следствие, TCP-буфера, в результате которой серверу придется снизить скорость передачи данных. Чтобы определить итоговую скорость, нужно учитывать, что при удалении f бит из клиентского буфера в нем появляется ровно f бит свободного пространства, позволяя серверу отправить те самые f бит. Таким образом, скорость отправки данных сервером не может превышать скорость их потребления на стороне клиента. Следовательно, *заполнение клиентского буфера косвенно ограничивает интенсивность, с которой видео может передаваться от сервера к клиенту, при условии, что вещание выполняется по протоколу HTTP.*

Принцип работы потокового видеовещания

Чтобы лучше понимать причины начальной задержки и «заморозки» воспроизведения в результате опустошения клиентского буфера, выполним простое моделирование. Пусть B — это размер клиентского буфера (в битах), а Q — количество бит, которые необходимо буферизировать для того, чтобы началось воспроизведение (естественно, $Q < B$). r будет представлять скорость потребления — интенсивность, с которой приложение извлекает биты из клиентского буфера во время воспроизведения. Итак, если видео воспроизводится с частотой 30 кадров в секунду и каждый (закодированный) кадр имеет размер 100 000 бит, то $r = 3$ Мбит/с. Чтобы не отвлекаться на мелкие детали, мы проигнорируем TCP-буферы отправителя и получателя.

Давайте будем исходить из того, что сервер отправляет биты с постоянной скоростью x при условии, что клиентский буфер не заполнен.

(Это грубое упрощение, поскольку из-за механизма предотвращения перегрузок, встроенного в протокол TCP, скорость неизбежно будет меняться; в конце этой главы мы рассмотрим более реалистичный пример, где частота передачи зависит от времени — $x(t)$). Допустим, в момент времени $t = 0$ видеоданные начинают появляться в клиентском буфере, который изначально пуст. Так когда же начнется воспроизведение ($t = t_p$)? И если уж на то пошло, когда клиентский буфер полностью заполнится ($t = t_f$)?

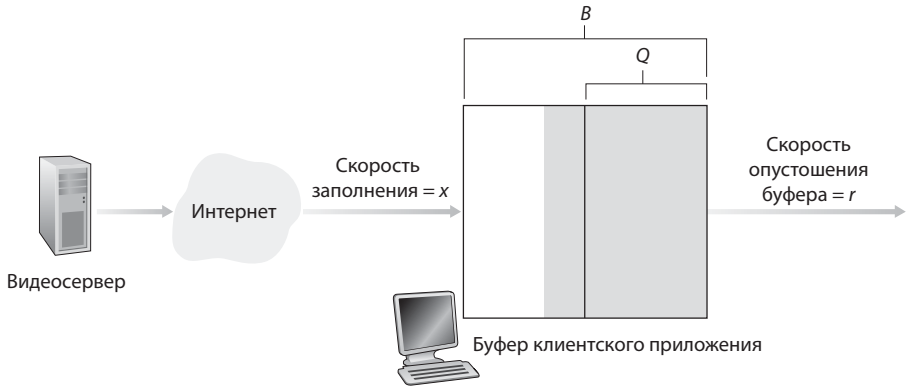


Рис. 7.3. Принцип работы клиентского буфера во время потокового видеовещания

Давайте сначала определим t_p — момент времени, когда Q бит попадает в клиентский буфер и когда начинается воспроизведение. Как вы помните, биты поступают в буфер с частотой x и *не* покидают его до тех пор, пока не стартует просмотр видео. Таким образом, количество времени, необходимое для накопления Q бит (задержка начальной буферизации), определяется как $t_p = Q/x$.

Теперь определим t_f — момент, когда буфер клиентского приложения становится полным. Отметим, что если $x < r$ (то есть если скорость передачи данных сервером меньше скорости их потребления), клиентский буфер никогда не заполнится до конца! Действительно, частота заполнения r должна быть выше частоты потребления x. В какой-то момент буфер полностью опустеет, что приведет к остановке воспроизведения и задержке в t_p секунд, за время которой должно поступить Q бит данных. *Следовательно, если пропускная способность сети меньше частоты потребления видеоданных, воспроизведение будет периодически останавливаться.* В домашнем задании вам нужно будет определить

длину всех периодов воспроизведения и простоя как функцию от Q, r и x. Теперь определим t_f для $x > r$. В этом случае буфер наполняется со скоростью $x - r$, начиная с Q, и заканчивая B; биты изымаются с частотой r, но поступают на скорости x, как показано на рис. 7.3. Пользуясь этими подсказками, вы должны будете определить t_f — момент полного заполнения клиентского буфера. Обратите внимание: *если пропускная способность сети превышает частоту потребления видеоданных, после начальной задержки, связанной с буферизацией, воспроизведение больше не будет останавливаться.*

Преждевременное завершение и перемотка воспроизведения

Системы HTTP-вещания часто используют в запросах типа GET заголовок **Range**, определяющий для видеофайла диапазон байт, которые клиент желает получить. Это особенно полезно в тех случаях, когда пользователь хочет перемотать видео вперед. При перемотке клиент отправляет HTTP-запрос, в котором указывается, с какого места сервер должен начинать передачу содержимого файла. Диапазон можно менять при каждом новом запросе.

Раз уж мы затронули эту тему, будет нелишним сказать, что при перемотке вперед или преждевременном завершении воспроизведения некоторые загруженные, но еще не показанные кадры так никто и не увидит — это будет пустая трата ресурсов сети и сервера. Представьте, что на определенной позиции в видеофайле t_0 клиентский буфер заполнится и будет содержать B бит, а в этот момент пользователь перемотает видео вперед на позицию $t>t_0+B/r$ и продолжит просмотр до самого конца. В этом случае все B бит, что находились в буфере, не попадут на экран; ресурсы, которые использовались для их передачи, будут потрачены зря. В Интернете множество сетевых ресурсов уходит в никуда из-за преждевременного завершения воспроизведения, особенно это касается беспроводных сетей[239]. В связи с этим многие системы потокового вещания ограничивают размер буфера приложения или количество заранее загружаемых видеоданных; для этого используется вышеупомянутый HTTP-заголовок Range[411].

Перемотка и преждевременное завершение воспроизведения — это все равно что приготовить шведский стол на одного человека и в результате выбросить большую часть еды. Поэтому в следующий раз, когда ваши родители будут вас упрекать за недоеденный ужин, вы можете смело ответить, что при перемотке фильмов в Интернете они зря тра-

тят ресурсы сервера и сети! Но, естественно, один неправильный поступок не может компенсировать другой, поэтому бережно относитесь как к еде, так и к переданным данным!

7.2.3. Адаптивное вещание и технология DASH

В предыдущем разделе мы узнали, что HTTP-вещание активно используется на практике (например, компанией YouTube с момента ее появления), однако у него есть большой недостаток: все клиенты, независимо от текущей пропускной способности их сетевого соединения, получают одни и те же видеоданные. В результате был разработан новый стандарт — **динамическое адаптивное вещание по HTTP** (Dynamic Adaptive Streaming over HTTP, **DASH**). Согласно ему закодированный видеофайл должен иметь несколько версий с разной шириной потока и, соответственно, с разным качеством. Клиент динамически запрашивает фрагменты разных версий длиной в несколько секунд. При наличии высокой пропускной способности сети клиент, естественно, выбирает самый качественный вариант; но если сетевое соединение имеет низкую скорость, он может выбрать версию с небольшой шириной потока. Каждый фрагмент запрашивается с помощью запросов типа GET по протоколу HTTP[16].

Во-первых, технология DASH позволяет клиентам с разным качеством сетевого соединения потреблять видеоданные с разной шириной потока. Пользователи медленных сетей 3G могут просматривать ужатые (и низкокачественные) версии, тогда как обладателям оптоволоконного подключения доступно самое высокое качество. Во-вторых, клиенты могут адаптироваться к изменениям пропускной способности сети в рамках одного сеанса. Это чрезвычайно важно для пользователей мобильных устройств, качество сетевого соединения которых меняется в зависимости от базовой станции. Например, Интернет-провайдер Comcast предоставляет систему адаптивного вещания, которая способна кодировать исходный видеоматериал в 8–10 разных форматов (основанных на MPEG-4); это позволяет клиентам получать видео оптимального качества и адаптироваться к изменениям состояния сети и самого устройства.

При использовании технологии DASH каждая версия видеофайла помещается на HTTP-сервер и имеет свой уникальный URL-адрес. На сервере также должен находиться **файл манифеста**, в котором указаны адреса и ширина потока для всех версий. Сначала клиент запрашивает

файл манифеста, чтобы узнать, какие видеоданные ему доступны. Затем он последовательно запрашивает каждый фрагмент с помощью команды GET, указывая подходящий URL-адрес и диапазон байтов. Загружая фрагменты, клиент оценивает пропускную способность сети, используя *специальный алгоритм*, чтобы решить, из какой версии запрашивать следующий фрагмент. Естественно, если удалось принять в буфер много видеоданных и если измеренная скорость оказалась высокой, будет выбран фрагмент самого лучшего качества. В противном случае можно будет довольствоваться ужатой версией. Таким образом, технология DASH позволяет свободно переключаться между данными с разной шириной потока. Резкое снижение скорости передачи при переключении на другую версию может неблагоприятно отразиться на качестве воспроизведения, поэтому для достижения плавного перехода может использоваться несколько промежуточных уровней, пока скорость передачи не опустится ниже пропускной способности сети клиента. Позже, когда состояние сетевого соединения улучшится, клиент сможет проделать тот же процесс в обратном порядке.

За счет динамического отслеживания состояния сети и уровня клиентского буфера, а также корректировки скорости передачи данных посредством переключения между разными версиями потоков технология DASH во многих случаях может обеспечить беспрерывное воспроизведение с оптимальным качеством. Более того, такой подход улучшает масштабируемость серверных ресурсов, поскольку выбор фрагментов для загрузки происходит на стороне клиента. Еще один плюс заключается в возможности использования HTTP-заголовка Range, благодаря чему можно точно контролировать объем предварительно загружаемых видеоданных, которые буферизируются локально.

В завершение нашего знакомства с технологией DASH стоит сказать, что во многих ее реализациях на версии разделяются не только видео-, но и аудиоданные. Каждый звуковой поток имеет свой уровень качества, размер и URL-адрес; аудио- и видеофрагменты выбираются отдельно, синхронизируясь на стороне клиента.

7.2.4. Сети распространения контента (CDN)

В наши дни многие Интернет-компании ежедневно распространяют видеоданные высокого качества среди миллионов пользователей. К примеру, YouTube каждый день предоставляет свою обширную видеотеку сотням миллионов людей по всему миру[135]. Потоковое вещание

всей этой информации в разные уголки планеты, а также обеспечение непрерывного воспроизведения с высокой степенью интерактивности является, вне всяких сомнений, нетривиальной задачей.

Наверное, наиболее очевидный способ предоставить услуги потокового видеовещания — построить единый большой вычислительный центр для хранения и распространения данных по всему миру. Но у этого подхода есть три существенных недостатка. Во-первых, если клиент находится слишком далеко, пакеты, отправляемые из такого центра, будут проходить множество сетевых узлов, принадлежащих разным провайдерам и, вероятно, размещенных на разных континентах. Если пропускная способность одного из этих узлов окажется меньше скорости потребления видеоданных, это сузит канал потребления на клиентской стороне и приведет к регулярным остановкам воспроизведения (в первой главе мы говорили, что пропускная способность канала связи определяется самым узким его местом). Вероятность возникновения этой проблемы растет вместе с числом промежуточных узлов. Второй недостаток заключается в том, что популярные видеоролики, скорее всего, будут многократно проходить по одним и тем же маршрутам. Это не только пустая трата сетевых ресурсов, но и лишние расходы для Интернет-компании за потребленный траффик, ведь по каналам связи передаются *одинаковые* наборы байт. Есть и третья проблема: полагаться только на один вычислительный центр — это как хранить все яйца в одной корзине. В случае серьезной неисправности пользователи лишатся доступа ко *всем* видеопотокам сразу.

Чтобы решить проблему доставки огромных потоков данных пользователям из разных стран, почти все крупные компании используют **сети распространения контента** (Content Distribution Network, **CDN**). Сеть CDN состоит из множества размещенных в разных местах серверов, в которых хранятся видеофайлы (и другие веб-ресурсы, такие как документы, изображения и аудиоданные); каждый пользователь перенаправляется в ту точку сети, которая является для него наиболее подходящей. CDN может принадлежать одной компании, предоставляющей данные (как в случае с компанией Google, которая распространяет видеоролики и другие данные через сервис YouTube), или использоваться сразу несколькими провайдерами (например, компания Akamai предоставляет CDN для многих сервисов, в том числе для Netflix и Hulu). Крайне интересный обзор современных сетей CDN содержится в статье Тома Лейтона[315].

При построении CDN обычно применяется один из двух подходов к размещению серверов[211].

СЛУЧАЙ ИЗ ПРАКТИКИ

Сетевая инфраструктура компании Google

Чтобы поддерживать обширный набор своих сервисов (таких как поиск, почта, календарь, YouTube, карты, документы и социальные сети), компания Google использует развитую частную сеть и инфраструктуру CDN. Последняя состоит из трех видов серверных кластеров:

- Восемь «гигантских вычислительных центров», шесть из которых находятся в США и еще два в Европе[186]. В каждом из них размещено порядка 100 000 серверов. Эти центры отвечают за выдачу динамических (часто персонализированных) данных, включая результаты поиска и почтовые сообщения.
- Около 30 «локальных» кластеров (см. раздел 7.2.4), каждый из которых состоит примерно из 100–500 серверов[9]. Кластеры разбросаны по всему миру и обычно находятся рядом с точкой присутствия провайдеров первого эшелона. Отвечают за раздачу статических данных, таких как видеоролики сервиса YouTube[9].
- Множество сотен кластеров «глубокого проникновения» (см. раздел 7.2.4), интегрированных в инфраструктуру Интернет-провайдеров. Кластер обычно состоит из десяти серверов, размещенных в одной стойке. Используются для так называемого расщепления TCP-пакетов (см. раздел 3.7) и раздачи статических данных[91], включая статические фрагменты веб-страницы, которые выводят результаты поиска.

Все эти вычислительные центры и кластеры связаны между собой в единую частную сеть компании Google и являются частью гигантской автономной системы (AS 15169). Поисковый запрос, сделанный пользователем, часто перенаправляется на кеширующий сервер локального Интернет-провайдера, откуда извлекаются статические данные; одновременно с этим кеширующий сервер передает запрос по частной сети Google в один из крупных вычислительных центров, откуда уже возвращаются персонализированные результаты поиска. Например, при просмотре видеоролика на сервисе YouTube данные могут поступать от одного из локальных серверов, тогда как фрагменты веб-страницы, на которой этот ролик размещен, могут браться из кластера глубокого проникновения; при этом рекламные объявления, скорее всего, раздаются из вычислительного центра. В целом, если не учитывать локальных Интернет-провайдеров, облачная инфраструктура компании Google во многом полагается на публичные сети.

- **Глубокое проникновение**. Этот подход был впервые реализован компанией Akamai. Он заключается в *глубоком проникновении* в сетевую инфраструктуру Интернет-провайдеров. Это означает, что

серверные кластеры должны размещаться как можно ближе к пользователям Интернета по всему миру (сети доступа описаны в разделе 1.3).

Кластеры компании Akamai, которая использует этот подход, размещены в 1700 разных географических точках. Главная цель заключается в том, чтобы оказаться как можно ближе к пользователю, уменьшая сетевые задержки и улучшая пропускную способность сети за счет снижения количества узлов и маршрутизаторов между потребителем и кластером CDN, который раздает данные. Однако из-за такой распределенной архитектуры управление и обслуживание кластеров превращается в довольно нетривиальную задачу.

- **Консолидация**. Другой подход, взятый на вооружение компанией Limelight (и другими сетями CDN), *заключается в построении* какого-то небольшого числа крупных кластеров (например, десяти) в ключевых географических регионах и объединении их в высокоскоростную частную сеть. Вместо того чтобы внедряться в инфраструктуру Интернет-провайдеров, такие CDN обычно имеют свои кластеры недалеко от точек присутствия провайдеров первого эшелона (см. раздел 1.3). Например, в больших городах один кластер может находиться на расстоянии нескольких километров от точек присутствия, принадлежащих компаниям AT&T и Verizon. В отличие от стратегии глубокого проникновения, этот подход обычно требует меньше затрат на обслуживание и управление, однако с ним могут быть связаны такие проблемы, как увеличение сетевых задержек и снижение скорости передачи данных конечным пользователям.

CDN дублирует данные на всех доступных кластерах. Возможно, некоторые видеоролики ввиду своей низкой популярности в определенных регионах не будут копироваться во все точки частной сети. На самом деле во многих сетях CDN используется пассивная стратегия: кластер копирует к себе видеоданные (как правило, из центрального хранилища или из другого кластера) только после того, как пользователь их запросит. По аналогии с кеширующими серверами (см. главу 2), в случае переполнения кластеры удаляют редко запрашиваемые данные.

Принцип работы CDN

Итак, мы определили два основных подхода к развертыванию сетей CDN. Теперь давайте сосредоточим свое внимание на том, как они работают. Когда пользователь с помощью своего веб-браузера пытается полу-

чить определенные видеоданные (по заданному URL-адресу), сеть CDN должна перехватить запрос и сделать две вещи: найти кластер, который лучше всего подходит для этого конкретного клиента, и перенаправить запрос на сервер в этом кластере. О поиске подходящего кластера мы поговорим чуть позже. Сначала давайте посмотрим, как происходят перехват и перенаправление запроса.

Большинство сетей CDN выполняют перехват и перенаправление на уровне DNS (если хотите узнать много интересного о таком способе использования DNS, см. Vixie 2009). Чтобы понять, как это делается, давайте рассмотрим простой пример. Компания NetCinema, которая занимается распространением видео, пользуется услугами сторонней сети CDN, принадлежащей компании KingCDN. На веб-сайте NetCinema у каждого видеоролика есть свой собственный URL-адрес, который содержит слово video и некий уникальный идентификатор; например, художественный фильм «Трансформеры-7» мог бы находиться по адресу **http://video.netcinema.com/6Y7B23V**. Весь процесс состоит за шести этапов (см. рис. 7.4):

1. Пользователь открывает страницу на веб-сайте NetCinema.
2. При переходе по ссылке вида **http://video.netcinema.com/6Y7B23V** компьютер пользователя отправляет DNS-запрос по адресу **video.netcinema.com**.
3. Локальный DNS-сервер пользователя (Local DNS или LDNS) передает запрос официальному DNS-серверу компании NetCinema, который ищет в доменном имени **video.netcinema.com** строку video. Чтобы не возвращать IP-адрес, а «перенаправить» запрос в сеть KingCDN, официальный DNS-сервер возвращает локальному LDNS соответствующее доменное имя — например, **a1105.kingcdn.com**.
4. С этого момента DNS-запрос поступает в частную сеть KingCDN. LDNS пользователя шлет второй запрос, теперь по адресу **a1105.kingcdn.com**, а инфраструктура KingCDN возвращает свои внутренние IP-адреса. Именно на этом этапе выбирается один из серверов кластера, который будет возвращать пользователю запрашиваемые данные.
5. LDNS передает IP-адрес узла, входящего в сеть CDN, компьютеру пользователя.
6. Получив IP-адрес сервера внутри сети KingCDN, клиент устанавливает с ним соединение по протоколу TCP и выполняет HTTP-

команду типа GET, чтобы получить доступ к видеоролику. Если используется технология DASH, клиент сперва получит файл манифеста со списком URL-адресов, представляющих разные версии видеоданных; после этого он сможет динамически загружать видеофрагменты разного качества.

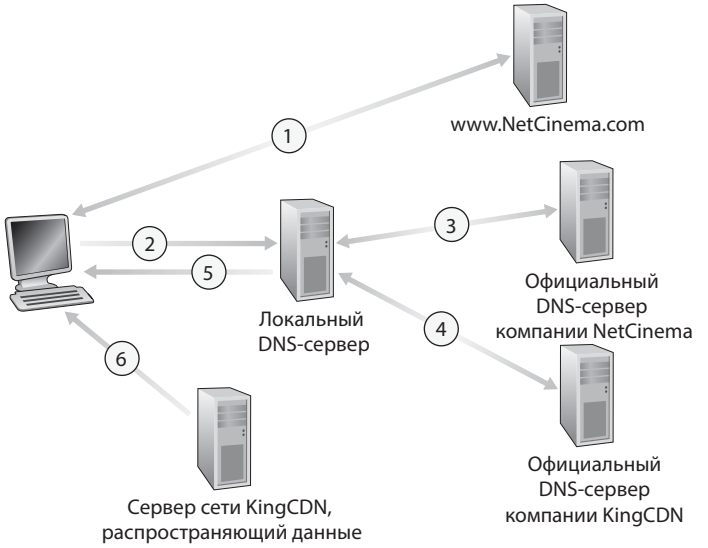


Рис. 7.4. DNS перенаправляет запрос пользователя к серверу, входящему в состав CDN

Стратегии выбора кластера

В основе структуры любой сети CDN лежит **стратегия выбора кластера** — то есть механизма для динамического перенаправления клиентов на определенный кластер или вычислительный центр. Как мы только что увидели, CDN получает IP-адрес клиентского LDNS-сервера посредством поискового DNS-запроса. Исходя из этого адреса, необходимо выбрать подходящий кластер. Обычно этот выбор основывается на принадлежности серверов к той или иной компании или сети. Мы же сосредоточим свое внимание на «нейтральных» стратегиях, каждая из которых имеет свои преимущества и недостатки.

Один из простых подходов заключается в перенаправлении клиента к кластеру, который **находится ближе всех**. С помощью коммерческой геолокационной базы данных, такой как Quova[403] или Max-Mind[335] каждый IP-адрес LDNS-сервера привязывается к определенному гео-

графическому местоположению. Получив DNS-запрос, сеть CDN выбирает близкий кластер (обычно он находится в нескольких километрах от LDNS-сервера). Такое решение довольно неплохо подходит для большинства клиентов12. Но в ряде ситуаций оно является неудачным, поскольку географическая близость не гарантирует кратчайший путь в сети. Кроме того, всем методикам, основанным на DNS, присуща общая проблема: некоторые пользователи указывают в настройках LDNS-серверы, которые находятся где-то далеко[598, 334], в результате чего клиент может находиться совсем не там, где ожидается. Есть еще один недостаток: эта стратегия игнорирует показатели сетевых задержек и пропускную способность сети между клиентом и кластером, основывая свой выбор исключительно на географическом местоположении.

Чтобы учитывать *текущее* состояние сети при выборе подходящего кластера, можно **периодически проверять** величину задержки и пропускную способность между кластерами и клиентами. Например, каждый кластер может время от времени слать проверочные пакеты (например, DNS-запросы или ICMP-сообщения) LDNS-серверам по всему миру. Одним из недостатков этого подхода является то, что некоторые LDNS-серверы в силу своей конфигурации не реагируют на такие проверки.

Но есть и другой вариант: вместо того чтобы измерять свойства участка сети посредством отдельных пакетов, это можно делать во время обмена данными между клиентом и CDN-сервером. Например, задержку между компьютером пользователя и кластером можно измерить при трехэтапном согласовании TCP-соединения, когда сервер шлет флаг SYNACK, а в ответ получает ACK. Однако это решение подразумевает, что иногда клиенты будут перенаправляться на не совсем оптимальные узлы сети — это требуется для измерения свойств сетевого пути к кластерам. И хотя для этого необходимо небольшое количество запросов, некоторые клиенты ощутят значительное снижение производительности при получении данных[23, 299]. Еще один подход к измерению сетевых задержек между кластером и компьютером пользователя заключается в использовании DNS-запросов. В частности, на этапе получения IP-адреса (см. пункт 4 в предыдущем списке) LDNS-сервер можно иногда перенаправлять на другие официальные DNS-серверы, размещенные в разных точках сети CDN; при этом можно произвести соответствующие замеры. В таком случае DNS-сервер всегда будет возвращать адрес оптимального кластера, поэтому клиенты не столкнутся ухудшением условий доставки видеоданных или других объектов[212].

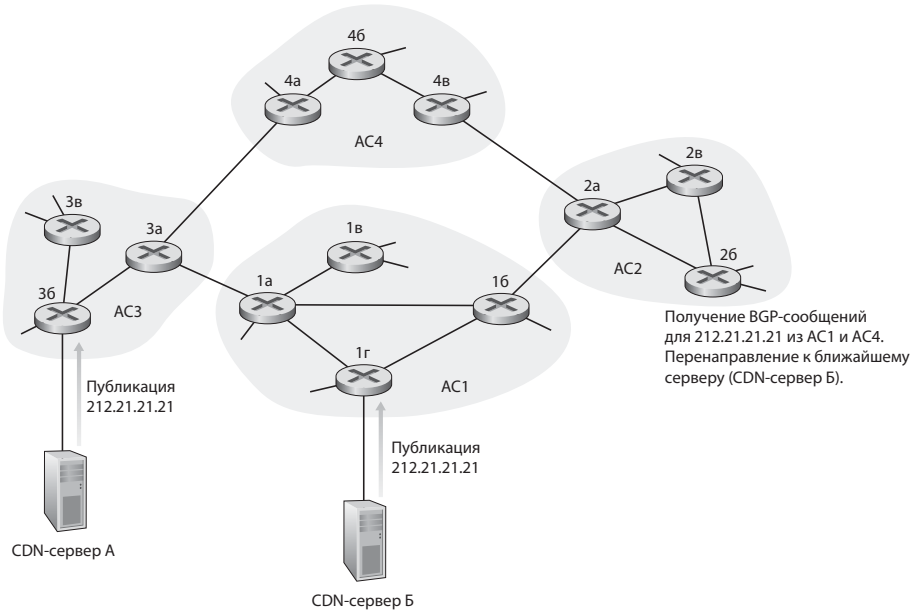


Рис. 7.5. Использование IP-anycast для перенаправления клиентов к ближайшему кластеру сети CDN

Но есть совершенно другой подход к поиску подходящих CDN-серверов, который называется **IP-anycast**[447]. Его идея заключается в том, что маршрутизаторы в Интернете перенаправляют пакеты клиента к «ближайшему» кластеру. В частности, как показано на рис. 7.5, на этапе конфигурирования IP-anycast владелец сети CDN назначает всем своим кластерам *один и тот же* IP-адрес, который публикуется каждым из кластеров, расположенных в разных географических точках *посредством протокола* BGP. Получая множество сообщений для одного IP-адреса, BGP-маршрутизатор работает так, как будто разные сетевые маршруты выдаются для одного и того же физического местоположения (хотя на самом деле эти сообщения приходят из *разных* мест). Следуя стандартным процедурам, BGP-маршрутизатор выбирает «лучший» маршрут (например, ближайший путь, исходя из суммы расстояний между узлами) к IP-адресу в соответствии со своим локальным механизмом выбора. Например, BGP-маршрутизатор обычно выбирает тот маршрут, который состоит из меньшего числа переходов (см. раздел 4.6). Выполнив начальную конфигурацию, CDN может заняться своей основной задачей — то есть раздавать данные. Когда какой-либо клиент, независимо от его местоположения, запрашивает видео, DNS-сервер сети CDN возвращает ему адрес IP-anycast. Пакеты, переданные по это-

му IP-адресу, перенаправляются к «ближайшему» кластеру (в соответствии с заранее подготовленными таблицами маршрутов), который был настроен с помощью протокола BGP вышеописанным образом. Преимущество данного подхода заключается в том, что он учитывает местоположение самого клиента, а не LDNS-сервера. Однако это не позволяет реагировать на изменения структуры сети, которые в Интернете случаются постоянно[37].

Помимо аспектов, связанных с сетью, таких как сетевые задержки, потери пакетов и ограничения пропускной способности, существует множество дополнительных факторов, которые берутся во внимание, когда разрабатывается стратегия выбора ближайшего кластера. В качестве примера можно привести загруженность серверов — клиенты не должны перенаправляться на кластер, который и без того слишком занят. Кроме того, нужно учитывать, какие именно Интернет-провайдеры используются для доставки данных между клиентом и сетью CDN, ведь у каждого из них могут быть свои условия предоставления услуг, в том числе и специфические контракты с операторами кластера.

7.2.5. Примеры: Netflix, YouTube и Kankan

Чтобы подытожить обсуждение потокового вещания хранимых видеоданных, рассмотрим три успешных крупномасштабных проекта: Netflix, YouTube и Kankan. Как вы скоро убедитесь, все они используют совершенно разные подходы, соблюдая при этом общие принципы, о которых мы говорили в этом разделе.

Netflix

Генерируя почти 30% от всего входящего интернет-трафика в США, компания Netflix является лидирующим дистрибутором фильмов и телевизионных передач Америки[581]. Чтобы иметь возможность быстро разворачивать свой крупномасштабный сервис, Netflix активно использует сети CDN. Действительно, это интересный пример того, как можно создать гигантскую инфраструктуру на основе арендуемых серверов, сетевых каналов, хранилищ и баз данных. В материале, приведенном ниже, мы отталкиваемся от исследования[10], в котором очень доходчиво разобрана архитектура этого сервиса. Как вы вскоре убедитесь, Netflix использует многие подходы, изложенные ранее в этом разделе, включая распространение видео посредством сети CDN (на самом деле таких сетей несколько) и адаптивное вещание по протоколу HTTP.

На рис. 7.6 показана основная архитектура платформы потокового видеовещания компании Netflix. Она состоит из четырех главных компонентов: серверы регистрации и оплаты, облачные серверы Amazon, несколько CDN-провайдеров и клиенты. Регистрация новых пользователей и обработка информации, связанной с кредитными картами, выполняется с помощью собственных аппаратных средств. Все остальные операции происходят внутри арендуемых серверов (или виртуальных машин) в облаке Amazon. В частности:

- *Потребление данных*. Прежде чем инфраструктура компании Netflix сможет раздавать данные своим клиентам, она должна сначала их получить и обработать. Netflix получает студийные версии фильмов и загружает их на серверы в облаке Amazon.
- *Обработка данных*. Серверы в облаке Amazon генерируют множество разных копий одного и того же фильма, которые подходят для разных клиентских проигрывателей, работающих на настольных компьютерах, смартфонах и игровых консолях, подключенных к телевизору. Каждая копия имеет свою ширину потока, что позволяет организовывать адаптивное вещание по протоколу HTTP с помощью технологии DASH.
- *Загрузка видеокопий в сети CDN*. Как только все версии фильма готовы, они загружаются в сети CDN.

Для доставки своих фильмов конечным потребителям компания Netflix активно применяет технологию CDN. Фактически, на момент написания этой книги (2012 год) она одновременно пользовалась услугами *трех* провайдеров — Akamai, Limelight и Level-3.

Разобравшись с отдельными компонентами инфраструктуры, можно перейти к рассмотрению взаимодействия между клиентом и различными серверами, задействованными в доставке фильмов. Веб-страницы для просмотра библиотеки обслуживаются серверами в облаке Amazon. Файл манифеста, который получает клиент, когда пользователь инициирует воспроизведение видеофайла, хранится там же. Он содержит различную информацию, включая упорядоченный список сетей CDN и URL-адреса разных версий фильма, которые используются при воспроизведении с помощью технологии DASH. Ранжирование сетей CDN выполняется администрацией Netflix и может отличаться от сеанса к сеансу. Обычно выбирается тот вариант, который идет первым в списке. После этого DNS-сервер выбранной сети перенаправляет клиента на конкретный сетевой узел, как было описано в разделе 7.2.4. Там меж-

ду ними происходит взаимодействие с помощью технологии DASH. В частности, как уже говорилось в разделе 7.2.3, клиент использует в своих HTTP-сообщениях типа GET заголовок Range, благодаря чему он может запрашивать фрагменты фильма из разных файлов. Netflix использует фрагменты длиной примерно в 4 с[10]. По мере загрузки данных клиент измеряет пропускную способность сети и запускает алгоритм определения ширины потока, чтобы при запросе следующего фрагмента выбрать версию наиболее подходящего качества.

Рис. 7.6. Платформа потокового видеовещания компании Netflix

В сервисе Netflix воплощено множество ключевых методик, которые мы рассматривали в этой главе, включая адаптивное потоковое вещание и распространение данных через сети CDN. Это прекрасный пример того, как крупная Интернет-компания, занимающая почти 30% сетевого трафика, может выполнять свои задачи за счет преимущественно сторонней облачной инфраструктуры и аренды сетей распространения данных.

YouTube

Имея в своей библиотеке около полумиллиарда видеороликов и обслуживая ежедневно сотни миллионов просмотров[135], YouTube является крупнейшим в мире веб-сайтом для распространения видео. Проект был основан в апреле 2005 года, а уже в ноябре 2006 его купила компа-

ния Google. И хотя в этом сервисе используются фирменные наработки и протоколы, некоторые независимые исследования[9, 690, 639] дают нам возможность судить о том, как это все работает.

Как и Netflix, YouTube активно применяет для распространения своих видеороликов технологию CDN[639], но при этом компания Google использует собственные сети распространения контента. Ее серверные кластеры находятся в сотнях разных мест, примерно 50 из которых отводится именно для функционирования сайта YouTube[9]. Пользовательские запросы перенаправляются к конкретным кластерам с помощью DNS-серверов, как было описано в разделе 7.2.4. Обычно выбирается сервер с самым низким показателем двусторонней задержки (Round-trip delay time, или RTT); но чтобы сбалансировать нагрузку между разными кластерами, иногда DNS-сервер направляет запрос к более удаленному узлу[639]. Кроме того, если кластер не содержит запрашиваемый видеоролик, он может просто вернуть HTTP-сообщение, которое перенаправит клиент в другую точку сети[639].

Как уже было сказано в разделе 7.2.2, YouTube выполняет потоковое вещание по протоколу HTTP. У большинства роликов есть несколько версий с разной шириной потока и соответствующим качеством. В 2011 году этот сервис не использовал технологии адаптивного вещания (такие как DASH), заставляя пользователя самостоятельно выбирать подходящую версию видеофайла. Чтобы сократить расход сетевых и серверных ресурсов, связанный с преждевременным завершением воспроизведения, используется HTTP-заголовок Range, который ограничивает количество данных, передаваемых после их предварительной загрузки.

На веб-сайт YouTube каждый день загружается несколько миллионов видеороликов. То есть нужно обслуживать не только потоковое вещание от сервера к клиенту, но и загрузку данных в обратном направлении. Все ролики конвертируются в специальный формат, для них создается несколько версий с разной шириной потока. Вся обработка полностью выполняется внутри вычислительных центров компании Google. В отличие от сервиса Netflix, который почти полностью зависит от сторонних решений, Google полагается исключительно на собственную инфраструктуру, состоящую из вычислительных центров, сетей распространения данных и глобальной частной сети, которая объединяет все это в единое целое (подробней об инфраструктуре компании Google вы узнаете в разделе 7.2.4).

Kankan

Как мы только что увидели на примере сервисов Netflix и YouTube, для доставки видеоданных клиенту используются серверы в составе сетей CDN (частных или сторонних). Компании Netflix и YouTube вынуждены платить не только за аппаратное обеспечение (либо напрямую, либо в виде арендных отчислений), но и за количество данных, переданных по сети. Учитывая масштабы этих сервисов и объемы потребляемого ими трафика, такое клиент-серверное распространение стоит очень больших денег.

Этот подраздел мы завершим рассмотрением совершенно иного подхода к широкомасштабному распространению видео через Интернет, который позволяет значительно снизить расходы на инфраструктуру и трафик. Как вы уже, наверное, догадались, речь пойдет о пиринговой (P2P) доставке, которая, в отличие от клиент-серверной, не требует наличия сетей CDN. Технология P2P успешно используется несколькими китайскими компаниями, включая Kankan (владельцем которой является корпорация Xunlei), PPTV (бывшая PPLive) и PPs (ранее известная под названием PPstream). На текущий момент Kankan является лидирующим пиринговым поставщиком видеопродукции в Китае, ежемесячно обслуживая 20 млн уникальных пользователей.

В целом, пиринговое вещание видеоданных очень похоже на загрузку файлов с помощью технологии BitTorrent (см главу 2). Когда один из узлов хочет посмотреть видео, он связывается с трекером (централизованным или распределенным с помощью технологии DHT), чтобы получить сведения о других узлах в системе, у которых есть искомый файл. Отличие от BitTorrent заключается в том, что загрузка видеофрагментов выполняется последовательно, чтобы обеспечить беспрерывное воспроизведение.

Данные отслеживаются с помощью трекера и собственной таблицы DHT. В распространение наиболее популярных видеороликов вовлечены десятки тысяч узлов, что, как правило, превышает показатели самых крупных скоплений участников сети BitTorrent[131]. Все протоколы, которые используются в сервисе Kankan (для взаимодействия между узлом и трекером, узлом и таблицей DHT, а также между разными узлами), являются закрытыми. Интересно, что по возможности видеофрагменты распространяются между узлами по протоколу UDP; благодаря этому Kankan имеет значительную долю в общем объеме UDP-трафика, передаваемого внутри Китая[685].

7.3. IP-телефония

Передачу голоса по Интернету в режиме реального времени часто называют **интернет-телефонией** или **IP-телефонией**, ведь с точки зрения пользователя это похоже на общение по обычному телефону. Также используется сокращенное название **VoIP** (Voice-over-IP). В данном разделе мы опишем принципы и протоколы, которые стоят за этой технологией. Видеовызовы во многом очень похожи на IP-телефонию, только помимо голоса они могут передавать еще и изображения участников разговора. Чтобы не отвлекаться на второстепенные вопросы, в этом разделе мы сосредоточимся именно на голосовой связи.

7.3.1. Ограничения, вызванные негарантированной доставкой данных

Протокол IP, на котором основан Интернет, имеет одну особенность. Он пытается сделать все для того, чтобы как можно быстрее переправить пакет из источника в место назначения, но при этом нельзя сказать наперед, какие задержки мы получим и какой процент пакетов будет утерян. Подобная непредсказуемость вызывает значительные трудности при разработке систем голосового общения, которые крайне чувствительны к сетевым задержкам, нестабильной скорости и потере данных.

В этом разделе мы рассмотрим несколько методик, которые позволяют улучшить производительность IP-телефонии в условиях, описанных выше. Мы сосредоточим свое внимание на подходах, которые работают на уровне приложений и не требует изменения сетевой инфраструктуры или даже транспортных протоколов. Чтобы наш разговор был более предметным, давайте обсудим ограничения, связанные с негарантированной доставкой данных, на примере реального сервиса VoIP. Отправитель генерирует 8000 байт в секунду; каждых 20 мс эти байты объединяются в отдельные фрагменты. Фрагмент и специальный заголовок (об этом чуть позже) передаются через сокет в виде UDP-сегмента. То есть, размер одного фрагмента равен (20 мс) × (8,000 байт/с) = 160 байт, а UDP-сегмент отправляется каждых 20 мс.

Если в сети есть постоянная задержка, каждый пакет будет доставляться получателю раз в 20 мс. В таких идеальных условиях получатель может просто воспроизводить все фрагменты по мере их поступления. Но, к сожалению, в реальных условиях некоторые пакеты окажутся утеряны, а у большинства остальных задержка будет варьироваться — это

касается даже слабо загруженных сетей. В связи с этим получатель должен более взвешенно подходить к определению времени начала воспроизведения и учитывать возможность потери пакетов.

Потеря пакетов

Возьмем один из UDP-сегментов, сгенерированных вашим приложением. Он находится внутри дейтаграммы протокола IP. Прежде чем достичь конечной точки, дейтаграмма передается по сети, проходя через буферы маршрутизаторов (которые представляют собой очереди пакетов). Вполне возможно, что один из буферов на этом пути окажется заполненным; в этом случае дейтаграмма может никогда не дойти до адресата.

Потерю пакетов можно исключить, если вместо UDP использовать протокол TCP (который гарантирует доставку данных). Однако механизм повторной передачи часто считается неприемлемым для обмена аудиоданными в режиме реального времени, как в случае с VoIP, поскольку он увеличивает время сетевой задержки[61]. Более того, из-за системы предотвращения перегрузок, реализованной в протоколе TCP, любая потеря пакетов может сделать скорость отправки данных меньше скорости их потребления, что приведет к опустошению буфера. Это, в свою очередь, может повлиять на разборчивость передаваемого голоса. Таким образом, большинство из ныне существующих приложений IP-телефонии используют по умолчанию протокол UDP. Есть сведения[43] о том, что сервис Skype переключается на протокол TCP, только если сеть пользователя защищена брандмауэром или механизмом NAT (во всех остальных случаях используется UDP).

Но потеря пакетов — это не такая уж большая проблема, как может показаться на первый взгляд. На самом деле допустимой является потеря до 20% данных, в зависимости от того, как они кодируются и передаются, а также от того, как это компенсируется принимающей стороной. Например, в таких условиях может помочь прямая коррекция ошибок, которая позволяет восстановить некоторые потерянные пакеты. Как вы вскоре увидите, этот подход подразумевает передачу некоторой избыточной информации, которая и делает возможным процесс восстановления. Тем не менее если один или несколько узлов между отправителем и получателем оказались перегруженными, и количество потерянных пакетов достигает 10–20%, вы никак не сможете достичь приемлемого качества звука. Негарантированная доставка данных накладывает определенные ограничения.

Сквозная задержка

Сквозная задержка — это общее время, которое тратится на простой в буферах маршрутизаторов, переход через сетевые узлы и обработку на принимающей стороне. Для пользователей приложений, которые обмениваются аудиоданными в режиме реального времени, сквозная задержка, не превышающая 150 мс, является незаметной; задержки в диапазоне от 150 до 400 мс считаются приемлемыми, но далекими от идеала; все, что выходит за эти рамки, может серьезно усложнить голосовое общение. VoIP-приложения обычно игнорируют любые пакеты, которые доходят до них со слишком большой задержкой (например, дольше 400 мс). Такие данные фактически считаются потерянными.

Колебание времени доставки

Крайне значимая часть сквозной задержки — непостоянное время прохождения пакетов через маршрутизаторы сети. Из-за этого непостоянства время от момента генерации пакета до его получения другой стороной тоже меняется от пакета к пакету (см. рис. 7.1). Такие колебания времени доставки называют **джиттером**. В качестве примера рассмотрим два последовательных пакета в VoIP-приложении. Второй пакет отправляется через 20 мс после первого. Но на принимающей стороне этот интервал может вырасти. Чтобы в этом убедиться, представьте, что первый пакет проходит через маршрутизатор в тот момент, когда у него пустой буфер, но прямо перед прохождением второго пакета этот буфер быстро заполняется из какого-то другого источника. Поскольку первый пакет успел пройти маршрутизатор достаточно быстро, а второму пришлось задержаться, интервал между ними превысил 20 мс. Промежуток между соседними пакетами может и уменьшиться — например, если первый пакет попал в заполненный буфер, а второй — в свободный, то интервал между ними станет минимальным.

Это похоже на езду автомобилей по дороге. Представьте, что вы со своим другом решили прокатиться на собственных машинах из Сан-Диего в Феникс. Допустим, у вас похожий стиль вождения и вы оба едете со скоростью 100 км/ч. Если ваш друг выедет на час раньше, вы прибудете в Феникс примерно через час после него — может, чуть раньше или позже, в зависимости от загруженности автострады.

Если принимающая сторона игнорирует колебания сетевых задержек и начинает воспроизводить звуковые фрагменты сразу после их поступления, качество звучания может сильно пострадать. К счастью, от

этих колебаний можно избавиться, используя **нумерации пакетов**, **временные метки** и **задержку воспроизведения**. Эти методики мы обсудим ниже.

7.3.2. Выравнивание колебаний на принимающей стороне

Наше VoIP-приложение, которое генерирует пакеты с заданной периодичностью, должно уметь воспроизводить принимаемые голосовые фрагменты в условиях непредсказуемых задержек в сети. Обычно для этого используется сочетание следующих механизмов:

- *Присваивание каждому фрагменту* ***временной метки***. Каждый отправляемый фрагмент содержит время своего создания.
- ***Задержка воспроизведения*** *фрагментов принимающей стороной*. Ранее, при обсуждении рис. 7.1, вы узнали, что задержка воспроизведения аудиофрагментов должна быть достаточно большой, чтобы большинство пакетов можно было принять до того, как они понадобятся. Задержка может оставаться постоянной на протяжении всего сеанса или же изменяться динамически.

Теперь мы поговорим о том, как одновременное применение этих механизмов может уменьшить или вовсе устранить все последствия сетевых колебаний. Будут рассмотрены две стратегии — с постоянным и динамическим временем задержки воспроизведения.

Постоянная задержка воспроизведения

В случае постоянной задержки принимающая сторона будет пытаться воспроизводить каждый фрагмент ровно через q миллисекунд после его создания. Поэтому, если фрагмент имеет временную метку t, момент его воспроизведения будет равен $t+q$ (если исходить из того, что он доставка произойдет вовремя). Пакеты, которые поступают после назначенного времени, отклоняются и считаются потерянными.

Какой должна быть оптимальная величина q? VoIP-приложения могут поддерживать задержки до 400 мс, но чтобы обеспечить достойное качество общения, этот показатель должен быть ниже. С одной стороны, слишком маленькое значение q может привести к тому, что многие пакеты не будут доходить до принимающей стороны в назначенное время из-за меняющихся сетевых условий. Грубо говоря, если изменения сквозной задержки происходят постоянно, значение q должно быть

большим. Но если мы имеем дело с незначительными задержками, уровень которых не сильно колеблется, показатель q лучше уменьшить (может даже до 150 мс).

Компромисс между задержкой воспроизведения и потерей пакетов проиллюстрирован на рис. 7.7; на нем отмечены моменты создания и использования звуковых фрагментов для вывода одного голосового потока. Рассматриваются два разных показателя начальной задержки воспроизведения. Левая ступенчатая линия демонстрирует генерирование пакетов с постоянным интервалом — скажем, каждых 20 мс. Первый пакет в этой цепочке доходит до потребителя в момент времени r. Как видно на рис., из-за сетевых колебаний моменты появления последующих пакетов распределены не равномерно.

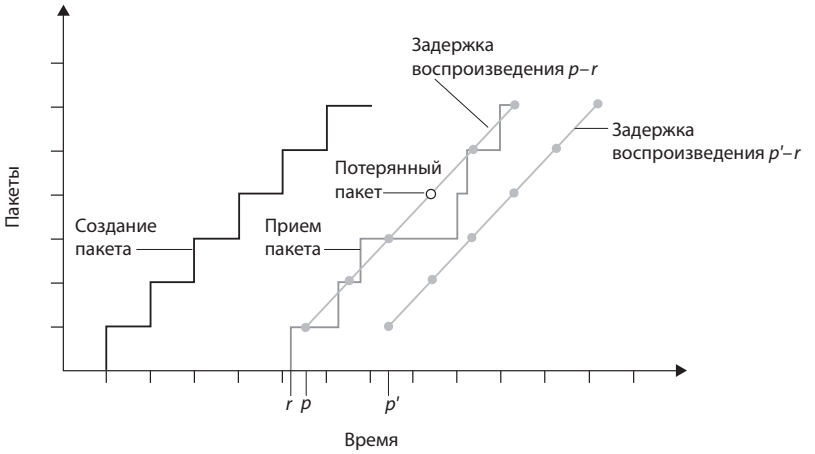


Рис. 7.7. Потеря пакетов при выборе разных показателей задержки воспроизведения

В первом случае задержка планируется из расчета $p–r$. Четвертый пакет не успевает появиться в назначенное время, и принимающая сторона считает его потерянным. Во втором случае задержка равняется $p'–r$, благодаря чему всего пакеты приходят раньше запланированного времени и, как результат, не теряются.

Динамическая задержка воспроизведения

Предыдущий пример демонстрирует неизбежность компромисса, на который приходится идти в случае выбора стратегии с постоянной задержкой воспроизведения. Высокий уровень задержки позволяет во-

время доставлять большинство пакетов, минимизируя их потерю; но для систем голосового общения, таких как IP-телефония, большая задержка является довольно неприятным, а иногда и вовсе неприемлемым фактором. В идеале нам бы хотелось свести величину задержки к минимуму, сохранив при этом потерю пакетов на уровне нескольких процентов.

Для того чтобы справиться с этой проблемой, логично было бы оценивать величину и колебания сетевой задержки перед началом передачи каждого голосового потока, регулируя соответствующим образом задержку воспроизведения. Такой подход позволит сжимать данные во время пауз в разговоре; если применять его в разумных пределах, пользователи не заметят никаких изменений.

Ниже представлен общий алгоритм динамического регулирования задержки воспроизведения, который может использоваться на принимающей стороне[409]. Итак, пусть

- t_i — временная метка i-го пакета (момент создания пакета на стороне отправителя).
- r_i — момент получения i-го пакета.
- p_i — момент воспроизведения i-го пакета.

Сквозная сетевая задержка i-го пакета равна $r_i - t_i$. Учитывая колебания задержек в сети, этот показатель может быть разным для разных пакетов. d_i обозначает *среднюю* оценочную продолжительность сетевой задержки до получения i-го пакета. Это можно выразить в виде следующей формулы:

$$d_i = (1-u)d_{i-1}+u(r_i-t_i)$$

где u является константой (например, $u = 0{,}01$). Таким образом, d_i — это нормализованный усредненный показатель наблюдаемых сетевых задержек: $r_1 - t_1, \ldots, r_i - t_i$. При расчете этого значения больший акцент делается на более свежих замерах. Такой подход должен вам кое-что напоминать; похожий принцип использовался в главе 3, когда мы оценивали время на передачу и подтверждение приема пакетов в протоколе TCP. Давайте выведем из d_i предположительное среднее отклонение задержки (назовем его v_i). Здесь нам тоже пригодятся временные метки.

$$v_i = (1-u)v_{i-1}+u|r_i-t_i-d_i|$$

Оценочные значения p_i и v_i вычисляются для каждого полученного пакета, хотя с их помощью определяется только момент начала воспроизведения голосового потока.

Имея эти значения, принимающая сторона использует при воспроизведении следующий алгоритм. Если i — это первый пакет в голосовом потоке, момент его воспроизведения, p_i, вычисляется так:

$$p_i = t_i + d_i + Kv_i$$

Здесь K — это положительная константа (например, $K = 4$). Слагаемое Kv_i позволяет отложить начало воспроизведения достаточно далеко, чтобы из-за опоздания было утеряно как можно меньше пакетов. Все последующие пакеты в голосовом потоке воспроизводятся со сдвигом относительно первого. В частности, пусть

$$q_i = p_i - t_i$$

будет промежутком времени между созданием и началом воспроизведения первого пакета. Если пакет j тоже является частью этого голосового потока, он будет использован в момент времени

$$p_j = t_j - q_i$$

Вышеописанный алгоритм имеет смысл при условии, что принимающая сторона способна определить начало голосового потока. Это можно сделать путем анализа уровня сигнала в каждом полученном пакете.

7.3.3. Восстановление потерянных пакетов

Мы уже обсудили то, как VoIP-приложения могут справляться с колебаниями задержек доставки в сети. Теперь рассмотрим несколько приемов, которые позволяют поддерживать приемлемое качество звука при потере пакетов. Такие приемы называются **восстановлением потерь**. Здесь мы подходим к факту потери в общем смысле — пакет может просто не дойти до получателя, или дойти, но уже после запланированного времени. В качестве примера мы опять возьмем IP-телефонию.

Как уже упоминалось в начале этого раздела, в приложениях для голосового общения в режиме реального времени повторная передача пакетов может быть просто невыполнима. В самом деле, нет никакого смысла заново передавать пакет, который уже пропустил момент своего воспроизведения. А данные, застрявшие в переполненных буферах маршрутизаторов, все равно нельзя доставить достаточно быстро. Учитывая эти обстоятельства, в IP-телефонии часто используется технология предупреждения потерь данных. Мы рассмотрим два подхода подобного рода: **прямую коррекцию ошибок** и **чередование пакетов**.

Прямая коррекция ошибок

Основная идея, которая лежит в основе прямой коррекции ошибок (ПКО), заключается в добавлении к исходному потоку дополнительной информации. За счет незначительного увеличения объема передаваемых данных вы получаете возможность восстанавливать приблизительные или точные копии потерянных пакетов. Можно выделить две основных разновидности ПКО[61, 387]. Первая подразумевает отправку дополнительной информации через каждые n фрагментов. Избыточный фрагмент извлекается путем применения исключающего ИЛИ к n исходным фрагментам[597]. Таким образом, если в группе n+1 потеряется один пакет, принимающая сторона сможет его полностью восстановить. Но при потере двух и более пакетов восстановление выполнить нельзя. Сделав группу n+1 достаточно маленькой, вы сможете справиться с потерей значительной части пакетов (если таковых не слишком много). Но с уменьшением размера группы возрастает объем передаваемых данных — а именно, в $1/n$ раз. Следовательно, если $n = 3$, ширина потока увеличится на 33%. Кроме того, этот простой подход увеличивает задержку воспроизведения, поскольку принимающая сторона должна получить всю группу пакетов целиком. Чтобы узнать больше подробностей об использовании ПКО для передачи мультимедийных данных, см. RFC 5109.

Вторая разновидность ПКО заключается в том, что в качестве избыточной информации отправляется звуковой поток более низкого качества. Например, к исходному потоку шириной 64 Кбит/с, закодированному в формат PCM, отправитель может добавить звуковую дорожку в стандарте GSM с частотой 13 Кбит/с, которая будет рассматриваться как избыточная. Как можно видеть на рис. 7.8, при создании n-го пакета из исходного и дополнительного потоков берутся, соответственно, n-й и $(n - 1)$-й фрагменты. Таким образом, принимающая сторона может компенсировать потерю произвольного пакета путем воспроизведения следующего звукового фрагмента из дополнительной звуковой дорожки. Конечно, качество этого фрагмента будет ниже, но в целом, учитывая отсутствие потери данных, качество потока останется на хорошем уровне. Стоит также отметить, что для начала воспроизведения принимающая сторона должна получить всего два пакета, что положительно сказывается на задержке. К тому же, если ширина потока вспомогательной звуковой дорожки намного ниже, чем у оригинала, увеличение объема передаваемых данных получается незначительным.

Чтобы справиться с потерей последовательных пакетов, мы можем предусмотреть некоторое отклонение. Помимо $(n - 1)$-го фрагмента

отправитель может добавлять к n-му пакету из оригинального потока $(n - 2)$-й или $(n - 3)$-й фрагмент. Чем больше фрагментов вы добавите, тем легче будет поддерживать приемлемое качество звука в жестких условиях негарантированной доставки данных. С другой стороны, это увеличивает объем передаваемых данных и задержку воспроизведения.

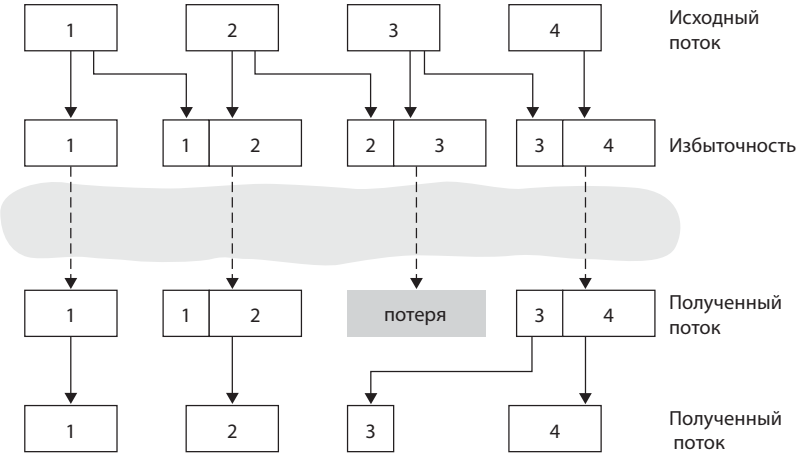


Рис. 7.8. Передача дополнительного потока низкого качества

Чередование пакетов

Помимо отправки избыточной информации VoIP-приложение может использовать чередование фрагментов данных. Как показано на рис. 7.9, прежде чем передавать пакеты, отправитель меняет их местами, чтобы пакеты, которые изначально находились рядом, были разделены. Такой подход может смягчить эффект, вызванный потерей данных. Например, если мы имеем пакеты длиной 5 мс и звуковой фрагмент длиной 20 мс (то есть четыре пакета в одном фрагменте), тогда первый фрагмент может содержать пакеты под номерами 1, 5, 9 и 13; во втором фрагменте могут находиться пакеты 2, 6, 10 и 14, и так далее. На рис. 7.9 мы можем видеть, что потеря одного пакета приводит к нескольким небольшим пропускам в восстановленной версии звукового потока; без чередования пакетов мы бы потеряли целый фрагмент.

Данный подход может значительно улучшить качество звукового потока с точки зрения субъективного восприятия[387]. К тому же, он не влечет за собой больших издержек. Однако главным его недостатком является увеличение задержки. Это значительно ограничивает его приме-

нение в приложениях для голосового общения, хотя он может подойти для потокового вещания статических аудиоданных. Основное преимущество чередования пакетов заключается в том, что оно не увеличивает объем передаваемых данных.

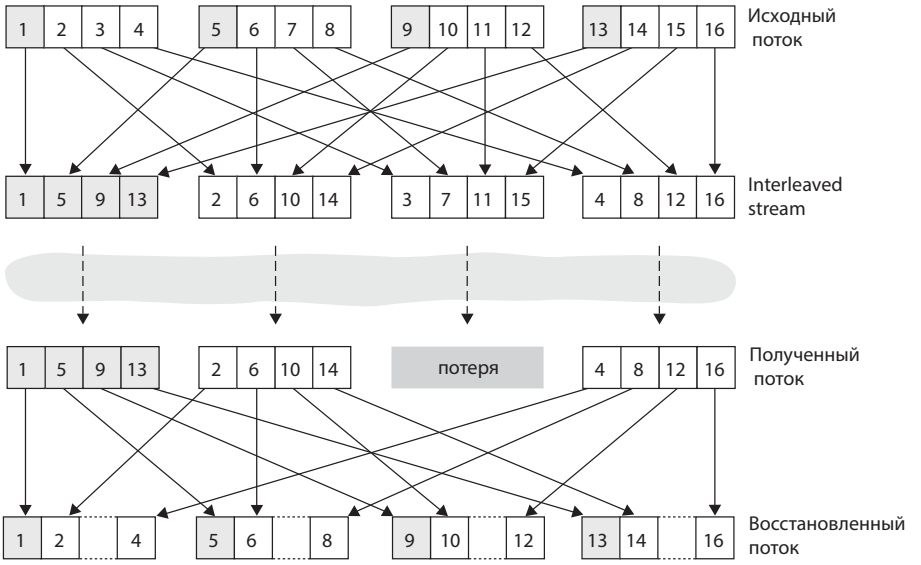


Рис. 7.9. Чередование пакетов при передаче аудиоданных

Маскировка ошибок

Маскировка ошибок — это попытка заменить утерянные пакеты данными, похожими на оригинал. Это вполне реальная задача, так как звуковые сигналы (в частности, человеческая речь) характеризуются большим количеством схожих мелких фрагментов[387]. Таким образом, подобные методики годятся при относительно небольших потерях (до 15%) и в условиях, когда пакеты хранят короткие звуковые отрезки (4–40 мс). Когда теряются отрезки потока, сравнимые с длиной фонемы (5–100 мс), эффективность этих методик резко снижается, поскольку из разговора могут исчезать целые слоги.

Наверное, простейший способ восстановления данных на принимающей стороне заключается в повторении пакетов. В качестве замены используются копии пакетов, которые были приняты непосредственно перед потерей. Этот алгоритм обладает низкой вычислительной сложностью и работает довольно неплохо. Есть и другой подход: можно

брать звуковые фрагменты до и после потери, интерполируя нужные пакеты. По сравнению с простым повторением интерполяция дает немного лучший результат, но ее требования к вычислительным ресурсам значительно выше[387].

7.3.4. Исследование VoIP-приложения на примере Skype

Skype — это чрезвычайно популярное VoIP-приложение, которым ежедневно пользуется более 50 миллионов человек. Оно позволяет выполнять звонки между сетевыми узлами и телефонами, организовывать видеоконференции с большим количеством участников (под сетевым узлом имеется в виду любое устройство, подключенное к Интернету, в том числе персональные компьютеры, планшеты и смартфоны). В 2011 году приложение Skype было куплено компанией Microsoft за более чем 8 млрд. долларов.

Skype использует закрытый протокол с шифрованием всех пакетов, поэтому сложно сказать, что именно происходит внутри. Тем не менее, благодаря документации, размещенной на официальном веб-сайте, а также нескольким исследованиям[43, 193, 89, 629, 686, 414], мы можем судить об общем принципе его работы. При кодировании видео и аудиоданных в Skype используется широкий спектр кодеков, что позволяет поддерживать разные уровни качества с разной шириной потока. Согласно измерениям, в сеансе с самым низким качеством данные передаются на скорости 30 Кбит/с; ширина высококачественного потока достигает 1 Мбит/с[686]. Как правило, Skype обеспечивает лучшее качество звука, чем в проводных телефонных сетях, за счет более высокой частоты дискретизации (16000 Гц против 8000 Гц). По умолчанию аудио- и видеоданные передаются по протоколу UDP. TCP используется для отправки управляющих пакетов и в случаях, когда UDP-потоки блокируются брандмауэром. Для восстановления потерянных пакетов применяется прямая коррекция ошибок. Клиентское приложение Skype способно адаптироваться к текущему состоянию сети, подбирая подходящие качество и объем избыточных данных для ПКО[686].

Компания Skype довольно оригинально применяет пиринговые технологии, еще раз доказывая, что с помощью P2P-сетей можно заниматься не только распространением хранимых данных. По аналогии с системами обмена мгновенными сообщениями, Интернет-телефония основывается на одноранговых сетях, соединяя пользователей (то есть, одноранговые сетевые узлы) напрямую, в режиме реального времени.

Однако в Skype технологии P2P выполняют еще две важных функции: обнаружение пользователей и обход систем NAT.

Как показано на рис. 7.10, сетевые узлы (пиры) в системе Skype организованы в иерархическую наложенную сеть, где участники делятся на главных и обычных. Skype ведет список, в котором имя пользователя привязывается к его текущему IP-адресу (и номеру порта). Этот список распространяется между главными узлами. Когда Алиса хочет позвонить Бобу, ее клиентское приложение выполняет поиск по распределенному списку, чтобы найти соответствующий IP-адрес. Поскольку в Skype используется закрытый протокол, нельзя с уверенностью сказать, как именно происходит привязка в главных сетевых узлах; вероятно, применяется некая разновидность распределенной хеш-таблицы (DHT).

Пиринговые методики также используются в **ретрансляторах**, что довольно полезно в ситуациях, когда вызов выполняется в рамках одной домашней сети. Как уже говорилось в главе 4, многие внутренние сети ограждены от Интернета с помощью NAT. NAT не дает внешнему узлу инициировать соединение с участником внутренней сети. Если *оба* участника разговора защищены с помощью NAT, у сервиса Skype возникает проблема, ведь ни один из них не может принимать внешние соединения, что, на первый взгляд, исключает всякую возможность выполнить вызов. Но нестандартный подход к организации главных узлов и ретрансляторов может решить эту проблему. Допустим, при входе в систему Алиса была привязана к главному узлу, не огражденному с помощью NAT. Теперь она пытается создать сеанс с этим узлом (NAT не станет помехой, поскольку Алиса является инициатором соединения), с помощью которого они смогут обмениваться управляющими сообщениями. То же самое будет касаться Боба, когда тот войдет в систему. Теперь, если Алиса хочет позвонить Бобу, она связывается со своим главным узлом, который, в свою очередь, информирует Боба о входящем вызове. Если Боб «снимает трубку», два главных узла находят так называемый ретранслятор, не защищенный с помощью NAT, основной задачей которого является передача данных между Алиса и Бобом. Затем оба узла связываются со своими пользователями и «объясняют» им, как установить сеанс с ретранслятором. Как видно на рис. 7.10, Алиса отправляет голосовые пакеты непосредственно ретранслятору (соединение с котором создала именно она), а тот передает их напрямую Бобу (по соединению, инициированному самим пользователем). В обратную сторону пакеты идут по тому же маршруту. И *вуаля*! Боб и Алиса могут общаться, даже несмотря на то, что ни один из них не может принимать подключения извне.

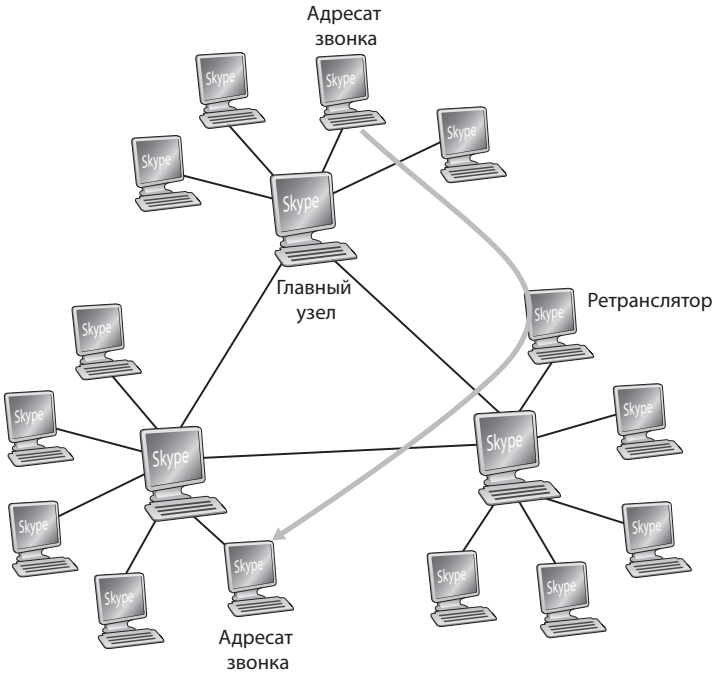


Рис. 7.10. Сетевые узлы в системе Skype

Пока что речь шла о разговоре двух людей. Теперь давайте рассмотрим групповые телефонные конференции. Допустим, количество участников N больше 2. Если каждый пользователь отправляет копию своего звукового потока $N-1$ участникам, тогда для поддержки общения по сети нужно иметь $N(N-1)$ потоков. Чтобы уменьшить объем передаваемых данных, Skype использует некоторые нестандартные приемы. В частности, все участники разговора шлют свои потоки пользователю, который инициировал конференцию. Там все данные объединяются в единый поток (фактически, путем суммирования звуковых сигналов) и рассылаются каждому из $N-1$ участников. Таким образом, количество потоков снижается до $2N-1$. Для обычных вызовов с двумя участниками Skype использует прямое соединение; исключение составляют ситуации, когда нужно обходить NAT — в таком случае данные идут через ретранслятор, как было описано выше. В групповых видеоконференциях потоки не объединяются в один; вместо этого видеоданные в количестве $N-1$ потоков передаются на серверный кластер (который по состоянию на 2011 находился в Эстонии), откуда переправляются каждому из $N-1$ участников[686]. Но зачем каждый пользователь отправляет свои данные на сервер, а не напрямую другим участникам? Действительно, в обо-

их случаях каждый из N пользователей получит $N(N–1$ потоков. Дело в том, что в большинстве точек доступа исходящая скорость передачи данных значительно ниже входящей, поэтому клиентское приложение может не справиться с отправкой $N–1$ потоков.

С появлением систем IP-телефонии, таких как Skype, QQ и Google Talk, начали возникать новые проблемы, связанные с конфиденциальностью данных. Вернемся к нашему примеру. Алиса имеет возможность извлечь IP-адрес Боба и с помощью геолокационных сервисов[403, 335] определить его текущее местоположение и название его Интернет-провайдера (домашнего или корпоративного). Фактически, при выполнении вызова в Skype Алиса может блокировать передачу определенных пакетов; это позволит ей скрыть отслеживание IP-адреса и выполнить соединение даже в том случае, если она не находится в списке контактов Боба. Более того, один и тот же адрес может использоваться как в Skype, так и в BitTorrent, поэтому теоретически у Алисы есть возможность следить за тем, какие файлы загружает себе Боб[313, 314]. К тому же, вызов, выполняемый с помощью Skype, можно расшифровать путем анализа размеров передаваемых пакетов[665].

7.4. Протоколы для общения в режиме реального времени

Приложения для общения в реальном времени, включая системы IP-телефонии и видеоконференций, на сегодняшний день являются довольно развитыми и популярными. Неудивительно, что такие организации как IETF и ITU уже много лет неустанно работают над стандартами для подобного рода сервисов. Благодаря этому, продукты, созданные разными компаниями, могут быть совместимыми друг с другом. В этом разделе вы узнаете, какое место в приложениях для общения в реальном времени занимают протоколы RTP и SIP. Оба эти стандарта получили широкое распространение в промышленных продуктах.

7.4.1. Протокол RTP

В предыдущем разделе мы узнали, что перед отправкой звуковых фрагментов VoIP-приложения добавляют в них заголовочные поля. Эти поля содержат последовательные номера и временные метки. Поскольку такой подход применяется в большинстве мультимедийных сетевых приложений, было бы логично стандартизировать структуру

пакетов, чтобы они включали в себя одни и те же заголовочные данные. Такая стандартизация была проделана в протоколе RTP[518]. RTP можно использовать для передачи популярных форматов аудио- (PCM, ACC и MP3) и видеоданных (MPEG и H.263). С его помощью можно передавать и закрытые мультимедийные форматы. На сегодняшний день этот протокол получил широкое распространение во многих продуктах и исследовательских прототипах. Кроме того, он совместим с другими популярными протоколами для общения в реальном времени — например, с SIP.

В этом разделе мы познакомимся с протоколом RTP. Мы рекомендуем вам посетить веб-сайт Хеннинга Шулзри[589], на котором собрано множество информации на эту тему. Вам также стоит ознакомиться с документацией к утилите Robust Audio Tool, в которой используется протокол RTP[412].

Основы протокола RTP

Протокол RTP обычно функционирует поверх UDP. Отправитель заключает мультимедийный фрагмент внутрь RTP-пакета, полученный результат заворачивается в UDP-сегмент и передается по протоколу IP. Получатель последовательно извлекает сначала RTP-пакет, а потом и сам мультимедийный фрагмент, передавая его для дальнейшего декодирования и воспроизведения.

В качестве примера использования протокола RTP рассмотрим передачу голосового потока. Предположим, что звук кодируется в формате PCM (что подразумевает дискретизацию, квантование и оцифровку) с шириной потока 64 кбит/с. Наше приложение должно разбивать закодированные данные на фрагменты по 20 мс (то есть, размер одного фрагмента равен 160 байт). К каждому фрагменту перед отправкой добавляется **RTP-заголовок**, содержащий информацию об аудио-формате, порядковый номер и временную метку (обычно все это занимает 12 байт). Из заголовка и звукового фрагмента формируется **RTP-пакет**, который потом отправляется посредством UDP-сокета. Принимающая сторона получает из сокета RTP-пакет, извлекает из него аудиофрагмент и с помощью информации, хранящейся в заголовке, пытается его декодировать и воспроизвести.

Приложение, использующее для передачи данных о звуковых фрагментах стандартные RTP-заголовки, легче адаптировать к взаимодействию с другими мультимедийными системами. Например, если две раз-

ных компании разрабатывают VoIP-приложения на основе протокола RTP, есть шанс, что в будущем их пользователи смогут общаться между собой. В разделе 7.4.2 вы увидите, что связка протоколов RTP и SIP является важным стандартом в Интернет-телефонии.

Нужно подчеркнуть, что RTP не предоставляет никаких механизмов для обеспечения своевременной доставки данных и не дает никаких гарантий относительно качества обслуживания (QoS); этот протокол не убережет вас даже от потери или изменения порядка следования пакетов. Фактически его действие можно наблюдать исключительно на конечных узлах. Для маршрутизаторов нет никакой разницы, использовался ли протокол RTP при передаче IP-дейтаграмм.

Каждый источник данных (например, камера или микрофон) может иметь свой собственный независимый RTP-поток. В случае с двусторонней видеоконференцией можно открыть четыре потока — два для передачи звука (по одному в каждом направлении) и еще два для передачи видео. Однако многие методы кодирования, включая MPEG 1 и MPEG 2, подразумевают упаковывание аудио и видео в единый поток. В таких условиях будет достаточно иметь один RTP-поток в каждом направлении.

RTP-пакеты можно использовать не только для однонаправленной передачи. Их можно распространять по принципу «один-ко-многим» и «многие-ко-многим». Во втором случае все отправители и источники обычно используют одну и ту же многоадресную группу RTP-потоков. Связанные вместе многоадресные RTP-потоки, генерируемые разными отправителями в рамках видеоконференции, формируют **RTP-сеанс**.

Заголовочные поля RTP-пакетов

Как видно на рис. 7.11, в заголовочных полях RTP-пакета хранятся тип данных, порядковый номер, временная метка и идентификатор источника.

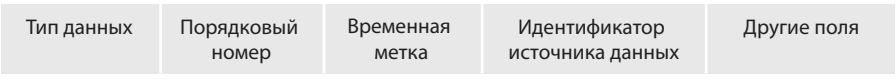


Рис. 7.11. Заголовочные поля RTP-пакета

Поле с типом данных занимает 7 бит. В случае со звуковым потоком оно используется для обозначения текущего формата аудиоданных (на-

пример, PCM, адаптивная дельта-модуляция, кодирование с линейным предсказанием). Если отправитель решит изменить способ кодирования посреди сеанса, он сможет проинформировать об этом получателя именно с помощью данного поля. Изменения могут быть вызваны необходимостью повысить качество звука или уменьшить ширину RTP-потока. В табл. 7.2 перечислены некоторые аудиоформаты, поддерживаемые протоколом RTP.

Табл. 7.2. Аудиоформаты, поддерживаемые протоколом RTP

Номер формата	Аудио-формат	Частота дискретизации, кГц	Ширина потока, кбит/с
0	PCM μ-law	8	64
1	1016	8	4,8
3	GSM	8	13
7	LPC	8	2,4
9	G.722	16	48–64
14	MPEG Audio	90	—
15	G.728	8	16

Это поле также используется для обозначения форматов кодирования в видеопотоках (например, motion JPEG, MPEG 1, MPEG 2, H.261). Опять же, отправитель может менять его значение на ходу, не дожидаясь завершения сеанса. Некоторые из видеоформатов, поддерживаемых протоколом RTP, описаны в табл. 7.3. Другие важные поля перечислены ниже:

Табл. 7.3. Некоторые видеоформаты, поддерживаемые протоколом RTP

Номер формата	Видео-формат
26	Motion JPEG
31	H.261
32	MPEG 1 video
33	MPEG 2 video

- *Порядковый номер*. Занимает 16 бит. Это значение инкрементируется на единицу при отправке каждого нового RTP-пакета. С его помощью на стороне получателя можно следить за потерями данных и выполнять их восстановление. Например, если приложение принимает поток RTP-пакетов и в последовательности порядковых номеров обнаружился промежуток между 86 и 89, это означает, что

пакеты с номерами 87 и 88 не дошли до адресата. В итоге получатель может попытаться как-то компенсировать эту потерю.

- *Временная метка.* Занимает 32 бита. Содержит время дискретизации первого байта в RTP-пакете. Как мы уже видели в предыдущем разделе, с помощью временных отметок можно нивелировать сетевые колебания на принимающей стороне и обеспечивать синхронное воспроизведение. Эти значения формируются на основе периода дискретизации на стороне отправителя (например, для частоты 8 кГц это будет 125 мкс). Если в одном закодированном звуковом фрагменте содержится 160 таких периодов, значение временной метки будет увеличиваться на 160 периодов для каждого RTP-пакета, генерируемого источником. Наращивание значения временной метки продолжается с постоянной частотой даже после того, как источник перестает отдавать данные.
- *Идентификатор источника данных (Synchronization source identifier, SSRC).* Занимает 32 бита. Идентифицирует источник RTP-потока. Обычно каждый поток в RTP-сеанса имеет собственный идентификатор. Это не IP-адрес, а случайное число, которое генерируется источником при создании нового потока. Вероятность совпадения двух идентификаторов крайне мала; в таких случаях источники выбирают новые значения.

7.4.2. Протокол SIP

Протокол установления сеанса (Session Initiation Protocol или SIP) описывается в документах RFC 3261 и RFC 5411. Это несложный и открытый стандарт, который выполняет следующие функции:

- Позволяет установить сетевое соединение между вызывающей и вызываемой стороной, информируя последнюю о начале вызова. Участники разговора имеют возможность согласовать формат кодирования мультимедийных данных. Завершить вызов может каждый из хостов.
- Позволяет инициатору вызова узнать текущий IP-адрес вызываемой стороны. Адресов может быть несколько (для разных устройств), и они могут меняться из-за использования DHCP.
- Дает возможность управлять вызовом — например, добавлять новые потоки, менять формат кодирования данных, приглашать новых участников, перенаправлять и откладывать вызов (все это во время разговора).

Подготовка вызова по известному IP-адресу

Чтобы лучше понять суть протокола SIP, рассмотрим конкретный пример. Представьте, что Алиса хочет позвонить Бобу и при этом оба они находятся за своими компьютерами, оснащенными VoIP-приложениями с поддержкой SIP. Мы будем исходить из того, что Алиса знает IP-адрес Боба. Весь процесс вызова проиллюстрирован на рис. 7.12.

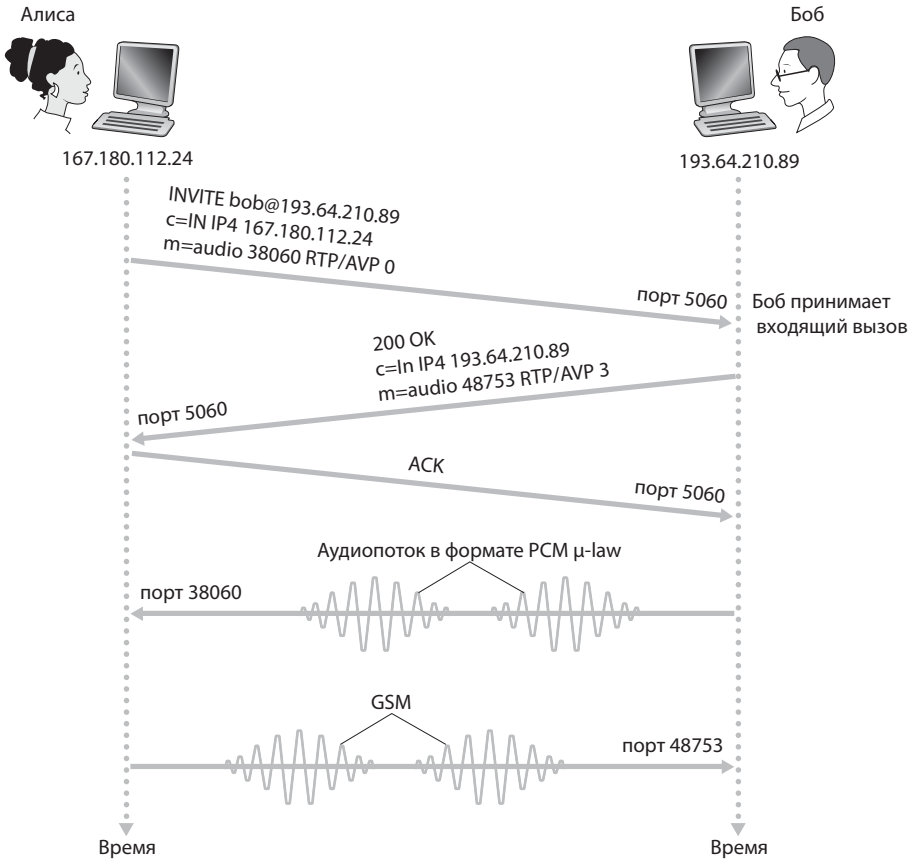


Рис. 7.12. Процесс вызова по протоколу SIP с заранее известным IP-адресом

Как видите, сеанс начинается с того, что Алиса шлет Бобу приглашительное сообщение (INVITE), которое чем-то напоминает HTTP-запрос. При этом используется протокол UDP и стандартный для SIP порт под номером 5060 (хотя это можно делать и по TCP). Сообщение содержит идентификатор Боба (bob@193.64.210.89), IP-адрес Алисы, обозначение того, что клиент хочет получать звук в формате AVP 0

(PCM μ-law), инкапсулированный в виде RTP-потока, а также номер порта для приема RTP-пакетов (38060). Получив приглашение, Боб отправляет ответ похожего вида, используя порт 5060. Ответ содержит код состояния (200 OK), IP-адрес, желательный формат кодирования и пакетирования, а также номер порта, по которому должны отправляться аудиопакеты. Стоит отметить, что в этом примере стороны выбрали разные способы кодирования данных: Алиса предпочла GSM, тогда как Боб запросил формат PCM μ-law. Получив ответ от Боба, Алиса отправляет ему подтверждение. После этой транзакции можно начинать непосредственное общение (на рис. 7.12 показано, что Алиса и Боб говорят по очереди, но это всего лишь упрощение — одновременный разговор является вполне нормальной ситуацией). Голос Боба кодируется и пакетируется согласно предъявленным требованиям и передается по IP-адресу 167.180.112.24 с номером порта 38060. Алиса, также соблюдая договоренность относительно формата аудиоданных, отправляет пакеты по IP-адресу 193.64.210.89 с номером порта 48753.

С помощью этого простого примера мы познакомились с рядом ключевых характеристик протокола SIP. Во-первых, для отправки и приема сообщений используются отдельные сокеты, не имеющие ничего общего с передачей мультимедийных данных. Во-вторых, сами сообщения представляют собой обычный текст в формате ASCII и чем-то напоминают HTTP-запросы. В-третьих, получение каждого сообщения должно подтверждаться, поэтому протокол SIP может работать поверх UDP или TCP.

Давайте представим, что бы случилось, если бы у Боба не оказалось нужного кодека (PCM μ-law). В этом случае вместо «200 OK» он вернул бы сообщение «606 Not Acceptable» и перечислил бы все кодеки, которые может использовать. Тогда Алиса выбрала бы другой кодек и заново послала пригласительное сообщение. Боб также мог бы просто отклонить вызов, вернув в ответ один из множества кодов отказа (например, busy, gone, payment required и forbidden).

SIP-адреса

В предыдущем примере у Боба был SIP-адрес вида sip:bob@193.64.210.89. Однако подобные идентификаторы в большинстве своем похожи на адреса электронной почты. В случае с Бобом он мог бы выглядеть как **sip:bob@domain.com**. Он бы точно так же содержался внутри пригласительного сообщения, которое отправляет SIP-устройство

Алисы; система, поддерживающая этот протокол, автоматически перенаправила бы сообщение по IP-адресу, к которому в данный момент подключено устройство Боба (об этом мы поговорим ниже). SIP-адреса могут также принимать вид телефонного номера или имени/фамилии/отчества (при условии, что они уникальные).

Интересной особенностью SIP-адресов является то, что их можно размещать на веб-страницах — точно так же, как это делается с адресами электронной почты при помощи URL вида mailto:. Представьте, что у Боба есть домашняя страница в Интернете и ему захотелось предоставить посетителям механизм, с помощью которого они смогут ему звонить. Для этого ему достаточно опубликовать ссылку sip:bob@domain.com. Перейдя по этому адресу, компьютер посетителя автоматически запустит SIP-приложение и отправит Бобу сообщение INVITE.

Сообщения в протоколе SIP

В этом беглом обзоре протокола SIP мы не станем рассматривать все виды сообщений и заголовков, которые он поддерживает, и сосредоточим свое внимание на самых распространенных из них — в частности, на сообщении SIP INVITE. Давайте опять представим, что Алиса хочет позвонить Бобу, но на этот раз ей известен только SIP-адрес его устройства, bob@domain.com. Ее пригласительное сообщение будет выглядеть следующим образом:

```
INVITE sip:bob@domain.com SIP/2.0
Via: SIP/2.0/UDP 167.180.112.24
From: sip:alice@hereway.com
To: sip:bob@domain.com
Call-ID: a2e3a@pigeon.hereway.com
Content-Type: application/sdp
Content-Length: 885

c=IN IP4 167.180.112.24
m=audio 38060 RTP/AVP 0
```

В первой строчке содержится версия протокола и тип запроса. Каждый раз, когда SIP-сообщение проходит через устройство (в том числе и через источник этого сообщения), к нему добавляется заголовок Via, в котором указан промежуточный IP-адрес (скоро вы увидите, что, пре-

жде чем дойти до адресата, приглашение проходит множество других устройств). Как и электронные письма, SIP-сообщения содержат заголовки From и To. Также они включают в себя уникальный идентификатор вызова, `Call-ID` (по аналогии с message-ID в электронных письмах). Стоит также обратить внимание на заголовки `Content-Type` и `Content-Length`. Первый определяет формат описания содержимого SIP-сообщением, а второй содержит сведения о его размере (в байтах). В конце, после пустой строчки и перевода строки находятся сами данные. В нашем случае это информация об IP-адресе Алисы и о том, в каком виде она хочет получать звуковой поток.

Трансляция доменных имен и местонахождение пользователя

В примере, представленном на рис. 7.12, мы исходили из того, что SIP-устройство, принадлежащее Алисе, знает IP-адрес, по которому можно связаться с Бобом. Но это предположение довольно нереалистичное — и не только потому, что IP-адреса часто назначаются динамически с помощью DHCP; дело в том, что у Боба может быть несколько VoIP-устройств (например, одно дома, другое на работе, третье в автомобиле). А теперь представьте, что Алисе известен только адрес электронной почты, bob@domain.com, который он также использует для звонков по протоколу SIP. Чтобы получить текущий IP-адрес Боба, Алиса создает пригласительное сообщение, которое начинается строчкой «INVITE bob@domain.com SIP/2.0», и отправляет его **прокси-серверу SIP**. Тот возвращает ответ, который может содержать искомый IP-адрес. Но это также может быть адрес голосовой почты или веб-страницы (например, с надписью «Боб решил вздремнуть. Оставьте меня в покое!»). К тому же, эта информация может зависеть от того, кто ее запрашивает. Например, Боб может принять звонок от жены, предоставив свой IP-адрес, а тещу перенаправить на вышеупомянутую веб-страницу!

Сейчас вам, наверное, интересно, каким образом прокси-сервер SIP определяет текущий IP-адрес по идентификатору bob@domain.com? Чтобы ответить на этот вопрос, нужно сказать несколько слов о еще одном SIP-устройстве — **регистраторе**. С ним должен быть связан каждый пользователь этого протокола. При запуске SIP-приложение отправляет регистратору соответствующее сообщение, информируя его о своем текущем IP-адресе. Например, клиентская программа, которую Боб запускает на своем коммуникаторе, должна отправить сообщение следующего вида:

```
REGISTER sip:domain.com SIP/2.0
Via: SIP/2.0/UDP 193.64.210.89
From: sip:bob@domain.com
To: sip:bob@domain.com
Expires: 3600
```

Информация о текущем IP-адресе Боба хранится в его регистраторе. Каждое устройство, которое начинает использовать Боб, отправляет регистрационное сообщение с обновленными данными. Если оно используется на протяжении какого-то длительного периода времени, регистратору все равно периодически отправляются сообщения, которые подтверждают, что последний указанный IP-адрес все еще остается актуальным (в примере, представленном выше, это делается раз в 3600 с).

Стоит отметить, что по своему принципу работы регистратор похож на доверенный DNS-сервер; он тоже транслирует некий фиксированный и понятный для человека идентификатор (например, bob@domain.com) в IP-адрес, только в случае с SIP этот адрес может быть динамическим. Регистраторы и прокси-серверы SIP часто работают на одном и том же сетевом узле.

Теперь давайте посмотрим, как прокси-сервер, связанный с Алисой, получает текущий IP-адрес Боба. Вы уже знаете, что для этого достаточно просто переправить исходное пригласительное сообщение регистратору или прокси-серверу Боба, откуда оно потом будет передано соответствующему устройству. После этого Боб сможет отослать обратно свой ответ.

В качестве примера возьмем рис. 7.13, на котором показано, как пользователь jim@umass.edu с IP-адресом 217.123.56.89 хочет позвонить пользователю keith@upenn.edu с IP-адресом 197.87.54.21. При этом выполняются следующие шаги:

1. Джим шлет пригласительное сообщение прокси-серверу umass.
2. Прокси-сервер выполняет DNS-запрос в попытке найти адрес регистратора с доменным именем upenn.edu (не показано на рис.) и затем переправляет сообщение этому регистратору.
3. Поскольку идентификатор keith@upenn.edu больше не зарегистрирован на сервере upenn, регистратор в ответ сигнализирует о том, что нужно попробовать адрес keith@eurecom.fr.

4. Прокси-сервер umass шлет пригласительное сообщение регистратору eurecom.
5. Регистратор eurecom знает IP-адрес пользователя keith@eurecom.fr, и поэтому переправляет пригласительное сообщение соответствующему клиенту (с адресом 197.87.54.21).

Шаги с 6 по 8 описывают процесс возвращения ответа через регистраторы и прокси-серверы обратно клиенту с IP-адресом 217.123.56.89. В конце пользователи могут обмениваться мультимедийными данными напрямую (на этой схеме не показано подтверждающее сообщение).

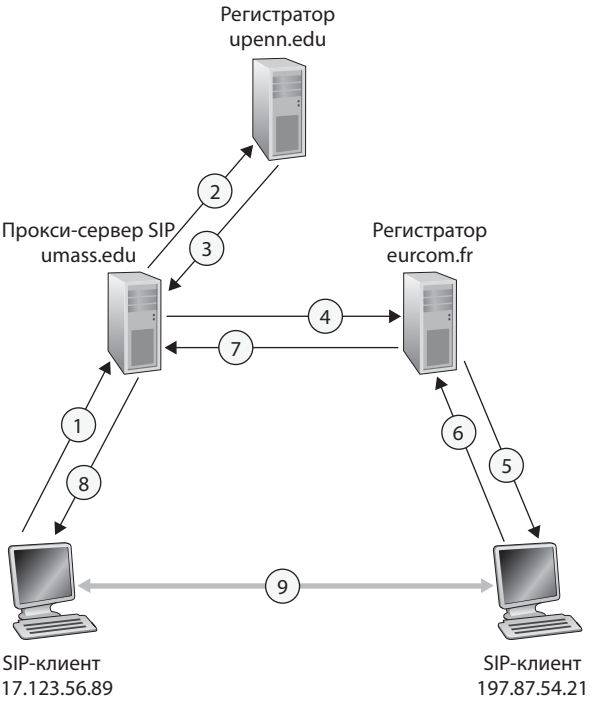


Рис. 7.13. Создание сеанса с участием прокси-серверов SIP и регистраторов

В этом подразделе мы, в основном, обращаем внимание на выполнение голосовых звонков. Однако SIP, как сигнальный протокол, может также использоваться для организации текстовых и видеоконференций. На самом деле он лежит в основе множества систем обмена мгновенными сообщениями. Чтобы получить больше информации об этом протоколе, посетите веб-сайт Хеннинга Шулзри[589]. К примеру, там вы сможете найти открытый программный код SIP-клиентов и серверов[601].

7.5. Поддержка мультимедийных сервисов на уровне сети

В разделах 7.2–7.4 мы узнали, как повысить производительность мультимедийных приложений с помощью таких программных механизмов, как клиентская буферизация, предварительная загрузка, регулирование качества в зависимости от доступной пропускной способности сети, адаптивное воспроизведение и восстановление потерянных пакетов. Мы также увидели, *каким образом* можно распространять мультимедийную информацию с помощью сетей CDN и P2P. Все эти методики и приемы были разработаны специально для использования в условиях, когда доставка данных попросту не гарантируется. Действительно, их популярность обусловлена именно тем, что подобные условия являются неизбежными в современном Интернете. Но как архитекторы компьютерных сетей мы не можем не спросить, можно ли обеспечить распространение контента за счет механизмов самой *сети* (не прибегая к использованию высокоуровневой программной инфраструктуры). Ответ на этот вопрос положительный, и в этом вы вскоре убедитесь! Но вместе с тем вы увидите, что многие из этих сетевых механизмов все еще не очень широко распространены. Это может быть вызвано их сложностью и тем фактом, что программные надстройки в сочетании с правильным подходом к оценке ресурсов сети могут быть вполне «приемлемыми» (хотя и не всегда идеальными) в условиях негарантированной доставки пакетов.

В табл. 7.4 приводится три популярных методики для поддержки мультимедийных приложений на сетевом уровне.

- *Оптимальное использование ресурсов в условиях негарантированной доставки*. Инфраструктурные и программные механизмы, которые мы изучили в разделах 7.2-7.4, могут успешно применяться в предсказуемых условиях, когда потеря и значительная сквозная задержка пакетов возникают довольно редко. Если повышение нагрузки поддается прогнозированию, провайдер может вовремя расширить пропускную способность сети, чтобы обеспечить приемлемые показатели[209]. **Измерение имеющихся сетевых ресурсов** будет рассмотрено в разделе 7.5.1.
- *Дифференцированное обслуживание*. С первых дней возникновения Интернета было ясно, что к предоставлению разных типов данных (которые, например, в IPv4-пакетах обозначаются заголовком Type-of-Service) лучше подходить индивидуально, а не использовать какой-то единый универсальный сервис.

Табл. 7.4. Три методики для поддержки мультимедийных приложений на сетевом уровне

Методика	Выборочность	Гарантии	Механизмы	Сложность	Распространенность
Оптимальное использование ресурсов в условиях негарантированной доставки	Все данные обрабатываются одинаково	Частичные или никаких	Поддержка на прикладном уровне, CDN, P2P, выделение ресурсов на сетевом уровне	Минимальная	Повсеместная
Дифференцированное обслуживание	Обработка данных зависит от их типа	Частичные или никаких	Маркировка пакетов, введение правил, планирование	Средняя	Частичная
Определение качества обслуживания для каждого соединения	Каждый поток данных обрабатывается поразному	Частичные или жесткие, для распознанных потоков	Маркировка пакетов, введение правил, планирование; прием и управление вызовом	Небольшая	Низкая

Дифференцированное обслуживание подразумевает наличие строгого приоритета одного вида информации перед другим на этапе буферизации в маршрутизаторах. Например, пакеты, принадлежащие приложению для общения в режиме реального времени, могут иметь более высокий приоритет по сравнению с другими пакетами ввиду жестких ограничений относительно сетевых задержек. Для реализации дифференцированного обслуживания на сетевом уровне необходимо воспользоваться новыми механизмами для маркировки пакетов (обозначения вида сервиса) и планирования их передачи. Этой теме отводится раздел 7.5.2.

- *Определение качества обслуживания (Quality-of-Service, QoS) для каждого соединения.* Эта методика позволяет выделять поток определенной ширины для каждого приложения отдельно, гарантируя стабильную производительность. Благодаря **жестким гарантиям** можно рассчитывать на соблюдение некоего уровня обслуживания. **Частичные гарантии** выдаются в том случае, когда этот уровень должен обеспечиваться с высокой долей вероятности. Например, если пользователь А захочет позвонить пользователю Б, VoIP-приложение зарезервирует поток определенной ширины во всех узлах на маршруте между двумя точками в сети. Но чтобы давать приложениям такие полномочия и обязывать сеть соблюдать выданные гарантии, необходимо произвести существенные изменения. Во-первых, нам нужен протокол, который по просьбе приложений будет резервиро-

вать определенные сетевые ресурсы на маршруте между отправляющей и принимающей стороной. Во-вторых, нам нужен новый подход к планированию очередей в маршрутизаторах, чтобы соблюдать гарантии для каждого соединения. И, наконец, чтобы зарезервировать эти самые ресурсы, приложение должно предоставить описание тех данных, которые оно намеревается передавать, а сети необходимо следить за тем, чтобы каждый пакет соответствовал этому описанию. Для реализации всех этих механизмов в узлах и маршрутизаторах требуется внедрить новое и сложное программное обеспечение. Эта методика не пользуется большой популярностью, поэтому в разделе 7.5.3 мы рассмотрим ее только в общих чертах.

7.5.1. Оценка сетевых ресурсов в условиях негарантированной доставки

В сущности, главная сложность в поддержке мультимедийных приложений связана с их жесткими требованиями к производительности (малая сквозная задержка, небольшие колебания времени доставки и незначительные потери пакетов), которые в условиях загруженной сети очень сложно соблюдать. Первое, что можно сделать для улучшения качества мультимедийного приложения (на самом деле, так можно решить практически любую задачу, связанную с ограниченными ресурсами) — это просто потратить большое количество денег, не вникая в суть проблемы. То есть необходимо обеспечить достаточную пропускную способность сети, чтобы в ней больше не возникали (или были крайне редкими) существенные сквозные задержки, колебания и потери пакетов. Это вполне реальное решение в условиях современного Интернета, при условии наличия достаточного объема ресурсов. Во многих отношениях это оптимальный вариант — мультимедийные приложения будут работать идеально, пользователи останутся довольны, и для этого даже не нужно менять архитектуру сети.

Вопрос только в том, сколько именно понадобится ресурсов, чтобы достичь такой благодати, и будет ли выгодно Интернет-провайдеру оплачивать их стоимость. Возможность предоставления сетевых узлов в условиях определенной структуры сети для достижения требуемой производительности часто называют **обеспечением достаточной полосы пропускания** (bandwidth provisioning). Но более важный вопрос заключается в том, каким образом спроектировать сеть (где разместить маршрутизаторы, как подключить их к отдельным узлам, какую нагруз-

ку им отвести), чтобы получить желаемый уровень производительности; этот процесс еще называют **определением основных параметров сети** (network dimensioning). Это довольно сложные темы, которые явно выходят за рамки данной книги. Однако ниже мы рассмотрим моменты, на которые следует обратить внимание при прогнозировании производительности приложений, находящихся в разных точках сети; это позволит вам рассчитывать объем ресурсов, которые необходимо выделять.

- *Моделирование загруженности сети между двумя ее точками.* Необходимость в моделировании может возникнуть как на уровне установления вызова (например, когда пользователи «появляются» в сети и инициируют межпрограммное соединение), так и на уровне передачи пакетов, генерируемых приложениями. Нужно понимать, что рабочая нагрузка может со временем измениться.

- *Четко определенные требования к производительности.* Это касается, например, систем, чувствительных к задержкам, таких как мультимедийные сервисы для общения. Необходимо просчитать вероятность того, что сквозная задержка при передаче пакетов превысит какое-то максимальное значение[171].

- *Прогнозирование производительности с учетом заданной загруженности сети; методики поиска оптимальной пропускной способности с точки зрения денежных затрат.* Речь идет о создании моделей, которые могут дать представление о производительности сети в условиях определенной загруженности, а также о методиках, которые помогают максимально снизить стоимость канала передачи данных, сохраняя при этом необходимую производительность.

Учитывая тот факт, что современные технические возможности позволяют передавать мультимедийные данные с соблюдением необходимого уровня производительности (при условии выполнения надлежащих расчетов), возникает закономерный вопрос: почему эти методики до сих пор не используются в Интернете? Причины лежат, преимущественно, в денежной и организационной плоскостях. Захотят ли клиенты Интернет-провайдеров оплачивать дополнительное расширение каналов связи для поддержки мультимедийных приложений? Однако решающими, наверное, являются организационные вопросы. Нужно понимать, что маршрут между двумя точками в сети состоит из множества сегментов, принадлежащих разным компаниям. Готовы ли эти компании совместно (возможно, путем разделения прибыли) обеспечивать поддержку мультимедийных приложений и проводить измерения про-

изводительности сети? Чтобы узнать подробности об этих обстоятельствах, см. работу Дейвиса[125]. Если хотите узнать, как операторы первого эшелона (tier-1) обеспечивают достаточную полосу пропускания для передачи данных с минимальными задержками, см. источник[171].

7.5.2. Предоставление нескольких категорий обслуживания

Наверное, само простое решение проблем универсальных сервисов с негарантированной доставкой пакетов в современном Интернете — это разделение передаваемых данных на отдельные категории и предоставление для каждой из них отдельного уровня обслуживания. Например, более высокие стандарты обслуживания могут применяться к сервисам интернет-телефонии видеоконференций, которые не способны работать в условиях больших задержек (естественно, за соответствующую плату). В то же время, электронная почта и веб-страницы могут передаваться в обычном режиме. Как вариант, провайдеры могут предоставлять повышенную производительность только для тех пользователей, которые готовы за нее платить. Подобная модель пользуется популярностью у кабельных и сотовых операторов — клиенты, выбравшие самый дорогой тариф, получают лучшее качество услуг.

Мы сталкиваемся с дифференцированным обслуживанием каждый день — авиапассажиры с билетами первого и бизнес- и эконом-класса получают разное внимание со стороны персонала авиалиний; знаменитости всегда могут свободно пройти на концерт, тогда как остальные люди должны отстаивать длинные очереди; в некоторых странах пожилые люди пользуются уважением, их всегда усаживают во главе стола и подают им лучшие блюда. Важно отметить, что подобная дифференциация касается категорий передаваемых данных, а не отдельных сетевых соединений. Если вернуться к нашей аналогии, все пассажиры первого класса обслуживаются одинаково — точно так же все пакеты VoIP-сервисов будут иметь одинаковый приоритет в рамках сети, независимо от того, откуда и куда они передаются. Как вы вскоре увидите, это существенно упрощает механизмы, предназначенные для поддержки дифференциации.

Люди, которые проектировали Интернет, явно держали в голове разделение потоков данных на категории. Напомним, что IPv4-пакеты содержат поле, определяющее тип обслуживания (см. рис. 4.13).

Поле с типом обслуживания описывалось еще в стандарте IEN123[246], принятом до появления IPv4-дейтаграмм:

Тип обслуживания (Type of Service или ToS) предоставляет обозначение абстрактных параметров, касающихся желаемого качества предоставления услуг. Эти параметры используются для непосредственного выбора средств, которые используются при передаче дейтаграммы в рамках конкретной сети. Некоторые сети разграничивают типы данных по приоритету, обслуживая их соответствующим образом.

Идея предоставлять разные уровни обслуживания для разных видов трафика была четко сформулирована более 40 лет назад! Однако только сейчас мы начинаем понимать ее суть.

Сценарии применения

Давайте начнем обсуждение механизмов дифференцированного обслуживания с нескольких реальных примеров, где бы они могли пригодиться.

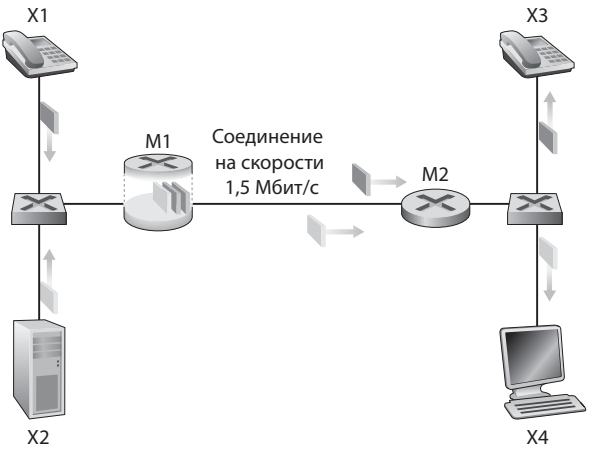


Рис. 7.14. Конкурирующая передача данных аудио- и HTTP-приложением

На рис. 7.14 проиллюстрирована простая ситуация, в которой два потока данных передаются из одной локальной сети в другую; в качестве источников выступают узлы X1 и X2, а адресатами являются узлы X3 и X4. Маршрутизаторы на обоих концах подключены друг к другу на скорости 1,5 Мбит/с. Давайте предположим, что скорость в локальных сетях значительно выше, чем во внешних. Сосредоточимся на исходящем буфере маршрутизатора M1; если общая скорость отправки пакетов узлами X1 и X2 превысит 1,5 Мбит/с, данные начнут задерживаться

и даже теряться. Теперь представьте, что канал между M1 и M2 занимают два приложения: одно передает звук с частотой 1 Мбит/с (к примеру, это может быть высококачественная телефонная связь), а другое загружает веб-страницы, размещенные на узле X4.

В обычных условиях аудио- и HTTP-пакеты будут смешаны внутри исходящего буфера M1 и переданы по принципу «первым пришел — первым ушел» (First-In-First-Out или FIFO). Это может привести к тому, что определенное количество HTTP-пакетов займет место в буфере, задерживая (или даже исключая) передачу части звуковых данных. Каким образом мы должны решать эту потенциальную проблему? Учитывая, что веб-браузеры не имеют жестких ограничений к задержкам, можно прийти к логичному выводу, что аудиопакеты должны иметь наивысший приоритет при прохождении через маршрутизатор M1. В случае жесткого соблюдения этого приоритета аудиоданные всегда будут покидать исходящий буфер первыми. Для аудиопакетов канал между M1 и M2 всегда будет иметь ширину 1,5 Мбит/с; в то же время веб-страницы будут передаваться только в те моменты, когда буфер свободен от приоритетных данных. Чтобы маршрутизатор M1 мог различать разные виды трафика, каждый пакет специально маркируется. Именно для этого в стандарте IPv4 было предусмотрено поле, определяющее тип обслуживания. Исходя из вышесказанного, первое утверждение касательно механизмов, необходимых для дифференцирования передаваемых данных, можно сформулировать так:

Утверждение №1: Маркировка пакетов позволяет маршрутизаторам распознавать разные типы данных.

Это относится не только к тем двум потокам данных, которые приводятся в нашем примере (мультимедийному и текстовому) — каждый пакет должен быть промаркирован в соответствии с принадлежностью к конкретному уровню обслуживания.

Теперь представим, что пакетам, принадлежащим аудио-приложению, выделен канал шириной 1 Мбит/с. Несмотря на то, что HTTP-пакеты получили более низкий приоритет, они все равно могут передаваться со средней скоростью 0,5 Мбит/с. Но что случится, если скорость передачи звуковых данных будет равна 1,5 Мбит/с или выше (неважно, по какой причине)? В этом случае веб-браузер получит отказ в обслуживании на отрезке сети M1–M2. Похожие проблемы возникнут и при работе нескольких приложений (например, нескольких видеовызовов) с одним и тем же уровнем обслуживания. В идеале разные типы переда-

ваемых данных должны быть изолированы, чтобы не мешать друг другу. Эта изоляция может быть реализована по-разному — например, в каждом маршрутизаторе, в узле, через который осуществляется доступ к сети, или за счет межсетевого барьера. Это приводит нас ко второму утверждению.

Утверждение №2: Желательно изолировать разные типы передаваемых данных, чтобы один тип, вышедший из-под контроля, не мешал всем остальным.

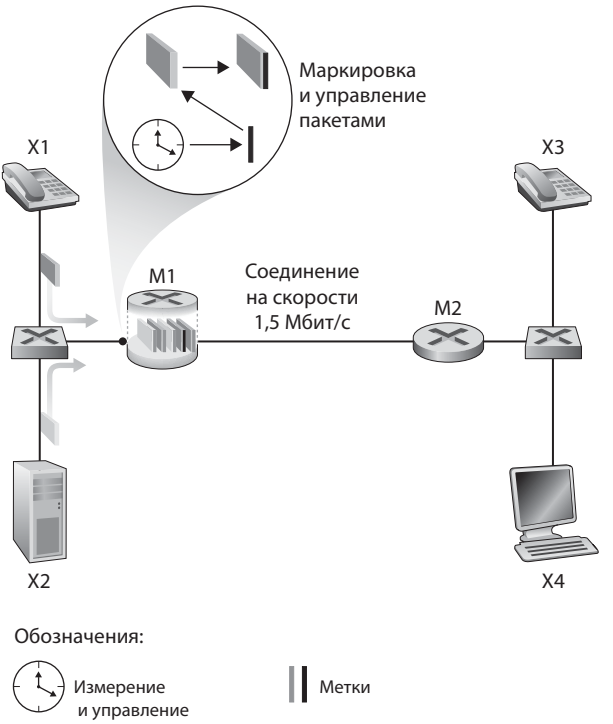


Рис. 7.15. Маркировка и управление аудио- и HTTP-пакетами

Мы рассмотрим несколько конкретных механизмов, которые обеспечивают подобного рода изоляцию. Все они относятся к одной из двух категорий. Первая категория подразумевает **управление потоками данных**, как показано на рис. 7.15. Если тип данных отвечает заданным критериям (например, если пиковая частота аудио-потока превышает 1 Мбит/с), этот механизм может гарантировать их отслеживание. Если приложение с управляемым потоком начинает вести себя некорректно, предпринимаются определенные меры (например, пакеты, которые на-

рушают критерии, могут задерживаться или сбрасываться), чтобы процесс передачи данных вернулся в нужное русло. Принцип «дырявого ведра», с которым мы скоро познакомимся, является, вероятно, наиболее распространенным способом управления потоками данных. На рис. 7.15 механизмы маркировки (утверждение №1) и управления (утверждение №2) пакетами реализованы на границе сети — либо на уровне конечной системы, либо в первом/последнем маршрутизаторе.

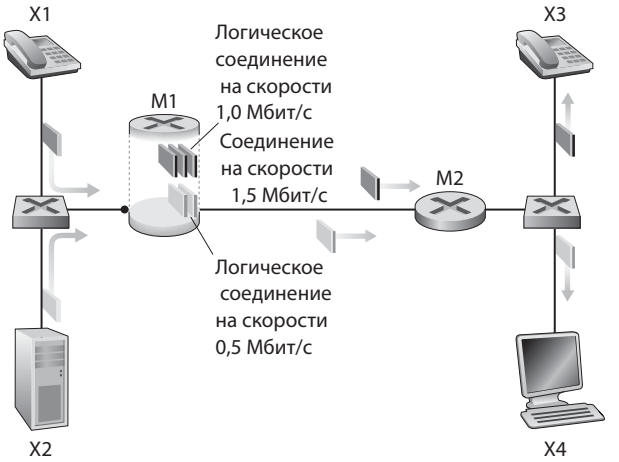


Рис. 7.16. Логическая изоляция аудио- и HTTP-пакетов

Альтернативный подход к изоляции сетевых потоков заключается в планировании передачи пакетов на канальном уровне и выделении каждому типу данных канала определенной ширины. Например, для аудиопакетов в маршрутизаторе M1 можно выделить канал шириной 1 Мбит/с, а для HTTP-пакетов — 0,5 Мбит/с. В этом случае музыка и веб-страницы будут загружаться через два разных логических соединения с соответствующей скоростью (см. рис. 7.16). Если строго контролировать эти ограничения, пакетам определенного типа будет доступен канал определенной ширины, но не более того — они не смогут передаваться на более высокой скорости, даже если для этого есть свободные ресурсы. Например, если в звуковом потоке наблюдается затишье (то есть, разговор прекратился, в результате чего пакеты больше не генерируются), соединение M1-M2 все равно будет передавать веб-страницы на скорости 0,5 Мбит/с. Сетевой канал — это «скоропортящийся» ресурс; если его не использовать, он просто зря продает. Поэтому нет никакого смысла ограничивать передачу HTTP-пакетов в то время, когда аудио-поток приостановлен. Нам бы хотелось использовать имеющиеся ресурсы как

можно более эффективно, не растрачивая их впустую. Из этого следует третье утверждение.

Утверждение №3: Изолируя разные типы и потоки данных, желательно как можно более эффективно использовать доступные ресурсы (например, пропускную способность канала или буфер маршрутизатора).

Механизмы планирования

Как уже говорилось в разделах 1.3 и 4.3, пакеты, принадлежащие разным сетевым потокам, смешиваются и выстраиваются в очередь в исходящем буфере соединения. Механизм, который определяет, какие именно из буферизированных пакетов будут передаваться дальше, называется **порядком планирования**. Ниже мы рассмотрим несколько разновидностей этого механизма.

Первым пришел — первым ушел (FIFO)

На рис. 7.17 показана схема работы механизма планирования по принципу FIFO. Пакеты поступают в исходящий буфер соединения и ждут, пока не будут переданы пакеты, полученные ранее.

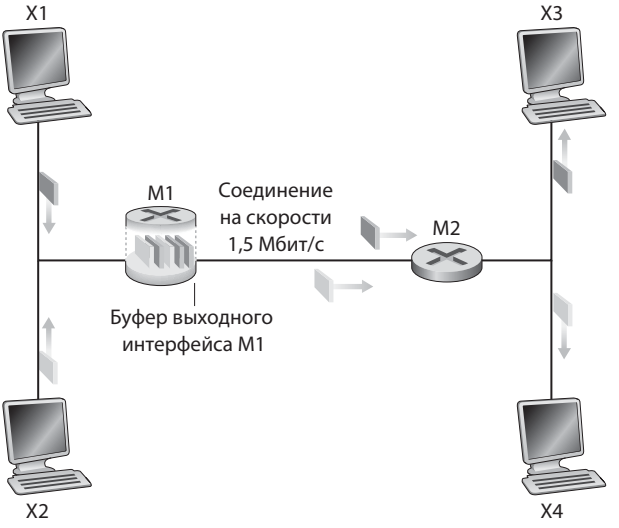


Рис. 7.17. Принцип планирования FIFO

Если в буфере закончится свободное место, специальный механизм решит, отказываться ли от приема новых данных или удалить из буфера

уже имеющиеся пакеты, чтобы освободить место. В наших примерах мы будем игнорировать возможность возникновения подобных ситуаций. Пакет, полностью прошедший через исходящее соединение (это означает, что он был обслужен), удаляется из буфера.

Порядок планирования FIFO (известный также под названием FCFS — «первым пришел — первым был обслужен») выбирает пакеты для дальнейшей передачи в том же порядке, в котором они поступили в буфер. С этим принципом знакомы все, кто когда-нибудь ждал автобуса на остановке (например, в Англии, где формирование очередей доведено до совершенства) или посещал сервисные центры, где посетители выстраиваются в одну длинную шеренгу, соблюдая порядок и ожидая своей очереди.

Принцип работы механизма планирования представлен на рис. 7.18. Появление пакетов обозначено пронумерованными стрелочками над верхней временной шкалой; по их номерам можно понять, в каком порядке они поступают. Отправка отдельных пакетов показана под нижней шкалой. Время, которое затрачивается на обслуживание пакета (при передаче) представлено в виде закрашенных прямоугольников между двумя осями. Поскольку мы имеем дело с порядком планирования FIFO, пакеты уходят в той же последовательности, в которой приходят. Стоит отметить, что после отправки четвертого пакета соединение переводится в холостой режим (поскольку больше нечего обслуживать), пока не появится пакет №5.

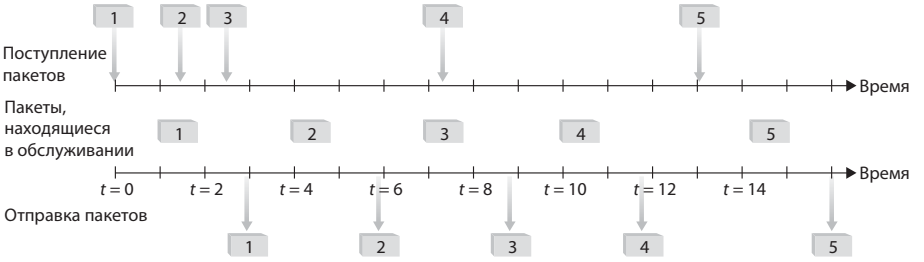


Рис. 7.18. Схема буфера, организованного по принципу FIFO

Приоритетное планирование

Приоритетное планирование подразумевает, что пакеты, попадающие в буфер исходящего соединения, классифицируются по приоритету (см. рис. 7.19). Как уже говорилось в предыдущем разделе, приоритет

пакета может зависеть от маркировки, которая содержится в его заголовке (например, в IPv4-пакетах хранится информация о типе обслуживания), от источника или места назначения, номера порта или других критериев. Для каждого приоритета обычно предусмотрен свой буфер. При выборе пакета, который будет передаваться дальше, первым делом рассматривается буфер с наивысшим приоритетом (при этом он не должен быть пустым). В рамках *одного* буфера, как правило, действует принцип FIFO.

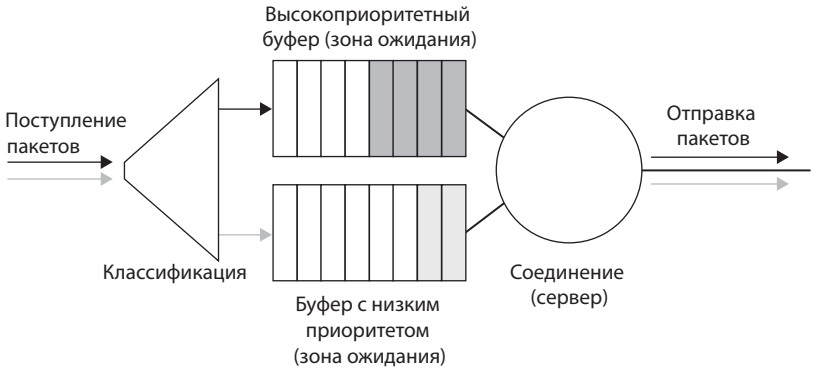


Рис. 7.19. Схема приоритетного планирования

На рис. 7.20 проиллюстрировано использование приоритетного планирования с двумя типами приоритетов. Пакеты 1, 3 и 4 имеют наивысший приоритет. Первый пакет поступает в пустой буфер и немедленно передается дальше.

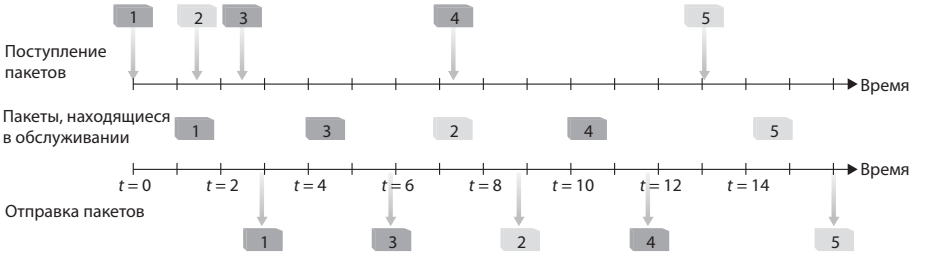


Рис. 7.20. Приоритетное планирование в действии

В это же время появляются пакеты 2 и 3, которые имеют, соответственно, низкий и высокий приоритет. Таким образом, вслед за первым пакетом будет передан пакет под номером 3, после чего начнется пере-

дача второго (он имеет низкий приоритет, хотя и появился раньше). В этот момент появляется высокоприоритетный четвертый пакет. При невытесняющем приоритетном планировании передача пакета, если она уже началась, не может быть прервана. В нашем случае четвертый пакет ждет своей очереди, пока не будет отправлен пакет под номером 2.

Циклический алгоритм и взвешенная справедливая очередь

Согласно **циклическому алгоритму**, пакеты распределяются по категориям, по аналогии с приоритетным планированием. Но вместо назначения строгих приоритетов, он чередует уровни обслуживания между разными категориями. Простейший пример: сначала идет пакет категории 1, потом пакет категории 2, потом пакет категории 1, опять пакет категории 2 и т. д. Если в сетевое соединение поступают пакеты (любой категории), ресурсосберегающее планирование не даст ему простаивать. **Ресурсосберегающий циклический алгоритм**, не найдя пакет подходящей категории, немедленно переключится на поиск пакетов следующего типа.

На рис. 7.21 проиллюстрирована работа циклического алгоритма с двумя категориями. Пакеты 1, 2 и 4 принадлежат к категории 1, а пакеты 3 и 5 — к категории 2. Первый пакет передается сразу, как только поступает в соединение. После этого планировщик ищет и передает пакет второй категории (пакет 3). Дальше алгоритм опять возвращается к первой категории и передает пакет 2. В очереди остается только пакет под номером 4, который сразу же отправляется дальше.

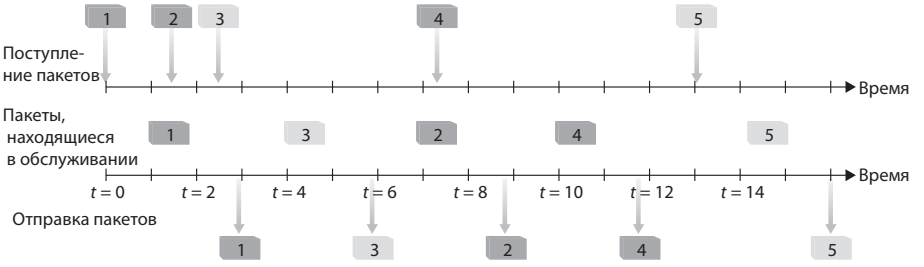


Рис. 7.21. Циклический алгоритм с двумя категориями трафика

Обобщенный вид циклического алгоритма, нашедший применение в архитектурах с поддержкой QoS, называется **взвешенной справедливой очередью** (Weighted Fair Queuing, **WFQ**)[128, 381]. Схема работы WFQ продемонстрирована на рис. 7.22. Приходящие пакеты класси-

фицируются и переносятся в зону ожидания подходящего типа. Как и в случае с циклическим алгоритмом, планировщик WFQ обслуживает пакеты по кругу — с первой по последнюю категорию, каждый раз возвращаясь обратно. Это ресурсосберегающий механизм, поэтому, если текущая категория оказывается пустой, он сразу же переходит к следующей.

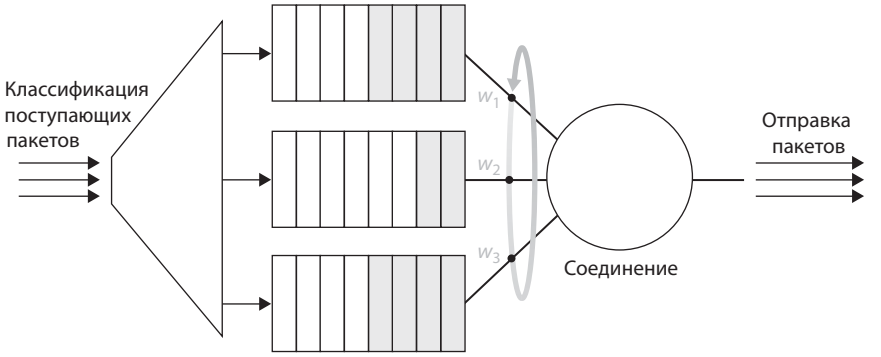


Рис. 7.22. Взвешенная справедливая очередь (WFQ)

Отличие от циклического алгоритма заключается в том, что уровень обслуживания пакетов любой категории может *поменяться* в любой момент. В частности, каждая категория, *i*, пакеты которой присутствуют в очереди, имеет свой «вес», w_i; на ее обслуживание гарантированно выделяется часть ресурсов, которая определяется формулой $w_i/(\sum w_j)$. В знаменателе находится сумма категорий, чьи пакеты ожидают отправки. В худшем случае, если все категории представлены в очереди, категория *i* все равно получит часть пропускной способности сети в размере $w_i/(\sum w_j)$. Таким образом, если сетевое соединение имеет скорость *R*, для пакетов категории *i* всегда будет доступен канал шириной как минимум $R\times(w_i/(\sum w_j))$. Нужно понимать, здесь речь идет об идеальном случае; мы не учитываем того факта, что пакеты — это отдельные фрагменты данных, и что передача одного пакета не будет прерываться из-за начала отправки другого[128, 381]. В следующих разделах вы увидите, что алгоритм WFQ лежит в основе архитектур, поддерживающих QoS. Он также доступен в современных маршрутизаторах[104].

Принцип дырявого ведра

Одно из утверждений, сделанных нами ранее, заключается в том, что регулирование скорости, с которой могут передаваться пакеты в рамках

определенной категории или потока (далее мы будем исходить из того, что речь идет именно о потоке), является важным механизмом для обеспечения качества обслуживания. Но каким именно аспектом потока необходимо управлять? Можно выделить три важных критерия, отличающиеся друг от друга временем, на протяжении которого регулируется скорость прохождения каждого пакета:

- *Средняя скорость*. Сеть способна ограничить долгосрочную среднюю скорость (количество пакетов в определенный отрезок времени), с которой данные могут передаваться в рамках потока. Важнейший момент, требующий внимания — это отрезок времени, на котором будет действовать вышеозначенное ограничение. Например, 100 пакетов в секунду — это более жесткое ограничение, чем 6000 пакетов в минуту, хотя в долгосрочной перспективе никакой разницы быть не должно. Дело в том, что во втором случае, в отличие от первого, за одну секунду может быть передано сразу 1000 пакетов.

- *Пиковая скорость*. Если средняя скорость описывает то, сколько пакетов можно отправить в сеть за какой-то относительно длинный промежуток времени, то пиковая скорость относится к коротким временным отрезкам. Возвращаясь к нашему предыдущему примеру, средняя скорость потока может быть ограничена 6000 пакетами в минуту, но при этом количество пакетов, которое позволяется передать за секунду, не должно превышать, скажем, 1500.

- *Взрыв объема трафика*. Сеть также позволяет ограничить количество пакетов, которое можно отправить за крайне короткий отрезок времени. Речь фактически идет о мгновенной отправке, поскольку временной интервал стремится к нулю. И хотя сеть физически не способна принять одновременно несколько пакетов (в конце концов, скорость любого сетевого соединения имеет свой предел, превысить который невозможно), сама концепция ограничения взрывного объема трафика имеет практическую ценность.

Принцип дырявого ведра — это абстракция, с помощью которой можно охарактеризовать вышеперечисленные ограничения. Как видно на рис. 7.23, ведро может хранить b маркеров, которые постоянно прибывают. Новые маркеры всегда генерируются с постоянной частотой, r маркеров в секунду (секунды выбраны исключительно для простоты восприятия). Если ведро не заполнено, новые маркеры могут поступать дальше; в противном случае они игнорируются, а количество маркеров в ведре остается неизменным — b.

Рис. 7.23. Принцип дырявого ведра

Теперь давайте посмотрим, как с помощью этого принципа можно управлять потоком пакетов. Представьте, что прежде чем попасть в сеть, пакет должен удалить из ведра один из маркеров. Если таковых в ведре нет, ему придется ждать их появления (как вариант, пакет может быть просто отклонен, но этот случай мы рассматривать не будем). Теперь давайте посмотрим, как это влияет на поток данных. Поскольку ведро вмещает не более b маркеров, число связанных с ним пакетов так же не может превышать b. Кроме того, создание маркеров происходит со скоростью r, поэтому максимальное количество пакетов, которое может быть передано в сеть за отрезок времени t, равно $rt+b$. Таким образом, в долгосрочной перспективе величина r ограничивает среднюю скорость вхождения пакетов. Этот принцип можно также использовать для управления пиковой скоростью (особенно если разместить два ведра подряд); подробней об этом в домашнем задании в конце этой главы.

Принцип дырявого ведра + взвешенная справедливая очередь = предсказуемость максимальной задержки в очереди

Давайте завершим наш разговор о планировании и управлении потоками данных примером того, как с помощью двух вышеприведенных алгоритмов можно ограничить задержку внутри буфера маршрутизатора. Давайтесь рассмотрим исходящий узел с n потоками, каждый из которых управляется по принципу дырявого ведра с параметрами b_i и r_i, $i = 1, \ldots, n$, используя планирование типа WFQ. Под термином «поток» подразумевается набор пакетов, которые не различаются планировщи-

ком. В реальных условиях поток может состоять из одного или нескольких сетевых соединений (см. рис. 7.24).

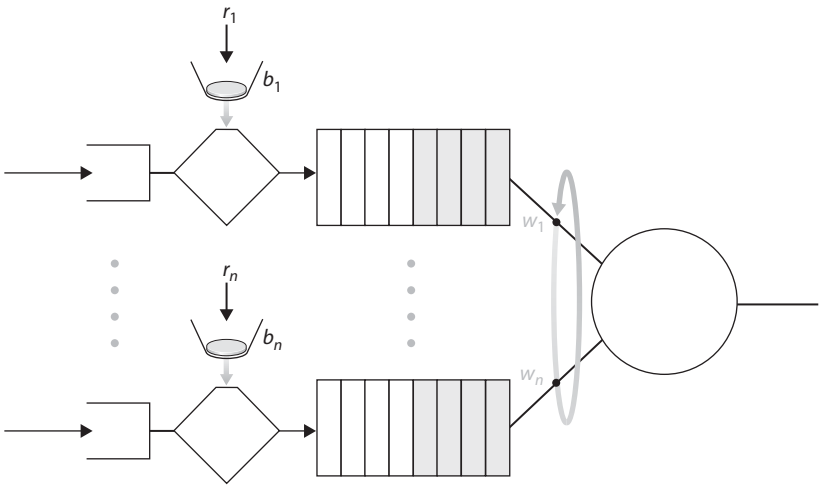


Рис. 7.24. *n* уплотненных потоков, организованных по принципу дырявого ведра с использованием планирования типа WFQ

При обсуждении взвешенной справедливой очереди мы уже упоминали, что каждый поток, i, гарантированно получает часть пропускного канала объемом $R \times w_i/(\sum w_j)$, где R — это скорость передачи данных целого соединения (в пакетах за секунду). Какой же будет максимальная задержка, вызванная ожиданием обслуживания со стороны планировщика (то есть, после прохождения дырявого ведра)? Давайте сосредоточимся на первом потоке. Допустим, его ведро изначально полностью заполнено, и тут в один момент появляется большое количество пакетов b_1. Все маркеры, находящиеся в ведре, мгновенно удаляются, а пакеты, связанные с ними, попадают в зону ожидания, где их должен обработать планировщик. Поскольку пакетам b_1 гарантирована минимальная скорость обслуживания $R \times w_i/(\sum w_j)$, максимальную задержку, d_{max}, будет иметь последний из них, где

$$d_{\max} = \frac{b_1}{R \times w_i/\sum w_j}$$

Эта формула имеет следующее логическое обоснование: если в очереди находится b_1 пакетов, а пакеты, которые были удалены из этой очереди, обслуживаются на скорости $R \times w_1/(\sum w_j)$, тогда количество времени, за которое должен быть передан последний бит последнего пакета, не может превышать $b_1/(R \times w_1/(\sum w_j))$. В упражнении к главе вам будет

предложено доказать, что если $r_1 < R \times w_1/(\sum w_j)$, то d_{max} точно является максимально возможной задержкой для любого пакета в первом потоке в рамках очереди WFQ.

7.5.3. Архитектура DiffServ

Рассмотрев назначение и конкретные механизмы дифференцированного обслуживания данных, можно подытожить наше обсуждение с помощью реального примера — архитектуры Diffserv[478, 283]. Diffserv поддерживает масштабирование и обеспечивает разделение передаваемой информации на разные категории. Необходимость масштабирования объясняется тем фактом, что базовый маршрутизатор может пропускать через себя миллионы параллельных потоков. Именно поэтому внутри сети реализуются только основные функции, а все сложные управляющие операции выносятся в граничные узлы.

Давайте начнем с простой сети, изображенной на рис. 7.25. Здесь мы остановимся только на одном варианте применения архитектуры Diffserv; но на самом деле таких вариантов может быть несколько[478]. Итак, архитектура Diffserv состоит из двух наборов функциональных элементов:

- *Граничные функции: классификация пакетов и нормирование трафика.* Входящие пакеты маркируются на границе сети (это может быть сетевой узел или маршрутизатор, который генерирует или пропускает через себя данные; главное — наличие поддержки Diffserv). Для этого полю в заголовке IPv4- и IPv6-пакетов (указывающему на определенный тип обслуживания) присваивается определенное значение[501]. Это поле должно заменить поля ToS (IPv4) и Traffic Class (IPv6), которые мы рассматривали в главе 4. Например, на рис. 7.25 видно, что пакеты, передаваемые между узлами X1 и X3, маркируются маршрутизатором M1; маркировка данных, отправленных с X2 в X4, происходит в точке M2. Маркеры определяют, к какому типу данных принадлежит тот или иной пакет. Разные типы данных будут получать разный уровень обслуживания внутри сети.
- *Основная функция: перенаправление.* Дойдя до маршрутизатора с поддержкой Diffserv, промаркированный пакет перенаправляется к следующему узлу, исходя из так называемого пошагового поведения, определенного с типом данных пакета. Пошаговое поведение влияет на то, каким образом маршрутизатор распределяет ресурсы буфера и сети между разными категориями трафика. Главный прин-

цип, на котором базируется архитектура Diffserv, заключается в том, что распределение пакетов основывается исключительно на их маркировке — то есть на их принадлежности к той или иной категории. Если пакетам, проходящим из точки X1 в точку X3 и из точки X2 в точку X4, назначать одинаковые маркеры, все маршрутизаторы внутри сети будут обслуживать их совершенно одинаково, независимо от того, откуда они пришли. Например, пакеты, идущие из узлов X1 и X2 в точку M4, будут считаться равноправными в маршрутизаторе R3. Таким образом, архитектура Diffserv полностью устраняет необходимость различать отдельные пары источник-адресат, что крайне положительно влияет на ее масштабируемость.

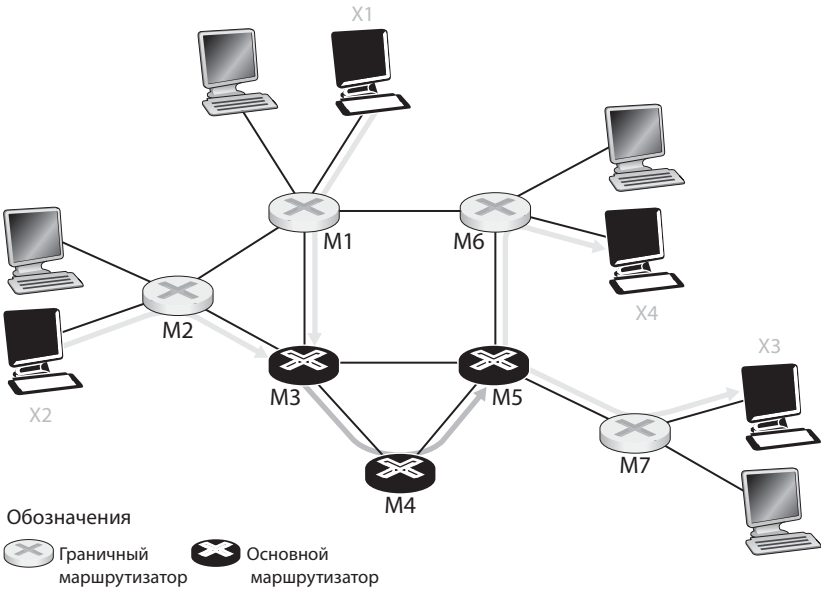


Рис. 7.25. Пример простой сети на основе архитектуры Diffserv

Здесь лучше обратиться к реальному примеру. Во время масштабных публичных мероприятий (например, в общественной приемной, ночном клубе, на концерте или футбольном матче) люди получают пропуска разных типов: для важных персон, для совершеннолетних (в случае, если внутри раздаются алкогольные напитки), для тех, кто хочет попасть за кулисы, для прессы и для самых обычных людей. Такие пропуска обычно распределяются на входе в заведение — то есть на его границе. Именно здесь выполняются интенсивные вычислительные операции, такие как оплата при входе, проверка соответствующего вида приглашения

и идентификация личности приглашенного. Кроме того, количество пропусков каждого типа может быть ограничено; в этом случае, прежде чем пройти на само мероприятие, людям придется подождать. Пройдя вовнутрь, человек, обладающий пропуском определенного вида, может рассчитывать на соответствующий уровень обслуживания в разных местах заведения — важные персоны будут иметь право употреблять бесплатные напитки, сидеть за лучшим столиком, заказывать лучшие блюда, заходить в закрытые помещения и пользоваться радушием обслуживающего персонала. Обычный же посетитель не сможет гулять везде, где ему вздумается; ему придется платить за выпивку, и обслуживать его будут не так внимательно. В обоих случаях отношение к людям в рамках одного мероприятия полностью определяется типом пропуска, которым они владеют. Один и тот же тип пропуска гарантирует одинаковый уровень обслуживания.

На рис. 7.26 схематически изображен процесс классификации и маркировки на границе маршрутизатора. Сначала входящие пакеты классифицируются на основе значений одного или нескольких полей, хранящихся в их заголовках (например, адрес отправителя, адрес получателя, порт отправителя, порт получателя и идентификатор протокола), затем они передаются в соответствующую функцию, которая проводит маркировку. Как уже упоминалось выше, маркер пакета хранится в специальном поле заголовка и указывает на определенный тип обслуживания.

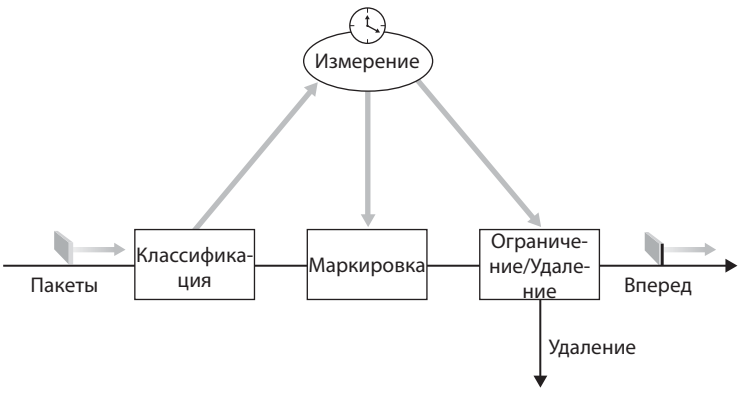


Рис. 7.26. Пример простой реализации архитектуры Diffserv

В некоторых случаях, чтобы соответствовать заявленному **профилю трафика**, конечный пользователь может согласиться на ограничение частоты отправки пакетов. Ограничение может касаться пиковой и взрыв-

ной скорости, как уже было продемонстрировано при рассмотрении принципа дырявого ведра. Если пользователь удовлетворяет всем требованиям оговоренного профиля, его пакеты маркируются в соответствии с заданным приоритетом и перенаправляются по своему маршруту прямо к адресату. В противном случае пакеты получат другую маркировку; это может привести, например, к ограничению максимальной скорости или даже потере данных на границе сети. Задача **функции измерения**, показанной на рис. 7.26, заключается в определении того, соответствует ли поток данных оговоренному профилю. Решение о повторной маркировке, передаче, задержке или сбросе пакета определяется политикой, установленной администратором вычислительной сети, и *не* имеет отношения к архитектуре Diffserv.

Мы рассмотрели процесс маркировки пакетов и управляющие функции. Но архитектура Diffserv подразумевает наличие еще одного компонента, который должен поддерживаться совместимыми маршрутизаторами. Речь идет о пошаговом поведении. Его формальное (и довольно запутанное) определение звучит так: «это описание механизма передачи данных, наблюдаемого извне, который действует в одном узле и применяется на всю совокупность узлов в рамках архитектуры Diffserv»[478]. Если внимательно к нему присмотреться, можно обнаружить несколько важных моментов:

- Пошаговое поведение может порождать разные категории трафика, которые передаются с разной скоростью (то есть «разные механизмы передачи данных, наблюдаемые извне»).
- Пошаговое поведение определяет разницу в производительности (поведении) между категориями, но оно не предусматривает никакого конкретного механизма для соблюдения этих различий. Фактически можно использовать любой механизм и любые правила выделения буфера (или ширины канала), если с точки зрения внешнего наблюдателя они удовлетворяют критериям. Например, пошаговое поведение не требует использования какой-то определенной методики формирования очереди пакетов (подойдет и WFQ, и FIFO). Выделение ресурсов и нюансы реализации — это всего лишь средство, а не цель.
- Разница в производительности должна быть доступна для наблюдения и, как следствие, для измерения.

Пошаговая пересылка пакетов бывает ускоренной[500] и гарантированной[482]. **Ускоренная пересылка** подразумевает, что скорость от-

правки маршрутизатором определенной категории пакетов не должна опускаться ниже некоторого определенного уровня. **Гарантированная пересылка** разделяет трафик на четыре категории, каждой из которых гарантируется какой-то минимальный объем пропускной способности сети и место в буфере.

Давайте завершим наше обсуждение архитектуры Diffserv несколькими фактами, касающимися ее модели обслуживания. Мы с самого начала исходили из того, что эта архитектура действует в рамках одного администраторского домена, но обычно между конечными точками передачи пакетов находится несколько интернет-провайдеров. Все эти провайдеры обязаны не только поддерживать соответствующие услуги, но также взаимодействовать друг с другом, чтобы предоставить своим клиентам полный цикл обслуживания. В противном случае конечный потребитель регулярно бы сталкивался с потерями или задержками данных из-за отсутствия договоренностей между разными провайдерами. Есть еще один немаловажный фактор: в большинстве случаев при средней загруженности сети разница между классическим подходом и архитектурой Diffserv почти не видна. В самом деле, сквозная задержка больше зависит от скорости доступа и загруженности отдельных участков сети, чем от буферизации в маршрутизаторах. Представьте себе недовольство клиента, который обнаруживает, что, несмотря на дополнительную плату за поддержку архитектуры Diffserv, он получает практически то же качество обслуживания, что и пользователи с обычными тарифами.

7.5.4. Гарантированное качество обслуживания для каждого соединения: резервирование ресурсов и допуск вызовов

В предыдущем разделе мы узнали, что с помощью маркировки пакетов, установления правил, изоляции трафика и планирования передачи данных на канальном уровне можно обеспечивать разные уровни обслуживания с разной производительностью. Например, приоритетное планирование позволяет практически «затмить» одну категорию трафика другой. При надлежащем измерении характеристик сети мы можем минимизировать потерю пакетов и задержки для данных с наивысшим приоритетом, обеспечивая тем самым производительность, как в случае с прямым соединением. Но может ли сеть *гарантировать*, что высокоприоритетный поток будет получать такое обслуживание на про-

тяжении всего своего существования, опираясь только на те механизмы, которые мы уже успели обсудить? Нет, не может. В этом разделе вы узнаете, почему для того чтобы строго соблюсти качество обслуживания отдельных соединений требуются дополнительные сетевые инструменты и протоколы.

Давайте вернемся к нашему примеру из раздела 7.5.2; рассмотрим работу двух аудиоприложений, которые работают на скорости 1 Мбит/с и вынуждены делить канал шириной 1,5 Мбит/с (см. рис. 7.27). Совокупная скорость двух потоков равняется 2 Мбит/с и превышает пропускную способность сети. Даже если разбить трафик на категории, воспользоваться маркировкой, изолировать потоки и разделить свободные ресурсы (которых в этом случае просто нет), это мало чем поможет. Пропускной способности сети физически не хватает для удовлетворения потребностей сразу двух приложений. Если разделить ресурсы поровну, то каждая сторона будет терять 25% всех пакетов. Качество обслуживания будет настолько низким, что приложения станут совершенно непригодными к использованию; передача аудиопакетов утратит всякий смысл.

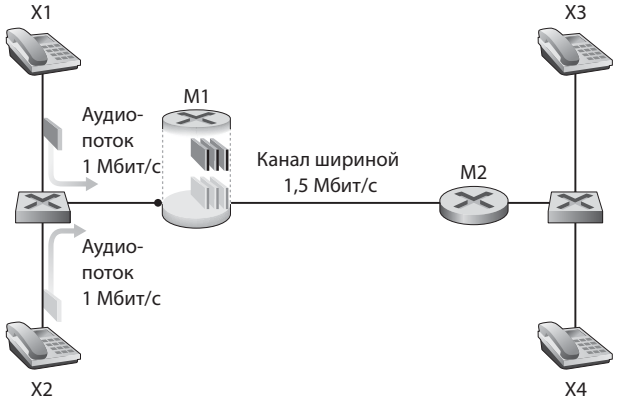


Рис. 7.27. Два аудиоприложения, работающих параллельно и перегружающих канал M1-M2

Итак, мы не можем обеспечить удовлетворительную работу сразу двух приложений, изображенных на рис. 7.27. Как в этой ситуации должна вести себя сеть? Если каждое из приложений будет обслужено на низком уровне, мы просто потеряем ресурсы, а конечный пользователь не получит никакой практической выгоды. Ответ достаточно прост: один поток должен быть заблокирован (то есть отрезан от до-

ступа к сети), а второй — работать дальше, пользуясь необходимой ему пропускной способностью (1 Мбит/с). В качестве примера такого подхода можно привести телефонную сеть — если при вызове не удается выделить достаточное количество ресурсов (чтобы соединить двух абонентов), вызов блокируется (закрывается доступ в сеть), а пользователь слышит короткие гудки. В нашем случае нет никакого смысла пропускать поток в сеть, если мы не сможем обслужить его на приемлемом уровне, который позволял бы извлечь из приложения какую-то пользу. Процесс определения того, располагает ли сеть достаточными ресурсами для работы приложения, связан с определенными издержками.

Пропуская или блокируя потоки, в зависимости от их требований к ресурсам и от того, сколько ресурсов занимают уже пропущенные потоки, сеть может гарантировать необходимое качество обслуживания. Для этого каждый поток должен как-то декларировать свои требования. Процесс объявления потоком требований к уровню обслуживания и определение того, можно ли пропускать этот поток в сеть (то есть, имеются ли для этого ресурсы), называется **допуском вызовов**. Из этого следует четвертое утверждение (в дополнение к предыдущим трем, которые приводятся в разделе 7.5.2) касательно механизмов, необходимых для обеспечения качества обслуживания.

> **Утверждение №4**: Если сеть не всегда предоставляет достаточный объем ресурсов, и если качество обслуживания должно быть *гарантировано*, необходим процесс допуска вызовов, при котором потоки декларируют свои требования и затем либо пропускаются в сеть (на требуемых условиях), либо блокируются (если сеть неспособна удовлетворить запрошенные требования).

Наш пример, изображенный на рис. 7.27, свидетельствует о том, что для обеспечения заданного качества обслуживания каждого вызова (сквозного потока) требуется несколько новых механизмов и протоколов.

- *Резервирование ресурсов.* Единственный способ *гарантировать* наличие ресурсов (таких как пропускная способность, место в буферах), необходимых для желаемого качества обслуживания — это ввести механизм, который на сетевом жаргоне называется **резервированием ресурсов**. Данный механизм обеспечивает непрерывное обслуживание вызова, независимо от требований остальных сторон. Если вызову была выделена определенная ширина канала и он ее не превышает, пользователь никогда не столкнется с потерей или задержкой пакетов.

- *Допуск вызовов*. Сеть должна предоставлять механизм, с помощью которого вызовы могут запрашивать зарезервированные ресурсы. При этом в некоторых случаях вызовы будут блокироваться, поскольку ресурсы не могут выделяться бесконечно. Такая система действует в телефонных сетях — запрос ресурсов происходит во время дозвона. Если абонент доступен, сеть устанавливает соединение, и дозвон проходит успешно. В противном случае вызов блокируется, и мы слышим короткие гудки. Заблокированный вызов может повторить попытку связаться с абонентом, но пока не будет успешно завершен процесс допуска, он не сможет отправлять свои данные в сеть. Конечно, маршрутизатор может выделить только ту ширину канала, которая ему доступна. Обычно резервируется только часть пропускной способности сети, поэтому одновременно могут обслуживаться сразу несколько вызовов. Но если к качеству обслуживания предъявляются жесткие требования, общая ширина канала, доступная для всех вызовов, не должна превышать физические возможности сети.
- *Протокол установления вызовов*. Вышеописанный процесс требует, чтобы вызов имел возможность зарезервировать необходимый ему объем ресурсов в рамках всех маршрутизаторов, которые находятся на пути к адресату — именно так обеспечивается качественное обслуживание соединения. Каждый маршрутизатор должен определить, достаточно ли у него локальных ресурсов, необходимых для работы сеанса, учитывая ту ширину канала, которая уже выделена на текущие нужды. Чтобы скоординировать действия всех узлов, размещенных на маршруте между двумя точками сети, нужен специальный протокол; он будет управлять выделением локальных ресурсов в маршрутизаторах и решать, способен ли каждый отдельный участок маршрута обслужить имеющийся вызов. Речь идет о **протоколе установления вызовов** (см. рис. 7.28). Одна из его реализаций, которая заложена в архитектуру Интернета[683, 470], называется **RSVP**. В ATM-сетях банкоматов действует протокол Q2931b[56], который занимается передачей данных между коммутаторами и терминалами.

Несмотря на огромное количество исследований, разработок и даже готовых продуктов, призванных обеспечить качество обслуживания отдельных соединений, внедрение подобных технологий находится в зачаточном состоянии. И этому есть множество объяснений. Первая, и главная причина заключается в том, что во многих случаях для «приемлемой» работы мультимедийных приложений достаточно использования простых программных механизмов (которые мы рассмотрели

в разделах 7.2-7.4) в сочетании с надлежащей оценкой характеристик сети (раздел 7.5.1). Кроме того, механизмы, гарантирующие качественное обслуживание каждого соединения, могут усложнить сетевую инфраструктуру и увеличить затраты на ее управление и обслуживание, в результате чего Интернет-провайдер может посчитать эти технологии слишком дорогими, учитывая ожидаемую прибыль от предлагаемых услуг.

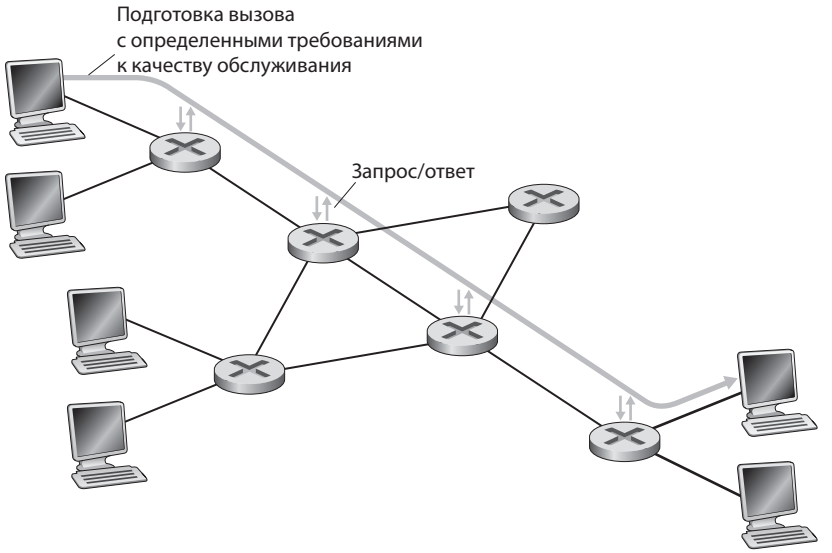


Рис. 7.28. Процесс допуска вызова

7.6. Заключение

Мультимедийные сетевые технологии — это одна из самых захватывающих областей современного Интернета. Люди все меньше пользуются традиционными радио и телевидением, предпочитая потреблять теле- и радиопередачи через сеть (речь идет как о живых трансляциях, так и о заранее записанных данных). По мере развития беспроводных сетей эта тенденция, несомненно, будет набирать обороты. Более того, благодаря таким веб-сайтам, как YouTube, пользователи могут не только потреблять, но и производить собственный мультимедийный материал. В наши дни Интернет используется не только для распространения видеоданных, но и как для телефонных звонков. Фактически в течении следующих десяти лет IP-телефония может вытеснить традиционные

коммутируемые каналы телефонной связи. Она обеспечивает не только низкие цены на разговоры, но и предоставляет богатый набор дополнительных услуг, таких как видеоконференции, службы каталогов, голосовая почта и интеграция с социальными сетями (например, Facebook и Google+).

В разделе 7.1 мы рассмотрели характерные свойства голосовых и видеоданных, разделив мультимедийные приложения на три категории: потоковое вещание хранимого аудио/видео, IP-телефония и видеосвязь, а также потоковое вещание аудио/видео в реальном времени.

В разделе 7.2 мы сосредоточились на изучении потокового вещания хранимого видео. Такие видеоданные хранятся на серверах и передаются по запросу пользователей. Мы узнали, что системы потокового вещания можно разделить на три вида: UDP-вещание, HTTP-вещание и адаптивное HTTP-вещание. Последние два используются в большинстве современных проектов, хотя протокол UDP тоже имеет свою сферу применения. Главным показателем производительности систем, занимающихся потоковой передачей видеоданных, является средняя пропускная способность. В этом разделе также было уделено внимание сетям CDN, которые помогают распространять большие объемы данных по всему миру. В конце мы обсудили технологии, которые лежат в основе трех крупных видеосервисов: Netflix, YouTube и Kankan.

В разделе 7.3 основное внимание уделялось голосовым системам, таким как VoIP-приложения, и их работе в условиях негарантированной доставки данных. Процесс передачи голоса крайне чувствителен к задержкам, поэтому фактор времени здесь имеет большое значение. При этом потеря отдельных пакетов не является существенной проблемой, вызывая лишь незначительные помехи при воспроизведении; к тому же, она может быть полностью или частично компенсирована. Мы увидели, что с помощью клиентских буферов, временных меток и нумерации пакетов можно минимизировать последствия сетевых колебаний. Мы также рассмотрели архитектуру сервиса Skype, который является лидером на рынке голосовых и видеоконференций. В разделе 7.4 мы познакомились с двумя наиболее важными стандартами IP-телефонии — протоколами RTP и SIP.

В разделе 7.5 было представлено несколько сетевых механизмов (таких как критерии планирования на канальном уровне или правила дифференцирования данных), которые позволяют обеспечивать разный уровень обслуживания для разных категорий трафика.

Глава 8
СЕТЕВАЯ БЕЗОПАСНОСТЬ

Еще в разделе 1.6 мы описали ряд наиболее распространенных и опасных классов Интернет-атак, в частности атаки с применением вредоносного ПО, отказ в обслуживании, анализ пакетов (сниффинг), имитация источника, а также удаление и изменение сообщений. В предыдущих главах мы изучили огромный объем информации о компьютерных сетях, но все еще не говорили о том, как защищаться от подобных атак. Вооружившись новоприобретенным опытом работы с компьютерными сетями и Интернет-протоколами, приступим к углубленному изучению безопасной коммуникации, в частности, обсудим, как защищать компьютерные сети от вторжения злоумышленников.

Позвольте вам представить Алису и Боба, двух молодых людей, жаждущих связи, но связи «безопасной». Поскольку мы говорим о сетях, Алиса и Боб могут быть двумя маршрутизаторами, которые хотят обменяться таблицами маршрутизации, клиентом и сервером, устанавливающими транспортное соединение, двумя приложениями электронной почты, пытающимися передать друг другу электронные письма. Все эти ситуации мы рассмотрим в этой главе. Вымышленные имена Алисы и Боба очень популярны в мире сетевой безопасности, возможно, потому что использовать их веселее, чем просто буквы «А» и «Б». Рискованная любовная интрижка, военные коммуникации, деловые транзакции — везде нужна безопасная связь. Правда, тема любви для нас гораздо интереснее, чем две другие, поэтому давайте предположим, что Алиса и Боб — наши отправитель и получатель, а затем рассмотрим их в контексте романтического сценария.

Итак, Алиса и Боб желают установить друг с другом безопасную связь, но что это означает? Как мы увидим, безопасность — сложное понятие, то есть у нее множество аспектов. Во-первых, Алиса и Боб хотели бы, чтобы содержание их разговоров не достигло любопытных ушей. Во-вторых, Алиса и Боб хотели бы удостовериться, что общаются они именно друг с другом, а если в их разговор вмешивается кто-то третий, то это вмешательство должно быть гарантированно обнаружено. В пер-

вой половине этой главы мы рассмотрим методы шифрования и дешифрирования, методы аутентификации собеседника, а также методы, гарантирующие целостность данных.

Во второй части главы речь пойдет о том, как фундаментальные криптографические принципы могут использоваться для создания безопасных сетевых протоколов. Мы по-прежнему будем придерживаться нисходящего подхода, поэтому исследуем протоколы безопасности, применяемые на каждом из четырех верхних уровней, начиная с прикладного. В частности, мы поговорим о том, как защитить передачу электронной почты, TCP-соединение, как обеспечить комплексную безопасность на сетевом уровне, а также как защитить беспроводную локальную сеть. В третьей части этой главы мы обсудим операционную безопасность — то есть поговорим о том, как защитить от атак сети организаций. Так, мы подробно остановимся на том, как можно повысить безопасность такой сети при помощи брандмауэров и систем обнаружения вторжений.

8.1. Понятие о сетевой безопасности

Начнем наше изучение вопроса сетевой безопасности с наших знакомых Алисы и Боба, желающих иметь безопасную связь. Что это означает? Разумеется, Алиса хочет, чтобы только Боб мог читать отправляемые ею сообщения, *несмотря на то*, что сигнал проходит по незащищенной линии связи, где злоумышленница (назовем ее Мэри) может перехватить сообщение, отправленное Алисой Бобу. Далее, Боб хочет быть уверен, что получаемые им сообщения действительно посланы Алисой, и Алиса также хочет быть уверенной, что она общается именно с Бобом. Кроме того, Алиса и Боб хотели бы быть уверенными, что содержание их сообщений не меняется при передаче. Наконец, они хотели бы общаться друг с другом без вмешательства посторонних (то есть чтобы никто не закрыл им доступ к ресурсам, необходимым для поддержания связи) Итак, с учетом всего вышесказанного, мы можем сформулировать следующие желательные свойства **безопасной связи**.

- *Конфиденциальность*. Только отправитель и предполагаемый получатель должны быть способны понимать содержимое передаваемых сообщений. Поскольку злоумышленники могут перехватить сообщение, оно должно быть каким-то образом **зашифровано** (его данные должны быть скрыты), так чтобы перехвативший сообщение злоумышленник не смог его расшифровать (понять). Вероятно,

именно этот аспект конфиденциальности имеется в виду, когда говорится о *безопасной связи*. Мы рассмотрим методы криптографии для шифрования и дешифрирования данных в разделе 8.2.

- *Целостность сообщения*. Алиса и Боб хотят быть уверены, что содержимое их переписки по пути не будет изменено (случайно или злонамеренно). Для обеспечения такой целостности зачастую применяются расширения методов проверки контрольных сумм вместе с протоколами надежной транспортировки и канального уровня. О целостности сообщений мы подробно поговорим в разделе 8.3.
- *Аутентификация конечной точки*. Как отправитель, так и получатель должны быть способны подтвердить личность собеседника — убедиться, что они общаются именно с тем человеком, за которого он себя выдает. При общении лицом к лицу эта проблема легко решается визуально. Когда же собеседники не могут «видеть» друг друга, аутентификация представляет собой значительно более сложную проблему. Например, если пользователь хочет проверить входящую почту на своем электронном ящике, как почтовый сервер определяет, что это действительно хозяин ящика? Аутентификация конечной точки будет подробно рассмотрена в разделе 8.4.
- *Операционная безопасность*. В настоящее время практически у всех организаций (компаний, университетов и т. д.) есть компьютерные сети с выходом в Интернет. Следовательно, эти сети потенциально подвержены вторжению извне. Злоумышленники могут пытаться внедрять червей на хосты корпоративной сети, выведывать секреты организации, трассировать (картографировать) конфигурации внутренних сетей и осуществлять DOS-атаки. В разделе 8.9 мы познакомимся со специальными устройствами — брандмауэрами и системами обнаружения вторжений — которые используются для предотвращения атак на сети организаций. Брандмауэр, располагаясь между сетью организации и общедоступной сетью, контролирует выход пакетов из внутренней сети и поступление пакетов в нее. Система обнаружения вторжений выполняет углубленную проверку пакетов (deep packet inspection), предупреждая администраторов вычислительной сети о любой подозрительной активности.

Итак, дав определение сетевой безопасности, рассмотрим, к какой информации может получить доступ злоумышленник и какие действия он может предпринять. Схема диалога Алисы и Боба с участием в нем злоумышленника изображена на рис. 8.1. Алиса хочет отправить данные Бобу. Чтобы обмениваться данными безопасным образом, а также

выполнить требования конфиденциальности, аутентификации конечной точки и целостности данных, Алиса и Боб обмениваются управляющими и информационными сообщениями (подобно тому, как TCP-отправители и TCP-получатели обмениваются управляющими и информационными сегментами). Как правило, все или некоторые из этих сегментов зашифровываются. Как было указано в разделе 1.6, злоумышленник потенциально может выполнять следующие действия:

- *Прослушивание* — анализ (сниффинг) и запись управляющих и информационных сообщений, проходящих по каналу.
- *Изменение, вставка или удаление* сообщений или их содержимого.

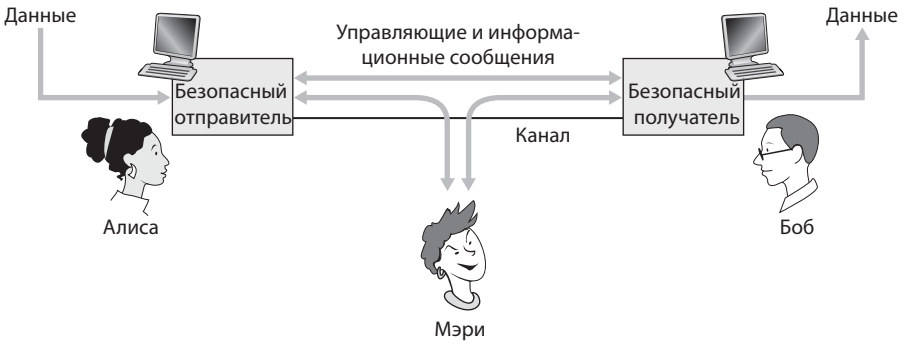


Рис. 8.1. Отправитель, получатель и злоумышленник (Алиса, Боб и Мэри)

Как мы увидим, если не предпринимать соответствующих контрмер, злоумышленник сможет прибегнуть к весьма изощренным атакам, начиная от прослушивания переговоров (при этом возможно похищение паролей и данных) до выдачи себя за одного из собеседников и нарушения работы системы путем ее перегрузки и т. д. Отчеты о замеченных атаках подготавливаются координационным центром CERT[83].

Итак, мы убедились, что в Интернете вполне можно столкнуться с серьезными угрозами. Разумеется, Боб и Алиса могут быть двумя обычными приятелями, которые хотят безопасно общаться в сети, но какие еще эквивалентные пары «отправитель-получатель» такого рода существуют в Интернете? Предположим, это реальные люди, которые переписываются по электронной почте. Это могут быть участники транзакции в интернет-магазине. Например, реальный Боб хочет безопасно передать свой номер кредитной карточки на веб-сервер, чтобы приобрести товар в виртуальном магазине. Аналогично, Алисе может потребоваться удаленно совершить какие-либо операции со своим банковским

счетом. Стороны, между которыми устанавливается безопасное соединение, сами могут быть элементами сетевой инфраструктуры. Как вы помните, система доменных имен (DNS, раздел 2.5) или служебные процессы (демоны) маршрутизации, обменивающиеся маршрутной информацией (раздел 4.6), требуют такой защищенной коммуникации между отправителем и получателем. Аналогичные требования возникают при работе с приложениями, осуществляющими управление сетью — о таких приложениях мы подробно поговорим в главе 9. Злоумышленник, способный активно вмешиваться в поиск DNS (о таком поиске подробно рассказано в разделе 2.5), вычисление таблиц маршрутизации[530] или в функции управления сетью[510] может посеять хаос во всем Интернете.

Итак, обрисовав предметную область данной главы, дав определения некоторых наиболее важных терминов и обосновав необходимость сетевой безопасности, давайте подробно поговорим о криптографии. Польза криптографии для обеспечения конфиденциальности очевидна, но мы вскоре также убедимся, что она совершенно необходима и для аутентификации конечной точки, и для обеспечения целостности сообщений. Таким образом, криптография является настоящим краеугольным камнем сетевой безопасности.

8.2. Основы криптографии

Хотя криптография имеет долгую историю (например, известно, что простейшие шифры применял Юлий Цезарь), современные криптографические методы, включая многие из тех, что используются в сегодняшнем Интернете, основаны на достижениях последних 30 лет. Захватывающая история криптографии отражена в книгах Кана[271] и Сингха[600]. Для полного обсуждения этой темы требуется целая книга (см., например, публикации Кауфмана[280] и Шнейера[587]), поэтому мы только коснемся существенных аспектов криптографии в том виде, в котором они практикуются в сегодняшнем Интернете. Отметим, что хотя основным в этом разделе является вопрос использования криптографии для обеспечения конфиденциальности, сами криптографические методы тесно связаны с вопросами аутентификации, проверки целостности сообщений, невозможности отказа от ранее совершенных действий и т. п.

Криптографические методы позволяют отправителю скрыть содержимое своих посланий, поэтому злоумышленник не может извлечь полезную информацию из перехваченных сообщений. Само собой, по-

лучатель должен быть способен восстановить исходные данные. Некоторые из наиболее важных понятий иллюстрирует рис. 8.2.

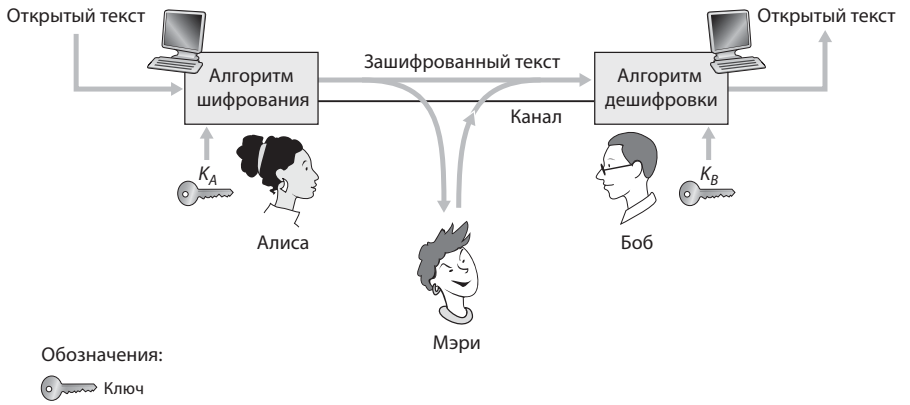


Рис. 8.2. Криптографические компоненты

Предположим, что Алиса хочет переслать Бобу сообщение. Сообщение Алисы в его исходном виде (например, «`Боб, я люблю тебя. Алиса`») называется **открытым текстом**. Алиса зашифровывает это сообщение при помощи **алгоритма шифрования**, в результате **зашифрованное сообщение**(или **шифротекст**) становится непонятным для злоумышленника. Интересно отметить, что во многих современных криптографических системах, включая те, что используются в Интернете, сам алгоритм шифрования *известен*, то есть опубликован[444, 514, 475, 369], стандартизован и доступен всем и каждому, даже потенциальному злоумышленнику! Очевидно, если метод шифрования данных известен всем, должна быть какая-то секретная информация, не позволяющая злоумышленнику расшифровать пересылаемые данные. Такой информацией является ключ.

На рис. 8.2 показано, что в качестве входных данных для алгоритма шифрования Алиса использует **ключ** K_A, представляющий собой текстовую строку. Алгоритм шифрования принимает ключ и открытый текст m на входе и выдает зашифрованное сообщение на выходе. Для обозначения сообщения, зашифрованного с помощью ключа K_A, используется запись $K_A(m)$. Аналогично, Боб предоставляет ключ K_B и зашифрованный текст **алгоритму дешифрирования**, выдающему на выходе исходный вариант открытого текста. Таким образом, если Боб получает зашифрованное сообщение $K_A(m)$, он расшифровывает его, вычисляя $K_B(K_A(m)) = m$. В **системах с симметричными ключами** Алиса и Боб

используют идентичные и секретные ключи. В **системах с открытым ключом** применяется пара ключей. Один из ключей известен как Бобу, так и Алисе (а также всем и каждому). Второй ключ известен только Бобу или Алисе (но не обоим сразу). В следующих двух подразделах мы рассмотрим системы с симметричными ключами и системы с открытым ключом более подробно.

8.2.1. Шифрование с симметричными ключами

Все криптографические алгоритмы заменяют один вариант данных другим: открытый текст — зашифрованным. Прежде чем обсудить современные криптографические системы, рассмотрим очень старый простой алгоритм с симметричными ключами, приписываемый Юлию Цезарю и известный как **шифр Цезаря** (шифр — это способ шифрования данных).

Чтобы зашифровать шифром Цезаря английский текст, нужно каждую букву открытого текста заменить буквой, располагающейся в алфавите на k символов дальше (при этом весь алфавит рассматривается как цикл, то есть за буквой z следует буква a). Например, если $k = 3$, тогда буква a в открытом тексте будет заменена в зашифрованном тексте буквой d; буква b станет буквой e и т. д. В данном шифре ключом служит значение k. Например, открытое сообщение `bob, i love you. alice` превратится в `ere, i oryx brx. dolfh`. Хотя зашифрованный текст выглядит как полная тарабарщина, для взлома такого шифра не понадобится много времени, если известно, что использовался шифр Цезаря, так как у этого алгоритма шифрования может быть только 25 значений ключа.

Усовершенствованием шифра Цезаря является **моноалфавитный шифр**, также заменяющий одну букву алфавита другой. Однако вместо того, чтобы заменять символы регулярным образом (например, смещением в алфавите на постоянную величину), любая буква может заменяться любой другой буквой, при условии, что соответствия букв при замене уникальны и взаимно однозначны. На рис. 8.3 показан пример такого шифра.

Незашифрованная буква	a b c d e f g h i j k l m n o p q r s t u v w x y z
Зашифрованная буква	m n b v c x z a s d f g h j k l p o i u y t r e w q

Рис. 8.3. Моноалфавитный шифр

В этом случае открытое сообщение `bob, i love you. alice` превратится в `nkn, s gktc wky. mgsbc`. Таким образом, как и в случае шифра Цезаря, зашифрованное сообщение выглядит непонятно. Однако моноалфавитный шифр значительно более надежен, нежели шифр Цезаря, так как существует 26 (величина порядка 10^{26}) возможных вариантов соответствий между алфавитами открытого и зашифрованного текста. Попытка найти ключ полным перебором всех 10^{26} возможных вариантов потребует слишком больших усилий. Однако статистический анализ позволяет значительно упростить задачу взлома подобного кода. Так, например, известно, что в английском тексте чаще всего встречаются буквы *e* и *t* (13 и 9% соответственно). Также известна частота появления двухбуквенных и трехбуквенных сочетаний (например, *in*, *it*, *the*, *ion*, *ing* и т. д.). В результате взломать такой шифр оказывается не так уж и сложно. Если же злоумышленник обладает некоторой информацией о сообщении, задача взлома шифра становится еще проще. Например, если злоумышленником является жена Боба, подозревающая Боба в том, что у него интрижка с Алисой, то она может предположить, что в тексте должны встречаться имена Алисы и Боба. В этом случае, перехватив приведенное выше зашифрованное сообщение, она может отгадать семь из 26 символов ключа, в результате для продолжения взлома путем полного перебора всех вариантов понадобится в 10^9 раз меньше времени. Само собой, если Мэри подозревает, что Боб изменяет ей, она вполне может отгадать и другие слова, которые наверняка будут содержаться в любовном сообщении.

Для каждого нового шифра необходимо определить, насколько легко злоумышленник может его взломать. При этом рассматриваются три сценария, зависящих от того, какой информацией обладает злоумышленник.

- *Атака с известным шифротекстом*. В некоторых случаях злоумышленник может получить доступ только к перехваченному им зашифрованному тексту без какой-либо информации о содержимом перехваченного сообщения. Как уже отмечалось, для **взлома такого шифра** может использоваться статистический анализ.
- *Атака с известным открытым текстом*. Ранее было показано, что если злоумышленник каким-либо образом сумел догадаться о том, что в зашифрованном сообщении встречаются слова *bob* и *alice*, то он может определить пары (открытый текст, зашифрованный текст) для букв *a*, *1*, *i*, *c*, *e*, *b* и *o*. Кроме того, злоумышленнику может повезти еще больше, если он случайно обнаружит расшифрованную версию

одного из сообщений, полученных Бобом. Если злоумышленнику известны хотя бы некоторые пары-соответствия букв из открытого и зашифрованного текста, то применяется атака **с известным открытым текстом**.

- *Атака с подобранным открытым текстом.* В этом случае злоумышленник способен выбирать открытый текст и получать соответствующий ему зашифрованный текст. При таком простом алгоритме шифрования (моноалфавитный шифр) злоумышленник может перехватить сообщение, содержащее все буквы алфавита, например `the quick brown fox jumps over the lazy dog`, и полностью взломать схему шифрования. Далее будет показано, что в более сложных схемах шифрования подобная атака **с подобранным открытым текстом** не обязательно приводит к взлому системы.

Пятьсот лет назад метод моноалфавитного шифрования был усовершенствован, в результате появился так называемый полиалфавитный шифр. Идея этого метода заключается в использовании нескольких моноалфавитных шифров, причем выбор шифра определяется позицией кодируемого символа в открытом сообщении. Таким образом, один и тот же символ открытого текста может кодироваться по-разному. Пример полиалфавитного шифра показан на рис. 8.4. В нем применяются два шифра Цезаря (со значениями $k = 5$ и $k = 19$). Эти два шифра Цезаря, C_1 и C_2, могут использоваться в следующем повторяющемся порядке: C_1, C_2, C_2, C_1, C_2. То есть первый символ открытого текста кодируется при помощи шифра C_1, второй и третий символы — с помощью шифра C_2, для шифрования четвертого символа опять используется шифр C_1 пятый символ шифруется с помощью шифра C_2. Затем вся последовательность повторяется, то есть шестой символ шифруется при помощи шифра C_1, седьмой — при помощи шифра C_2 и т. д. В результате открытое сообщение `bob, i love you` превращается в `ghu, n etox dhz`. Обратите внимание, что первый символ `b` открытого сообщения кодируется шифром C_1, тогда как второй символ `b` — шифром C_2. В этом примере «ключ» шифрования и дешифровки предполагает знание двух ключей Цезаря ($k = 5$ и $k = 19$) и последовательности C_1, C_2, C_2, C_1, C_2.

Блочные шифры

Давайте перенесемся в современность и рассмотрим, как шифрование с симметричными ключами осуществляется сегодня. Все методы такого шифрования подразделяются на две обширные категории: потоковые шифры и блочные шифры. Мы кратко рассмотрим потоковые

шифры в разделе 8.7, когда будем исследовать проблемы безопасности в беспроводных локальных сетях. А в этом разделе мы займемся блочными шифрами, которые применяются во многих защищенных интернет-протоколах, таких как PGP (для безопасной передачи электронной почты), SSL (для защиты TCP-соединений) и IPsec (для защиты передачи данных на сетевом уровне).

```
Буквы открытого текста  a b c d e f g h i j k l m n o p q r s t u v w x y z
    C1(k = 5):          f g h i j k l m n o p q r s t u v w x y z a b c d e
    C2(k = 19):         t u v w x y z a b c d e f g h i j k l m n o p q r s
```

Рис. 8.4. Полиалфавитный шифр, содержащий два шифра Цезаря

В блочном шифре сообщение, которое требуется зашифровать, обрабатывается блоками по k бит. Например, если $k = 64$, то сообщение разбивается на 64-битные блоки, и каждый такой блок шифруется независимо. Для шифрования блока в данном механизме применяется взаимно-однозначное соответствие k-битного блока открытого текста и k-битного блока зашифрованного текста. Рассмотрим пример. Допустим, $k = 3$. В таком случае блочный шифр соотносит 3-битные входные фрагменты (открытый текст) с 3-битными выходными (зашифрованный текст). Вариант такого соответствия дан в табл. 8.1. Обратите внимание: здесь мы имеем взаимно-однозначное соответствие. Это означает, что если отличаются все входные блоки, то отличаются и все соответствующие им выходные. Данный шифр разбивает сообщение на 3-битные блоки и шифрует каждый блок согласно вышеуказанным соответствиям. Можете убедиться, что сообщение 010110001111 в зашифрованном виде превращается в 101000111001.

Табл. 8.1. Разновидность трехбитного блочного шифра

вход	выход	вход	выход
000	110	100	011
001	111	101	010
010	101	110	000
011	100	111	001

Продолжая пример с 3-битными блоками, отметим, что в табл. 8.1 приведен лишь один из возможных вариантов взаимно-однозначного соответствия. Сколько же таких вариантов существует? Чтобы ответить

на этот вопрос, отметим, что указанное соответствие — всего лишь перестановка всех возможных входных значений. Существует 2^3 (= 8) вариантов входных значений (они перечислены в столбцах входа). Эти входные последовательности допускают 8! = 40 320 различных вариантов перестановки. Каждую такую перестановку можно рассматривать как соответствие, а соответствие — как пару симметричных ключей. Если Алиса и Боб знают соответствие (ключ), то способны зашифровывать и расшифровывать сообщения, которыми обмениваются.

Попытка взлома этого шифра простым перебором требует применить для дешифровки все соответствия. Когда существует всего 40 320 вариантов таких соответствий (при k=3), с такой задачей легко справится обычный ПК. Для защиты от атак простым перебором в блочных шифрах обычно применяются гораздо более крупные блоки, состоящие из k = 64 бит и даже еще больше. Следует отметить, что количество возможных соответствий для типичного k-блочного шифра составляет 2^k!, что дает астрономические величины даже для сравнительно небольших значений k (например, k = 64).

Итак, полнотабличные блочные шифры, подобные описанному выше, дают надежные схемы шифрования с симметричными ключами даже при небольших значениях k. К сожалению, такие шифры очень сложны в реализации. При использовании конкретного соответствия со значением k = 64 Алисе и Бобу придется сверяться с таблицей, где содержится 2^{64} значений ввода, что практически неосуществимо. Более того, если бы Алиса и Боб поменяли ключи, то и таблицу им пришлось бы перестроить. Таким образом, полнотабличный блочный шифр, где существуют заранее определенные соответствия между всеми возможными входными и выходными значениями (как в вышеприведенном примере), просто не рассматривается.

Вместо этого в блочных шифрах обычно используются функции, моделирующие таблицы, значения в которых переставляются случайным образом. Пример такой функции (адаптированный вариант из работы Кауфмана[280]) для k = 64 показан на рис. 8.5. Сначала такая функция разбивает 64-битный блок на 8 фрагментов по 8 бит каждый. Каждый фрагмент заменяется по таблице на новое 8-битное значение (такая таблица «8 бит в 8 бит» имеет приемлемый размер). Например, первый 8-битный фрагмент обрабатывается таблицей T_1. Далее 8 фрагментов, полученных в результате замен, пересобираются снова в 64-битный блок. Затем позиции в этом блоке скремблируются (перемешиваются), давая новый 64-битный выходной блок. Этот результат вновь подается

на 64-битный вход, после чего начинается следующая итерация. После *n* таких итераций функция выдает зашифрованный текст. Цель многократных повторов — добиться, чтобы каждый входной бит повлиял на большинство выходных битов (если не на все). Если бы у нас был не цикл, а всего одна итерация, то каждый входной бит менял бы всего 8 из возможных 64 выходных битов. Ключом для такого алгоритма блочного шифрования будут 8 таблиц перестановки (предполагается, что функция скремблирования является общедоступной).

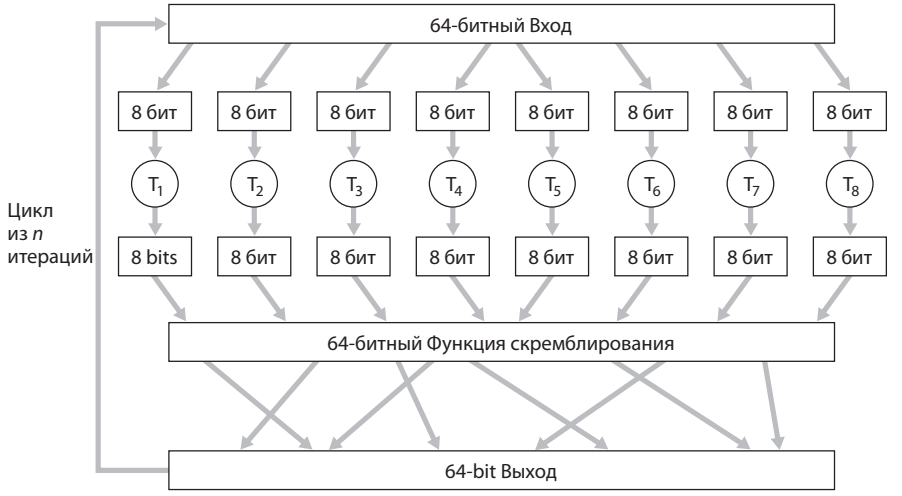


Рис. 8.5. Пример блочного шифра

В настоящее время применяется ряд популярных блочных шифров, в том числе, DES (Data Encryption Standard — Стандарт Шифрования Данных), 3DES и AES (Advanced Encryption Standard — Улучшенный Стандарт Шифрования). Во всех этих стандартах применяются функции, а не заготовленные таблицы, в духе рис. 8.5 (хотя механизм более сложен и специфичен для каждого отдельного шифра). Во всех этих алгоритмах в качестве ключа используется последовательность битов. Например, в алгоритме DES применяются 64-битные блоки с 56-битным ключом. AES использует 128-битные блоки и может оперировать ключами по 128, 192 и 256 бит. Ключ алгоритма определяет конкретные «мини-табличные» соответствия и правила перестановки внутри алгоритма. Для взлома таких шифров простым перебором пришлось бы циклически проверить все ключи, применить к каждому из них алгоритм дешифровки. Как вы помните, при длине ключа равной *n* существует 2^n возможных ключей. По оценке института NIST[369] такая вычислитель-

ная машина, которая взломала бы шифр DES за одну секунду (то есть, перебрала бы за секунду все 2^{56} ключей), справилась бы со 128-битным алгоритмом AES примерно за 149 триллионов лет.

Сцепление блоков шифротекста

В приложениях для обслуживания компьютерных сетей обычно требуется шифровать длинные сообщения (или длинные потоки данных). Если применить блочный шифр так, как было описано выше — просто разбив сообщение на k блоков и независимо зашифровав каждый блок — то возникает неявная, но очень важная проблема. Обратите внимание: ведь два блока открытого текста могут оказаться идентичными. Допустим, два или более блоков текста представляют собой последовательность символов HTTP/1.1. Разумеется, при кодировании таких блоков блочный шифр даст идентичный шифротекст. Злоумышленник вполне может восстановить часть открытого текста, опираясь на одинаковые блоки зашифрованного текста либо даже расшифровать все сообщение, сопоставляя эту информацию с известной базовой структурой протокола[280].

Для устранения такой проблемы следует добавить в шифротекст некоторую долю случайности, таким образом, чтобы идентичные блоки открытого текста в зашифрованном виде выглядели по-разному. Чтобы объяснить эту концепцию, обозначим i-й блок открытого текста как $m(i)$, i-й блок зашифрованного текста как $c(i)$, а при помощи нотации $a \oplus b$ обозначим исключающее ИЛИ (XOR) двух битовых последовательностей — a и b. (Как вы помните, $0 \oplus 0 = 1 \oplus 1 = 0$ и $0 \oplus 1 = 1 \oplus 0 = 1$, а XOR для двух битовых последовательностей выполняется поразрядно. Например, 10101010 ⊕ 11110000 = 01011010). Кроме того, обозначим алгоритм блочного шифрования с ключом S как K_S. Основная идея такова: отправитель создает случайное k-разрядное число $r(i)$ для i-го блока и вычисляет $c(i) = K_S(m(i) \oplus r(i))$. Обратите внимание: случайное k-разрядное число выбирается заново для каждого блока. Затем отправитель посылает $c(1)$, $r(1)$, $c(2)$, $r(2)$, $c(3)$, $r(3)$ и т. д. Поскольку получатель принимает $c(i)$ и $r(i)$, он может восстановить любой блок открытого текста, вычислив $m(i) = K_S(c(i)) \oplus r(i)$. Важно отметить, что, хотя $r(i)$ и отсылается открытым текстом (а значит, Мэри может перехватить эту информацию), Мэри не сможет получить открытый текст $m(i)$, поскольку не знает ключа K_S. Кроме того, если два блока открытого текста — $m(i)$ и $m(j)$ — будут одинаковы, то соответствующие блоки шифротекста $c(i)$ и $c(j)$ получатся разными, так как с очень высокой вероятностью разными окажутся случайные числа $r(i)$ и $r(j)$.

Например, рассмотрим блочный 3-разрядный шифр из табл. 8.1. Допустим, у нас есть открытый текст 010010010. Если Алиса зашифрует его напрямую, без фактора случайности, то на выходе получим результирующий шифротекст 101101101. Если Мэри перехватит и проанализирует этот шифротекст, то сможет верно предположить, что все три блока обычного текста одинаковы. Теперь допустим, что вместо этого Алиса сгенерирует случайные блоки $r(1) = 001$, $r(2) = 111$ и $r(3) = 100$, а затем воспользуется вышеописанным методом, чтобы получить шифротекст $c(1) = 100$, $c(2) = 010$ и $c(3) = 000$. В таком случае все три блока шифротекста являются разными, тогда как блоки открытого текста одинаковы. Затем Алиса посылает $c(1)$, $r(1)$, $c(2)$ и $r(2)$. Можете убедиться, что Боб правильно восстановит исходный открытый текст, воспользовавшись ключом K_S, который известен и ему, и Алисе.

Проницательный читатель заметит, что при введении факторов случайности решается одна проблема, но создается другая, а именно: Алисе приходится передавать вдвое больше бит, чем ранее. Действительно, на каждый бит шифра требуется посылать еще и случайный бит — соответственно, ширина полосы, необходимая на пересылку этих данных, удваивается. Чтобы в данном случае найти золотую середину, в блочных шифрах обычно применяется метод, называемый **сцеплением блоков шифра шифрования** (Cipher Block Chaining, **CBC**). Суть этого метода такова: *случайное значение посылается только с самым первым блоком информации, после чего отправитель и получатель работают с вычисленными блоками кода, заменяющими последующие случайные числа*. Фактически CBC работает так:

1. Перед шифрованием сообщения (или потока данных) отправитель генерирует последовательность из k бит**,** которая называется **вектором инициализации** (Initialization Vector, **IV**)**.** Обозначим этот вектор как $c(0)$**.** Отправитель отсылает получателю вектор инициализации *открытым текстом*.

2. Для первого блока отправитель вычисляет $m(1) \oplus c(0)$, то есть, исключающее ИЛИ первого блока открытого текста, содержащего вектор инициализации. Затем он прогоняет результат через алгоритм блочного шифрования, чтобы получить соответствующий блок шифротекста, то есть, $c(1) = K_S(m(1) \oplus c(0))$. Отправитель посылает зашифрованный блок $c(1)$ получателю.

3. Для i-го блока отправитель генерирует i-й блок шифротекста из $c(i) = K_S(m(i) \oplus c(i - 1))$.

Теперь давайте исследуем некоторые последствия применения этого метода. Во-первых, получатель должен иметь возможность восстановить исходное сообщение. Действительно, имея $c(i)$, получатель дешифрует его при помощи K_S, чтобы найти значение $s(i) = m(i) \oplus c(i - 1)$; поскольку получатель также знает $c(i - 1)$, он далее может восстановить блок открытого текста с помощью выражения $m(i) = s(i) \oplus c(i - 1)$. Во-вторых, даже если два блока открытого текста окажутся идентичными, соответствующие им блоки шифротекста (практически всегда) будут разными. В-третьих, хотя отправитель посылает вектор инициализации открытым текстом, злоумышленник не сможет дешифровать блоки шифротекста, так как не знает секретного ключа S. Наконец, отправителю приходится послать «в нагрузку» к шифру всего один блок (вектор инициализации). В результате расход полосы передачи данных лишь пренебрежимо возрастает даже для очень длинных сообщений (которые могут состоять из сотен блоков).

Для примера давайте определим шифротекст для 3-битного блочного шифра из табл. 8.1. Пусть наш открытый текст — 010010010, а вектор инициализации равен $c(0) = 001$. Сначала отправитель применяет вектор инициализации для вычисления $c(1) = K_S(m(1) \oplus c(0)) = 100$. Затем отправитель вычисляет $c(2) = K_S(m(2) \oplus c(1)) = K_S(010 \oplus 100) = 000$ и $c(3) = K_S(m(3) \oplus c(2)) = K_S(010 \oplus 000) = 101$. Можете убедиться, что получатель, знающий значения IV и K_S, способен восстановить исходный открытый текст.

Режим сцепления блоков шифра учитывается при разработке защищенных сетевых протоколов: протокол требуется оснащать специальным механизмом, позволяющим передавать вектор инициализации от отправителя к получателю. Ниже в этой главе мы рассмотрим, как эта задача решается в некоторых протоколах.

8.2.2. Шифрование с открытым ключом

В течение более 2000 лет (со времен Цезаря до 70-х годов XX века) для шифрованной связи требовалось, чтобы две общающиеся стороны хранили общий секрет — для шифрования и дешифрирования использовался один и тот же ключ. С этим методом была связана определенная сложность: обе стороны должны были как-то договориться об общем ключе, но для этого им опять же требовалась (предположительно *безопасная*) связь! Поэтому обеим сторонам необходимо было сначала лично встретиться и договориться о ключе (например, два центуриона

могли побеседовать в римских банях), и лишь после этого они получали возможность общаться при помощи шифрованных посланий. Однако в сетевом мире два общающихся собеседника могут никогда не встретиться. Поэтому возникает вопрос: могут ли две стороны обмениваться по сети шифрованными сообщениями, не имея изначально общего секретного ключа? В 1976 году Диффи и Хеллман[132] продемонстрировали алгоритм (называемый теперь алгоритмом обмена ключа Диффи и Хеллмана), обеспечивающий именно такую возможность — принципиально иной и изумительно красивый метод безопасной связи, приведший к разработке современных криптографических систем с открытым ключом. Далее будет показано, что криптографические системы с открытым ключом обладают замечательными свойствами, благодаря которым могут использоваться не только для шифрования данных, но также для аутентификации и цифровых подписей. Интересно, что идеи, сходные с описываемыми в работах Диффи[132] и Ривеста[574], независимо от них высказывались в начале 70-х годов в ряде секретных отчетов британских исследователей из компании CESG (Communications-Electronics Security Group — группа безопасности связи и электроники)[150]. Как это часто бывает, замечательные идеи возникают независимо в разных местах. К счастью, в настоящее время технические достижения, связанные с шифрованием открытым ключом, уже стали достоянием не только отдельных специалистов, но и широкой общественности.

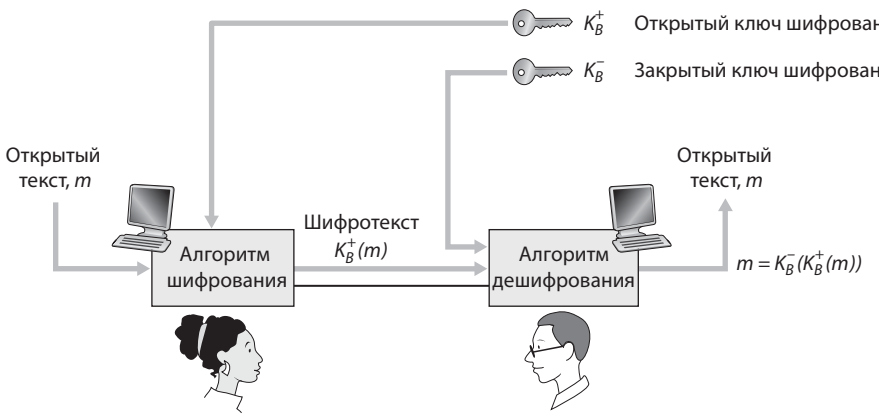


Рис. 8.6. Криптография с открытым ключом

Концепция шифрования с открытым ключом очень проста. Допустим, Алиса хочет связаться с Бобом. Как показано на рис. 8.6, вместо того чтобы использовать один общий секретный ключ (как это делается

в системах с симметричными ключами), у Боба (получателя сообщений Алисы) есть два ключа — **открытый ключ**, доступный *всем и каждому* (в том числе злоумышленнику), и **закрытый ключ**, известный только Бобу. Для открытого и закрытого ключей Боба мы будем использовать обозначения K_B^+ и K_B^-, соответственно. Для общения с Бобом Алиса сначала получает открытый ключ Боба. Затем она зашифровывает свое сообщение m при помощи известного (возможно, стандартизованного) алгоритма шифрования и открытого ключа Боба. То есть Алиса вычисляет $K_B^+(m)$. Боб получает зашифрованное Алисой сообщение и расшифровывает его известным (например, стандартизованным) алгоритмом дешифрирования с помощью своего закрытого ключа. То есть Боб вычисляет $K_B^-(K_B^+(m))$. Далее будет показано, что существуют алгоритмы шифрования и дешифрирования, а также методы выбора открытого и закрытого ключей, таких что $K_B^-(K_B^+(m)) = m$; то есть, применив открытый ключ Боба K_B^+ к сообщению m (вычисляем $K_B^+(m)$), а затем применив закрытый ключ Боба K_B^- к зашифрованной версии сообщения m (вычисляем $K_B^-(K_B^+(m))$) получаем в результате исходный вариант сообщения m. Это замечательный результат! Таким образом, Алиса может использовать доступный всем и каждому ключ Боба, чтобы посылать Бобу секретные сообщения. При этом отпадает необходимость в передаче секретного ключа! Как вы узнаете далее, можно поменять местами открытый и закрытый ключи, получив все тот же замечательный результат: $K_B^-(^+_B(m)) = K_B^+(K_B^-(m)) = m$.

Итак, концепция шифрования с открытым ключом проста. Однако тут же возникают две проблемы. Во-первых, злоумышленнику, перехватившему сообщение Алисы, известен открытый ключ Боба (ведь это общедоступная информация) и алгоритм шифрования. Таким образом, хотя злоумышленник и увидит в письме Алисы лишь тарабарщину, он может предпринять атаку подбором открытого текста. То есть Мэри может попытаться закодировать открытым ключом предполагаемые варианты сообщений или фрагментов сообщений и сравнить результат с перехваченной шифровкой. Разумеется, система шифрования с открытым ключом должна быть создана так, чтобы злоумышленник не мог по открытому ключу получить закрытый (или это должно быть настолько трудно, что можно считать невыполнимым), либо каким-то другим образом расшифровать тайное сообщение, отправленное Алисой Бобу. Вторая проблема заключается в том, что, поскольку для шифрования используется открытый ключ Боба, то, очевидно, описанная выше простая схема не обеспечивает гарантии подлинности, а значит, кто угодно (включая Алису) может послать Бобу сообщение *от имени* Алисы. В си-

стеме с симметричными секретными ключами сам факт того, что отправителю известен секретный ключ, удостоверял его личность. В системе же с открытым ключом это не так, поскольку кто угодно может взять открытый ключ Боба и зашифровать им свое сообщение. Для привязки сообщения к отправителю требуется еще и цифровая подпись — эту тему мы подробно обсудим в разделе 8.3.

Алгоритм RSA

Описанные проблемы решаются при помощи разнообразных алгоритмов шифрования, но мы остановимся на одном из них, **алгоритме RSA** (названный по инициалам его разработчиков, Рона Ривеста, Ади Шамира и Леонарда Адлемана), который практически стал синонимом шифрования с открытым ключом. Рассмотрим сначала, как работает алгоритм RSA, а затем поговорим о том, почему он работает.

В алгоритме RSA широко применяются арифметические операции с использованием арифметики по модулю n. Давайте кратко ознакомимся с модулярной арифметикой. Как вы, возможно, знаете, $x \bmod n$ означает просто остаток от деления x на n. Например, $19 \bmod 5 = 4$. В модулярной арифметике используются обычные операции сложения, умножения и возведения в степень. Но результат каждой такой операции заменяется целочисленным остатком, который получается, если разделить этот результат на n. Сложение и умножение в модулярной арифметике упрощается благодаря следующим удобным фактам:

$$[(a \bmod n) + (b \bmod n)] \bmod n = (a + b) \bmod n$$
$$[(a \bmod n) - (b \bmod n)] \bmod n = (a - b) \bmod n$$
$$[(a \bmod n) \times (b \bmod n)] \bmod n = (a \times b) \bmod n$$

Из третьего выражения следует, что $(a \bmod n)^d \bmod n = a^d \bmod n$ — вскоре мы убедимся в исключительной полезности этого свойства.

Итак, предположим, что Алиса хочет послать Бобу письмо, зашифрованное алгоритмом RSA, как показано на рис. 8.6. Говоря об алгоритме RSA, необходимо иметь в виду, что сообщение — это просто последовательность битов, а любая такая последовательность может быть уникально представлена целым числом (с указанием длины этой последовательности). Допустим, у нас имеется последовательность битов 1001; это сообщение может быть представлено десятичным числом 9. Соответственно, при шифровании сообщения алгоритмом RSA мы фактически шифруем целое число, представляющее это сообщение.

Алгоритм RSA включает два тесно взаимосвязанных этапа.

- Выбор открытого и закрытого ключей.
- Шифрование и дешифрирование.

Чтобы получить открытый и личный ключи RSA, Боб должен выполнить следующие действия.

1. Выбрать два больших простых числа p и q. Насколько большими должны быть эти числа? Чем они больше, тем труднее взломать шифр RSA, но тем дольше продлятся шифрование и дешифрирование. Сотрудники компании RSA Laboratories рекомендуют выбирать числа p и q так, чтобы их произведение было длиной около 1024 бит. Тема нахождения больших простых чисел обсуждается в публикациях Калдвелла[73].
2. Вычислить $n = pq$ и $z = (p - 1)(q - 1)$.
3. Выбрать число e, меньшее, чем n, у которого нет общих делителей (кроме 1) с числом z. (В этом случае говорят, что числа e и z являются относительно простыми.) Буква e используется потому, что с нее начинается английское слово encryption (шифрование).
4. Найти число d, такое, чтобы $ed - 1$ без остатка делилось на 2. Буква d используется потому, что с нее начинается английское слово decryption (дешифрирование). Другими словами, при заданном числе e мы выбираем число d такое, чтобы
$$ed \bmod z = 1$$
5. Открытый ключ K_B^+, доступ к которому Боб предоставляет всему миру, — это пара чисел (n, e), а закрытый ключ Боба, K_B^- — пара чисел (n, d).

Операции шифрования со стороны Алисы и дешифрирования со стороны Боба выполняются следующим образом.

- Предположим, Алиса хочет послать Бобу последовательность битов, представляемую числом m, так, что $m < n$. Для шифрования Алиса выполняет возведение в степень m^e, а затем вычисляет целочисленный остаток от деления m^e на n. Иными словами, зашифрованное значение c, соответствующее открытому Алисиному сообщению m, равно
$$c = m^e \bmod n$$
Последовательность битов, соответствующая данному зашифрованному тексту c, отсылается Бобу.
- Чтобы расшифровать полученное сообщение c, Боб вычисляет
$$m = c^d \bmod n$$
для чего требуется его закрытый ключ (n, d).

Рассмотрим работу алгоритма RSA на простом примере. Допустим, Боб выбирает числа $p = 5$ и $q = 7$. (Разумеется, эти значения слишком малы, чтобы обеспечивать безопасность.) В этом случае $n = 35$ и $z = 24$. Далее Боб выбирает число $e = 5$, так как у чисел 5 и 24 нет общих делителей. Наконец, Боб выбирает число $d = 29$, так как число $5 \times 29 - 1$ (то есть $ed - 1$) без остатка делится на 24. Боб открывает всем два значения $n = 35$ и $e = 5$ и сохраняет в секрете число 29. Пусть теперь Алиса хочет послать Бобу символы *l*, *o*, *v* и *e*, зашифровав их открытым ключом Боба. Алиса интерпретирует каждую букву числом в диапазоне от 1 до 26 (*a* соответствует 1, а *z* соответствует 26). Алиса и Боб выполняют шифрование и дешифрование, представленные в табл. 8.2 и 8.3 соответственно. Обратите внимание: в данном примере мы считаем каждую из четырех букв отдельным сообщением. В более реалистичном примере потребовалось бы преобразовать четыре буквы в их 8-битные представления в кодировке ASCII, а затем зашифровать целое число, соответствующее результирующему 32-битному фрагменту. Числа, которые получаются в таком реалистичном примере, слишком длинные, чтобы напечатать их в книге!

Итак, в табл. 8.2 и 8.3 был приведен, в сущности, «игрушечный» пример, а у нас уже получились огромные числа. Поскольку выше мы уже убедились, что каждое из чисел p и q должно иметь несколько сотен бит в длину, налицо определенные проблемы, возникающие при практическом применении алгоритма RSA.

Табл. 8.2. Алиса выполняет шифрование по алгоритму RSA; $e = 5$, $n = 35$

Буква открытого текста	m: числовое представление	m^e	Шифротекст $c = m^e \bmod n$
l	12	248832	17
o	15	759375	15
v	22	5153632	22
e	5	3125	10

Табл. 8.3. Боб выполняет расшифровку по алгоритму RSA; $d = 29$, $n = 35$

Шифротекст c	c^d	$m = c^d \bmod n$	Буква открытого текста
17	481968572106750915091509141182522307169 7	12	l
15	12783403940394885893911123275756835937 5	15	o
22	85164331908653770195619449972110603059 2	22	v
10	10000000000000000000000000000000	5	e

Как выбираются большие простые числа? Как затем выбрать числа e и d? Каким образом вычислять степени больших чисел? Обсуждение этих важных вопросов выходит за рамки темы нашей книги; подробности см. в работе Кауфмана[280] и приведенных в этом издании справочных материалах.

Сеансовые ключи

Следует заметить, что возведение в степень, применяемое в алгоритме RSA, требует массу процессорного времени. Для сравнения, алгоритм DES работает в 100 раз быстрее в программном исполнении и от 1000 до 10 000 раз быстрее в аппаратной реализации[575]. В результате на практике алгоритм RSA часто применяется в комбинации с криптографией симметричных ключей. Например, если Алиса хочет срочно послать Бобу большое количество зашифрованных данных, она может поступить так. Сначала Алиса выбирает ключ, который будет применяться для шифрования самих данных, иногда называемый *сеансовым* ключом и обозначаемый как K_S. Алиса должна сообщить Бобу сеансовый ключ, так как это будет разделяемый симметричный ключ, применяющийся при симметричном шифровании (например, с алгоритмом DES или AES). Этот ключ Алиса зашифровывает открытым ключом Боба, то есть вычисляет $c = (K_S)^e \bmod n$. Боб получает сеансовый ключ c и расшифровывает его для получения конкретного значения этого ключа.

Принцип работы алгоритма RSA

Описанные операции шифрования и дешифрирования по алгоритму RSA напоминают какой-то фокус. Почему при применении алгоритма шифрования, а впоследствии — алгоритма дешифрования удается восстановить исходное сообщение? Чтобы понять, как работает алгоритм RSA, вновь обозначим $n = pq$, где p и q являются большими простыми числами.

Вспомним, что при шифровании по алгоритму RSA сообщение (представляемое в виде целого числа) m сначала возводится в степень e при помощи арифметики по модулю n, вот так:

$$c = m^e \bmod n$$

Дешифрирование осуществляется также возведением зашифрованного символа в степень d опять же при помощи арифметики по модулю n. Таким образом, результат выполнения этих двух операций представ-

ляет собой $(m^e \bmod n)^d \bmod n$. Что можно сказать об этой величине? Как было указано выше, важное свойство модулярной арифметики заключается в том, что равенство $(a \bmod n)^d \bmod m = a^d \bmod n$ соблюдается для любых значений a, n и d. Соответственно, воспользовавшись $a = m^e$ в таком качестве, имеем:

$$(m^e \bmod n)^d \bmod n = m^{ed} \bmod n$$

Остается доказать, что $m^{ed} \bmod n = m$. Хотя мы и стараемся прояснить механизмы работы алгоритма RSA, для этого нам придется опираться на довольно сложный результат, для получения которого применяется теория чисел. Нам нужен результат, удовлетворяющий следующим условиям[280]: если числа p и q простые, $n = pq$ и $z = (p - 1)(q - 1)$, то $x^y \bmod n$ равно $x^{(y \bmod z)} \bmod n$. Применив этот результат с $x = m$ и $y = ed$, имеем:

$$m^{ed} \bmod n = m^{(ed \bmod z)} \bmod n$$

Но не забывайте, что мы выбрали такие значения e и d, что $ed \bmod z = 1$. Таким образом:

$$m^{ed} \bmod n = m^1 \bmod n = m$$

Именно на этот результат мы и рассчитывали! Сначала возводя число m в степень e (то есть зашифровывая), а затем возводя в степень d (то есть расшифровывая), мы получаем исходное значение m. Еще *более* замечателен тот факт, что если мы сначала возведем число m в степень d, а затем в степень e, то есть выполним операции шифрования и дешифрирования в обратном порядке, мы также получим исходное значение m. Этот результат непосредственно проистекает из свойств модулярной арифметики:

$$(m^d \bmod n)^e \bmod n = m^{de} \bmod n = m^{ed} \bmod n = (m^e \bmod n)^d \bmod n$$

Безопасность алгоритма RSA основывается на том факте, что алгоритм быстрого определения делителей числа, в данном случае алгоритм нахождения делителей p и q известного числа n, пока не найден. Если вам известны значения p и q, тогда, зная открытое число e, нетрудно вычислить секретный ключ d. С другой стороны, быстрые алгоритмы нахождения делителей числа всего лишь не найдены, *отсутствие* таких алгоритмов не доказано, поэтому, строго говоря, безопасность алгоритма RSA не гарантирована.

Еще один популярный алгоритм шифрования носит имя своих первооткрывателей — Диффи и Хеллмана. Этот алгоритм мы кратко иссле-

дуем в домашнем задании к данной главе. Он не столь универсален, как RSA, поскольку не может использоваться для шифрования сообщений произвольной длины. Правда, с его помощью удобно создавать симметричные сеансовые ключи, а уже такие ключи применять для шифрования сообщений.

8.3. Целостность сообщений и цифровые подписи

В предыдущем разделе было рассмотрено, как шифрование можно использовать для обеспечения конфиденциальной передачи данных между двумя участниками коммуникации. В этом разделе мы обсудим не менее важную криптографическую тему — обеспечение **целостности сообщения** (синоним — аутентификация сообщения). Наряду с целостностью сообщения мы изучим еще две тесно связанные с ней темы: цифровые подписи и аутентификацию конечной точки.

Эту проблему мы вновь опишем на примере общения наших героев, Алисы и Боба. Допустим, Боб получает письмо (которое может быть как зашифрованным, так и открытым) и считает, что оно было отослано Алисой. Чтобы аутентифицировать это сообщение, Боб должен удостовериться в следующих двух фактах:

1. Сообщение действительно было отправлено Алисой
2. На пути к Бобу Алисино сообщение не **было изменено (**подделано**)**

В разделах 8.4–8.7 мы убедимся, что такая проблема целостности сообщения критически важна практически во всех безопасных сетевых протоколах.

В качестве конкретного примера рассмотрим компьютерную сеть, в которой применяется алгоритм маршрутизации с учетом состояния канала (например, OSPF). Этот алгоритм требуется для определения путей между каждыми двумя маршрутизаторами, работающими в сети (см. главу 4). В алгоритме с учетом состояния канала каждый маршрутизатор должен широковещательно распространять сообщение о состоянии канала всем остальным маршрутизаторам в сети. В таком сообщении маршрутизатор дает список непосредственно подключенных к нему (смежных) узлов и стоимость путей до этих узлов. Когда маршрутизатор получит сообщения о состоянии канала от всех других маршрутизаторов, он может построить полную карту сети, запустить свой алгоритм

для вычисления наименьшей стоимости маршрутизации и сконфигурировать свою таблицу маршрутизации. Мэри может предпринять сравнительно простую атаку на алгоритм маршрутизации: распространить по сети фиктивные сообщения о состоянии канала, в которых будет содержаться неверная информация о них

Поэтому нам и требуется удостовериться в целостности сообщения: когда маршрутизатор Б получает сообщение о состоянии канала от маршрутизатора А, он должен убедиться, что, во-первых, это сообщение создано именно маршрутизатором А, и, во-вторых, никто не изменил его в ходе передачи.

В этом разделе будет описана популярная техника проверки целостности сообщений, применяемая во многих безопасных сетевых протоколах. Но перед тем как заняться данным вопросом, требуется изучить еще одну важную тему из области криптографии — криптографические хэш-функции.

8.3.1. Криптографические хэш-функции

Как показано на рис. 8.7, хэш-функция принимает ввод m и на его основе вычисляет значение фиксированной разрядности $H(m)$. Это значение (битовая последовательность) называется «хэш». Контрольная сумма, используемая в Интернете (глава 3), и код циклического контроля (глава 4) соответствуют такому определению. Но **криптографическая хэш-функция** также должна иметь еще одно важное свойство:

- Не существует метода вычислений, который позволил бы найти любые два различных сообщения x и y, удовлетворяющих условию: $H(x) = H(y)$.

На практике это означает, что злоумышленник не сможет при помощи вычислений заменить конкретное сообщение другим, если оно защищено хэш-функцией. Таким образом, если $(m,H(m))$ — это сообщение и хэш этого сообщения, созданный отправителем, то злоумышленник не сможет создать сообщение с иным содержимым, y, которое обладало бы таким же хэш-значением, как и исходное.

Убедимся в том, что простая контрольная сумма, вроде той, что применяется в Интернет-протоколах, плохо подходит для вычисления хэш-значения. Вместо того чтобы выполнять арифметические действия с дополнением до единицы (как в Интернет-протоколах), мы будем вы-

числять контрольную сумму, обращаясь с каждым символом как с байтом и суммируя все байты блоками по четыре. Допустим, Боб задолжал Алисе 100 долларов и 99 центов и отправляет ей долговую расписку в виде текстовой строки: `IOU100.99BOB`. В формате ASCII (в шестнадцатеричной нотации) эти символы выглядят следующим образом: `49`, `4F`, `55`, `31`, `30`, `30`, `2E`, `39`, `39`, `42`, `4F`, `42`.

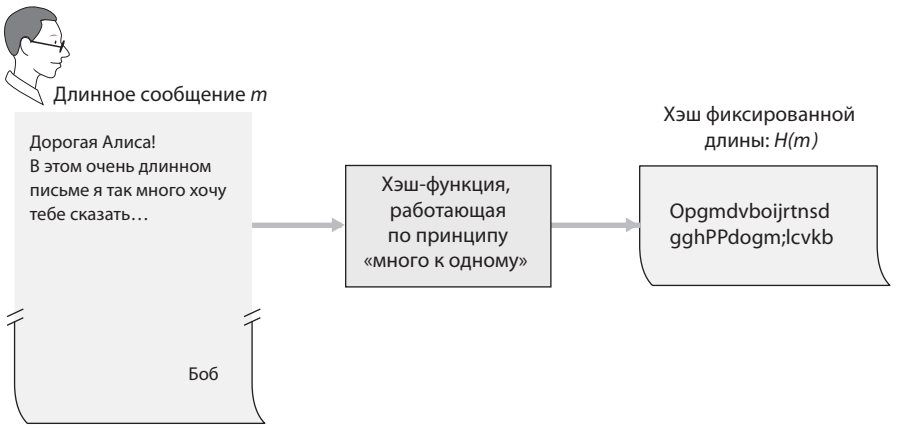


Рис. 8.7. Хэш-функции

В верхней части рис. 8.18 показано, что четырехбайтовая сумма для этого сообщения равна `B2 C1 D2 AC`. Немного отличающееся сообщение (со значительно большей суммой) показано в нижней части рисунка. Однако у сообщений `IOU100.99BOB` и `IOU900.19BOB` *одинаковая* контрольная сумма! Таким образом, этот простой алгоритм нарушает вышеприведенное требование. При использовании данного алгоритма нетрудно найти другое сообщение с той же контрольной суммой. Очевидно, что для обеспечения безопасности нам необходима более мощная, нежели контрольная сумма, хэш-функция.

Сегодня получил широкое распространение алгоритм вычисления хэша сообщения MD5[444], разработанный Роном Ривестом. Он позволяет вычислять 128-битный хэш сообщения в четыре этапа. Сначала к сообщению добавляется единица и нули для дополнения его длины до определенного норматива. Затем к сообщению добавляется 64-разрядное представление исходной длины сообщения. Потом инициализируется сумматор, и, наконец, сообщение обрабатывается блоками в четырехэтапном цикле. Подробное описание MD5 (в том числе — исходный код его реализации на языке C) дается в стандарте RFC 1321[444].

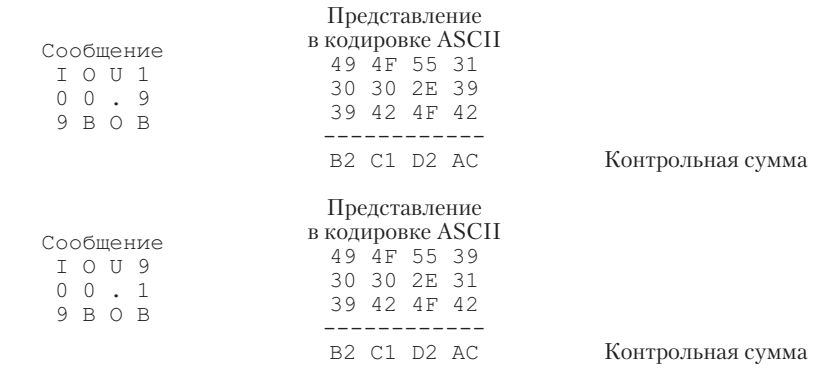


Рис. 8.8. Исходное и поддельное сообщение обладают одинаковыми контрольными суммами!

Другим распространенным сегодня алгоритмом вычисления хэш-значения является алгоритм SHA-1 (Secure Hash Algorithm — безопасный алгоритм хэширования, версия 1)[161]. Этот алгоритм основан на принципах, сходных с используемыми в алгоритме MD4[443], который появился раньше MD5. Алгоритм SHA-1 является федеральным стандартом США. Его предписывается использовать для федеральных приложений, когда требуется надежное вычисление хэш-значений. Алгоритм формирует 160-битный хэш сообщения. Благодаря большей разрядности алгоритм SHA-1 считается более надежным.

8.3.2. Код аутентификации сообщения

Вернемся к проблеме целостности сообщения. Теперь, понимая принципы работы хэш-функций, давайте в общих чертах опишем, как можно проверить целостность сообщения:

1. Алиса создает сообщение m и вычисляет для него хэш $H(m)$ (например, при помощи алгоритма SHA-1)
2. Затем Алиса прикрепляет значение $H(m)$ к сообщению m, создавая расширенное сообщение $(m, H(m))$, которая посылает Бобу.
3. Боб получает расширенное сообщение (m, h) и вычисляет $H(m)$. Если $H(m) = h$, то Боб может быть уверен, что все в порядке.

Очевидно, у такого подхода есть серьезный недостаток. Мэри может создать фальшивое сообщение m', в котором выдаст себя за Алису, высчитает $H(m')$ и отошлет Бобу сообщение $(m', H(m'))$. Когда Боб по-

лучит такую информацию, она будет признана корректной на этапе 3, и никакого мошенничества Боб не заметит.

Для обеспечения целостности сообщений Бобу и Алисе придется использовать не только криптографические хэш-функции, но и разделенный секрет *s*. Разделенный секрет — это просто последовательность битов, называемая ключом аутентификации. При помощи разделенного секрета целостность сообщения подтверждается следующим образом:

1. Алиса создает сообщение *m*, сцепляет его с ключом *s* для создания *m*+*s* и вычисляет хэш *H*(*m*+*s*) (например, при помощи алгоритма SHA-1). Информация *H*(*m* + *s*) называется **кодом аутентификации сообщения** (message authentication code, **MAC**)
2. Затем Алиса прикрепляет MAC к сообщению *m*, создавая расширенное сообщение (*m*,*H*(*m*+*s*)), которое и посылает Бобу.
3. Боб получает расширенное сообщение (*m*, *h*) и, зная разделенный секрет *s*, вычисляет MAC *H*(*m*+*s*). Если *H*(*m*+*s*) = *h*, то Боб убеждается, что все в порядке.

Вся эта процедура схематически представлена на рис. 8.9. Обратите внимание: здесь аббревиатура MAC означает «код аутентификации сообщения», а совсем не «управление доступом к среде передачи», как в главе о протоколах канального уровня!

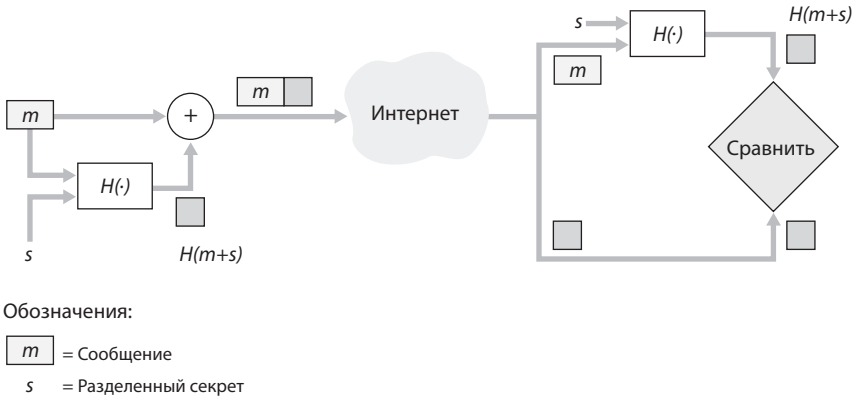


Рис. 8.9. Код аутентификации сообщения (MAC)

Удобное свойство MAC заключается в том, что он работает без применения алгоритма шифрования. Действительно, во многих прикладных областях, в частности, при использовании алгоритмов, учитывающих

состояние канала, обменивающиеся информацией узлы обычно должны обеспечивать лишь целостность сообщения, но не его конфиденциальность. Применяя MAC, эти стороны могут аутентифицировать сообщения, которыми обмениваются, но при этом подтверждать целостность информации без привлечения сложных алгоритмов шифрования.

Как вы уже, вероятно, догадываетесь, за годы работы были предложены разнообразные стандарты работы с MAC. Наиболее популярный из них называется **HMAC**. Стандарт HMAC может использоваться как с MD5, так и с SHA-1. На самом деле, HMAC дважды прогоняет через хэш-функцию данные и ключ аутентификации[280, 465].

Остается решить еще одну важную проблему. Как раздать разделяемый ключ аутентификации двум сторонам, обменивающимся информацией? Например, в алгоритме маршрутизации с учетом состояния канала нам нужно каким-то образом доставить секретный ключ аутентификации на каждый из маршрутизаторов, работающих в автономной системе. При этом отметим, что все маршрутизаторы могут использовать один и тот же ключ аутентификации. Обычно администратор вычислительной сети решает эту задачу, просто обойдя все маршрутизаторы и записав на них нужные значения. Если же администратор вычислительной сети — человек ленивый, а каждый маршрутизатор обладает собственным открытым ключом, то вполне можно разослать ключ аутентификации по всем маршрутизаторам, в каждом случае шифруя сообщение с ключом открытым ключом маршрутизатора, а затем посылая зашифрованный таким образом ключ по сети соответствующему маршрутизатору.

8.3.3. Цифровые подписи

Вспомните, сколько раз вы ставили свою подпись на различных документах в течение последней недели. Вы подписываете чеки, квитанции о получении кредитных карт, письма и прочие документы. Ваша подпись подтверждает факт, что вы (а не кто-либо иной) ознакомились и/или согласились с содержимым документа. В цифровом мире также часто возникает необходимость указать автора или согласиться с содержимым документа. Для этого в цифровом мире применяется основанный на криптографии метод, называемый **цифровой подписью**.

Как и обычная, цифровая подпись должна быть выполнена таким образом, чтобы ее можно было проверить и нельзя было подделать. Таким

образом, цифровая подпись призвана доказать, что документ подписан действительно тем, кем он подписан, и *только* он мог его подписать (т. е. подпись нельзя подделать, и подписавшаяся сторона не имеет возможности заявить, что она не подписывала документ).

Рассмотрим, как можно разработать схему цифровой подписи. Отметим, что, когда Боб подписывает сообщение, он должен оставить на нем какой-либо уникальный автограф, принадлежащий именно Бобу. Так, можно снабдить подпись кодом аутентификации сообщения (MAC), причем такой код создается путем прикрепления к сообщению специального ключа (принадлежащего именно Бобу) с последующим взятием хэша. Но чтобы Алиса могла проверить подпись, у нее также должна быть копия ключа — следовательно, ключ Боба уже не будет уникальным. Соответственно, MAC-коды не подходят для решения этой задачи.

Как вы помните, при применении криптографии с открытым ключом у Боба есть два ключа — открытый и закрытый, — причем оба эти ключа есть только у Боба (они уникальны). Таким образом, криптография с открытым ключом очень хорошо подходит для создания цифровых подписей. Теперь давайте изучим практическую сторону этого вопроса.

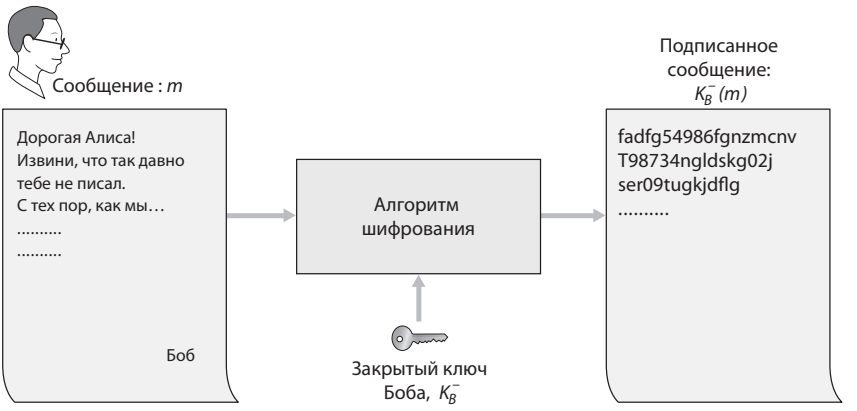


Рис. 8.10. Создание цифровой подписи для документа

Предположим, Боб хочет снабдить цифровой подписью документ m. Это может быть файл или сообщение, которое Боб собирается послать. Как показано на рис. 8.10, чтобы подписать этот документ, Боб просто вычисляет значение $K_B^-(m)$ с помощью своего закрытого ключа K_B^-. На первый взгляд может показаться странным, что Боб использует для подписи свой закрытый, а не открытый ключ (в разделе 8.2 мы узна-

ли, что закрытый ключ используется для дешифрования сообщения, которое было зашифровано открытым ключом). Однако не забывайте, что шифрование и дешифрование — это просто математические операции, возведение в степень e или d по алгоритму RSA; см. раздел 8.2, и в данной ситуации цель Боба является диаметрально противоположной шифрованию документа. В данном случае необходимо не скрыть данные, а подписать их, обеспечив их доступность проверке и невозможность подделки. Цифровая подпись, которую Боб ставит на документе — это $K_B^-(m)$.

Удовлетворяет ли цифровая подпись $K_B^-(m)$ нашим требованиям проверяемости и невозможности подделки? Предположим, у Алисы есть документ m и цифровая подпись $K_B^-(m)$. Она хочет доказать в суде, что Боб действительно подписал этот документ и что он единственный, кто мог его подписать. Алиса расшифровывает цифровую подпись $K_B^-(m)$, ассоциированную с документом m, при помощи открытого ключа Боба K_B^+, то есть вычисляет $K_B^+(K_B^-(m))$. В результате (вуаля!) она получает документ m, точно совпадающий с оригиналом. Затем Алиса заявляет, что только Боб мог подписать этот документ по следующим причинам:

- Тот, кто подписал этот документ, должен знать закрытый ключ Боба K_B^-, чтобы вычислить значение $K_B^-(m)$, такое что $K_B^+(K_B^-(m)) = m$.
- Единственный человек, которому известен личный ключ Боба K_B^- это сам Боб. Как отмечалось в разделе 8.2 при обсуждении алгоритма RSA, знание открытого ключа K_B^+ не поможет узнать закрытый ключ Боба K_B^-. Таким образом, единственным, кто может знать личный ключ Боба K_B^- является тот человек, который сгенерировал пару ключей (K_B^+, K_B^-). Предполагается, что Боб никому не давал свой личный ключ K_B^-, и никто не мог украсть личный ключ Боба K_B^-.

Также необходимо отметить, что если изменить оригинальный документ m, преобразовав его к виду m' созданная Бобом подпись не будет действительной для модифицированного документа m', так как $K_B^+(K_B^-(m))$ не равно m'. Соответственно, мы также видим, как цифровые подписи способствуют и соблюдению целостности сообщения: получатель может убедиться, что сообщение не было изменено, а также узнать его источник

Важная проблема, связанная с шифрованием и дешифрованием данных, заключается в том, что эти процедуры требуют значительных вычислительных затрат. Учитывая издержки, связанные с шиф-

рованием и дешифрованием, подпись данных путем их полного шифрования/дешифрования можно считать излишней перестраховкой. Более эффективный метод — использование для цифровой подписи хэш-функций. Как вы помните из раздела 8.3.2, алгоритм хэширования принимает сообщение m произвольной длины и вычисляет «отпечаток» сообщения, имеющий фиксированную длину. Этот отпечаток обозначают $H(m)$. При помощи хэш-функции Боб подписывает хэш сообщения, а не само это сообщение. Таким образом, Боб вычисляет $K_B^-(H(m))$. Поскольку $H(m)$ обычно гораздо меньше, чем исходное сообщение m, вычислительная мощность, требуемая на создание такой цифровой подписи, значительно снижается.

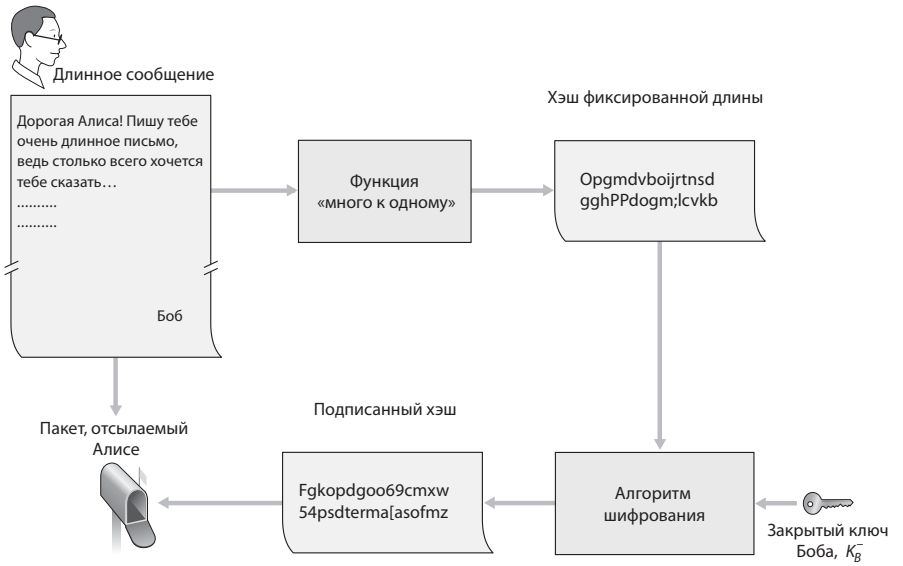


Рис. 8.11. Отправка сообщения, снабженного цифровой подписью

Продолжая примеры с перепиской Боба и Алисы, рассмотрим на рис. 8.11 обобщенную процедуру создания цифровой подписи. Боб написал длинное письмо и прогоняет его через хэш-функцию. Затем он подписывает результирующее значение хэша своим закрытым ключом. Исходное сообщение (в виде открытого текста), а также хэш сообщения, снабженный цифровой подписью (мы называем его просто «цифровая подпись»), отправляется Алисе. На рис. 8.12 в обобщенном виде представлена процедура проверки цифровой подписи. Алиса применяет к полученному сообщению открытый ключ отправителя, чтобы получить хэш-результат. Кроме того, она обрабатывает этим же ключом

и исходное сообщение (открытый текст), чтобы получить второй хэш-результат. Если два хэша совпадают, то Алиса может не беспокоиться о целостности сообщения и быть уверенной в том, кто его автор.

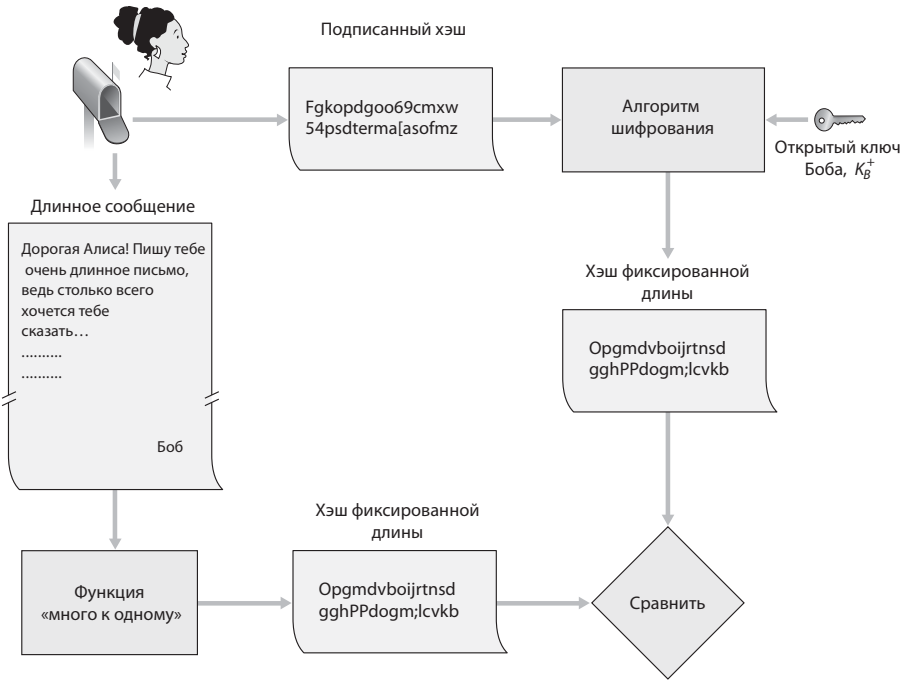


Рис. 8.12. Проверка подписанного сообщения

Прежде чем переходить к изучению дальнейшего материала, давайте кратко сравним цифровые подписи и коды MAC, поскольку у этих сущностей есть параллели, но в то же время между ними существуют важные неявные отличия. Как цифровая подпись, так и MAC создается на основе сообщения (или документа). Чтобы создать код MAC из сообщения, мы прикрепляем к этому сообщению ключ аутентификации, а затем берем хэш результата. Обратите внимание: при создании MAC не применяется ни шифрование открытым ключом, ни симметричными ключами. Для создания цифровой подписи мы сначала берем хэш сообщения, а затем шифруем сообщение нашим закрытым ключом (здесь применяется криптография по методу открытого ключа).

Итак, метод с применением цифровой подписи более «тяжеловесный», так как он работает на основе инфраструктуры открытых ключей (Public Key Infrastructure, PKI), взаимодействующей с органами

сертификации, описанными ниже. В разделе 8.4 мы также поговорим о PGP — популярной системе для защищенного обмена электронной почтой — которая использует цифровые подписи для обеспечения целостности сообщений. Выше мы уже рассмотрели, как алгоритм OSPF гарантирует целостность сообщений при помощи кодов MAC. В разделах 8.5 и 8.6 будет рассказано о том, что коды MAC также используются в популярных протоколах, отвечающих за безопасность на транспортном и сетевом уровне.

Сертификация с применением открытых ключей

Важная область практического применения цифровых подписей — это **сертификация с открытым ключом**. Такая сертификация удостоверяет, что открытый ключ принадлежит конкретному лицу. Сертификация с открытым ключом применяется во многих популярных сетевых протоколах — в частности, в IPsec и SSL.

Чтобы заглянуть в глубь этой задачи, рассмотрим пример торговли через Интернет, а именно — классический «розыгрыш с пиццей». Предположим, Алиса занимается доставкой пиццы, принимая заказы по Интернету. Любитель пиццы Боб посылает Алисе открытым текстом сообщение, в котором указывает свой домашний адрес и сорт пиццы, который ему нужен. В это сообщение Боб также помещает цифровую подпись (то есть подписанный хэш оригинального сообщения), чтобы доказать Алисе, что сообщение исходит именно от него. Для того чтобы убедиться в подлинности этой подписи, Алиса может получить открытый ключ Боба (например, по электронной почте) и проверить цифровую подпись. Таким образом она убеждается в том, что это Боб, а не какой-либо юный шутник заполнил заказ.

Все это прекрасно до тех пор, пока не появляется Мэри. Как показано на рис. 8.13, она решает подшутить. Она посылает Алисе сообщение, в котором объявляет себя Бобом, указывает домашний адрес Боба и заказывает пиццу. Мэри также прилагает к заказу цифровую подпись, выполненную, естественно, при помощи собственного закрытого ключа. Алиса расшифровывает цифровую подпись открытым ключом Мэри (полагая, что это ключ Боба), и приходит к выводу, что сообщение действительно создано Бобом. Вероятно, Боб будет очень удивлен, когда ему доставят пиццу!

Как явствует из этого примера, чтобы шифрованием с открытым ключом можно было пользоваться, сетевые объекты (пользователи, бра-

узеры, маршрутизаторы и т. д.) должны знать наверняка, что у них есть открытый ключ именно того адресата, с которым они общаются. Например, когда Алиса связывается с Бобом, используя шифрование с открытым ключом, она должна быть уверена, что у нее на руках действительно открытый ключ Боба.

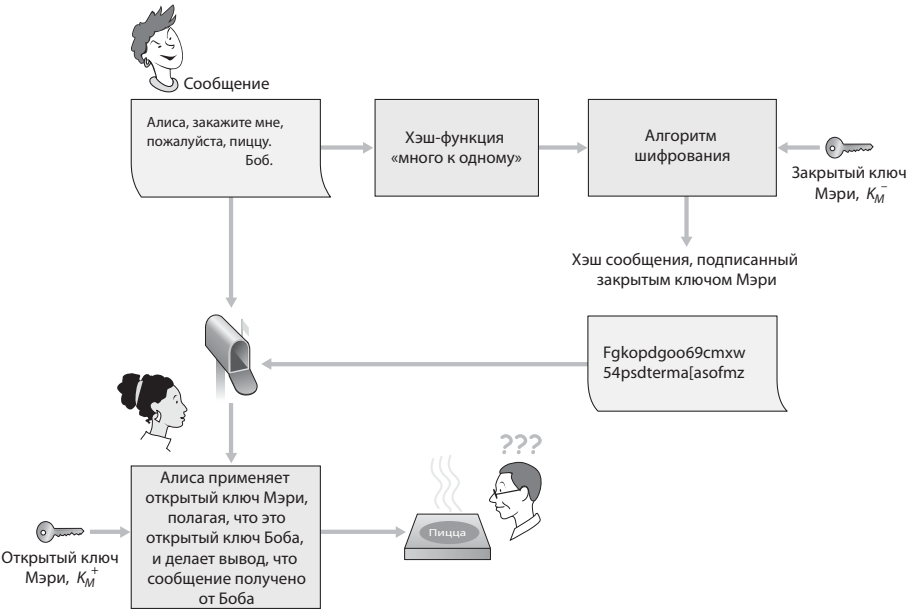


Рис. 8.13. Мэри выдает себя за Боба, применяя криптографию с открытым ключом

Привязка открытого ключа к определенным сетевым объектам, как правило, осуществляется **сертификационным центром** (Certification Authority, CA), работа которого заключается в подтверждении подлинности и выдаче сертификатов. Такой орган выполняет следующие задачи:

1. Сертификационный центр проверяет, является ли объект (человек, маршрутизатор и т. д.) тем, за кого он себя выдает. Конкретный способ аутентификации никак не регламентируется. Сетевые объекты должны доверять сертификационному центру выполнение строгой процедуры проверки подлинности. Например, если бы Мэри могла просто зайти в некий сертификационный центр, заявить «Я — Алиса» и получить сертификат, ассоциированный с личностью под именем «Алиса», тогда можно было бы особо не доверять сертификатам,

выданным этим центром. С другой стороны, некоторые пользователи склонны (или, наоборот, не склонны) больше полагаться на государственный сертификационный центр. Доверять «личности», ассоциированной с открытым ключом, можно только в той степени, в которой вы доверяете сертификационному центру и его методам проверки.

2. После проверки подлинности объекта сертификационный центр создает **сертификат**, который связывает открытый ключ объекта с идентификатором этого объекта. Сертификат содержит открытый ключ и глобально уникальный идентификатор владельца этого ключа (например, имя человека или IP-адрес). Сертификат снабжается цифровой подписью сертификационного центра. Эти этапы показаны на рис. 8.14.

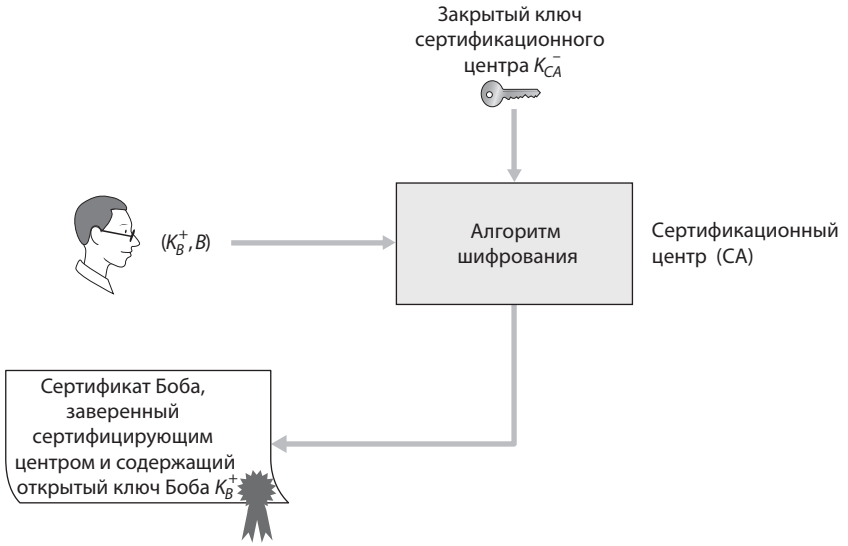


Рис. 8.14. У Боба есть свой закрытый ключ, сертифицированный ответственным органом

Посмотрим теперь, как сертификаты могут использоваться для борьбы с шутниками, заказывающими пиццу тем, кто ее не любит, а также с другими злоумышленниками. Когда Боб делает заказ, вместе с заказом он отсылает и свой сертификат, заверенный сертифицирующим органом. Алиса применяет открытый ключ этого сертифицирующего органа, чтобы проверить действительность сертификата Боба и извлечь принадлежащий Бобу открытый ключ.

Стандарты для сертификационных центров были разработаны международным союзом телекоммуникаций (International Telecommunications Union, ITU) и проблемной группой разработок для Интернета (Internet Engineering Task Force, IETF). Стандарт ITU X.509[251] описывает службу аутентификации, а также синтаксис сертификатов. В стандарте RFC 1422[446] представлена система управления ключами на основе сертификационных центров для безопасности электронной почты, пересылаемой через Интернет. Этот стандарт совместим с X.509, но он шире, так как устанавливает процедуры и соглашения для архитектуры управления ключами. Некоторые наиболее важные поля сертификата описаны в табл. 8.4.

Табл. 8.4. Некоторые поля сертификата открытого ключа стандартов X.509 и RFC 1422

Название поля	**Описание**
Версия	Номер версии спецификации стандарта X.509
Порядковый номер	Уникальный идентификатор сертификата, назначаемый сертификационным центром
Подпись	Алгоритм для подписи сертификата, используемый сертификационным центром
Имя эмитента	Идентификатор сертификационного центра, выпустившего данный сертификат, в формате DN (Distinguished Name — различимое имя)[542]
Срок действия	Время начала и окончания действия сертификата
Имя субъекта	Идентификатор объекта, чей открытый ключ ассоциирован с этим сертификатом в формате DN
Открытый ключ субъекта	Открытый ключ субъекта, а также обозначение алгоритма шифрования с открытым ключом (а также параметры алгоритма), используемого с этим ключом

8.4. Аутентификация конечной точки

Аутентификацией конечной точки называют процесс подтверждения чьей-либо личности в компьютерной сети — например передачу этой информации другому лицу по компьютерной сети. Люди могут убедиться в личности друг друга различными способами: мы распознаем лица при встрече, а голоса по телефону, сотрудники различных государственных служб могут проверить личность гражданина, сравнив его лицо с фотографией в паспорте.

В этом разделе мы поговорим об аутентификации при взаимодействии в сети. Здесь обсуждается аутентификация «активных» участ-

ников, в момент времени, когда протекает общение между двумя сторонами. Конкретный пример — аутентификация пользователя на почтовом сервере. Это немного отличается от проверки того, на самом ли деле сообщение, полученное в какой-то момент в прошлом, пришло от указанного в нем адресата — такая ситуация была рассмотрена в разделе 8.3.

При аутентификации по сети общающиеся стороны не могут полагаться на биометрическую информацию, такую как внешний вид или тембр голоса. В самом деле, как будет показано далее, аутентификация часто требуется сетевым элементам, таким как маршрутизаторы и процессы клиентов и серверов. Таким образом, аутентификация должна осуществляться исключительно на основе сообщений, которыми обмениваются участники **протокола аутентификации**. Как правило, протокол аутентификации запускается *прежде*, чем две общающиеся стороны запустят другой протокол (например, надежный протокол передачи данных, обмена таблицами маршрутизации или электронной почты). Сначала протокол аутентификации устанавливает аутентичность участников сеанса связи, и только после этого они получают возможность продолжать сеанс.

Как и при разработке протокола надежной передачи данных `rdt` (см. главу 3), мы разработаем несколько версий протокола аутентификации, который назовем **`ap`** (authentication protocol — протокол аутентификации), отмечая недостатки каждой версии. Если вам нравится подобный эволюционный подход к проектированию, рекомендуем работу Брянта[65] где излагается воображаемая история о разработчиках системы аутентификации для открытой сети, обнаруживающих в процессе все новые и новые проблемы.

Предположим, Алиса должна подтвердить свою личность Бобу.

8.4.1. Протокол аутентификации `ap1.0`

Пожалуй, самым простым протоколом аутентификации, который мы можем себе представить, является такой протокол, при котором Алиса просто посылает Бобу сообщение о том, что она Алиса (рис. 8.15). Недостаток такого протокола очевиден — у Боба нет способа убедиться в том, что тот, кто послал ему сообщение «Я Алиса», действительно Алиса. Точно такое же сообщение может быть послано Бобу злоумышленником.

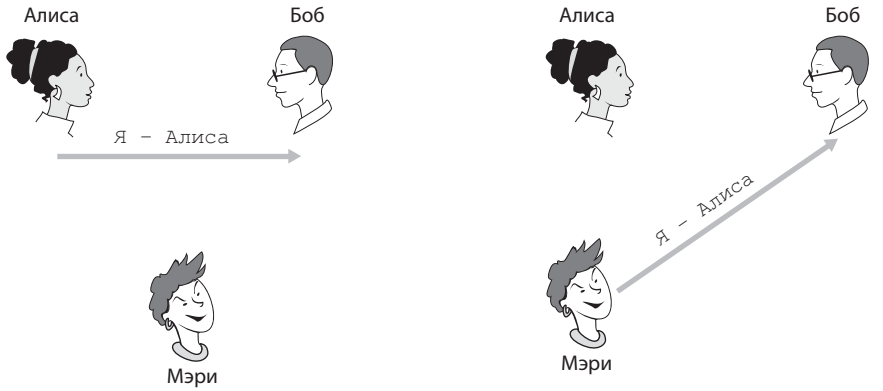


Рис. 8.15. Протокол аутентификации `ap1.0` и сценарий, в котором он отказывает

8.4.2. Протокол аутентификации `ap2.0`

В том случае, если у Алисы есть известный сетевой адрес, используемый ею при связи (например, IP-адрес), по нему Боб может попытаться аутентифицировать Алису. Ему потребуется убедиться, что адрес отправителя IP-дейтаграммы, содержащей аутентификационное сообщение, совпадает с известным адресом Алисы. Однако это может остановить только самого начинающего злоумышленника, но не защитит от целеустремленного студента, читающего эту книгу, а также от многих других!

Поскольку мы уже изучили сетевой и канальный уровни, мы знаем, что узнать сетевой адрес не так уж и трудно (например, если у злоумышленника есть доступ к исходным текстам операционной системы, что характерно, например, для Linux, он может построить собственное ядро операционной системы). В этом случае злоумышленник сможет создавать IP-дейтаграммы и указывать в них IP-адрес нужного источника (например, IP-адрес Алисы). Эти дейтаграммы злоумышленник станет передавать в сеть ближайшему маршрутизатору по протоколу канального уровня. С этого момента сеть послушно переправит Бобу дейтаграмму с фиктивным адресом отправителя. Этот метод, который иллюстрирует рис. 8.16, представляет собой хорошо известную разновидность атаки с поддельным адресом отправителя (так называемого спуфинга). Избежать подобной атаки можно, если настроить первый маршрутизатор в сети злоумышленника на отправку только тех дейтаграмм, которые содержат исходный IP-адрес злоумышленника[486]. Однако такой метод борьбы со злоумышленниками, к сожалению, применяется не везде.

Поэтому Бобу не следует полагаться на то, что администратор вычислительной сети злоумышленника (которым, кстати, может быть сам злоумышленник) установил подобную защиту от подделки IP-адресов.

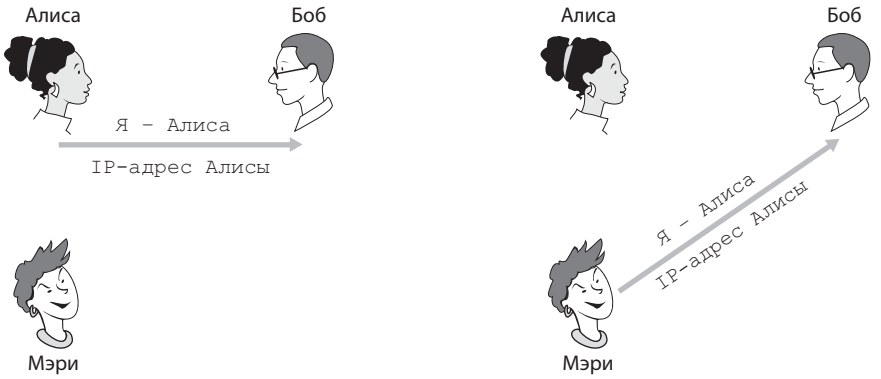


Рис. 8.16. Протокол аутентификации `ap2.0` и сценарий, в котором он отказывает

8.4.3. Протокол аутентификации `ap3.0`

Классический подход к аутентификации связан с использованием паролей. Пароль — это разделенный секрет, используемый одновременно аутентификатором и аутентифицируемым лицом. Аутентификация с паролем применяется в Gmail, Facebook, telnet, FTP и во многих других службах. Поэтому при работе по протоколу `ap3.0` Алиса посылает свой секретный пароль Бобу, как показано на рис. 8.17.

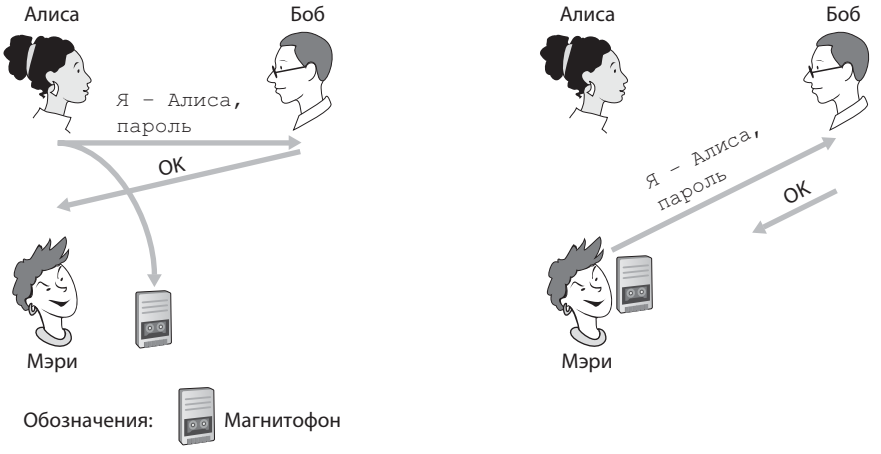


Рис. 8.17. Протокол аутентификации `ap3.0` и сценарий, в котором он отказывает

Поскольку пароли так широко применяются, мы можем предположить, что протокол аутентификации ap3.0 вполне надежен. Однако это не так. Взломать его совсем несложно. Злоумышленнику нужно всего лишь перехватить сообщение, посылаемое Алисой, чтобы узнать ее пароль. Если это кажется вам маловероятным — задумайтесь, что при использовании протокола telnet для работы на удаленном сервере пароль также пересылается в открытом виде. С любого компьютера, как подключенного к локальной сети telnet-клиента, так и с сервера этой сети, можно перехватывать все пакеты, посылаемые по этой сети, и, следовательно, украсть передаваемый в открытом виде пароль. Этот способ кражи давно и широко известен[268]. Подобная угроза вполне реальна, поэтому протокол аутентификации ap3.0, очевидно, не годится.

8.4.4. Протокол аутентификации ap3.1

Таким образом, очевидно, пересылаемый пароль необходимо зашифровывать. Тем самым мы не позволим злоумышленнику узнать пароль. Если предположить, что Алиса и Боб пользуются общим симметричным ключом K_{A-B}, тогда Алиса может зашифровать пароль и послать его Бобу вместе с собственным зашифрованным идентификационным сообщением «Я — Алиса». Затем Боб расшифрует пароль и, при условии, что он верный, убедится, что этот пакет действительно послан Алисой. Боб уверен в том, что это Алиса, так как она не только знает пароль, но также знает общий секретный ключ, необходимый для шифрования пароля. Назовем этот протокол ap3.1.

Хотя протокол аутентификации ap3.1 действительно не позволяет никому узнать пароль Алисы, он не решает проблемы аутентификации, так как злоумышленнику все равно, в каком виде, зашифрованном или открытом, перехватить пароль Алисы, а потом повторить его в своем пакете, посланном Бобу — то есть притвориться Алисой. Подобная разновидность атаки называется **атакой повторного воспроизведения**. Таким образом, ситуация практически не отличается от случая использования протокола аутентификации ap3.0, продемонстрированного на рис. 8.17.

8.4.5. Протокол аутентификации ap4.0

Отказ, проиллюстрированный на рис. 8.17, обусловлен тем, что Боб не смог отличить аутентификацию Алисы от последовавшей атаки по-

вторного воспроизведения. В данном случае Боб не мог определить, участвовала ли в сеансе связи Алиса (то есть, действительно ли она находилась на противоположном конце провода), либо это было всего лишь повторное воспроизведение записанных аутентификаций Алисы, перехваченных злоумышленником. Очень (*очень*!) наблюдательный читатель вспомнит, что TCP-протокол с тройным рукопожатием был призван решить такую же проблему. Серверная сторона TCP-соединения не хочет принимать соединения, если полученный SYN-сегмент оказывается устаревшей (повторно переданной) копией сегмента SYN из более раннего соединения. Как сервер при TCP-соединении определяет, активен ли на самом деле нужный ему клиент? Он выбирает начальный номер последовательности, которая не использовалась достаточно давно, посылает его клиенту, а затем ожидает, пока клиент пришлет в ответ ACK-сегмент с таким же номером. Здесь мы вновь задействуем эту идею, но уже для аутентификации.

Число, используемое протоколом всего один раз, называют **одноразовым номером** (nonce). Наш протокол `ap4.0` использует одноразовые номера следующим образом.

1. Алиса отправляет Бобу сообщение «`Я — Алиса`».
2. Боб выбирает одноразовый номер, *R*, и посылает его Алисе.
3. Алиса зашифровывает одноразовый номер с помощью симметричного секретного ключа K_{A-B}, совместно используемого Алисой и Бобом, и посылает зашифрованный одноразовый номер $K_{A-B}(R)$ обратно Бобу. Как и в протоколе `ap3.1`, Боб понимает, что полученное им сообщение послано Алисой, потому что отправителю этого сообщения известен общий для Алисы и Боба ключ K_{A-B}, а номер гарантирует, что Алиса действительно участвует в сеансе.
4. Боб расшифровывает полученное сообщение. Если расшифрованный одноразовый номер совпадает с тем, который он послал Алисе, то аутентификация Алисы считается успешной.

Работа протокола `ap4.0` проиллюстрирована на рис. 8.18. Пользуясь одноразовым значением *R*, а затем проверяя возвращаемое значение $K_{A-B}(R)$, Боб может быть уверен не только в том, что общается именно с Алисой (поскольку она знает значение секретного ключа, необходимое, чтобы зашифровать *R*), но и в том, что Алиса активна в настоящий момент (так как она зашифровала одноразовый номер *R*, только что созданный Бобом).

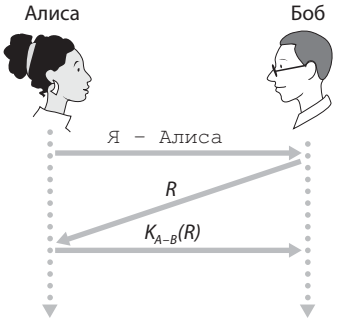


Рис. 8.18. Протокол аутентификации `ap4.0` и сценарий, в котором он отказывает

Применение одноразовых номеров и шифрования симметричными ключами закладывает основу протокола `ap4.0`. Разумеется, возникает вопрос: можно ли использовать одноразовые номера при шифровании открытым ключом (а не с симметричными ключами) для решения проблемы аутентификации? На этот вопрос мы попытаемся ответить в задачах, приведенных в конце данной главы.

8.5. Обеспечение безопасности электронной почты

В предыдущих разделах мы исследовали фундаментальные вопросы сетевой безопасности — шифрование симметричными ключами и открытым ключом, аутентификацию конечной точки, распределение ключей, целостность сообщений, цифровые подписи. Теперь поговорим о том, как эти инструменты применяются для обеспечения безопасности в Интернете.

Интересно отметить, что сервисы по обеспечению безопасности можно реализовать на любом из четырех верхних уровней стека Интернет-протоколов. Когда безопасность обеспечивается для конкретного протокола прикладного уровня, приложение, использующее этот протокол, будет задействовать один или несколько сервисов для повышения безопасности — они могут быть связаны с конфиденциальностью, аутентификацией или целостностью данных. Когда безопасность обеспечивается для протокола транспортного уровня, все приложения, использующие этот протокол, располагают такими сервисами безопасности. Когда безопасность обеспечивается на сетевом уровне на основе межхостовой связи, все сегменты транспортного уровня (и, следовательно, все данные прикладного уровня) могут пользоваться сервисами безопасности сете-

вого уровня. Когда безопасность обеспечивается на канальном уровне, сервисы безопасности, предоставляемые в этом канале, оказываются доступны для всех проходящих по нему кадров.

В разделах 8.5-8.8 мы поговорим о том, как инструменты обеспечения безопасности применяются на прикладном, транспортном, сетевом и канальном уровне. Следуя общей структуре этой книги, мы начнем обзор с верхней части стека протоколов, а именно — с прикладного уровня. Чтобы исследовать безопасность прикладного уровня на конкретном примере, возьмем за основу приложение для работы с электронной почтой. Затем двинемся вниз по стеку протоколов. Мы исследуем следующие протоколы: SSL (обеспечивающий безопасность на транспортном уровне), IPsec (безопасность на сетевом уровне), а также протокол IEEE 802.11, связанный с безопасностью в беспроводных локальных сетях.

Может возникнуть вопрос: а почему безопасность обеспечивается сразу на нескольких уровнях Интернета? Почему, например, не реализовать такие функции на сетевом уровне и не удовлетвориться этим? Существует два ответа на этот вопрос. Во-первых, хотя при реализации функций безопасности на сетевом уровне мы действительно добились бы всеобъемлющего эффекта (то есть, зашифровали бы все данные в дейтаграммах и все сегменты транспортного уровня, аутентифицировав все исходные IP-адреса), это не гарантировало бы нам безопасности на уровне отдельных пользователей. Например, сайт Интернет-магазина не мог бы полагаться на безопасность IP-уровня и удостоверять личность пользователя, приобретающего товары в этом магазине. Соответственно, требуется реализовать функции безопасности и на верхних уровнях, а широкий охват обеспечивать уже на нижних. Во-вторых, новые Интернет-службы, и службы безопасности в том числе, как правило, бывает проще развертывать на верхних уровнях стека протоколов. Ожидая, пока службы обеспечения безопасности начнут широко внедряться на сетевом уровне (что, вероятно, произойдет еще через много лет), многие разработчики приложений просто решают конкретные проблемы, самостоятельно реализуя функции безопасности в своих программах. Классический пример такого рода — программа PGP (аббревиатуру можно расшифровать как «Просто Замечательная Приватность»), обеспечивающая защищенный обмен электронной почтой. О такой переписке мы поговорим ниже в этой главе. Для PGP достаточно лишь клиентского и серверного приложений, и она является одной из первых технологий обеспечения безопасности, получившей широкое распространение в Интернете.

8.5.1. Безопасная электронная почта

В этом разделе мы попытаемся создать безопасную электронную почту, используя методы, описанные в разделах 8.2 и 8.3. Мы будем развивать наш проект постепенно, на каждом этапе добавляя новые службы безопасности. При разработке этой системы мы продолжим опираться на романтический сюжет об Алисе и Бобе, завязавшийся в разделе 8.1. Пусть Алиса хочет послать Бобу письмо по электронной почте, а Мэри жаждет вмешаться.

Прежде чем перейти к проектированию безопасной электронной почты для Алисы и Боба, мы должны сначала определить, какие функции безопасности необходимы им в первую очередь. Прежде всего, им требуется *конфиденциальность*. Как уже отмечалось в разделе 8.1, ни Алиса, ни Боб, не хотят, чтобы Мэри прочитала письмо Алисы. Кроме того, Алисе и Бобу нужна *аутентификация отправителя*. В частности, получив от Алисы сообщение «`Я не люблю тебя больше. Я не хочу тебя видеть. Когда-то твоя, Алиса`», Боб, разумеется, хотел бы удостовериться в том, что это сообщение пришло от Алисы, а не от Мэри. Еще нашим возлюбленным требуется обеспечить *целостность сообщения*, то есть нужна уверенность в том, что отправленное Алисой сообщение не изменено по дороге. Наконец, электронная почта должна обеспечивать *аутентификацию получателя*, то есть Алиса хочет быть уверенной в том, что она действительно посылает письмо Бобу, а не кому-либо еще, кто выдает себя за Боба (например, Мэри).

Итак, начнем с решения первоочередного требования к безопасной системе электронной почты, а именно обеспечения конфиденциальности. Наиболее простой способ достичь этой цели состоит в шифровании Алисой сообщений при помощи алгоритма с симметричными ключами (например, DES или AES). Получив письмо, Боб должен сначала расшифровать его. Как отмечалось в разделе 8.2, если длина симметричного ключа достаточно велика и ключ известен только Алисе и Бобу, то кому-либо еще (например, Мэри) прочитать зашифрованное сообщение будет весьма затруднительно. Хотя такой метод довольно прост, у него есть существенный недостаток, о котором мы тоже говорили, — трудно передать ключ так, чтобы его копии были только у Алисы и Боба. Таким образом, мы, естественно, рассмотрим альтернативный подход, а именно алгоритм шифрования с открытым ключом (например, RSA). При шифровании с открытым ключом Боб публикует свой открытый ключ (например, на сервере открытых ключей или на своей личной веб-странице), Алиса зашифровывает свое сообщение открытым ключом Боба

и посылает зашифрованное сообщение по адресу электронной почты Боба. Получив сообщение, Боб расшифровывает его своим закрытым ключом. Если Алиса уверена в том, что используемый ею открытый ключ действительно принадлежит Бобу, а длина этого ключа достаточно велика, подобный метод превосходно обеспечивает требуемую конфиденциальность. Однако один из его недостатков заключается в том, что шифрование с открытым ключом относительно неэффективно, особенно для длинных сообщений.

Чтобы решить проблему неэффективности, будем использовать сеансовый ключ (см. раздел 8.2.2). Во-первых, Алиса случайным образом выбирает симметричный сеансовый ключ K_S, во-вторых, зашифровывает этим ключом K_S свое сообщение m, в-третьих, зашифровывает сеансовый ключ K_S открытым ключом Боба K_B^+, в-четвертых, формирует из всех зашифрованных данных один пакет и, в-пятых, посылает этот пакет по адресу электронной почты Боба. Эти действия иллюстрирует рис. 8.19. (На этом и на последующих рисунках плюс обозначает операцию конкатенации, а минус — противоположную операцию, то есть разъединение.) Получив сообщение, Боб, во-первых, с помощью своего закрытого ключа K_B^- получает сеансовый ключ K_S, во-вторых, с помощью сеансового ключа сеанса K_S расшифровывает сообщение m.

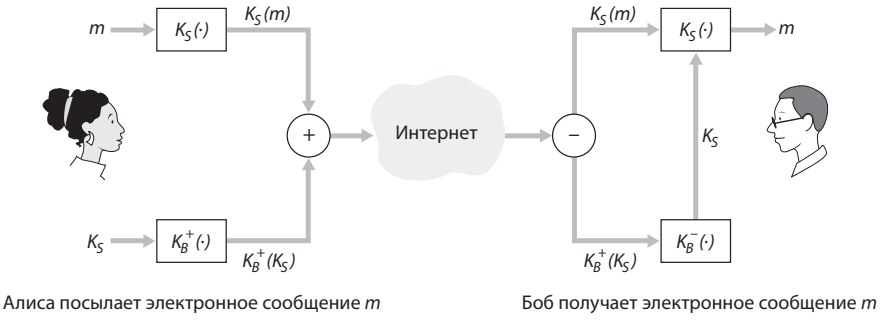


Рис. 8.19. Алиса применяет симметричный сеансовый ключ K_S для отправки тайного сообщения Бобу

Итак, мы спроектировали безопасную систему электронной почты, обеспечивающую конфиденциальность. Сделаем теперь другую систему, обеспечивающую аутентификацию отправителя и целостность сообщения. Предположим на время, что Алисе и Бобу более не требуется конфиденциальность (они готовы поделиться своими чувствами со всеми!), и их интересуют только аутентификация отправителя и целостность сообщения. Для решения данной задачи мы воспользуемся цифровыми

подписями и хэшами сообщений, о которых рассказывалось в разделе 8.3. Итак, Алиса, во-первых, применяет к сообщению *m* хэш-функцию *H* (например, MD5) и получает хэш сообщения, во-вторых, подписывает результат хэш-функции своим закрытым ключом K_A^-, в-третьих, объединяет исходное (незашифрованное) сообщение с подписью, формируя пакет, и, в-четвертых, отправляет этот пакет по адресу электронной почты Боба. Получив сообщение, Боб, во-первых, применяет открытый ключ Алисы K_A^+ к подписи хэша сообщения и, во-вторых, сравнивает результат этой операции с хэшем *H* сообщения, который он вычисляет сам. Эти действия проиллюстрированы на рис. 8.20. Как отмечалось в разделе 8.3, если результаты этих двух вычислений совпадают, Боб может быть уверен, что сообщение пришло от Алисы и оно не модифицировано.

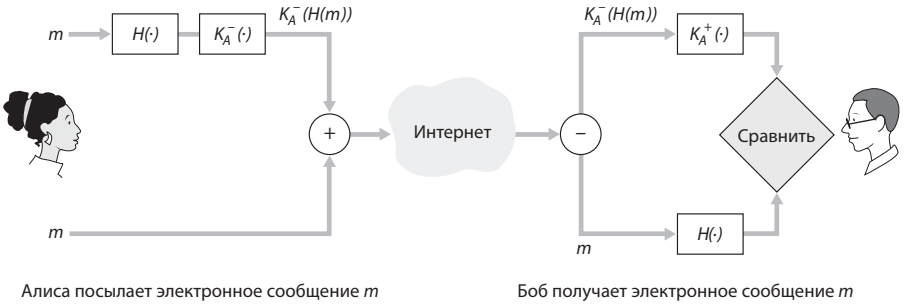


Рис. 8.20. При помощи хэш-функций и цифровых подписей обеспечивается аутентификация отправителя и целостность сообщения

Теперь попробуем объединить в одной системе службы обеспечения конфиденциальности, аутентификации отправителя *и* целостности сообщения. Это делается путем объединения процедур, описанных на рис. 8.19 и 8.20. Сначала Алиса создает предварительный пакет (см. рис. 8.20) состоящий из исходного сообщения и подписанного цифровой подписью хэша этого сообщения. Затем весь предварительный пакет зашифровывается как единое сообщение и формируется новый пакет, который и отправляется Бобу. Вместе эти действия иллюстрирует рис. 8.21. Получив пакет, Боб сначала применяет к нему действия, показанные на рис. 8.19, а затем — на рис. 8.20. Очевидно, что такая схема обеспечивает конфиденциальность, аутентификацию отправителя и целостность сообщения. Обратите внимание, что Алиса дважды использует шифрование с открытым ключом: один раз со своим закрытым ключом и второй раз с открытым ключом Боба. Соответственно, Боб также использует шифрование с открытым ключом дважды: один раз со своим закрытым ключом

и второй раз с открытым ключом Алисы. Показанная на рис. 8.21 схема безопасной электронной почты, вероятно, обеспечит достаточный уровень безопасности для большинства пользователей. Но нам необходимо решить еще один важный вопрос. В данной схеме Алиса должна получить открытый ключ Боба, а Боб — открытый ключ Алисы. Как отмечалось в разделе 8.3, для безопасного распространения открытых ключей их обычно *удостоверяют* специальные центры сертификации.

ИСТОРИЯ

Фил Циммерман и PGP

Филипп Циммерман — автор программы PGP (Pretty Good Privacy, что можно перевести как «Просто замечательная приватность»). За это достижение Фил находился под следствием в течение трех лет, поскольку когда эта программа стала распространяться по всему миру, власти США усмотрели в этом нарушение действовавших ограничений на распространение криптографического ПО. Действительно, еще в 1991 году PGP была выложена в общий доступ как свободное ПО. После выхода условно-бесплатной версии PGP кто-то разместил в Интернете и ее, а затем программу сразу же начали скачивать иностранные граждане. По федеральному законодательству США криптографические программы считаются военным снаряжением и не могут экспортироваться за рубеж.

Несмотря на все препятствия — отсутствие финансирования и, соответственно, штатных разработчиков, компании, которая вела бы проект, — а также вопреки активному вмешательству правительства PGP стала самой распространенной в мире программой для шифрования электронной почты. Как ни странно, само правительство США могло ненамеренно поспособствовать распространению PGP в силу всеобщего внимания к делу Циммермана.

Дело было закрыто в начале 1996 года. Объявление об этом было восторженно воспринято в кругу Интернет-активистов. Дело Циммермана стало символом неравной борьбы невиновного человека с государственной машиной за свои права. Отказ правительства от продолжения этого дела пришелся весьма кстати, отчасти потому, что Конгресс США одобрил программу цензуры в Интернете, а ФБР выступило с инициативой по расширению полномочий для правительственной слежки в сети.

Когда дело было закрыто, Циммерман основал компанию PGP Inc., которая была приобретена фирмой Network Associates в декабре 1997 года. В настоящее время Циммерман работает независимым консультантом в области криптографии.

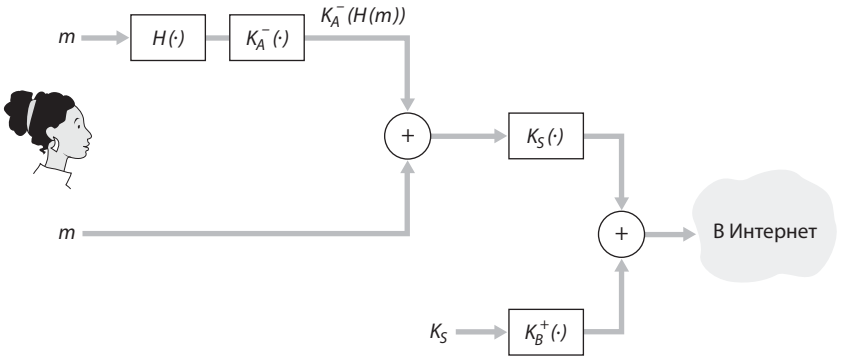


Рис. 8.21. Алиса использует криптографию с симметричными ключами, криптографию с открытым ключом, хэш-функцию и цифровую подпись, чтобы обеспечить секретность, аутентификацию отправителя и целостность сообщения

8.5.2. PGP

Программа **PGP (Pretty Good Privacy)** была написана Филом Циммерманом в 1991 году. В ней применяется схема шифрования электронной почты, которая *де-факто* сегодня является стандартом. Сайт программы ежемесячно обслуживает для различных пользователей из 166 стран мира более миллиона экземпляров страниц[391]. В широком доступе имеется ряд версий PGP; вы можете найти программу PGP для вашей любимой платформы, а также почитать множество интересных материалов на международной домашней странице PGP[391]. Отдельно обращаем ваше внимание на статьи самого автора программы[689]. В сущности, структура PGP именно такова, как показано на рис. 8.21. В зависимости от версии программа использует алгоритм MD5 или SHA для вычисления хэш-функции сообщения; CAST, тройной DES или IDEA для шифрования симметричными ключами и RSA для шифрования открытым ключом.

При установке программа PGP создает для пользователя пару ключей: открытый и закрытый. Открытый ключ пользователь может опубликовать на своем сайте, либо положить на сервере открытых ключей. Закрытый ключ защищается паролем. Для доступа к ключу пользователь должен всякий раз указывать этот пароль. Программа PGP позволяет пользователю снабдить сообщение цифровой подписью, зашифровать его, либо одновременно применить и цифровую подпись, и шифрование. На рис. 8.22 показано сообщение с подписью PGP. Оно следует за MIME-заголовком. Закодированные данные, содержащиеся в этом сообщении: $K_A^-(H(m))$, то есть, закодирован хэш сообщения, снаб-

женный цифровой подписью. Как было указано выше, чтобы Боб мог удостовериться в целостности сообщения, ему нужно знать открытый ключ Алисы.

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
Bob:
Can I see you tonight?
Passionately yours, Alice
-----BEGIN PGP SIGNATURE-----
Version: PGP for Personal Privacy 5.0
Charset: noconv
yhHJRHhGJGhgg/12EpJ+lo8gE4vB3mqJhFEvZP9t6n7G6m5Gw2
-----END PGP SIGNATURE-----
```

Рис. 8.22. Подписанное PGP-сообщение

На рис. 8.23 показано секретное PGP-сообщение. Это сообщение также следует за MIME-заголовком. Разумеется, в секретное электронное сообщение не включается та информация, которая была записана открытым текстом. Когда отправителю (в данном случае — Алисе) требуется соблюсти как конфиденциальность, так и целостность данных, PGP будет содержать такое сообщение, как на рис. 8.23, в таком сообщении, как на рис. 8.22.

```
-----BEGIN PGP MESSAGE-----
Version: PGP for Personal Privacy 5.0
u2R4d+/jKmn8Bc5+hgDsqAewsDfrGdszX68liKm5F6Gc4sDfcXyt
RfdS10juHgbcfDssWe7/K=lKhnMikLo0+1/BvcX4t==Ujk9PbcD4
Thdf2awQfgHbnmKlok8iy6gThlp
-----END PGP MESSAGE
```

Рис. 8.23. Секретное PGP-сообщение

PGP также предоставляет механизм для сертификации открытых ключей, но он заметно отличается от процедуры, применяемой в обычных сертифицирующих центрах. Открытые ключи PGP удостоверяются при помощи механизма, называемого *сетью доверия*. Алиса может самостоятельно сертифицировать любую пару ключ/значение, если

верит, что они действительно образуют пару. Кроме того, PGP позволяет Алисе заявить, что она доверяет другому пользователю, который ручается за подлинность других ключей. Некоторые пользователи PGP даже собираются на специальные вечеринки, где подписывают ключи друг другу. Люди собираются вместе, обмениваются открытыми ключами и удостоверяют их своими закрытыми ключами.

8.6. Защита TCP-соединений при помощи технологии SSL

В предыдущем разделе мы рассмотрели на примере электронной почты, как могут использоваться на прикладном уровне обсуждавшиеся в начале этой главы механизмы обеспечения безопасности — такие как шифрование, аутентификация, распределение ключей, целостность данных и цифровые подписи. В этом разделе мы перейдем в стеке протоколов на уровень ниже и исследуем, как криптография позволяет оснастить функциями безопасности протокол TCP, обеспечивая для него конфиденциальность, целостность данных и аутентификацию конечной точки. Такая усовершенствованная версия TCP называется **SSL** (Secure Socket Layer — **уровень защищенных сокетов**). Несколько улучшенной является 3-я версия SSL, протокол **TLS** (Transport Layer Security — **безопасность на транспортном уровне**). Протокол TLS стандартизирован институтом IETF[540].

Разработка протокола SSL началась в компании Netscape, но основные идеи о защите протокола TCP зародились еще до того, как Netscape приступила к этой работе[672]. С самого своего появления технология SSL использовалась весьма активно. SSL поддерживается во всех популярных браузерах и на веб-серверах, а также применяется практически на всех крупнейших коммерческих сайтах в Интернете (в том числе, Amazon, eBay, Yahoo!, MSN и т. д.). Каждый год на SSL тратятся десятки миллиардов долларов. На самом деле, если вам когда-либо доводилось приобретать какие-то товары по Интернету при помощи банковской карты, то можно не сомневаться, что все общение между вашим браузером и сервером осуществлялась по SSL. Легко определить, когда в вашем браузере используется SSL: в таком случае электронный адрес веб-страницы начинается с https, а не с http.

Чтобы убедиться в необходимости SSL, рассмотрим типичный сценарий Интернет-коммерции. Боб путешествует по Интернету и попада-

ет на сайт компании Alice Incorporated, продающей парфюмерию. На сайте Боб обнаруживает форму, в которой нужно указать требуемое ему количество товара и марку духов, свой адрес и номер кредитной карты для оплаты. Боб вводит нужную информацию, щелкает мышью на кнопке отправки данных, а затем ожидает получения (например, по обычной почте) купленных им товаров. Кроме того, он рассчитывает увидеть результат покупки в следующем отчете о состоянии свой кредитной карты. Все это звучит прекрасно, но, если не предпринять специальных мер, таких как шифрование и аутентификация, Боба могут ожидать несколько сюрпризов.

- Если не соблюдается конфиденциальность (не используется шифрование), то злоумышленник может перехватить заказ и, таким образом, получить информацию о кредитной карте Боба. Впоследствии злоумышленник сможет совершать покупки за счет Боба.

- При несоблюдении целостности данных злоумышленник может изменить заказ Боба, купив в десять раз больше флакончиков с духами, чем требовалось

- Наконец, если не применяется аутентификация сервера, то на сайте может красоваться знаменитый логотип компании Alice Incorporated, но в действительности это будет замаскированный сайт злоумышленника. Тот может взять деньги Боба и сбежать. Или совершить кражу персональных данных, завладев такой информацией, как имя Боба, его адрес и реквизиты банковской карты.

Технология SSL помогает решать подобные проблемы, дополняя TCP такими возможностями, как конфиденциальность, целостность данных, аутентификация сервера и аутентификация клиента.

Протокол SSL часто применяется для обеспечения безопасности транзакций, совершаемых по протоколу HTTP. Правда, поскольку SSL защищает TCP, он может использоваться любым приложением, работающим по протоколу TCP. SSL предоставляет простой интерфейс программирования приложений (API) с сокетами, который похож на аналогичный API протокола TCP. Когда в приложении предполагается использовать SSL, в программу включаются классы и библиотеки протокола SSL. Как продемонстрировано на рис. 8.24, хотя SSL технически располагается на прикладном уровне, с точки зрения разработчика это транспортный протокол, обеспечивающий как передачу данных по TCP, так и службы обеспечения безопасности.

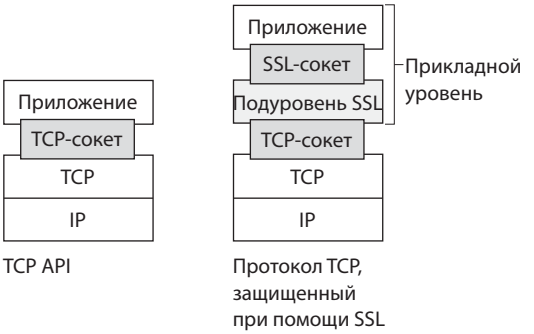


Рис. 8.24. Хотя SSL технически располагается на прикладном уровне, с точки зрения разработчика это протокол транспортного уровня

8.6.1. Общая картина

Начнем с описания упрощенной версии SSL, которая позволит нам приобрести общее представление о том, *что и как* функционирует в SSL. Такую версию назовем «почти-SSL». Познакомившись с почти-SSL, в следующем разделе мы опишем реальный протокол SSL, уточнив различные детали. В работе протокола почти-SSL выделяется три этапа: *рукопожатие* (также называемое «квитированием»), *формирование ключа* и *передача данных*. Ниже мы опишем три этих этапа сеанса связи между клиентом (Бобом) и сервером (Алисой). Будем исходить из того, что у Алисы есть пара ключей — закрытый и открытый, — а также сертификат, связывающий ее идентификационные данные с открытым ключом.

Рукопожатие

На этапе рукопожатия Боб должен (а) установить TCP-соединение с Алисой, (б) убедиться, что Алиса — это *действительно* Алиса и (в) послать Алисе главный секретный ключ (master secret key), который в дальнейшем и Алиса, и Боб будут использовать для генерации всех симметричных ключей, что потребуются им для SSL-сеанса. Три эти этапа проиллюстрированы на рис. 8.25. Обратите внимание: когда соединение по протоколу TCP установлено, Боб отправляет Алисе «приветственное» сообщение (hello). На это сообщение Алиса отвечает своим сертификатом, в котором содержится ее открытый ключ. Как рассказано в разделе 8.3, поскольку сертификат удостоверяется сертифицирующим органом, Боб может быть уверен, что открытый ключ из

сертификата действительно принадлежит Алисе. Затем Боб генерирует главный секретный ключ (MS), который в дальнейшем будет использоваться только на протяжении данного SSL-сеанса. Боб шифрует главный секретный ключ открытым ключом Алисы, создавая таким образом зашифрованный главный секретный ключ (EMS), после чего отсылает EMS Алисе. Алиса расшифровывает EMS своим закрытым ключом и получает MS. После этого этапа Боб и Алиса (только они!) знают главный секретный ключ, действующий в рамках данного SSL-сеанса.

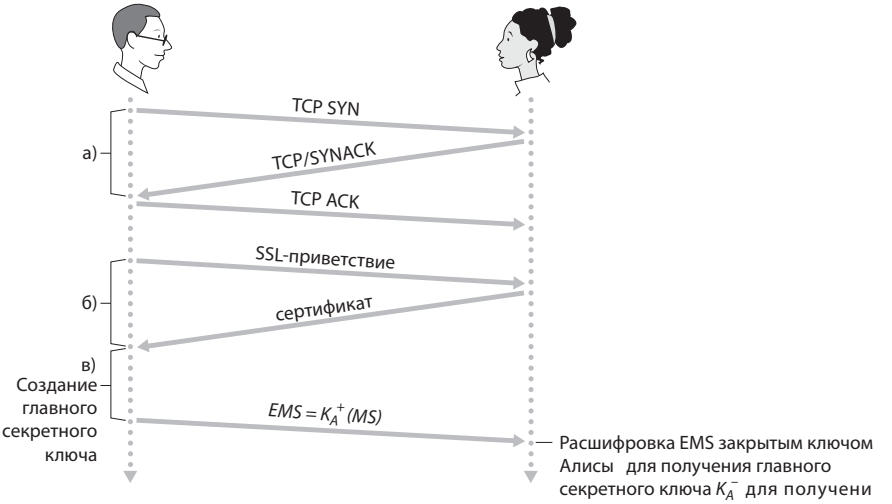


Рис. 8.25. Рукопожатие по упрощенной версии протокола SSL, начинающееся с TCP-соединения

Формирование ключа

Теперь главный секретный ключ есть и у Боба, и у Алисы. В принципе, его можно использовать как симметричный сеансовый ключ при всех последующих операциях шифрования и проверки целостности данных. Однако более безопасна другая практика, при которой Алиса и Боб должны использовать разные криптографические ключи. Итак, и Алиса, и Боб применяют главный секретный ключ, чтобы сгенерировать по четыре ключа:

- E_B — сеансовый ключ для шифрования данных, пересылаемых от Боба к Алисе
- M_B — сеансовый MAC-ключ для данных, пересылаемых от Боба к Алисе

- E_A — сеансовый ключ для шифрования данных, пересылаемых от Алисы к Бобу
- M_A — сеансовый MAC-ключ для данных, пересылаемых от Алисы к Бобу

И Алиса, и Боб обзаводятся четырьмя ключами, сформированными на основе одного главного секретного ключа. Для этого достаточно просто разрезать главный секретный ключ на четыре части. Правда, в *реальном* протоколе SSL ситуация несколько сложнее — об этом мы поговорим ниже. На последнем этапе фазы формирования ключа у Алисы и Боба будет по четыре ключа. Два ключа шифрования будут применяться для шифрования данных, а два MAC-ключа — для проверки целостности данных.

Передача данных

Теперь, когда Алиса и Боб используют два одинаковых набора по четыре сеансовых ключа (E_B, M_B, E_A и M_A), они могут приступать к обмену защищенными данными по протоколу TCP. Поскольку TCP — это протокол потока байтов, было бы логично использовать SSL для шифрования данных приложения «на лету», а затем «на лету» передать зашифрованные данные по TCP. Но если бы мы так поступили, где мы поместили бы код MAC для проверки целостности данных? Разумеется, мы не хотим дожидаться завершения TCP-сеанса, чтобы лишь после этого проверить целостность всех данных Боба, которые он успел передать за этот сеанс! Чтобы устранить эту проблему, SSL разбивает поток данных на *записи*, прикрепляет к каждой записи код MAC для проверки целостности, а затем шифрует пару *запись+MAC*. Для создания MAC Боб вводит информацию записи вместе с ключом M_B в хэш-функцию, как это было описано в разделе 8.3. Чтобы зашифровать пакет, состоящий из записи и MAC, Боб применяет свой сеансовый ключ шифрования E_B. Затем этот зашифрованный пакет передается протоколу TCP и для транспортировки по Интернету.

Хотя такой подход весьма надежен, он все-таки уязвим, если требуется обеспечить целостность данных во всем потоке байтов. В частности, предположим, что Мэри предпринимает атаку посредника и получает возможность вставлять, удалять и заменять сегменты в потоке информации, передаваемом по протоколу TCP между Алисой и Бобом. Например, Мэри может перехватить два сегмента, отосланных Бобом, поменять их местами, исправить номера последовательности TCP (ко-

торые не шифруются), а затем отослать два этих сегмента Алисе — уже в обратном порядке. Если предположить, что в каждом TCP-сегменте содержится ровно одна запись, то вот как Алиса после этого будет обрабатывать сегменты:

1. Протокол, работающий на машине Алисы, «ничего не заметит» и передаст две записи на подуровень SSL.
2. SSL на компьютере Алисы дешифрует две записи
3. SSL на компьютере Алисы воспользуется MAC-кодами обеих записей, чтобы проверить целостность данных, содержащихся в этих записях
4. Затем SSL передаст дешифрованные байтовые потоки двух записей на прикладной уровень, однако полный поток байт, полученный Алисой, будет построен в неверном порядке — поскольку Мэри поменяла местами записи!

Можете самостоятельно проработать подобные сценарии — случаи, в которых Мэри удаляет или повторно воспроизводит сегменты.

Как вы уже, вероятно, догадались, такая проблема решается при помощи порядковых номеров. Вот как это делается в SSL. У Боба работает счетчик порядковых номеров, который начинается с нуля и увеличивается на единицу с каждой SSL-записью, отсылаемой Бобом. Боб не включает в саму запись порядковый номер, но при вычислении кода MAC он использует порядковый номер в этих расчетах. Следовательно, код MAC теперь включает в себя код MAC плюс MAC-ключ плюс сеансовый ключ M_B, *плюс текущий порядковый номер*. Алиса отслеживает порядковые номера пакетов Боба и может проверить целостность данных, включив соответствующий номер в вычисление кода MAC. Такой способ использования порядковых номеров SSL не позволяет Мэри совершить атаку посредника, переупорядочить или повторно воспроизвести сегменты (Почему?)

SSL-запись

SSL-запись (такая же, как и запись в нашем варианте с «почти SSL») показана на рис. 8.26. Запись состоит из поля типа, поля версии, поля длины, поля данных и поля MAC. Обратите внимание на то, что первые три поля не шифруются. Поле типа указывает, применяется ли данное сообщение для рукопожатия, либо оно содержит информацию приложения. Оно также требуется для закрытия SSL-соединения, о чем мы

поговорим ниже. Протокол SSL на конечном узле получает поле длины для извлечения SSL-записей из байтового потока, поступающего по TCP. Поле версии должно быть понятно без объяснения.

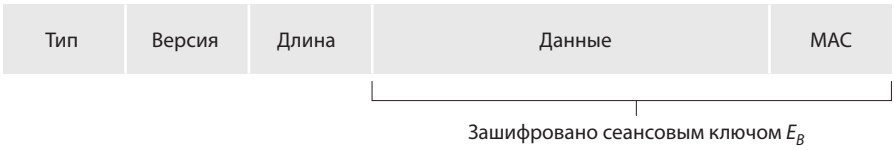


Рис. 8.26. Формат записи для SSL

8.6.2. Детализированная картина

В предыдущем разделе была рассмотрена модель протокола, которую мы назвали «Почти-SSL». Она была призвана пояснить, что и как функционирует в SSL. Теперь, имея общее представление о SSL, мы можем изучить этот протокол более внимательно и разобраться в его существенных деталях. Параллельно с чтением этого раздела рекомендуем выполнить лабораторную работу Wireshark, которая содержится в дополнительных материалах к этой главе.

SSL-рукопожатие

При связи по протоколу SSL Алиса и Боб не обязаны использовать конкретный алгоритм шифрования с симметричными ключами, открытым ключом или определенным MAC. Вместо этого SSL позволяет Алисе и Бобу оговорить криптографические алгоритмы в самом начале SSL-сеанса, на этапе рукопожатия. Кроме того, на этапе рукопожатия Алиса и Боб посылают друг другу одноразовые номера, которые применяются при создании сеансовых ключей (E_B, M_B, E_A и M_A). Реальное рукопожатие по протоколу SSL состоит из следующих этапов:

1. Клиент отсылает список криптографических алгоритмов, которые поддерживает, а также клиентский одноразовый номер
2. Сервер выбирает из этого списка симметричный алгоритм (например, AES), алгоритм открытого ключа (например, RSA с конкретной длиной ключа) и MAC-алгоритм. Затем сервер отсылает клиенту выбранные варианты, а вместе с ними — сертификат и серверный одноразовый номер.
3. Клиент проверяет сертификат, извлекает из него открытый ключ сервера, генерирует предварительный главный секретный ключ

(PMS), шифрует PMS открытым ключом сервера, после чего отсылает зашифрованный PMS на сервер.

4. Воспользовавшись одной и той же функцией произведения ключа, которая указана в стандарте SSL, клиент и сервер независимо друг от друга вычисляют главный секретный ключ (MS) на основании предварительного главного секретного ключа и одноразовых номеров. Затем главный секретный ключ разрезается на четыре части для получения двух ключей шифрования и двух MAC-ключей. Далее, когда выбранный симметричный шифр выполнит сцепление блоков шифротекста (например 3DES или AES), мы получим из главного секретного ключа еще и два вектора инициализации — по одному с каждой стороны соединения. В дальнейшем все сообщения, пересылаемые между клиентом и сервером, шифруются и аутентифицируются (при помощи MAC)
5. Клиент отсылает MAC всех сообщений рукопожатия
6. Сервер отсылает MAC всех сообщений рукопожатия

Последние два шага защищают рукопожатие от изменения информации. Чтобы в этом убедиться, обратите внимание, что на этапе 1 клиент как правило предлагает список алгоритмов — некоторые стойкие, некоторые слабые. Этот список алгоритмов отсылается в незашифрованном виде, поскольку договоренность об использовании алгоритмов шифрования и ключей пока не достигнута. Мэри, предпринимающая атаку посредника, может удалить из этого списка сравнительно стойкие алгоритмы, вынудив клиент выбрать более слабый алгоритм. Чтобы предотвратить такую атаку с изменением информации, на этапе 5 клиент отсылает MAC-код цепочки всех сообщений рукопожатия, которые он послал и получил. Сервер сравнивает этот MAC-код с MAC-кодом всех сообщений рукопожатия, которые отослал и получил он. При обнаружении несовпадения сервер может разорвать соединение. Аналогично, сервер посылает клиенту список всех просмотренных сообщений рукопожатия, позволяя клиенту проверить возможные несоответствия.

Может возникнуть вопрос: а зачем нужны одноразовые номера на этапах 1 и 2? Не достаточно ли порядковых номеров для предотвращения атаки с повторным воспроизведением сегментов? Ответ — да, но одноразовые номера позволяют защититься не только от атаки с повторным воспроизведением. Предположим, Мэри просматривает все сообщения, курсирующие между Алисой и Бобом. На другой день Мэри

притворяется Бобом и посылает Алисе ровно такую последовательность сообщений, какую ей посылал Боб днем ранее. Если Алиса не использует одноразовых номеров, то она отправит в ответ такую же последовательность сообщений, которую вчера передавала Бобу. Она не заподозрит никакого подвоха, поскольку все сообщения, которые она получит, успешно пройдут проверку целостности. Если в роли Алисы выступает сервер Интернет-магазина, то система просто предположит, что Боб делает еще один заказ (такой же, как и вчера). С другой стороны, указывая в протоколе одноразовый номер, Алиса будет посылать в каждом TCP-сеансе разные одноразовые номера, и ключи шифрования в разные дни также будут отличаться. Следовательно, когда Алиса получит от Мэри повторно воспроизведенную запись SSL, такая запись будет выявлена при проверке целостности и фальшивая транзакция не пройдет. Итак, одноразовые номера в SSL применяются для того, чтобы предотвращать атаки с повторным воспроизведением соединения, а порядковые номера — для предотвращения повторного воспроизведения отдельных пакетов в ходе текущего сеанса.

Закрытие соединения

В какой-то момент Боб или Алиса захотят завершить SSL-сеанс. Чтобы это сделать, Боб может просто разорвать базовое соединение по протоколу TCP — то есть Боб должен послать Алисе сегмент TCP FIN. Но такое упрощенное решение создает условия для *атаки с отсечением* (truncation attack), при которой Мэри также вмешивается в текущий SSL-сеанс и преждевременно завершает его, отсылая сегмент TCP FIN. Если бы Мэри так поступила, Алиса думала бы, что получила от Боба все нужные данные, тогда как на самом деле получила бы только их часть. Для устранения такой проблемы следует указать в поле типа, предназначена ли эта запись для завершения SSL-сеанса. Хотя SSL-тип и отсылается открытым текстом, он аутентифицируется на стороне получателя по коду MAC. Если в сегменте будет такое поле, то при получении Алисой сегмента TCP FIN в момент, когда запись о завершении SSL-сеанса еще не получена, Алиса с полным правом может заподозрить что-то нечистое.

На этом мы завершаем знакомство с протоколом SSL. Мы увидели, что этот протокол использует многие криптографические принципы, рассмотренные в разделах 8.2 и 8.3. Читатели, которые хотели бы подробнее исследовать тему SSL, могут прочитать очень интересную книгу Рескорлы[415] об этой технологии.

8.7. Безопасность на сетевом уровне: IPsec и виртуальные частные сети

Протокол защиты IP, чаще обозначаемый аббревиатурой IPsec, обеспечивает безопасность на сетевом уровне. IPsec защищает IP-дейтаграммы, передаваемые между любыми двумя сетевыми устройствами, в частности, хостами и маршрутизаторами. Как вскоре будет описано, многие организации (корпорации, правительственные учреждения, некоммерческие организации и т. д.) используют IPsec для создания **виртуальных частных сетей** (virtual private networks, **VPN**), работающих в общедоступном Интернете.

Прежде чем перейти к изучению тонкостей IPsec, давайте отступим на шаг назад и поговорим о том, что означает «обеспечить конфиденциальность на сетевом уровне». В данном случае речь идет о конфиденциальности связи между двумя сетевыми устройствами (например, между двумя маршрутизаторами, двумя хостами или между хостом и маршрутизатором). Устройство-отправитель шифрует полезную нагрузку всех дейтаграмм, отсылаемых устройству-получателю. Такая зашифрованная полезная нагрузка может представлять собой TCP-сегмент, UDP-сегмент, ICMP-сообщение и т. д. Если бы на сетевом уровне существовала такая служба, то любые данные, пересылаемые от одного устройства к другому — в частности, электронные письма, веб-страницы, сообщения о рукопожатии по TCP, а также управляющие сообщения (например, ICMP или SNMP) — были бы скрыты от любого стороннего наблюдателя, который мог бы анализировать сетевой трафик. Поэтому принято говорить, что безопасность на сетевом уровне обеспечивает «сплошной охват».

Наряду с конфиденциальностью, протокол безопасности на сетевом уровне потенциально мог бы предоставлять и другие службы. Например, он мог бы выполнять аутентификацию источника, чтобы устройство-получатель проверяло, откуда пришла защищенная дейтаграмма. Протокол безопасности сетевого уровня мог бы обеспечивать целостность данных, что позволяло бы узлу-получателю проверять наличие любых изменений, которые могли быть внесены в дейтаграмму по пути следования. Наконец, такой протокол мог бы служить для предотвращения атак повторного воспроизведения — то есть, Боб получал бы возможность обнаруживать любые дублирующиеся дейтаграммы, которые злоумышленник вставляет в поток данных. Вскоре мы убедимся, что протокол IPsec на самом деле предоставляет механизмы, обеспечивающие

все подобные сервисы безопасности — то есть способствует соблюдению конфиденциальности, аутентификации источника, целостности данных и предотвращает атаки повторного воспроизведения.

8.7.1. IPsec и виртуальные частные сети (VPN)

Если организация является распределенной и существует сразу в нескольких регионах, то ей зачастую требуется собственная IP-сеть — такая, в которой между хостами и серверами можно было бы наладить безопасный и конфиденциальный обмен данными. Для этого организация может развернуть автономную физическую сеть — с маршрутизаторами, каналами и DNS-инфраструктурой — которая была бы полностью отделена от общедоступного Интернета. Такие сети называются **частными** (private network). Неудивительно, что такая частная сеть — дорогое удовольствие, поскольку для ее обустройства организация должна приобрести, установить и обслуживать всю собственную физическую сетевую инфраструктуру.

В настоящее время многие организации не развертывают и не поддерживают таких сетей, а создают виртуальные частные сети (VPN), существующие прямо на базе общедоступного Интернета. При использовании VPN весь внутренний трафик конкретной организации передается по Интернету, а не по автономной физической сети. Но для обеспечения конфиденциальности он сначала шифруется и лишь потом поступает в Интернет. Простая модель виртуальной частной сети показана на рис. 8.27. В данном случае организация состоит из головного офиса и филиала, а также к ней относятся коммивояжеры, которые обычно выходят в Интернет со своих ноутбуков из гостиничных номеров (здесь показан только один такой коммивояжер). Если в данной виртуальной частной сети два хоста из головного офиса посылают IP-дейтаграммы друг другу, либо если требуется связаться двум хостам из филиала, при таком сеансе применяется старый добрый IPv4 (без всяких IPsec-сервисов). Однако если два хоста этой организации связываются по пути, который пролегает по общедоступному Интернету, то трафик шифруется перед попаданием в Интернет.

Чтобы составить впечатление о том, как работают VPN, давайте рассмотрим простой пример в контексте рис. 8.27. Когда хост из головного офиса отправляет IP-дейтаграмму на ноутбук коммивояжера, находящегося в гостинице, шлюзовой маршрутизатор головного офиса преобразует обычную дейтаграмму IPv4 в дейтаграмму IPsec, и уже в таком виде

отправляет эту дейтаграмму по Интернету. Кстати, такая дейтаграмма IPsec обладает обычным заголовком, как у дейтаграмм IPv4, поэтому маршрутизаторы в общедоступном Интернете обрабатывают ее как обычную дейтаграмму IPv4, не замечая никакой разницы. Но, как показано на рис. 8.27, в поле данных дейтаграммы IPsec содержится IPsec-заголовок, который и применяется при обработке IPsec. Кроме того, IPsec-дейтаграмма зашифровывается. Когда она прибывает на ноутбук коммивояжера, операционная система ноутбука расшифровывает поле данных (а также предоставляет другие функции безопасности, в частности, проверяет целостность данных), после чего передает расшифрованное содержимое протоколу верхнего уровня (например, TCP или UDP).

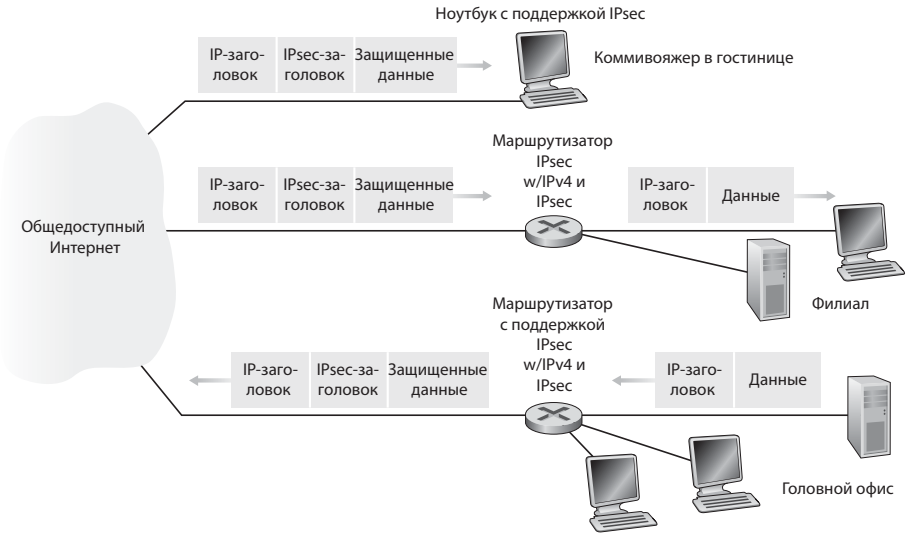


Рис. 8.27. Виртуальная частная сеть (VPN)

Итак, мы только что в общем виде рассмотрели, как применяется IPsec для создания виртуальных частных сетей. Чтобы разглядеть лес за деревьями, нам пришлось опустить многие важные детали. Давайте изучим их поподробнее.

8.7.2. Протоколы AH и ESP

Протокол IPsec — непростая штука. Он описывается более чем десятком стандартов RFC. Два важнейших из них — это RFC 4301, где определяется общая архитектура безопасности IP и RFC 6071, где рассматривается набор протоколов IPsec. Как и в других разделах этой книги, мы

стремимся не просто освежить в памяти сухие и мудреные RFC, а описать эти протоколы с более практической и образовательной точки зрения.

В наборе протоколов IPsec наиболее важны следующие два: **протокол аутентификации заголовка** (Authentication Header, **AH**) и **протокол безопасной инкапсуляции содержимого** (Encapsulation Security Payload, **ESP**). Когда устройство-отправитель, использующее IPsec (обычно это хост или маршрутизатор) отправляет дейтаграмму устройству-получателю (также хост или маршрутизатор), это делается по протоколу AH или по протоколу ESP. Протокол AH обеспечивает аутентификацию источника и целостность данных, *но не* конфиденциальность. Протокол ESP обеспечивает аутентификацию источника, целостность данных *и* конфиденциальность. Поскольку конфиденциальность обычно критически важна для VPN и других прикладных вариантов использования IPsec, протокол ESP распространен гораздо шире, чем AH. Чтобы развеять завесу тайны вокруг IPsec и не слишком вдаваться в его сложности, далее мы сосредоточимся на изучении протокола ESP. Если читатель также хотел бы разобраться с протоколом AH, отсылаем его к соответствующим RFC и онлайновым ресурсам.

8.7.3. Безопасные ассоциации

Каждая дейтаграмма IPsec пересылается между двумя сетевыми устройствами, например, между двумя хостами, двумя маршрутизаторами или между хостом и маршрутизатором. Прежде чем отправить IPsec-дейтаграмму по такому пути от источника к цели, между двумя этими устройствами создается логическое соединение на сетевом уровне, которое называется **безопасной ассоциацией** (security association, **SA**). Безопасная ассоциация — это симплексное логическое соединение, то есть оно является однонаправленным и пролегает от источника к получателю. Если оба устройства планируют посылать друг другу защищенные дейтаграммы, то необходимо установить две безопасные ассоциации (то есть, два логических соединения) — по одному в каждом направлении.

Например, вновь рассмотрим виртуальную частную сеть организации, продемонстрированную на рис. 8.7. В данном примере предположим, что между головным офисом и филиалом передается двунаправленный IPsec-трафик, а также он передается между головным офисом и ноутбуком каждого из коммивояжеров. Сколько безопасных ассоциаций насчитывается в этой сети VPN? Для ответа на этот вопрос необходимо учесть, что что у нас имеется по две безопасные ассоциации между

шлюзами головного офиса и филиала (по одной в каждом направлении). Кроме того, по две безопасные ассоциации существует между ноутбуком каждого из коммивояжеров и головным офисом (опять же, по одной в каждом направлении). Итак, всего имеем (2+2*n*) безопасных ассоциаций. *При этом следует иметь в виду, что не весь трафик, направляемый в Интернет шлюзовыми маршрутизаторами или ноутбуками будет защищен по технологии IPsec*. Например, хост из головного офиса может обратиться к серверу, расположенному в большом Интернете (например, к серверу Amazon или Google). Следовательно, шлюзовой маршрутизатор (а также ноутбуки) будут отсылать в Интернет как обычные дейтаграммы IPv4, так и защищенные дейтаграммы IPsec.

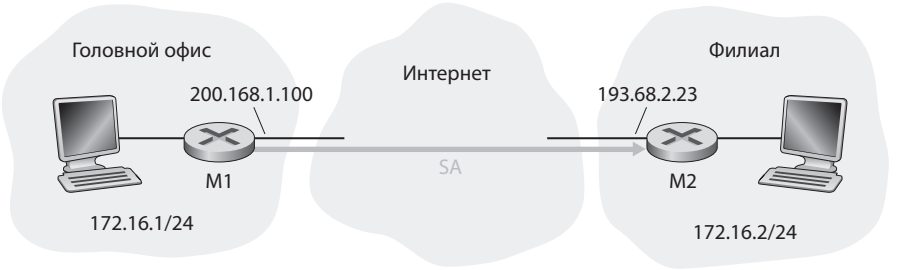


Рис. 8.28. Безопасная ассоциация (SA) между маршрутизаторами M1 и M2

Теперь давайте заглянем «внутрь» безопасной ассоциации. Чтобы разговор получился доступным и конкретным, рассмотрим безопасную ассоциацию, устанавливаемую между маршрутизаторами M1 и M2 (см. рис. 8.28). Можете считать, что эти маршрутизаторы находятся в виртуальной частной сети с рис. 8.27, причем M1 установлен в головном офисе, а M2 — в филиале компании. Маршрутизатор M1 будет хранить информацию о состоянии данной безопасной ассоциации, в частности:

- 32-разрядный идентификатор безопасной ассоциации, называемый **индексом параметра защиты** (Security Parameter Index, SPI)
- Исходный интерфейс защитной ассоциации (здесь — 200.168.1.100) и целевой интерфейс этой ассоциации (здесь — 193.68.2.23)
- Тип шифрования, который должен использоваться (например, 3DES с CBC)
- Ключ шифрования
- Тип проверки целостности (например, HMAC с MD5)
- Ключ аутентификации

Всякий раз, когда маршрутизатору M1 требуется создать IPsec-дейтаграмму для передачи по данной безопасной ассоциации, он обращается к вышеприведенной информации о состоянии, чтобы определить, как ему аутентифицировать и шифровать эту дейтаграмму. Аналогично, маршрутизатор M2 будет хранить такую же информацию об этой ассоциации, аутентифицируя и дешифруя с ее помощью любые IPsec-дейтаграммы, поступающие к нему через эту ассоциацию.

IPsec-устройство (маршрутизатор или хост) часто хранит информацию сразу о нескольких ассоциациях защиты. Так, в примере с виртуальной частной сетью с рис. 8.27, к которой подключаются n коммивояжеров, шлюзовой маршрутизатор хранит информацию о состоянии $(2+2n)$ безопасных ассоциаций. Всю информацию о своих безопасных ассоциациях сетевое устройство, работающее по IPsec, хранит в **базе данных о безопасных ассоциациях** (Security Association Database, **SAD**). Это особая структура данных, расположенная в ядре операционной системы устройства.

8.7.4. Дейтаграмма IPsec

Теперь, описав безопасные ассоциации, можно перейти собственно к обсуждению IPsec-дейтаграмм. В IPsec применяются пакеты двух видов, которые предназначены для так называемого **туннельного режима** (tunnel mode) и для **транспортного режима** (transport mode). Туннельный режим более удобен для работы с виртуальными частными сетями, и поэтому применяется шире, чем транспортный. Чтобы не усложнять тему IPsec, далее мы сосредоточимся исключительно на туннельном режиме. Когда вы хорошо усвоите туннельный режим, вы сможете без труда изучить транспортный режим самостоятельно.

Формат пакета дейтаграммы IPsec показан на рис. 8.29. Кому-то может показаться, что форматы пакетов — это скучная и пресная тема, но это не так — вскоре вы убедитесь, что IPsec-дейтаграмма на вкус и цвет не уступает самым острым деликатесам! Рассмотрим поля IPsec-дейтаграммы в контексте рис. 8.28. Предположим, маршрутизатор M1 получает обычную дейтаграмму IPv4 с хоста 172.16.1.17 (сеть головного офиса), которая должна быть отослана на хост 172.16.2.48 (в сети филиала). Маршрутизатор M1 преобразует эту «исходную дейтаграмму IPv4» в дейтаграмму IPsec по следующему принципу:

- Прикрепляет к хвосту исходной дейтаграммы IPv4 (содержащей исходные поля заголовка!) поле с заключительной частью ESP-заголовка (так называемый ESP trailer).

- Зашифровывает результат при помощи алгоритма и ключа, диктуемых безопасной ассоциацией.
- Перед этим зашифрованным блоком ставит поле заголовка («ESP header»). Результирующий пакет условно назовем «солянка».
- Далее для *всей этой сборной солянки* создается MAC-код аутентификации с применением алгоритма и ключа, заданного в рамках текущей ассоциации.
- MAC-код прикрепляется к хвосту пакета-солянки, и вся эта информация образует *полезное содержимое* IP-дейтаграммы.
- Наконец, создается совершенно новый IP-заголовок с традиционными полями IPv4 (общая длина — обычно 20 байт), который ставится перед только что сформированными данными.

Обратите внимание: в результате у нас получается полноценная дейтаграмма IPv4, где за обычными полями IPv4-заголовка следует поле данных. Но здесь в этих данных содержатся ESP-заголовок, исходная IP-дейтаграмма, хвостовой фрагмент ESP и поле аутентификации ESP (причем исходная IP-дейтаграмма и хвостовой фрагмент ESP зашифрованы). IP-адреса отправителя и получателя исходной IP-дейтаграммы — соответственно 172.16.1.7 и 172.16.2.48. Поскольку в состав IPsec-дейтаграммы входит исходная IP-дейтаграмма, эти адреса включаются (и зашифровываются) как часть полезного содержимого IPsec-пакета. Но что мы можем сказать об IP-адресах из нового IP-заголовка — того, который расположен в крайнем левом заголовке IPsec-дейтаграммы? Как вы, вероятно, уже догадались, сюда записываются адреса двух интерфейсов, расположенных по обе стороны туннеля, а именно — 200.168.1.100 и 193.68.2.23. Кроме того, в этом новом IP-заголовке идентификатор протокола указывает на привычные протоколы: TCP, UDP или ICMP. Вместо этого находим здесь значение 50, указывающее, что перед нами — дейтаграмма IPsec, использующая протокол ESP.

После того как маршрутизатор M1 отправит дейтаграмму IPsec в общедоступный Интернет, она пройдет через множество маршрутизаторов, прежде чем достигнет M2. Все эти маршрутизаторы обработают нашу дейтаграмму как самую обычную — их совершенно не смущает тот факт, что она несет зашифрованные данные в формате IPsec. С точки зрения этих маршрутизаторов из общедоступного Интернета важно то, что во внешнем IP-заголовке в качестве узла назначения указан маршрутизатор M2, именно поэтому дейтаграмма должна быть доставлена на M2.

Изучив на примере состав и создание IPsec-дейтаграммы, давайте подробнее изучим ингредиенты нашей «солянки». На рис. 8.29 показано, что хвостовой фрагмент IPsec состоит из трех полей: заполнитель, длина заполнителя и следующий заголовок. Как вы помните, блочные шифры требуют, чтобы длина шифруемого сообщения была кратна длине блока. Заполнитель (состоящий из произвольных байтов, не несущих полезной информации) используется именно для того, чтобы вместе с ним (а также с полями длины заполнителя и следующего заголовка) исходная дейтаграмма и все сообщение образовывали целое число блоков. Поле длины заполнителя указывает узлу назначения, сколько заполняющих байтов было вставлено в дейтаграмму (соответственно, сколько байтов нужно удалить). Поле следующего заголовка указывает тип данных (например, UDP), содержащихся в поле полезного содержимого. Сами эти данные (как правило, это исходная IP-дейтаграмма) объединяются с хвостовым фрагментом ESP, после чего зашифровываются.

Рис. 8.29. Формат IPsec-дейтаграммы

Перед зашифрованным блоком данных ставится ESP-заголовок, который отсылается в незашифрованном виде и состоит из двух полей: индекса параметра защиты и порядкового номера. Индекс указывает устройству назначения, к какой безопасной ассоциации относится дейтаграмма. Устройство назначения может дополнить свою базу данных SAD этим индексом и определить подходящие алгоритмы аутентификации и дешифрования, а также ключи. Поле с порядковым номером помогает защититься от атак повторного воспроизведения.

Устройство-отправитель также прикрепляет к дейтаграмме MAC-код аутентификации. Как было указано выше, это устройство вычисляет MAC-код для всей «солянки» (состоящей из ESP-заголовка, исходной

IP-дейтаграммы, а также хвостового фрагмента ESP, где дейтаграмма и заключительный фрагмент передаются в зашифрованном виде). Также напоминаем, что для вычисления MAC отправитель прикрепляет к «солянке» секретный MAC-ключ, а затем вычисляет для результата хэш фиксированной длины.

Когда маршрутизатор M2 получает IPsec-дейтаграмму, он убеждается (по IP-адресу в заголовке), что именно он сам, M2, и является получателем этой IP-дейтаграммы. После этого M2 обрабатывает дейтаграмму. Поскольку поле протокола (в крайнем левом IP-заголовке) имеет значение 50, M2 делает вывод, что должен поступить с этой дейтаграммой как с информацией IPsec. Заглянув в «солянку», M2 первым делом берет индекс параметра защиты и по нему определяет ту безопасную ассоциацию, к которой относится дейтаграмма. Далее он вычисляет MAC-код «солянки» и удостоверяется, что этот результат соответствует значению, записанному в поле ESP MAC. Таким образом, маршрутизатор M2 узнает, что данная дейтаграмма пришла с маршрутизатора M1 и по пути не была изменена. В-третьих, он проверяет поле порядкового номера и убеждается, что полученная дейтаграмма — свежая (а не повторно воспроизведенная). В-четвертых, он дешифрует зашифрованную информационную единицу, применяя при этом алгоритм дешифрования и ключ, связанные с безопасной ассоциацией. В-пятых, он отбрасывает заполняющие байты и извлекает исходную обычную IP-дейтаграмму. Наконец, в-шестых, он передает исходную дейтаграмму в сеть офиса-филиала, куда она и предназначалась. Ух, ну и сложный рецепт, правда? Но ведь и вкусную солянку сварить не так-то просто!

Здесь существует еще одна важная тонкость, которую следует обсудить отдельно. Она заключается в ответе на следующий вопрос: если маршрутизатор M1 получает (незащищенную) дейтаграмму от хоста, расположенного в головном офисе, и эта дейтаграмма направляется на какой-либо IP-адрес вне головного офиса, как M1 узнает, следует ли преобразовать эту дейтаграмму в формат IPsec? А если она должна быть обработана при помощи IPsec, как M1 узнает, какая безопасная ассоциация (или несколько таких ассоциаций из базы данных SAD) должна задействоваться для создания IPsec-дейтаграммы? Вот как решается эта проблема. Наряду с базой данных SAD IPsec-устройство поддерживает еще одну структуру данных, которая называется **базой данных политики безопасности** (Security Policy Database, SPD). Она указывает, какие типы дейтаграмм (в зависимости от исходного IP-адреса, конечного IP-адреса и типа протокола) необходимо обрабатывать с применением

IPsec. Для таких дейтаграмм SPD указывает, какая безопасная ассоциация должна учитываться при обработке. Можно сказать, что информация из базы данных SPD позволяет понять, что делать с прибывшей дейтаграммой, а информация из SAD — как это сделать.

Сервисы IPsec — заключение

Итак, какие именно сервисы предоставляются в протоколах IPsec? Давайте рассмотрим их с точки зрения злоумышленника, например Мэри. Она находится где-то между маршрутизаторами M1 и M2 на рис. 8.28 и предпринимает атаку посредника. Будем исходить из того, что Мэри не знает ключей аутентификации и шифрования, применяемых в рамках безопасной ассоциации. Что сможет и чего не сможет сделать Мэри? Во-первых, Мэри не сможет просмотреть исходную дейтаграмму. Фактически от Мэри будет скрыта не только информация из этой дейтаграммы, но и ее номер протокола, IP-адреса отправителя и получателя. Если дейтаграммы посылаются в рамках безопасной ассоциации, Мэри может определить лишь то, что дейтаграмма направляется с какого-то хоста подсети 172.16.1.0/24 на другой хост, расположенный в подсети 172.16.2.0/24. Мэри не знает, данные для какого протокола находятся в этой дейтаграмме — TCP, UDP или ICMP; она не знает, каков тип прикладных данных в этой дейтаграмме — HTTP, SMTP или еще какой-нибудь. Таким образом, достигается гораздо более полная конфиденциальность, чем при применении SSL. Далее предположим, что Мэри пытается изменить эту дейтаграмму, инвертировав в ней те или иные биты. Если ей это удастся, и такая измененная дейтаграмма попадет на маршрутизатор M2, она не пройдет на этом узле проверку целостности. Наконец, поскольку IPsec-дейтаграммы снабжаются порядковыми номерами, Мэри не сможет выполнить атаку повторного воспроизведения. Итак, как и было сказано в начале этого раздела, технология IPsec обеспечивает конфиденциальность, аутентификацию источника, целостность данных и предотвращение атак повторного воспроизведения при обмене информацией между любыми двумя устройствами, обрабатывающими пакеты на сетевом уровне.

8.7.5. IKE: управление ключами при применении IPsec

Если в виртуальной частной сети насчитывается сравнительно немного конечных точек (например, всего два маршрутизатора, как на рис. 8.28), администратор вычислительной сети может вручную ввести

информацию о безопасных ассоциациях (алгоритмы шифрования/аутентификации, ключи, индексы параметров защиты) в базы данных SAD каждой из конечных точек. Но, разумеется, такая работа вручную очень непрактична в больших виртуальных частных сетях, которые могут состоять из сотен и даже из тысяч маршрутизаторов и хостов, использующих технологию IPsec. Развертывание масштабных географически распределенных сетей предполагает наличие автоматизированного механизма для создания безопасных ассоциаций. В IPsec эта задача решается при помощи протокола обмена ключами в Интернете (Internet Key Exchange, IKE), описанного в стандарте RFC 5996.

Протокол IKE обладает некоторым сходством с механизмом рукопожатия в технологии SSL (см. раздел 8.6). Каждое IPsec-устройство обладает сертификатом, в котором указывается открытый ключ этого устройства. Как и в случае с SSL, протокол IKE предполагает, что два устройства будут обмениваться сертификатами, согласовывать алгоритмы аутентификации и шифрования, а также в защищенном режиме обмениваться информацией о ключах, создавая сеансовые ключи в рамках безопасных ассоциаций IPsec. В отличие от SSL, IKE решает эти задачи в два этапа.

Рассмотрим эти этапы в контексте двух маршрутизаторов M1 и M2, показанных на рис. 8.28. Первый этап заключается в двух операциях обмена парами сообщений между маршрутизаторами M1 и M2.

- При первом обмене сообщениями обе стороны используют алгоритм Диффи — Хеллмана (подробнее см. в заданиях для самостоятельной работы к этой главе) для создания двунаправленной безопасной ассоциации **IKE SA** между маршрутизаторами. Чтобы хорошенько заморочить всем голову, отметим, что эти двунаправленные безопасные ассоциации IKE коренным образом отличаются от безопасных ассоциаций IPsec, рассмотренных в разделах 8.6.3 и 8.6.4. Безопасная ассоциация IKE позволяет установить между двумя маршрутизаторами аутентифицированный и зашифрованный канал. В ходе этого первого обмена сообщениями создаются ключи, которые будут использоваться в данной безопасной ассоциации IKE для шифрования и аутентификации. Также создается главный секретный ключ, который будет применяться для вычисления ключей безопасной ассоциации IPsec позже — уже на втором этапе. Обратите внимание: на первом этапе не используются ни открытые, ни закрытые ключи RSA. В частности, ни M1, ни M2 не открывают своей идентификационной информации, так как не подписывают сообщение закрытым ключом.

- В ходе второго обмена сообщениями обе стороны открывают друг другу свою идентификационную информацию, подписывая свои сообщения. Однако такая идентификационная информация остается недоступна для пассивного перехватчика-анализатора (сниффера), поскольку она передается по защищенному каналу безопасной ассоциации IKE. Кроме того, на данном этапе две стороны согласовывают алгоритмы IPsec-шифрования и аутентификации, которые будут задействоваться в безопасных ассоциациях IPsec.

На втором этапе использования IKE две стороны создают в каждом из двух направлений безопасные ассоциации. В конце второго этапа на обеих сторонах создаются сеансовые ключи шифрования и аутентификации для двух этих безопасных ассоциаций. Затем обе стороны применяют свои безопасные ассоциации для отправки защищенных дейтаграмм, как это описано в разделах 8.7.3 и 8.7.4. Два этапа в работе IKE требуются в первую очередь для снижения вычислительных издержек. Поскольку на втором этапе не применяется какая-либо криптография с открытыми ключами, IKE позволяет создавать большое количество безопасных ассоциаций между двумя IPsec-устройствами с минимальными издержками.

8.8. Защита беспроводных локальных сетей

Безопасность особенно важна в беспроводных локальных сетях, где кадры переносятся радиоволнами, способными распространяться далеко за пределы здания, где находится беспроводная базовая станция и хосты. В этом разделе будет сделано краткое введение в тему безопасности беспроводных сетей. Гораздо глубже эта тема исследована в увлекательной книге Эдни[147].

Проблема безопасности в сетях стандарта 802.11 привлекает повышенное внимание как в технической сфере, так и в СМИ. На эту тему ведутся активные дискуссии, причем споров возникает сравнительно немного — по-видимому, все согласны, что в исходной спецификации 802.11 существует ряд серьезных недоработок по части безопасности. Действительно, сейчас не составляет труда скачать программы, которые эксплуатируют эти изъяны. Соответственно, те, кто пытается обеспечить безопасность при помощи одних только возможностей стандарта 802.11, не менее беззащитны перед лицом злоумышленников, чем те, кто полностью пренебрегает вопросами безопасности.

В следующем разделе мы рассмотрим механизмы безопасности, которые были описаны в исходной версии спецификации 802.11 и обозначаются собирательным термином «**конфиденциальность на уровне проводных сетей**» (Wired Equivalent Privacy, WEP). Как понятно из названия, система WEP призвана обеспечить в беспроводных сетях уровень безопасности, сопоставимый с тем, что достигается в обычных стационарных сетях. Мы рассмотрим ряд недостатков в области безопасности, характерных для WEP, а также обсудим стандарт 802.11i — принципиально более безопасную версию 802.11, принятую в 2004 году.

8.8.1. Конфиденциальность на уровне проводных сетей (WEP)

Протокол WEP стандарта IEEE 802.11 был разработан в 1999 году для обеспечения аутентификации и шифрования при обмене данными между хостом и беспроводной точкой доступа (то есть базовой станцией) с использованием симметричных разделяемых ключей. Стандарт WEP не указывает алгоритм управления ключами, поэтому предполагается, что хост и беспроводная точка доступа должны каким-либо образом согласовать ключ для совместного использования, сделав это внеполосным методом. Аутентификация выполняется следующим образом.

1. Беспроводной хост запрашивает аутентификацию у точки доступа
2. Точка доступа откликается на этот запрос 128-байтным одноразовым номером
3. Беспроводной хост зашифровывает одноразовый номер при помощи симметричного ключа, который он использует совместно с точкой доступа
4. Точка доступа расшифровывает одноразовый номер, зашифрованный хостом

Если дешифрованный одноразовый номер совпадает с тем значением одноразового номера, которое было изначально отправлено на хост, то точка доступа аутентифицирует этот хост.

Алгоритм шифрования данных, применяемый в технологии WEP, проиллюстрирован на рис. 8.30. Предполагается, что секретный 40-битный симметричный ключ K_S известен как хосту, так и точке доступа. Кроме того, к этому 40-битный ключу прикрепляется 24-битный вектор инициализации (*IV*). Получается 64-битный ключ, которым будет

зашифровываться отдельный кадр. Разным кадрам соответствуют разные векторы инициализации, поэтому каждый кадр зашифровывается своим 64-битным ключом. Вот как выполняется шифрование. Сначала для данных, передаваемых в качестве полезной нагрузки, вычисляется 4-байтное значение CRC (код циклического контроля, см. раздел 5.2). После этого полезная нагрузка и 4-байтный код циклического контроля зашифровываются при помощи потокового шифра RC4. Мы не будем подробно рассматривать здесь шифр RC4, его исчерпывающее описание дается в работах Шнейера[587] и Эдни[147].

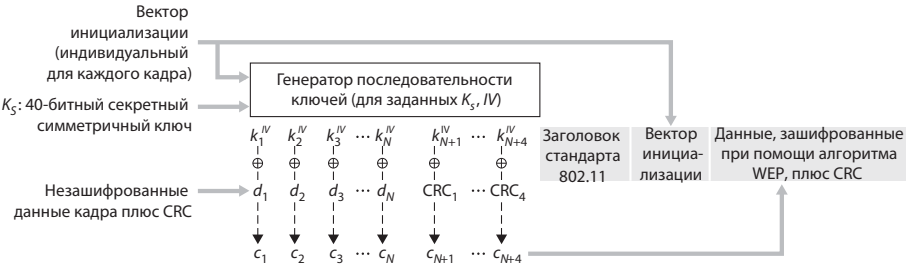


Рис. 8.30. Протокол WEP стандарта 802.11

В нашей книге достаточно сказать, что на основе значения ключа (в данном случае, имеется в виду 64 бита ключа (K_S) и вектора инициализации (IV) алгоритм RC4 дает поток значений ключа k_1^{IV}, k_2^{IV}, k_3^{IV} ... которые используются для шифрования в кадре как самих данных, так и кода циклического контроля. На практике можно считать, что такие операции выполняются по одному байту в единицу времени. Шифрование осуществляется путем применения исключающего ИЛИ к i-ому байту данных, d_i, причем для этого задействуется i-й ключ k_i^{IV} из потока значений, сгенерированных парой (K_S, IV). На выходе имеем i-й байт шифротекста c_i:

$$c_i = d_i \oplus k_i^{IV}$$

Значение вектора инициализации изменяется от кадра к кадру и открытым текстом записывается в заголовке каждого кадра 802.11, зашифрованного по алгоритму WEP, как показано на рис. 8.30. Получатель берет секретный 40-битный симметричный ключ, который он использует совместно с отправителем, прикрепляет к нему вектор инициализации и применяет для дешифрования кадра 64-битный ключ (идентичный тому, что применялся при шифровании кадра):

$$d_i = c_i \oplus k_i^{IV}$$

Для корректной работы алгоритма RC4 требуется, чтобы одно и то же 64-битное значение ключа обязательно использовалось *всего один раз*. Как вы помните, ключ WEP меняется от кадра к кадру. Для конкретного ключа K_S (который меняется редко, если вообще меняется) это означает, что такой ключ может иметь всего 2^{24} уникальных значений.

Если такие ключи выбираются случайным образом, то легко подсчитать[655, 147], что вероятность выбора двух одинаковых значений для вектора инициализации (с использованием одного и того же 64-битного ключа) достигает 99 процентов всего после 12 000 кадров. Когда каждый кадр имеет размер 1 кбайт, а скорость передачи данных составляет 11 кбит/с, на передачу 12 000 кадров затрачивается всего несколько секунд. Более того, поскольку вектор инициализации передается в кадре в незашифрованном виде, злоумышленник легко сможет определить, когда используется дублируемое значение вектора инициализации.

Чтобы оценить одну из многих проблем, возникающих при использовании дублируемых векторов инициализации, рассмотрим следующую атаку с подобранным открытым текстом, которую Мэри предпринимает против Алисы. Мэри (возможно, с применением IP-спуфинга) посылает Алисе запрос (например, по протоколу HTTP или FTP) на передачу файла с известным содержимым d_1, d_2, d_3, d_4,.... Кроме того, Мэри наблюдает зашифрованные данные c_1, c_2, c_3, c_4. . . . Поскольку $d_i = c_i \oplus k_i^{IV}$, при применении исключающего ИЛИ к c_i с каждой стороны уравнения имеем:

$$d_i \oplus c_i = k_i^{IV}$$

При наличии такого отношения Мэри может использовать известные значения d_i и c_i для вычисления k_i^{IV}. Когда Мэри в следующий раз заметит уже применявшееся ранее значение вектора инициализации, она также будет знать последовательность ключей k_1^{IV}, k_2^{IV}, k_3^{IV} ... и, соответственно, сможет дешифровать зашифрованное сообщение.

С алгоритмом WEP связан и ряд других проблем из области безопасности. В книге Флюрера[168] описана атака, в ходе которой эксплуатируется брешь в RC4, когда выбираются известные слабые ключи. В книге Стабблефилда[624] описываются эффективные способы реализации и эксплойта этой атаки. Еще одна проблема из области WEP связана с битами кода циклического контроля, показанными на рис. 8.30 и передаваемыми в кадре 802.11 для отслеживания возможных изменений битов в полезном содержимом.

Однако злоумышленник, модифицирующий зашифрованное содержимое (например, меняющий исходные зашифрованные данные на тарабарщину), вычисляет код циклического контроля для такой тарабарщины, а затем перезаписывает это значение CRC в WEP-кадр. Получается кадр 802.11, который будет нормально принят получателем. Здесь нам могут помочь приемы обеспечения целостности сообщения — например, те, что были изучены в разделе 8.3 и помогают в обнаружении подмены содержимого или забивки его бессмысленной информацией. Тема безопасности в технологии WEP подробнее рассматривается в работах Эдни[147], Уолкера[655], Уитерспуна[660] и в тех работах, на которые ссылаются эти авторы.

8.8.2. Стандарт IEEE 802.11i

Вскоре после того, как в 1999 году вышел стандарт IEEE 802.11, началась работа по созданию его новой, улучшенной версии, в которой были бы реализованы более совершенные механизмы безопасности. Новый стандарт, названный 802.11i, был окончательно ратифицирован в 2004 году. Как было показано выше, алгоритм WEP дает относительно слабое шифрование, предусматривает всего один способ выполнения аутентификации и не предоставляет механизмов для распространения ключей. IEEE 802.11i оснащен значительно более сильными видами шифрования, расширяемым набором механизмов аутентификации, а также поддерживает специальную процедуру распространения ключей. Здесь приводится лишь краткий обзор стандарта 802.11i; отличный технический обзор 802.11i (потоковое аудио) приведен в вебкасте Protected Wireless Networks[632].

На рис. 8.31 схематически представлена структура 802.11i. Наряду с беспроводным клиентом и точкой доступа в стандарте 802.11i определяется сервер аутентификации, с которым может обмениваться информацией точка доступа. При отделении точки доступа от сервера аутентификации появляется возможность обслуживать одним сервером множество точек доступа, централизованно принимая (зачастую конфиденциальные) решения об аутентификации и доступе. Стоимость обслуживания точек доступа остается сравнительно невысокой. Работа 802.11i осуществляется в четыре этапа:

1. ***Обнаружение***. На этапе обнаружения точка доступа сообщает о собственном наличии и о формах аутентификации и шифрования, которые она может предоставить беспроводному клиентскому узлу.

Затем клиент запрашивает конкретные формы аутентификации и шифрования, которые ему нужны. Хотя клиент и точка доступа уже обмениваются сообщениями, клиент пока не аутентифицировался, а также пока не имеет ключа шифрования. Не выполнены и еще несколько этапов, которые обязательно нужно пройти, чтобы клиент мог обмениваться информацией с произвольным удаленным хостом по беспроводному каналу.

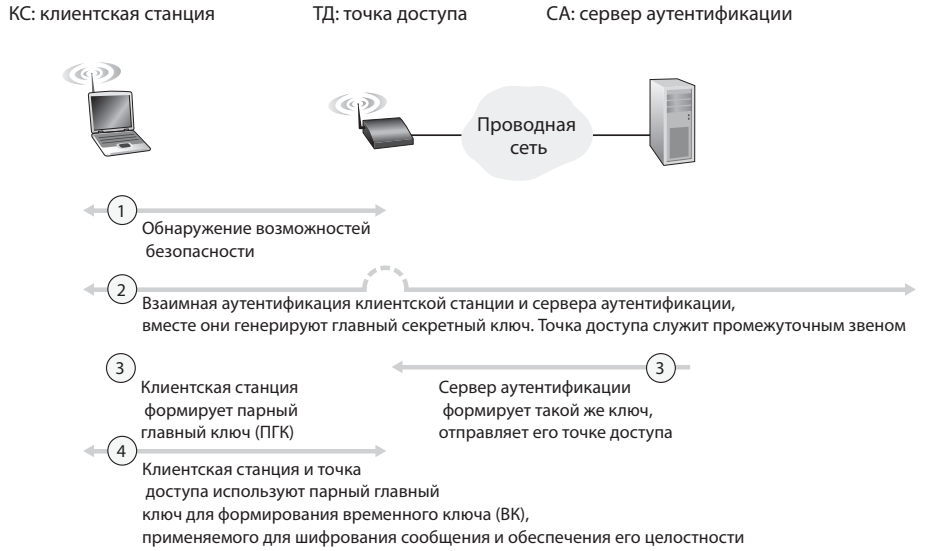


Рис. 8.31. Четыре этапа работы стандарта 802.11i

2. *Взаимная аутентификация и формирование главного секретного ключа (ГСК).* Аутентификация осуществляется между беспроводным клиентом и сервером аутентификации. На данном этапе точка доступа действует, в сущности, как ретранслятор, пересылая сообщения между клиентом и сервером аутентификации. Протокол **EAP (расширяемый протокол аутентификации** (Extensible Authentication Protocol), описан в стандарте RFC 3748523) определяет форматы двунаправленных сообщений, применяемых в режиме простого взаимодействия запрос-ответ между клиентом и сервером аутентификации. Как показано на рис. 8.32, сообщения EAP инкапсулируются в рамках протокола **EAPoL** (EAP по локальной сети[235]) и отправляются по беспроводному каналу стандарта 802.11. Затем на точке доступа эти сообщения извлекаются, после чего повторно инкапсулируются в протокол **RADIUS**, предназначенный для пере-

дачи информации по UDP/IP на сервер аутентификации. Отметим, что сервер и протокол RADIUS[487] не являются обязательными для работы в соответствии с протоколом 802.11i, но *де факто* уже стали неотъемлемыми компонентами этого протокола. Вероятно, в ближайшем будущем на смену протоколу RADIUS придет недавно стандартизованный протокол **DIAMETER**[520].

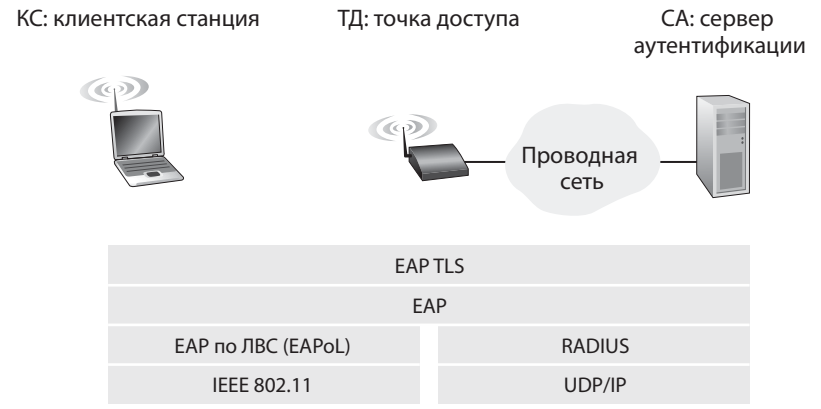


Рис. 8.32. EAP — «сквозной» протокол. Сообщения EAP инкапсулируются при помощи протокола EAPoL и передаются по беспроводному каналу между клиентом и точкой доступа, а при помощи протокола RADIUS — по UDP/IP между точкой доступа и сервером аутентификации

При работе по протоколу EAP сервер может выбирать один из нескольких способов выполнения аутентификации. Хотя стандарт 802.11i не регламентирует какого-либо обязательного метода, часто применяется схема аутентификации EAP-TLS[552]. В схеме EAP-TLS задействуются методы шифрования открытым ключом (в том числе, шифрование с применением одноразовых номеров и хэшей сообщений) подобные тем, что были описаны в разделе 8.3. Они позволяют клиенту и серверу аутентификации взаимно удостоверять друг друга, а также формировать главный секретный ключ (ГСК), который будет известен обеим сторонам.

3. *Формирование парного главного ключа (ПГК).* Главный ключ — это разделенный секрет, известный только клиенту и серверу аутентификации. Каждая из сторон использует этот ключ для формирования второго ключа, который именуется «парным главным ключом». Затем сервер аутентификации посылает этот парный ключ точке доступа. Этого мы и добивались! Теперь у клиента и точки доступа есть разделяемый ключ (как вы помните, в алгоритме WEP проблема

распределения ключа вообще не решалась), к тому же они взаимно аутентифицировались. Они уже готовы приступить к работе.

4. *Формирование временного ключа (ВК).* Имея ПГК, беспроводной хост и точка доступа могут генерировать дополнительные ключи, которые будут использоваться при обмене информацией. В данном случае особенно интересен временный ключ (ВК), используемый для шифрования на канальном уровне применительно к данным канального уровня, которые пересылаются по беспроводному каналу на произвольный удаленный хост.

Стандарт 802.11i обеспечивает несколько видов шифрования, включая схему на основе алгоритма AES и усиленную версию WEP-шифрования.

8.9. Эксплуатационная безопасность: брандмауэры и системы обнаружения вторжений

В этой главе мы убедились, что Интернет — довольно небезопасное место. В нем повсюду шныряют негодяи, постоянно готовые на какие-то пакости. Учитывая негостеприимность большого Интернета, давайте рассмотрим в таком контексте компьютерную сеть отдельно взятой организации и администратора вычислительной сети, который ее обслуживает. С точки зрения администратора вычислительной сети, мир четко делится на два полярных лагеря: свои (те, кто относится к сети организации и должны иметь возможность относительно беспрепятственно обращаться к ресурсам внутри этой корпоративной сети) и чужие (все остальные; следовательно, любой, кто пытается получить доступ к внутрикорпоративным ресурсам, должен тщательно проверяться). Во многих организациях, которые могут быть расположены как в средневековых замках, так и в современных офисных комплексах, есть единая точка входа/выхода, где проходят проверку и свои, и чужие, приходящие в организацию и покидающие ее. Например, в замке такая проверка осуществлялась у ворот около подъемного моста, а в современной корпорации — на посту охраны у входа. Когда на границе компьютерной сети появляется новый трафик, он может быть проверен на предмет безопасности, зарегистрирован, отброшен или переадресован. Эти задачи решаются специальными устройствами, которые называются «брандмауэрами» или «сетевыми экранами», а также системами обнаружения вторжений (IDS) и системами предотвращения вторжений (IPS).

8.9.1. Брандмауэры

Брандмауэр — это сочетание программных и аппаратных средств, которые изолируют внутреннюю сеть организации от большого Интернета, пропуская одни пакеты и блокируя другие. Брандмауэр позволяет администратору вычислительной сети контролировать доступ к ресурсам корпоративной сети, предпринимаемый извне, а также управлять ресурсами в администрируемой сети, регулируя входящий и исходящий трафик, связанный с этими ресурсами. Действующий брандмауэр решает три практические задачи:

- *Через брандмауэр проходит весь трафик, поступающий в корпоративную сеть извне, а также идущий в обратном направлении.* На рис. 8.33 показан брандмауэр, расположенный на границе администрируемой сети и Интернета. В больших организациях может существовать несколько уровней брандмауэров, либо распределенные брандмауэры[603]. Однако, располагая брандмауэр в конкретной точке, через которую осуществляется доступ в сеть, как показано на рис. 8.33, мы можем гораздо надежнее обеспечивать политику безопасного доступа и управлять ею.

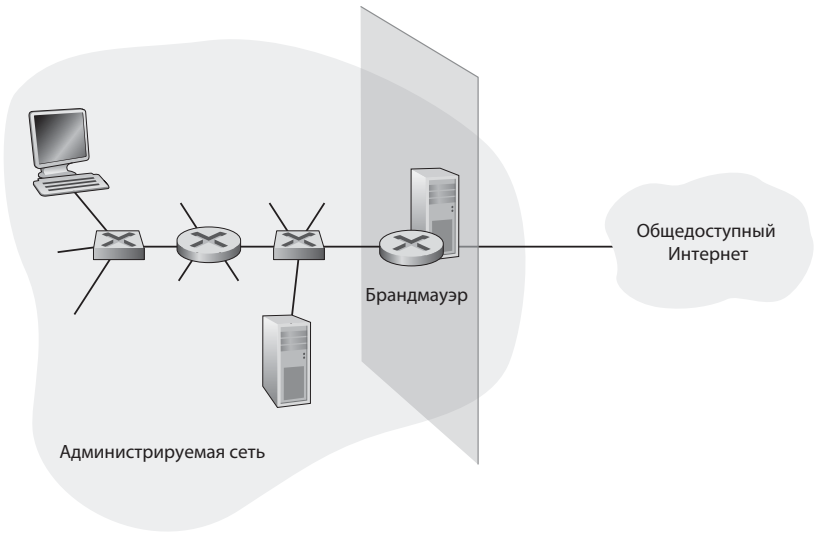


Рис. 8.33. Размещение брандмауэра между администрируемой сетью и внешним миром

- *В систему попадает только тот трафик, который удовлетворяет политике безопасности, определенной в локальной сети.* Поскольку

весь входящий и исходящий трафик корпоративной сети проходит через брандмауэр, он может ограничить этот поток, допуская в него лишь трафик, удовлетворяющий политике безопасности.

- *Сам брандмауэр должен быть неуязвим для вторжения.* Брандмауэр — это устройство, подключенное к сети. Если он неправильно спроектирован или установлен, и его можно взломать, в случае чего создается ложное ощущение безопасности — это еще хуже, чем полное его отсутствие!

В настоящее время основными производителями брандмауэров являются компании Cisco и Check Point. Можно с легкостью создать брандмауэр (фильтр пакетов) на основе Linux, воспользовавшись инструментом `iptables` (это общедоступная программа, которая обычно предоставляется в дистрибутивах Linux).

Все брандмауэры можно подразделить на три категории: **традиционные фильтры пакетов**, **фильтры, учитывающие состояние соединения** и **шлюзы приложений**. Все эти разновидности мы по очереди рассмотрим в следующих подразделах.

Традиционные фильтры пакетов

Как показано на рис. 8.33, организация обычно обладает шлюзовым маршрутизатором, который соединяет ее внутреннюю сеть с Интернет-провайдером (и, следовательно, с большим Интернетом). Весь входящий и исходящий трафик внутрикорпоративной сети проходит через этот маршрутизатор, и именно на немпроисходит **фильтрация пакетов**. Фильтр пакетов отдельно проверяет каждую дейтаграмму, определяя, как поступить с ней в соответствии с правилами, установленными администратором вычислительной сети: пропустить в сеть или отбросить. Решения, связанные с фильтрацией, обычно основываются на следующих факторах:

- Исходный или конечный IP-адрес
- Тип протокола в соответствующем поле IP-дейтаграммы: TCP, UDP, ICMP, OSPF **и т. д.**
- Порты и отправителя и получателя TCP- или UDP-соединения
- Флаговые биты TCP: SYN, ACK и т. д.
- Тип сообщения ICMP
- Различные правила, характеризующие входящие и исходящие дейтаграммы данной сети
- Различные правила, касающиеся интерфейсов маршрутизатора

Администратор вычислительной сети конфигурирует брандмауэр, опираясь на правила, действующие в организации (политику). Политика может учитывать интенсивность работы пользователя и активность расхода полосы передачи данных, а также соображения безопасности, действующие в организации. В табл. 8.5 перечислен ряд возможных политик, которые могут действовать в организации, а также описано, как они потенциально реализуемы при помощи пакетного фильтра. Например, если компания не желает принимать никаких входящих TCP-соединений кроме тех, которые инициированы ее веб-сервером, то брандмауэр такой сети может блокировать все входящие сегменты TCP SYN за исключением тех, у которых указан порт 80 и IP-адрес веб-сервера. Если организация не хочет, чтобы ее пользователи полностью заняли полосу доступа к Интернету радио-приложениями, то брандмауэр может заблокировать весь некритичный UDP-трафик (поскольку Интернет-радио часто транслируется по протоколу UDP). Если организация стремится предотвратить картографирование (трассировку) своей внутренней сети, на которое могут пойти злоумышленники, то она может блокировать все ICMP-сообщения с истекшим значением TTL (предписанным временем жизни), направляемые за пределы корпоративной сети.

Табл. 8.5. Политики и соответствующие правила фильтрации для корпоративной сети 130.207/16 с веб-сервером, расположенным по адресу 130.207.244.203

Политика	**Настройка брандмауэра**
Запрет на доступ к Интернету за пределами корпоративной сети	Отбрасывание всех исходящих пакетов, идущих на любой адрес через порт 80
Запрет входящих TCP-соединений за исключением тех, которые поступают с общедоступного веб-сервера организации	Отбрасывание всех входящих пакетов TCP SYN с любого IP кроме 130.207.244.203 порт 80
Недопущение того, чтобы полоса передачи данных активно расходовалась под трафик Интернет-радио	Отбрасывание всех входящих UDP-пакетов — кроме DNS-пакетов
Недопущение использования вашей сети для атак ICMP-запросами с измененными адресами (smurf attack) и последующего отказа в обслуживании	Отбрасывание всех ping-пакетов протокола ICMP, направляемых по широковещательным адресам (например, 130.207.255.255)
Защита вашей сети от трассировки	Отбрасывание всего исходящего трафика, передаваемого по протоколу ICMP и содержащего истекшее значение TTL

В основе политики фильтрации также могут лежать комбинации адресов и номеров портов. Например, фильтрующий маршрутизатор может передавать только те Telnet-дейтаграммы (идущие на порт 23),

которые исходят от множества IP-адресов, указанных в списке либо направляются на адреса из этого списка. Такая политика допускает установление Telnet-соединений только от хостов и к хостам, занесенным в список разрешенных адресов. К сожалению, при выстраивании политики на основе внешних адресов невозможно защититься от дейтаграмм, адреса которых подверглись спуфингу.

Фильтрация также может осуществляться по признаку того, установлен ли бит TCP ACK. Такой прием будет очень полезен, если организация планирует позволить своим внутренним клиентам подключаться к внешним серверам, но одновременно не разрешает внешним клиентам подключаться к внутренним серверам организации. Как вы помните из раздела 3.5, у первого сегмента в каждом TCP-соединении ACK-бит установлен в 0, тогда как у всех остальных сегментов в рамках соединения он установлен в 1. Соответственно, если в организации требуется не допустить соединений между внешними клиентами и внутренними серверами, то система должна будет просто отфильтровать все входящие сегменты, ACK-бит в которых имеет значение 0. Такая политика блокирует все TCP-соединения, устанавливаемые извне, но допускает соединения, инициируемые изнутри корпоративной сети.

Правила брандмауэра реализуются в маршрутизаторах при помощи списков контроля доступа, причем каждый интерфейс маршрутизатора располагает собственным. Пример списка контроля доступа для организации 222.22/16 дан в табл. 8.6. Этот список контроля доступа соответствует интерфейсу, соединяющему маршрутизатор с внешним Интернет-провайдером, обслуживающим организацию. Правила применяются к каждой дейтаграмме, которая проходит через данный интерфейс сверху вниз. Два первых правила вместе позволяют пользователям путешествовать по Интернету. Первое правило позволяет пакету с портом назначения 80 покинуть сеть организации; второе правило гарантирует, что любой пакет с исходным портом 80 и установленным ACK-битом сможет войти в сеть организации. Обратите внимание: если внешний источник пытается установить TCP-соединение с внутренним хостом, такое соединение блокируется, даже если портом отправителя или получателя для него является порт 80. Вторые два правила в сумме позволяют DNS-пакетам входить в сеть организации и покидать ее. В целом, такой довольно строгий список контроля доступа блокирует весь трафик за исключением, во-первых, Интернет-трафика, исходящего из организации и, во-вторых, DNS-трафика. В документе Packet Filtering for Firewall Systems[84] дается список рекомендуемых вариантов

фильтрации по принципу порт/протокол, позволяющий избежать ряда распространенных изъянов в типичных сетевых прикладных задачах.

Табл. 8.6. Список контроля доступа для интерфейса маршрутизатора

Действие	Исходный адрес	Конечный адрес	Протокол	Исходный порт	Конечный порт	Флаговый бит
Разрешить	222.22/16	Вне 222.22/16	TCP	>1023	80	Любой
Разрешить	Вне 222.22/16	222.22/16	TCP	80	>1023	ACK
Разрешить	222.22/16	Вне 222.22/16	UDP	>1023	53	–
Разрешить	Вне 222.22/16	222.22/16	UDP	53	>1023	–
Запретить	Все	Все	Все	Все	Все	Все

Фильтры, учитывающие состояние соединения

В традиционном фильтре пакетов решения о фильтрации принимаются отдельно по каждому пакету. Фильтры, учитывающие состояние соединения, отслеживают TCP-соединения и выполняют фильтрацию на основе этой информации.

Чтобы изучить фильтры, учитывающие состояние соединения, давайте вернемся к списку контроля доступа, представленному в табл. 8.6. Этот список довольно строгий, но тем не менее фильтр пропускает любые приходящие извне пакеты, удовлетворяющие двум условиям: бит ACK равен 1, а порт отправителя — 80. Злоумышленники могут использовать подобные пакеты, пытаясь обрушить внутренние системы организации посылкой искаженных пакетов, организуя DoS-атаку либо трассируя внутреннюю сеть. Примитивное решение такой проблемы — заблокировать и все пакеты TCP ACK, но в таком случае пользователи, работающие внутри организации, не смогут выходить со своих компьютеров в Интернет.

Гораздо удобнее решать такую проблему при помощи фильтров, учитывающих состояние соединения. Они отслеживают все текущие TCP-соединения в специальной таблице соединений. Такая возможность существует потому, что брандмауэр фиксирует начало нового соединения, отслеживая акты тройного рукопожатия (пакеты SYN, SYNACK и ACK). Кроме того, брандмауэр способен зафиксировать и окончание соедине-

ния, отследив соответствующий этому соединению пакет FIN. Брандмауэр также может (консервативно) предположить, что соединение неактивно, если оно фактически бездействует, скажем, в течение 60 секунд. Пример таблицы соединений для брандмауэра показан в табл. 8.7. Таблица соединений указывает, что в настоящий момент имеется три текущих TCP-соединения, и все они инициированы изнутри организации. Кроме того, в списке контроля доступа фильтра, учитывающего состояние соединения, находим новый столбец: «проверка соединения», как показано в табл. 8.8. Обратите внимание: табл. 8.8 идентична списку контроля доступа из табл. 8.6, за тем исключением, что в табл. 8.8 требуется проверить соединение на соответствие двум правилам.

Табл. 8.7. Таблица соединений для фильтра, учитывающего состояние соединений

Исходный адрес	Конечный адрес	Исходный порт	Конечный порт
222.22.1.7	37.96.87.123	12699	80
222.22.93.2	199.1.205.23	37654	80
222.22.65.143	203.77.240.43	48712	80

Табл. 8.8. Список контроля доступа для фильтра, учитывающего состояние соединений

Действие	Исходный адрес	Конечный адрес	Протокол	Исходный порт	Конечный порт	Флаговый бит	Проверка соединения
Разрешить	222.22/16	Вне 222.22/16	TCP	>1023	80	Любой	
Разрешить	Вне 222.22/16	222.22/16	TCP	80	>1023	ACK	X
Разрешить	222.22/16	Вне 222.22/16	UDP	>1023	53	–	
Разрешить	Вне 222.22/16	222.22/16	UDP	53	>1023	–	X
Запретить	Все	Все	Все	Все	Все	Все	

Давайте рассмотрим ряд примеров и изучим, как взаимодействуют таблица соединений и расширенный список контроля доступа. Предположим, злоумышленник пытается послать искаженный пакет во внутреннюю сеть организации, и для этого отправляет пакет с TCP-портом назначения 80 и с установленным флагом ACK. Далее предположим, что исходный IP-адрес и номер исходного порта у этого пакета —

150.23.23.155 и 12543 соответственно. Когда этот пакет прибывает на брандмауэр, тот сверяется со списком контроля доступа, приведенным в табл. 8.7. Этот список указывает, что перед допуском пакета в корпоративную сеть также требуется проверить таблицу соединений. Брандмауэр так и делает, определяет, что данный пакет не относится к текущему TCP-соединению и отбрасывает его. Другой пример: предположим, что пользователь внутренней сети хочет выйти на какой-то внешний сайт в Интернете. Поскольку пользователь сначала отсылает сегмент TCP SYN, пользовательское TCP-соединение записывается в таблицу соединений. Когда веб-сервер отправляет пакеты обратно (в этих пакетах должен быть обязательно установлен бит ACK), брандмауэр сверяется с таблицей и убеждается, что соответствующее соединение продолжается. Поэтому он пропускает в сеть такие пакеты, не мешая деятельности пользователя внутренней сети, который ходит по Интернету.

Шлюз приложений

В вышеприведенном примере мы убедились, что фильтрация на уровне пакетов позволяет организации выполнять грубое отсеивание информации на основе содержимого заголовков IP и TCP/UDP. В частности, таким образом можно работать с IP-адресами, номерами портов и битами рукопожатия. Но что делать, если организация планирует предоставить сервис Telnet ограниченному множеству внутренних пользователей (в отличие от IP-адресов)? Что если организация собирается обязать таких привилегированных пользователей сначала проходить аутентификацию, лишь после этого предоставляя им возможность устанавливать Telnet-сеансы с внешним миром? Такие задачи невозможно решить при помощи традиционных фильтров и фильтров, работающих с учетом состояния соединения. Действительно, информация об идентификационных данных внутренних пользователей — это содержимое прикладного уровня, которое не включается в заголовки IP/TCP/UDP.

Для обеспечения более адресной безопасности, брандмауэры должны комбинировать при работе пакетные фильтры и шлюзы приложений. Шлюз приложений просматривает не только заголовки IP/TCP/UDP, и принимает решения о соблюдении политики на основании данных прикладного уровня. **Шлюз приложений** — это сервер, работающий именно на прикладном уровне, и через такой шлюз должны протекать все данные приложений (как входящие, так и исходящие). На одном хосте может работать сразу несколько шлюзов приложений, но каждый шлюз — это самостоятельный сервер со своим собственным набором процессов.

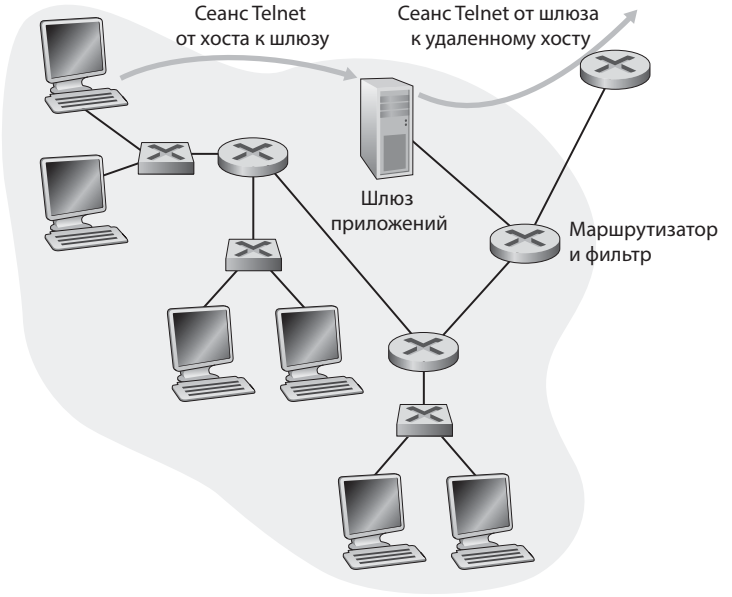


Рис. 8.34. Брандмауэр, состоящий из шлюза приложений и фильтра

Чтобы подробнее разобраться в шлюзах приложений, давайте спроектируем брандмауэр, который позволяет лишь ограниченному множеству внутренних пользователей выходить в большой Интернет по протоколу Telnet, а для клиентов, не относящихся к компании, полностью блокирует возможность таких сеансов. Для воплощения такой политики нам понадобится реализовать комбинацию фильтра пакетов (в маршрутизаторе) и шлюза приложений Telnet, как показано на рис. 8.34. Фильтр маршрутизатора сконфигурирован так, что он блокирует все Telnet-соединения кроме тех, что инициируются с IP-адреса шлюза приложений. Такая конфигурация фильтра приводит к тому, что все исходящие Telnet-соединения обязательно проходят через шлюз приложений. Предположим, что пользователь внутренней корпоративной сети хочет установить Telnet-соединение с каким-либо хостом в большом Интернете. Сначала этот пользователь должен установить Telnet-соединение со шлюзом приложений. Приложение, работающее на шлюзе и слушающее все входящие Telnet-соединения, предлагает пользователю ввести его идентификатор (ID) и пароль. Когда он предоставляет эту информацию, шлюз приложений проверяет, есть ли у данного пользователя право на установление Telnet-соединений с большим Интернетом. При отсутствии такого права шлюз приложений разрывает то Telnet-соединение, которое существует между ним и пользователем.

При наличии права на внешние соединения шлюз (1) предлагает пользователю ввести имя внешнего хоста, с которым предполагается установить соединение, (2) устанавливает Telnet-соединение между собой (шлюзом) и внешним хостом и (3) ретранслирует на внешний хост все данные, поступающие от пользователя, а также в обратном — идущие с внешнего хоста на хост пользователя. Итак, шлюз приложений Telnet не только выполняет авторизацию пользователей, но также совмещает в себе функции сервера и клиента Telnet, осуществляя ретрансляцию данных между пользователем и удаленным Telnet-сервером. Обратите внимание: фильтр допускает этап 2, так как шлюз инициирует Telnet-соединение с внешним миром.

ПРИНЦИПЫ В ДЕЙСТВИИ

Анонимность и приватность

Допустим, вы собираетесь перейти на сайт с неоднозначной репутацией (например, посвященный политическому экстремизму) и при этом (1) не хотите раскрывать свой IP-адрес на этом сайте, (2) хотите скрыть от локального Интернет-провайдера (который обслуживает вашу офисную или домашнюю сеть), что вы посещали этот сайт и (3) не хотите, чтобы локальный Интернет-провайдер мог просматривать те данные, которыми вы обмениваетесь с сайтом. Если вы выходите в Интернет традиционным способом — то есть просто подключаетесь к сайту без какого-либо шифрования, то ни одну из трех вышеперечисленных мер предосторожности соблюсти не удается. Даже если вы работаете с SSL, то все равно не решаете первых двух задач. Во-первых, ваш исходный IP-адрес будет присутствовать абсолютно во всех дейтаграммах, которые вы посылаете в Интернет. Во-вторых, Интернет-провайдер может легко узнать адрес назначения, по которому вы посылаете каждый пакет.

Чтобы добиться приватности и анонимности, можно воспользоваться комбинацией двух средств: доверенного прокси-сервера и SSL, как показано на рис. 8.35. В этом случае вы сначала устанавливаете SSL-соединение с доверенным прокси-сервером. Затем по данному SSL-соединению вы отправляете HTTP-запрос желаемой страницы, которую собираетесь просмотреть на сайте. Когда прокси-сервер получает SSL-зашифрованный HTTP-запрос, он расшифровывает этот запрос и переадресует на сайт HTTP-запрос уже в форме открытого текста. Сайт отвечает прокси-серверу, который, в свою очередь, по SSL пересылает этот отклик вам. Поскольку сайт «видит» лишь IP-адрес прокси-сервера, но не ваш клиентский адрес, вы действительно получили анонимный доступ к этому

сайту. Кроме того, весь трафик между вами и прокси-сервером передается в зашифрованном виде, а это означает, что Интернет-провайдер не может посягнуть на вашу приватность, зарегистрировав у себя посещенный вами сайт или записав данные, которыми вы обмениваетесь с сервером. Сегодня подобные услуги доступа к прокси-серверам предоставляются многими компаниями (в частности, proxify.com).

Разумеется, при таком решении абсолютно вся информация о соединении оказывается в распоряжении прокси-сервера: он знает и ваш IP-адрес, и IP-адрес того сайта, к которому вы обращаетесь. Кроме того, прокси-сервер может просматривать весь трафик, которым вы обмениваетесь с веб-сервером, так как эта информация проходит через него в незашифрованном виде. Следовательно, такое решение целиком и полностью зависит от благонадежности прокси-сервера. Более надежный подход, реализуемый службой TOR (эта компания занимается обеспечением анонимности и приватности в Интернете) связан с передачей вашего трафика через серию не зависящих друг от друга прокси-серверов[638]. В частности, TOR позволяет независимым лицам подключать свои серверы к ее пулу. Когда пользователь подключается к серверу через службу TOR, она случайным образом выбирает из своего пула цепочку из трех прокси-серверов, после чего весь трафик между клиентом и запрошенным веб-сервером начинает передаваться по этой цепочке. Таким образом, исходя из невозможности «сговора» между этими прокси-серверами, никому не известно, с каким именно веб-сервером связывается ваш IP-адрес. Вдобавок, несмотря на то, что между последним прокси-сервером и веб-сервером вся информация передается в незашифрованном виде, последний прокси-сервер не знает, какой клиентский IP-адрес посылает и получает этот текст.

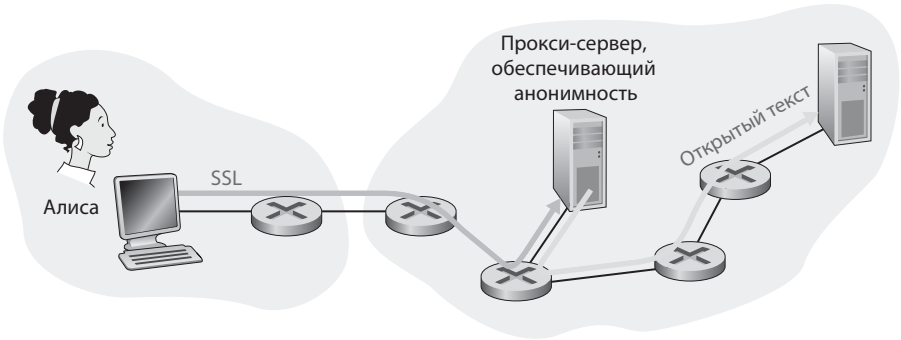


Рис. 8.35. Обеспечение анонимности и приватности при помощи прокси-серверов

Зачастую в корпоративных сетях работает сразу несколько шлюзов приложений: например, по шлюзу для таких видов трафика, как Telnet, HTTP, FTP и электронная почта. Фактически, почтовый сервер организации (см. раздел 2.4) и веб-кэш — это шлюзы приложений.

Разумеется, у шлюзов приложений есть и свои недостатки. Во-первых, для каждого приложения нужен свой шлюз. Во-вторых, использование таких шлюзов негативно сказывается на производительности, так как все данные ретранслируются через шлюз. Эта проблема встает особенно остро, если множество пользователей или приложений одновременно работают с одной машиной-шлюзом. Кроме того, в клиентском ПО должно быть прописано, как связываться со шлюзом (если пользователь выполняет запрос). Наконец, клиентские программы должны правильно сообщать шлюзу приложений, с каким внешним сервером ему следует связаться.

8.9.2. Системы обнаружения вторжений

Выше было рассмотрено, как пакетные фильтры (традиционные и учитывающие состояние соединения) просматривают поля заголовков в протоколах IP, TCP, UDP и ICMP, принимая решение о том, какие пакеты можно пропустить через брандмауэр. Однако для обнаружения многих типов атак нам потребуется выполнять **углубленную проверку пакетов** — то есть анализировать не только поля заголовков, но и сами данные приложений, содержащиеся в пакете. Как было рассказано в разделе 8.9.1, шлюзы приложений зачастую осуществляют углубленную проверку пакетов. Но шлюз решает эту задачу лишь для конкретного приложения.

Разумеется, в данном случае нам пригодилось бы еще одно устройство — такое, которое не только проверяет заголовки всех проходящих через него пакетов (подобно фильтру пакетов), но и выполняет их углубленную проверку (что не под силу фильтру пакетов). Если такое устройство заметит подозрительный пакет либо серию из них, то оно сможет предотвратить попадание таких пакетов в корпоративную сеть. Если же выяснится, что то или иное действие лишь показалось этому устройству подозрительным, оно пропустит пакеты, но отошлет соответствующее уведомление администратору вычислительной сети. Администратор сможет более внимательно проанализировать трафик и принять необходимые меры. Устройство, генерирующее такие уведомления о потенциально опасном трафике, называется **системой обнаружения**

вторжений (intrusion detection system, **IDS**). Похожее устройство, которое отфильтровывает подозрительный трафик, называется **системой предотвращения вторжений** (intrusion prevention system, **IPS**). В этом разделе мы изучим сразу оба класса систем — как IDS, так и IPS — поскольку наиболее интересный технический аспект таких устройств заключается в том, как они обнаруживают подозрительный трафик (а не в том, предупреждают ли они администратора вычислительной сети, либо просто отбрасывают пакеты). Поэтому мы будем говорить обо всех этих устройствах как об IDS — системах обнаружения вторжений.

Система IDS позволяет обнаруживать разнообразные атаки, в частности, трассировку сети (которое может исходить, например, от nmap), сканирование портов, сканирование стеков TCP, атаки отказа в обслуживании, связанные с лавинным переполнением полосы передачи данных (bandwidth flooding), применение червей и вирусов, атаки на уязвимости операционной системы либо отдельных приложений (обзор сетевых атак был сделан в разделе 1.6). В настоящее время системы обнаружения вторжений используются в тысячах организаций. Многие из таких систем являются проприетарными, предоставляются компаниями Cisco, Check Point и другими представителями этого специализированного рынка. Однако на практике также активно применяются разнообразные общедоступные системы такого рода, в частности, исключительно популярная программа Snort IDS (которую мы рассмотрим ниже).

В сети организации могут работать одна или несколько систем обнаружения вторжений. На рис. 8.36 показана организация, в сети которой установлено три IDS-сенсора. При одновременном развертывании нескольких таких сенсоров они обычно взаимодействуют согласованно, пересылая информацию о подозрительном трафике в центральный процессор IDS, который собирает и систематизирует все такие сведения, а также посылает уведомления администратору вычислительной сети, если это представляется необходимым. На рис. 8.36 видим, что организация разделила свою сеть на два сегмента: в первом из них соблюдается высокий уровень безопасности, поддерживаемый при помощи фильтра пакетов и шлюза приложений. Ситуация в этом сегменте также отслеживается сенсорами IDS. Второй сегмент, где уровень безопасности, несколько ниже, именуется **демилитаризованной зоной**. За защиту демилитаризованной зоны отвечает только фильтр пакетов, но в ней также работают сенсоры IDS. Обратите внимание: именно в демилитаризованной зоне установлены серверы организации, обеспечивающие связь с внешним миром — например, общедоступный веб-сервер и полномочный DNS-сервер.

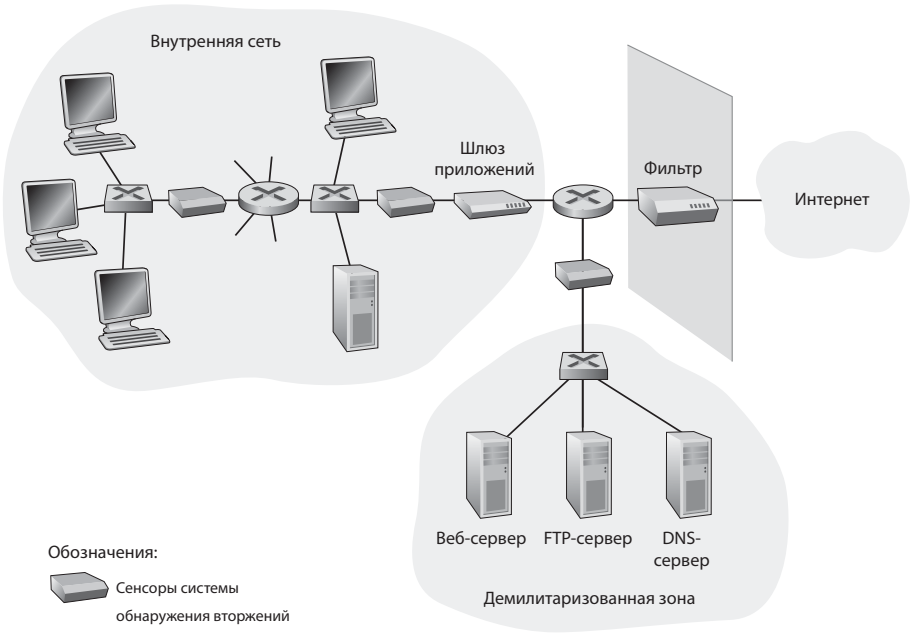


Рис. 8.36. Организация, использующая фильтр пакетов, шлюз приложений и сенсоры систем обнаружения вторжений

На данном этапе может возникнуть вопрос: а зачем нужны несколько сенсоров IDS? Почему бы не установить всего один такой сенсор непосредственно за фильтром пакетов, как показано 8.36 (более того — что мешает нам интегрировать его в такой фильтр)? Вскоре мы убедимся, что система обнаружения вторжений должна не только выполнять углубленную проверку пакетов, но и сравнивать каждый проходящий через нее пакет с десятками тысяч «сигнатур». Это может требовать значительных вычислительных затрат, особенно если организация ежесекундно принимает из Интернета гигабиты трафика. Если же установить сенсоры IDS несколько ниже в организационной сети, каждый из этих сенсоров получит на обработку лишь долю от всего трафика организации, справиться с такой работой будет гораздо легче. Тем не менее сегодня доступны высокопроизводительные системы обнаружения и предотвращения вторжений, и многие организации обходятся всего одним таким сенсором, установленным рядом с маршрутизатором доступа.

Все системы обнаружения вторжений можно разделить на две большие категории: **работающие на основе проверки сигнатур** и **работающие на основе выявления аномалий**. Система обнаружения вторжений, работающая на основе проверки сигнатур, ведет обширную базу данных

сигнатур атак. Каждая сигнатура — это набор правил, описывающих способы борьбы с вторжением того или иного рода. Сигнатура может представлять собой обычный набор характеристик отдельно взятого пакета (например, номера исходного и конечного портов, тип протокола, конкретная последовательность битов, наличие которой проверяется в полезной нагрузке пакета) и относиться не только к отдельному пакету, но и к серии пакетов. Как правило, формированием сигнатур занимаются опытные специалисты по сетевой безопасности, занимающиеся исследованиями известных типов атак. Администратор вычислительной сети организации может корректировать сигнатуры, а также добавлять в базу данных свои варианты сигнатур.

На практике система обнаружения вторжений, работающая на основе сигнатур, анализирует каждый проходящий через нее пакет, сравнивая его содержимое с сигнатурами, которые присутствуют в ее базе данных. Если пакет (или серия пакетов) совпадает с сигнатурой, имеющейся в базе данных, то система обнаружения вторжений генерирует предупреждение. Предупреждение может быть отправлено администратору вычислительной сети в виде электронного сообщения, либо просто записано в журнал для последующей проверки.

Хотя системы обнаружения вторжений, анализирующие сигнатуры, распространены очень широко, им свойственны некоторые ограничения. Важнее всего то, что для генерации точной сигнатуры такая система должна заранее знать о существовании той или иной атаки. Иными словами, подобная система совершенно бессильна против новых атак, которые еще нигде не зарегистрированы. Другой нюанс заключается в том, что совпадение сигнатуры вполне может быть и не связано с атакой — в таком случае, система даст ложноположительный результат. Наконец, поскольку каждый пакет приходится сравнить с огромной коллекцией сигнатур, система обнаружения вторжений может попросту не справиться с такой работой и «прозевать» множество вредоносных пакетов.

Система обнаружения вторжений, работающая на основе выявления аномалий, создает профиль безопасного трафика, наблюдаемого в штатном режиме. Затем она отслеживает такие потоки пакетов, которые обладают статистическими странностями. Например, в потоке может прослеживаться непропорциональное увеличение количества ICMP-пакетов либо внезапный резкий скачок интенсивности сканирования портов или эхо-тестирования адресов. Замечательная черта таких систем обнаружения вторжений заключается в том, что при работе они не зависят от заранее известных сведений о существующих атаках — то

есть, потенциально они способны обнаруживать совершенно новые, еще не описанные атаки. С другой стороны, исключительно сложно различать нормальный и статистически необычный трафик. В настоящее время среди систем обнаружения вторжений преобладают работающие на основе сигнатур, хотя встречаются и такие, которые в той или иной степени занимаются выявлением аномалий.

Система Snort

Snort — это общедоступная свободно распространяемая система обнаружений. В настоящее время развернуты сотни тысяч ее образцов[607, 296]. Snort работает на платформах Linux, UNIX и Windows. Эта система использует универсальный анализирующий интерфейс libcap, также применяемый в инструменте Wireshark и во многих других анализаторах пакетов. Snort легко справляется с объемами трафика порядка 100 Мбит/с. В системах, где трафик измеряется гигабитами в секунду, может потребоваться несколько сенсоров Snort.

Чтобы подробнее познакомиться со Snort, рассмотрим пример сигнатур из этого анализатора:

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any
(msg:"ICMP PING NMAP"; dsize: 0; itype: 8;)
```

Этой сигнатуре соответствует любой ICMP-пакет, входящий в сеть организации (`$HOME_NET`) извне (`$EXTERNAL_NET`), относящийся к типу 8 (ICMP ping) и имеющий пустое поле данных (`dsize: 0`). Поскольку программа nmap (раздел 1.6) генерирует ping-пакеты именно с такими характеристиками, данная сигнатура применяется для обнаружения эхо-тестирующих запросов, поступающих от nmap. Если пакет совпадает с этой сигнатурой, то Snort генерирует предупреждение, в котором содержится сообщение `"ICMP PING NMAP"`.

Вероятно, самая впечатляющая черта Snort заключается в том, какое огромное количество пользователей и экспертов по безопасности занимаются поддержкой базы данных сигнатур этой программы. Как правило, не проходит и нескольких часов после того, как разразится новая атака — а сообщество Snort уже успевает записать и опубликовать ее сигнатуру. Затем эта информация оперативно закачивается на сотни тысяч экземпляров системы Snort, распределенных по всему миру. Более того, работая с синтаксисом сигнатур Snort, администратор вычислительной сети может адаптировать сигнатуры под нужды своей организации, либо модифицируя имеющиеся, либо создавая совершенно новые.

8.10. Заключение

В этой главе были рассмотрены различные способы, при помощи которых наши тайные возлюбленные Боб и Алиса могут безопасно секретничать друг с другом. Мы убедились, что Боб и Алиса заинтересованы в сохранении конфиденциальности своей переписки (чтобы никто, кроме них, не мог узнать ее содержание), им требуется аутентификация конечной точки (чтобы они были уверены, что общаются именно друг с другом) а также целостность сообщения (гарантия, что передаваемые сообщения по пути не будут подвергаться изменениям). Разумеется, секретность коммуникации может понадобиться не только тайным возлюбленным. Как мы убедились в разделах 8.5–8.8, функции безопасности используются на различных уровнях сетевой архитектуры для защиты от злоумышленников, которые атакуют сеть самыми разнообразными способами.

В первой части этой главы были описаны разнообразные принципы, лежащие в основе безопасного обмена информацией. В разделе 8.2 мы рассмотрели криптографические приемы шифрования и дешифрования данных, в частности, криптографию с симметричными ключами и криптографию с открытым ключом. Были изучены алгоритмы DES и RSA — конкретные образцы двух этих обширных классов криптографических техник, применяемых в современных сетях.

В разделе 8.3 мы поговорили о двух подходах к обеспечению целостности сообщения: с применением кодов аутентификации сообщений (MAC) и цифровых подписей. Два этих метода во многом схожи. В обоих случаях используются криптографические хэш-функции, обе техники позволяют проверять источник и целостность сообщения как такового. Важная разница между ними заключается в том, что MAC-коды не основываются на шифровании, тогда как цифровые подписи предполагают наличие инфраструктуры шифрования с открытым ключом. В разделах 8.5–8.8 мы подробно рассмотрели применение обоих этих методов на практике. Кроме того, цифровые подписи используются при создании цифровых сертификатов, что очень важно для верификации подлинности открытых ключей. В разделе 8.4 мы обсудили аутентификацию конечной точки и познакомились с одноразовыми номерами, которые помогают защищаться от атак повторного воспроизведения.

В разделах 8.5–8.8 мы исследовали несколько протоколов, обеспечивающих сетевую безопасность, причем эти протоколы широко используются на практике. Мы убедились, что криптография с открытым

ключом играет важнейшую роль в технологиях PGP, SSL, IPsec и при обеспечении безопасности в беспроводных сетях. Мы узнали, что криптография с открытым ключом имеет принципиальное значение как в PGP, так и в SSL. Как было показано в этой главе, программа PGP обеспечивает целостность сообщения посредством цифровых подписей, тогда как в SSL и IPsec эта задача решается при помощи кодов MAC. Теперь, понимая базовые принципы криптографии и изучив, как они применяются на деле, вы даже можете разрабатывать собственные протоколы сетевой безопасности!

Вооружившись знаниями, приобретенными в разделах 8.2–8.8 этой главы, Боб и Алиса могут не опасаться за конфиденциальность своего общения. Остается надеяться, что наши герои — студенты с факультета информатики, которые прослушали этот курс, и теперь они могут быть уверены, что никакая Мэри не помешает их общению. Но конфиденциальность — лишь один аспект в обширной картине сетевой безопасности. Как было рассказано в разделе 8.9, все более серьезное внимание в архитектуре сетей уделяется защите сетевой инфраструктуры от массированных атак злоумышленников. В последней части главы мы уделили внимание брандмауэрам и системам обнаружения вторжений, проверяющим пакеты, которые входят в корпоративную сеть и покидают ее.

В этой главе было рассмотрено множество теоретических вопросов, затрагивающих наиболее важные аспекты современной сетевой безопасности. Читателям, желающим подробнее разобраться в этой теме, рекомендуем обратиться к книгам, на которые даются ссылки в этой главе. В частности, стоит обратить внимание на следующие труды: работа Скоудиса[603] посвящена атакам и эксплуатационной безопасности, Кауфман[280] пишет о криптографии и ее использовании при обеспечении сетевой безопасности, в работе Рескорлы[415] сделан глубокий, но доходчиво описанный анализ технологии SSL и в работе Эдни[147] подробно обсуждается стандарт безопасности 802.11, есть множество ценных замечаний об алгоритме WEP и его слабых местах.

Глава 9
АДМИНИСТРИРОВАНИЕ ВЫЧИСЛИТЕЛЬНОЙ СЕТИ

Итак, осилив восемь глав этой книги, мы хорошо понимаем, что сеть состоит из *многих* сложных взаимосвязанных аппаратных и программных элементов — каналов, коммутаторов, маршрутизаторов, хостов и прочих устройств, составляющих физические компоненты сети, а также различных протоколов (реализованных как аппаратно, так и программно), осуществляющих управление этими устройствами. Когда сотни или тысячи подобных компонентов собираются вместе, образуя сеть, не удивительно, что отдельные компоненты вдруг начинают неправильно работать, сетевые элементы оказываются настроенными неверно, сетевые ресурсы используются неэффективно, а некоторые компоненты просто ломаются (например, кто-то спотыкается и рвет кабель или опрокидывает на маршрутизатор стакан с газировкой). Администратор вычислительной сети, работа которого заключается в поддержании сети в работоспособном состоянии, должен уметь правильно реагировать на такие несчастные случаи (а еще лучше, не допускать их). При множестве сетевых компонентов, распределенных по большой территории, очевидно, что администратору вычислительной сети необходимы специальные средства, помогающие следить за состоянием сети и управлять ею. В этой главе мы изучим архитектуру, протоколы и информацию, используемые администратором при решении данной задачи.

9.1. Понятие администрирования вычислительной сети

Прежде чем углубиться в вопросы администрирования вычислительной сети, рассмотрим несколько примеров из реального (не сетевого) мира, в которых администратор должен следить за состоянием сложной системы, состоящей из множества взаимодействующих компонентов, контролировать ее работу, управлять ею. На электростанциях есть диспетчерская, где различные циферблаты, датчики и индикаторы показывают состояние (температуру, давление, скорость) расположенных по всей территории клапанов, труб, камер и других компонентов

электростанции, а некоторые из устройств могут даже сигнализировать о неполадках (знаменитые мигающие красные огоньки). Такие устройства помогают оператору контролировать текущее состояние станции и предпринимать те или иные меры в зависимости от ситуации. Подобным же образом различными циферблатами и лампочками оснащена кабина самолета, что позволяет пилоту следить за состоянием узлов самолета и управлять ими. В этих двух примерах администратор отслеживает состояние (осуществляет *мониторинг*) удаленных устройств, *анализирует* показания датчиков, чтобы удостовериться в работоспособности устройств и соответствии их параметров заданным критериям (например, в том, что температура ядерного реактора не превышает определенного предела, или в том, что топлива в самолете достаточно), *оперативно контролирует* систему, то есть реагирует на изменение параметров системы или ее окружения, а также осуществляет *упреждающее управление* системой (например, обнаружив тенденции аномального поведения, предпринимает определенные действия по предотвращению серьезной проблемы). Аналогичным образом администратор вычислительной сети следит за состоянием доверенной ему системы, контролирует ее и управляет ею.

На заре развития сетевой инфраструктуры, когда компьютерные сети представляли собой скорее объект исследования, нежели собственно инфраструктуру, которой пользуются миллионы людей в день, об администрировании вычислительной сети никто не слыхал. Если в сети возникала проблема, ее источник обнаруживался при помощи нескольких команд ping, после чего пользователь, обнаруживший проблему, мог сам изменить параметры, перезагрузить аппаратуру или программное обеспечение, вызвать удаленного коллегу и попросить его сделать это. (Очень занимательное обсуждение первого крупного «краха» сети ARPAnet, случившегося 27 октября 1980 года, задолго до того как появились инструменты администрирования вычислительной сети, а также усилий по устранению неисправности и попыток разобраться в причинах «краха», приводится в документе RFC 789[418].) По мере развития Интернета и превращения частных внутренних сетей в огромные глобальные структуры потребность в систематическом управлении огромным количеством аппаратных и программных компонентов этих сетей становилась все насущнее.

Начнем наше изучение администрирования вычислительной сети с простого примера. На рис. 9.1 показана небольшая сеть, состоящая из трех маршрутизаторов и нескольких хостов и серверов.

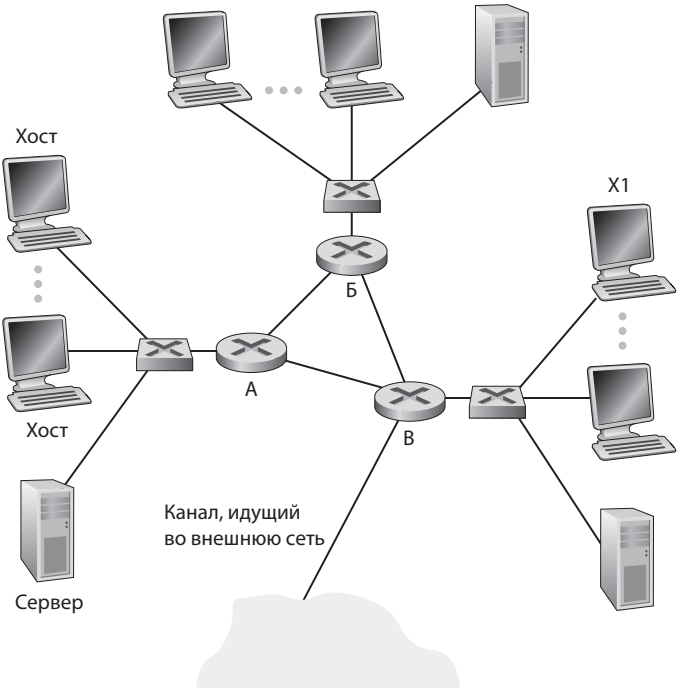


Рис. 9.1. Простой сценарий, демонстрирующий практическое применение администрирования вычислительной сети

Даже в такой простой сети возможны ситуации, в которых соответствующие инструменты сетевого управления окажутся необходимы администратору вычислительной сети.

- *Обнаружение неисправности интерфейсной карты хоста или маршрутизатора.* С помощью соответствующих инструментов сетевого управления сетевое устройство (например, маршрутизатор А) может сообщить администратору о том, что один из его интерфейсов вышел из строя. (Разумеется, это предпочтительнее телефонного звонка в сетевой операционный центр от разгневанного пользователя, сообщающего о том, что сетевое соединение не работает!) Администратор вычислительной сети, активно отслеживающий и анализирующий сетевой трафик, может обнаружить неисправность интерфейсной карты и заменить ее еще *до того*, как она окончательно выйдет из строя. Так, например, администратор может заметить увеличение ошибок контрольной суммы в кадрах, посылаемых «умирающим» интерфейсом.
- *Мониторинг хостов.* Администратор вычислительной сети может периодически проверять состояние подключенных к сети хостов.

И в этом случае он способен обнаружить проблему прежде, чем она станет заметна пользователю.

- *Мониторинг трафика с целью контроля над распределением ресурсов.* Администратор вычислительной сети может отслеживать характер сетевого трафика и заметить, например, что трафик, пересекающий множество сегментов локальных сетей, существенно снижается, если соединить определенные серверы напрямую. Представьте, как обрадуются пользователи сетей (особенно руководители высшего звена), если увеличения производительности удастся добиться без дополнительных капиталовложений. Аналогично, отслеживая коэффициент использования каналов, администратор вычислительной сети может определить наличие перегрузки в сегменте локальной сети или канале связи с внешним миром, в связи с чем потребуется установить линию с большей пропускной способностью (увы, дополнительные расходы). Администратор также может настроить программу мониторинга, чтобы та автоматически уведомляла его о превышении определенного уровня загрузки каналов — это даст возможность заранее подготовиться к высоким уровням загрузки.
- *Обнаружение быстрых изменений в таблицах маршрутизации.* Частые изменения в таблицах маршрутизации (так называемая нестабильность маршрута, route flapping) указывают на нестабильность маршрутов или на неверно настроенный маршрутизатор. Разумеется, администратор вычислительной сети предпочтет сам обнаружить ошибку, прежде чем сеть окажется неработоспособной.
- *Мониторинг уровней обслуживания.* **Соглашения об уровне обслуживания** (Service Level Agreement, **SLA**), определяют специфические параметры производительности и соответствующие им приемлемые показатели работы сети[214]. Verizon и Sprint — всего лишь два из множества провайдеров, предоставляющих гарантии соблюдения соглашений SLA своим клиентам[29, 647]. Эти соглашения включают параметры доступности службы, задержки, пропускной способности и требования к уведомлению о простоях. Очевидно, что если критерии производительности должны входить в соглашение об обслуживании между сетевым провайдером и его клиентами, тогда измерение и контроль производительности являются важной задачей администратора вычислительной сети.
- *Обнаружение вторжения.* Вероятно, администратор вычислительной сети хотел бы знать о случаях проникновения сетевого трафика от подозрительного источника (например, хоста или номера порта)

или к подозрительному получателю. Кроме того, администратор, очевидно, хотел бы иметь возможность обнаруживать (и во многих случаях фильтровать) определенные типы трафика (например, пакеты с внутренней маршрутизацией от источника или большое количество SYN-пакетов, направляющихся к определенному хосту), что может свидетельствовать об атаках на систему безопасности (см. главу 8).

Международная организация по стандартизации (International Organization for Standardization, ISO) разработала модель администрирования вычислительной сети. Были определены пять областей администрирования вычислительной сети.

- *Управление производительностью*. Цель заключается в том, чтобы измерить производительность, сообщить о ней, проанализировать полученные данные и предпринять необходимые действия по поддержанию определенного уровня производительности (например, коэффициента использования или пропускной способности) в различных сетевых компонентах. Этими компонентами могут быть конкретные устройства (например, линии связи, маршрутизаторы, хосты) или абстракции (например, сетевые маршруты). Вскоре мы увидим, что центральную роль в управлении производительностью в Интернете играют стандарты протоколов, таких как SNMP (Simple Network Management Protocol — простой протокол сетевого администрирования)[508].
- *Контроль неисправностей*. Цель заключается в обнаружении неисправностей, их регистрации и принятия соответствующих ответных мер. Связь между контролем неисправностей и контролем производительности довольно расплывчата. Мы можем понимать контроль неисправностей как немедленное исправление небольших сетевых поломок (например, аппаратных или программных сбоев), тогда как задача управления производительностью состоит в долгосрочном обеспечении ее приемлемых уровней, несмотря на изменяющиеся требования трафика и выход из строя сетевых устройств. Как и при управлении производительностью, в контроле неисправностей центральную роль играет протокол SNMP.
- *Управление конфигурацией*. Позволяет администратору определить, какие устройства входят в администрируемую сеть, а также аппаратную и программную конфигурацию этих устройств. Обсуждение управления конфигурацией и требования к IP-сетям можно найти в спецификации RFC 3139[495].

- Выделение сетевых ресурсов. Позволяет администратору указать правила доступа для пользователя или устройства к сетевым ресурсам, регистрировать запросы доступа, а также контролировать его. К работе с ресурсами также относятся квоты пользователей, взимание с них платы, объем которой зависит от используемых ими ресурсов, а также предоставление пользователям привилегий на доступ к ресурсам.

- Управление безопасностью. Цель заключается в контроле доступа к сетевым ресурсам в соответствии с некоторой политикой. Компонентами управления безопасностью являются центры распределения ключей и сертификационные центры, рассматривавшиеся в разделе 8.3. Для мониторинга и контроля доступа к сетевым ресурсам извне используются брандмауэры, обсуждавшиеся в разделе 8.9.

В этой главе мы изучим основы администрирования вычислительной сети. Мы намеренно сужаем сферу рассматриваемых вопросов, ограничиваясь только *инфраструктурой* администрирования вычислительной сети — общей архитектурой, протоколами сетевого управления и информационной базой, позволяющей администратору вычислительной сети поддерживать сеть в рабочем состоянии. Мы *не* описываем сам процесс принятия решений администратором вычислительной сети, призванный помочь ему планировать, анализировать информацию, поступающую в сетевой операционный центр и реагировать на нее. К этой области относятся такие темы, как обнаружение и контроль неисправностей[302, 279, 341, 615, 158, 674, 633], профилактическое обнаружение аномалий[41, 306, 307] и др. Мы также не станем затрагивать более широкую тему управления службами[585, 494]. Она сводится к предоставлению необходимых ресурсов: полосы передачи данных, серверных мощностей, других коммуникационных и вычислительных ресурсов, необходимых для выполнения критически важных эксплуатационных требований на конкретном предприятии.

Итак, что такое администрирование вычислительной сети? Выше мы обсудили необходимость администрирования вычислительной сети и привели несколько примеров. Мы завершим этот раздел определением администрирования вычислительной сети из публикации Сейдема[585].

> *Администрирование вычислительной сети включает развертывание, интеграцию и координацию аппаратуры, программного обеспечения и людей для мониторинга, тестирования, опроса, конфигурирования, анализа, оценки и контроля сети и ресурсов для удовлетворения требований работы в реальном времени, производительности и качества обслуживания за приемлемую цену.*

Это пространное, но хорошее рабочее определение. В следующих разделах мы постараемся «нарастить плоть» на его «скелет».

9.2. Инфраструктура администрирования вычислительной сети

В предыдущем разделе было показано, что для администрирования вычислительной сети необходима возможность «мониторинга, тестирования, опроса, конфигурации... и контроля» аппаратных и программных компонентов сети. Поскольку сетевые устройства распределены на большой территории, для этого как минимум требуется, чтобы администратор вычислительной сети мог собирать данные с удаленного объекта (например, для мониторинга) и производить необходимые изменения на этом удаленном объекте (например, контролировать его). Чтобы лучше понять инфраструктуру администрирования вычислительной сети, рассмотрим аналогию из мира людей.

Представьте, что вы руководите большой организацией с разветвленной сетью офисов по всему миру. Ваша работа состоит в том, чтобы гарантировать, что все фрагменты вашей организации работают слаженно друг с другом. Как этого добиться? Как минимум вы должны периодически собирать информацию из ваших филиалов в виде отчетов и различных численных данных, отражающих активность, производительность и бюджет. Иногда (но не всегда) вас будут явно уведомлять о наличии проблем в одном из филиалов. Управляющий филиалом, желающий продвинуться вверх по служебной лестнице (возможно, нацеливаясь на ваше место), может по собственной инициативе послать вам сообщение о том, как замечательно все работает в его офисе. Вы просматриваете получаемые сообщения, надеясь узнать, что все работает прекрасно, но, без сомнения, обнаружите проблемы, требующие вашего внимания. Вы можете начать диалог с одним из ваших филиалов, собрать больше информации, чтобы понять проблему, а затем передать менеджеру филиала нужную директиву.

В этом общем сценарии неявно задействована инфраструктура контроля организации — директор (вы), удаленные контролируемые сайты (филиалы), ваши удаленные агенты (менеджеры филиалов), протоколы связи (для передачи стандартных отчетов и диалогов) и данные (содержимое отчетов и численные данные, отражающие активность, производительность и бюджет). У каждого из этих компонентов есть аналоги в компьютерных сетях.

Архитектура сетевой системы управления концептуально идентична этому простому примеру. В сетевом управлении используется своя специфическая терминология. Как показано на рис. 9.2, архитектура сетевого управления состоит из трех основных компонентов: управляющего объекта (директор), управляемых устройств (филиалов) и протокола сетевого управления.

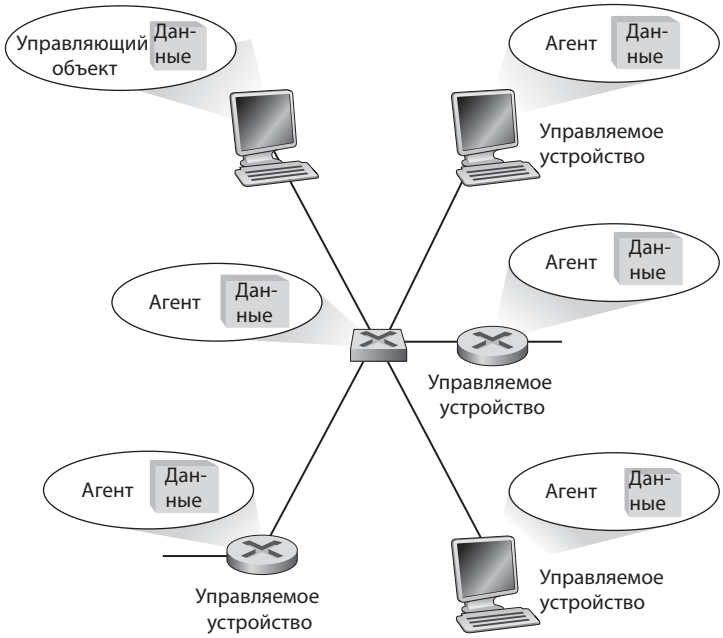


Рис. 9.2. Основные компоненты в архитектуре управления сетью

Управляющий объект представляет собой приложение, в работе которого, как правило, принимает участие человек, сидящий за терминалом в сетевом операционном центре. Управляющий объект — сердце сетевого управления. Он контролирует сбор, обработку, анализ и/или отображение сетевой информации. Именно здесь инициируются действия по управлению работой сети, и здесь администратор (человек) взаимодействует с сетевыми устройствами.

Управляемое устройство представляет собой часть сетевого оборудования (включая программное обеспечение), располагающееся в управляемой сети. Это филиал в нашей аналогии. Управляемым устройством может быть хост, маршрутизатор, мост, концентратор, принтер или модем. В управляемом устройстве может быть несколько так называемых

управляемых объектов. Они являются физическими фрагментами управляемых устройств (такова, например, сетевая интерфейсная карта), а также наборами настраиваемых параметров для этих аппаратных и программных фрагментов (таков, например, протокол внутренней маршрутизации, RIP). В нашей аналогии из человеческого мира управляемые объекты могут быть отделами филиала. С этими управляемыми объектами ассоциированы определенные данные, объединяемые в **базу управляющей информации** (Management Information Base, **MIB**). Мы увидим, что эта информация доступна (и во многих случаях может задаваться) управляющим объектом. В нашей корпоративной аналогии базе управляющей информации соответствуют количественные данные (измерения активности, производительности, финансовая информация, при этом управляющий объект может активно влиять на финансовые данные!), которыми обмениваются главный офис и филиалы. Более подробно мы рассмотрим базу управляющей информации в разделе 9.3. Наконец, в каждом управляемом устройстве находится **агент сетевого управления**, то есть процесс, работающий в управляемом устройстве и непосредственно взаимодействующий с управляющим объектом. В нашей аналогии из мира людей агенту сетевого управления соответствует менеджер филиала.

Третий фрагмент нашей архитектуры сетевого управления представляет собой **протокол сетевого управления**. Этот протокол работает между управляющим объектом и управляемыми устройствами, позволяя управляющему объекту запрашивать состояние этих устройств и неявно предпринимать действия по управлению ими через его агентов. Агенты могут использовать протокол сетевого управления для информирования управляющего объекта об исключительных ситуациях (например, о выходе из строя определенного компонента или о нарушении показателей производительности). Важно отметить, что протокол сетевого управления сам по себе не управляет сетью, а является инструментом, с помощью которого администратор вычислительной сети может это делать (осуществлять мониторинг, тестирование, опрос, конфигурирование, анализ, оценку и контроль). Данное различие неочевидно, но важно.

Хотя инфраструктура сетевого управления концептуально проста, часто возникает путаница с его терминологией, то есть такими понятиями, как «управляющий объект», «управляемое устройство», «управляющий агент», «база управляющей информации». Например, в нашем простом сценарии «управляющие агенты», располагающиеся на «управляемых устройствах», периодически опрашиваются «управляемым

устройством» — идея проста, но язык можно сломать! В любом случае корпоративная аналогия пригодится нам здесь и далее в этой главе.

Выше мы в общем виде обсудили целостную архитектуру сетевого управления, которая применима ко многим стандартам. Стандарты сетевого управления начали развиваться в конце 80-х годов после запуска международной программы стандартизации обмена данными между компьютерными системами различных производителей под названием OSI (Open System Interconnection — взаимодействие открытых систем), в рамках которой были приняты такие стандарты, как **CMISE** (Common Management Information Services Element — **служба общей управляющей информации**) и **CMIP** (Common Management Information Protocol — **протокол общей управляющей информации**)[398, 613, 183], а также интернет-протокол **SNMP** (Simple Network Management Protocol — **простой протокол сетевого администрирования**)[508, 614, 571]. Стандарты CMISE/CMIP и SNMP оказались наиболее важными[625]. Оба они ориентированы на универсальную работу с продуктами и сетями и не зависят от производителя. Поскольку протокол SNMP был разработан и внедрен, когда необходимость в сетевом управлении стала очевидной, он быстро получил широкое распространение. Сегодня он стал наиболее популярным и широко применяемым протоколом в архитектуре сетевого управления. Мы подробно рассмотрим его в следующем разделе.

ПРИНЦИПЫ В ДЕЙСТВИИ

Сетевой операционный центр Comcast

Компания Comcast располагает первоклассной глобальной оптоволоконной IP-сетью и предоставляет комплекс продуктов и услуг для 49 миллионов пользователей различных сервисов — по работе с видео, данными и голосовыми приложениями. Сеть Comcast включает в себя более 990 тыс. километров заводских сетей, более 220 тыс. километров оптоволоконных сетей, около 50 тыс. километров магистральных сетей, 122 тыс. оптических узлов, а также огромные массивы накопителей для хранения данных в сети доставки контента (CDN) Comcast. В частности, сервис предоставления видео по требованию (Video on Demand) оперирует более чем 134 Тб данных. Работа всех компонентов сети Comcast, вплоть до домашних и рабочих компьютеров пользователей, отслеживается в сетевых операционных центрах этой организации.

Comcast эксплуатирует два Национальных Сетевых Операционных Центра, которые управляют национальной магистральной сетью,

региональными сетями, национальными приложениями и конкретными платформами, поддерживающими инфраструктуру для работы с данными, видео и голосовой связью для индивидуальных, коммерческих и оптовых клиентов. Comcast располагает тремя филиальными операционными центрами, управляющими локальной инфраструктурой, которая поддерживает всех клиентов. Как национальные, так и филиальные операционные центры занимаются упреждающим наблюдением за производительностью всех сетей и служб круглосуточно и без выходных. Для этого применяются стандартные системы и методы. Так, различным событиям в сетях национального и локального уровня назначены заранее определенные уровни опасности, соответствующие процедуры и ожидаемое время восстановления. Национальные и филиальные центры могут подстраховывать друг друга, если в каком-то регионе возникает проблема, затрудняющая эксплуатацию сети. Кроме того, между национальными и филиальными операционными центрами развернута обширная виртуальная частная сеть, позволяющая инженерам безопасно выходить в сеть и удаленно выполнять упреждающие или ремонтные сетевые работы.

Модель управления сетями, принятая в компании Comcast, включает в себя все пять основных аспектов: управление производительностью, контроль неисправностей, управление конфигурацией, управление учетными записями и управление безопасностью.

На этих экранах отображаются используемые в Comcast инструменты для поддержки корреляции, управления пороговыми значениями и учета (оформления) неисправностей. Опубликовано с разрешения Comcast.

Основной задачей **управления производительностью** является понимание того, как работают сети/системы и приложения (такая структура собирательно именуется термином «экосистема») в тех условиях, которые задаются в зависимости от времени суток, дня

недели или особых событий (например, скачки напряжения из-за грозы или всплески платежей — таких как активная закупка билетов перед футбольным матчем). Такие заранее заданные меры по управлению производительностью действуют по всему пути предоставления сервиса, от домашнего или рабочего компьютера клиента по всей сети, а также на интерфейсных точках, связывающих пользователя с партнерами и коллегами. Кроме того, выполняются искусственные транзакции, призванные обеспечить стабильную безотказную работу всей экосистемы. **Управление неисправностями** определяется как способность обнаружения, регистрации и понимания тех аномалий, которые могут негативно сказаться на работе пользователей. Comcast использует механизмы корреляции событий, чтобы адекватно оценивать степень серьезности инцидента и правильно на него реагировать, исключая или устраняя потенциальные проблемы прежде, чем они станут заметны пользователям. **Управление конфигурацией** должно гарантировать, что на всех компонентах экосистемы устанавливается подходящее аппаратное и программное обеспечение. Поддерживая эти компоненты на пиковом «золотом» уровне производительности, удается избегать нежелательных последствий. Выделение сетевых ресурсов должно гарантировать, что в операционном центре существует четкое понимание того, как экосистема снабжается ресурсами, и как они используются. Это особенно важно для того, чтобы в любой момент времени операционный центр мог рационально перенаправлять трафик. **Управление безопасностью** гарантирует наличие необходимых средств контроля, обеспечивающих эффективную защиту экосистемы от нежелательного доступа.

Сетевые операционные центры и поддерживаемые ими экосистемы не статичны. Технический персонал таких центров постоянно пересматривает и заново оценивает действующие меры обеспечения производительности и применяемые для этого инструменты, чтобы оправдывать самые высокие запросы пользователей к качеству обслуживания.

9.3. Архитектура управляющих Интернет-стандартов

Аббревиатура SNMP (Simple Network Management Protocol) означает «простой протокол сетевого администрирования». Но на практике администрирование вычислительной сети в Интернете представляет собой нечто большее, чем просто протокол для перемещения данных между управляющим объектом и его агентами, и нечто гораздо более

сложное. Современная архитектура управляющих Интернет-стандартов (Internet-Standard Management Framework) обязана своим происхождением протоколу SGMP (Simple Gateway Monitoring Protocol — простой протокол мониторинга шлюза)[429]. Протокол SGMP был разработан группой университетских исследователей, пользователей и администраторов, чей опыт работы позволил спроектировать, реализовать и внедрить протокол SNMP всего за несколько месяцев[329]. Не сравнить с сегодняшними днями, когда процессы стандартизации стали такими длительными и сложными. С тех пор сменилось уже три версии протокола SNMP. Третья версия SNMPv3 была выпущена в апреле 1999 года и обновлена в декабре 2002 года[508].

При описании любой архитектуры администрирования вычислительной сети необходимо ответить на определенные вопросы.

- За чем (с семантической точки зрения) следует следить? Какую форму контроля должен применять администратор вычислительной сети?
- В какой форме должна передаваться информация в отчетах и директивах?
- Какой протокол связи должен использоваться для обмена этой информацией?

Вспомним нашу аналогию из предыдущего раздела. Директор и менеджеры филиалов должны договориться о единицах измерения активности, производительности и бюджета, используемых в отчетах о состоянии филиалов. Кроме того, им следует согласовать меры, которые может принимать директор (например, урезать бюджет, приказать менеджеру изменить определенный аспект работы филиала или уволить сотрудников и закрыть филиал). На более низком уровне следует договариваться о форме, в которой будут передаваться отчеты. Например, в какой валюте (в долларах, евро) будут сообщаться финансовые данные? В каких единицах будет измеряться производительность? Наконец, следует определить способ, которым должна переправляться информация между головным офисом и филиалами (то есть протокол связи).

На эти вопросы отвечает архитектура управляющих Интернет-стандартов. Она состоит из четырех частей.

- Определения *объектов администрирования вычислительной сети*, называемых *MIB-объектами*. В архитектуре управляющих Интернет-стандартов управляющая информация представляется как мно-

жество объектов, которые вместе образуют виртуальное хранилище информации, называемое базой управляющей информации (MIB). MIB-объект может быть счетчиком, например, измеряющим количество IP-дейтаграмм, отброшенных маршрутизатором из-за ошибок в заголовке, или количество ошибок опроса несущей в интерфейсной Ethernet-карте; описательная информация, например версия программного обеспечения, работающего на DNS-сервере; информация о состоянии устройства (например, правильно ли оно функционирует) или такая специфическая для протокола информация, как маршрут до получателя. Таким образом, MIB-объекты определяют информацию управления, поддерживаемую управляемым устройством. Связанные MIB-объекты объединяются в **MIB-модули**. В нашей корпоративной аналогии база управляющей информации определяет информацию, переправляемую между филиалом и главным офисом.

- *Язык определения данных*, называемый структурой управляющей информации (Structure of Management Information, SMI), определяет типы данных, модель объектов и правила для записи и изменения управляющей информации. На этом языке описываются MIB-объекты. В нашей аналогии язык SMI используется для определения деталей формата обмениваемых данных.

- *Протокол SNMP* служит для передачи данных и команд между управляющим объектом и агентом, работающим от имени этого объекта в управляемом сетевом устройстве.

- *Функции безопасности и управления*. Добавление этих функций отличает протокол SNMPv3 от протокола SNMPv2.

Таким образом, архитектура управляющих Интернет-стандартов состоит из отдельных модулей с независимыми от протокола языком определения данных и базой управляющей информации, а также с протоколом, независимым от MIB. Интересно, что эта модульная архитектура изначально была разработана, чтобы облегчить переход от администрирования вычислительной сети, основанного на протоколе SNMP, к конкурирующей с этим стандартом архитектуре управляющих Интернет-стандартов, разрабатываемой Международной организацией по стандартизации (ISO). Этот переход так и не состоялся. Однако со временем модульность протокола SNMP позволила ему пройти через три ревизии, при этом были усовершенствованы все четыре составляющие протокола SNMP, независимо друг от друга. Таким образом, реше-

ние о модульном устройстве протокола SNMP было верным, несмотря на то что изначальная цель, для которой оно выбиралось, оказалась ложной.

В следующем разделе мы более подробно рассмотрим четыре основных компонента архитектуры управляющих Интернет-стандартов.

9.3.1. Структура управляющей информации: SMI

Структура управляющей информации (Structure of Management Information, **SMI**) — это язык (с весьма странным названием), используемый для описания управляющей информации, хранящейся в объекте управляемой сети. Этот язык определений призван гарантировать, что синтаксис и семантика данных администрирования вычислительной сети хорошо определены и непротиворечивы.

Обратите внимание, что SMI определяет не конкретный элемент данных в объекте управляемой сети, а язык, на котором описывается подобная информация. Документация на язык SMI для протокола SNMPv3 (которая почему-то называется SMIv2, что вносит дополнительную путаницу) имеется в стандартах RFC 2578[479], RFC 2579[480] и RFC 2580[481]. Рассмотрим структуру управляющей информации снизу вверх, начав с основных типов данных SMI, а затем поговорим о том, как управляемые объекты описываются в SMI, а также как связанные управляемые объекты группируются в модули.

Основные типы данных SMI

В стандарте RFC 2578 определены основные типы данных языка определения MIB-модулей, то есть языка SMI. Язык SMI базируется на языке ASN.1 (Язык абстрактного синтаксиса данных-1)[248], предназначенном для описания объектов — о нем рассказывается в разделе 9.4. Но благодаря введению некоторых специфических типов данных SMI можно рассматривать как вполне самостоятельный язык. В табл. 9.1 перечислены 11 основных типов данных, определенных в RFC 2578. Кроме этих скалярных объектов на множество MIB-объектов также можно наложить табличную структуру с помощью конструкции SEQUENCE OF (подробности см. в RFC 2578). Большая часть представленных в табл. 9.1. типов данных должна быть знакома читателям. Более детально мы рассмотрим тип данных OBJECT IDENTIFIER, используемый для именования объекта.

Табл. 9.1. Основные типы данных языка SMI

Тип данных	Описание
INTEGER	32-разрядное целое, как определено в ASN.1; значения могут изменяться в диапазоне от -2^{31} до $2^{31}-1$ или значение из набора именованных констант
Integer32	32-разрядное целое; значения могут изменяться в диапазоне от -2^{31} до $2^{31}-1$ включительно
Unsigned32	32-разрядное целое без знака; значения могут изменяться в диапазоне от 0 до $2^{32}-1$ включительно
OCTET STRING	Байтовая строка формата ASN.1 длиной до 65 535 байт, содержащая произвольные двоичные или текстовые данные
OBJECT IDENTIFIER	Список (структурированное имя) целых чисел формата ASN. 1 (см. подраздел 9.3.2.)
IPaddress	32-разрядный Интернет-адрес, с тем порядком следования байтов, в котором они используются в сети
Counter32	32-разрядный циклический счетчик, увеличивающийся от 0 до $2^{32}-1$
Counter64	64-разрядный циклический счетчик
Gauge32	Целое 32-разрядное число без знака, не переходящее через 0
TimeTicks	Время, измеряемое в сотых долях секунды, начиная от некоторого момента
Opaque	Неинтерпретируемая строка формата ASN.1, используемая для обратной совместимости

Высокоуровневые конструкции SMI

Помимо основных типов данных язык определения данных SMI также предоставляет высокоуровневые конструкции языка.

Конструкция OBJECT TYPE используется для обозначения типа данных, состояния и семантики управляемого объекта. В подобных управляемых объектах содержатся данные, составляющие ядро администрирования вычислительной сети. В различных документах RFC определено около 10 000 объектов[508]. Конструкция OBJECT-TYPE состоит из четырех предложений. Предложение SYNTAX конструкции OBJECT-TYPE описывает тип основных данных, ассоциированный с объектом. Предложение MAX-ACCESS показывает, доступен ли управляемый объект для чтения, записи, создания и т. д. Предложение STATUS показывает, является ли определение объекта действительным, устаревшим (в этом случае оно не должно исполняться и остается только для истории) или частично устаревшим (в этом случае оно может исполняться для совместимости со старыми реализациями). Пред-

ложение DESCRIPTION содержит текстовое описание объекта, «документирующее» его назначение. Это описание должно предоставлять всю семантическую информацию, необходимую для управления объектом.

В качестве примера конструкции OBJECT-TYPE рассмотрим определение типа объекта `ipSystemStatsInDelivers` из стандарта RFC 4293[533]. Этот объект определяет 32-разрядный счетчик, учитывающий количество IP-дейтаграмм, полученных управляемым устройством и успешно доставленных протоколу верхнего уровня. Последняя строка определения касается имени объекта (это мы обсудим в следующем разделе).

```
ipSystemStatsInDelivers OBJECT-TYPE
  SYNTAX Counter32
  MAX-ACCESS read-only
  STATUS current
  DESCRIPTION
    "The total number of datagrams successfully
    delivered to IPuser-protocols (including ICMP).
    When tracking interface statistics, the counter
    of the interface to which these datagrams were
    addressed is incremented. This interface might
    not be the same as the input interface for
    some of the datagrams.
    Discontinuities in the value of this counter can
    occur at re-initialization of the management
    system, and at other times as indicated by the
    value of ipSystemStatsDiscontinuityTime."
::= { ipSystemStatsEntry 18 }
```

Конструкция MODULE-IDENTITY позволяет объединять связанные объекты в «модули». Например, в стандарте RFC 4293[533] описывается MIB-модуль, определяющий управляемые объекты (включая `ipSystemStatsInDelivers`) для управляющих реализаций протокола IP и связанного с ним протокола ICMP. В стандарте RFC 4022[526] описывается MIB-модуль для протокола TCP, а в стандарте RFC 4113[527] — для протокола UDP. MIB-модуль для удаленного мониторинга (Remote MONitoring, RMON) описывается в стандарте RFC 4502[541]. Наряду

с определениями управляемых объектов OBJECT-TYPE в модуле конструкция MODULE-IDENTITY содержит предложения для документирования сведений об авторе модуля, даты последнего обновления, истории обновлений и текстового описания модуля. Для примера рассмотрим определение модуля для управления протоколом IP:

```
ipMIB MODULE-IDENTITY
  LAST-UPDATED "200602020000Z"
  ORGANIZATION "IETF IPv6 MIB Revision Team"
  CONTACT-INFO
    "Editor:
    Shawn A. Routhier
    Interworking Labs
    108 Whispering Pines Dr. Suite 235
    Scotts Valley, CA 95066
    USA
    EMail: <sar@iwl.com>"
  DESCRIPTION
    "The MIB module for managing IP and ICMP
    implementations, but excluding their
    management of IP routes.
    Copyright (C) The Internet Society (2006).
    This version of this MIB module is part of
    RFC 4293; see the RFC itself for full legal
    notices."
  REVISION   "200602020000Z"
  DESCRIPTION
    "The IP version neutral revision with added
    IPv6 objects for ND, default routers, and
    router advertisements. As well as being the
    successor to RFC 2011, this MIB is also the
    successor to RFCs 2465 and 2466. Published
    as RFC 4293."
  REVISION   "199411010000Z"
  DESCRIPTION
```

```
    "A separate MIB module (IP-MIB) for IP and
    ICMP management objects. Published as RFC
    2011."
  REVISION   "199103310000Z"
  DESCRIPTION
    "The initial revision of this MIB module was
    part of MIB-II, which was published as RFC
    1213."
::= { mib-2 48}
```

Конструкция NOTIFICATION-TYPE применяется для спецификации информации, касающейся сообщений «SNMPv2-Trap» и «InformationRequest», генерируемых агентом или управляющим объектом (см. подраздел 9.3.3). Эта информация включает текстовое описание (предложение DESCRIPTION) того, когда могут посылаться подобные сообщения, а также список значений, которые должны включаться в генерируемое сообщение (подробности см. в спецификации RFC 2578[479]). Конструкция MODULE-COMPLIANCE определяет набор управляемых объектов в модуле, который должен быть реализован агентом. Конструкция AGENT-CAPABILITIES описывает возможности агента в отношении определений объекта или уведомлений о событиях.

9.3.2. База управляющей информации (MIB)

Ранее уже отмечалось, что **базу управляющей информации** (Management Information Base, **MIB**) можно рассматривать как виртуальное хранилище управляемых объектов, значения которых совместно отражают текущее состояние сети. Эти значения могут запрашиваться и/или устанавливаться управляющим объектом при помощи SNMP-сообщений, посылаемых агенту, работающему на управляемом устройстве от имени управляющего объекта. Управляемые объекты специфицируются при помощи обсуждавшейся выше конструкции OBJECT-TYPE языка SMI и объединяются в **MIB-модули** с помощью конструкции MODULE-IDENTITY.

Стандарты MIB-модулей для маршрутизаторов, хостов и другого сетевого оборудования были разработаны группой IETF. В MIB-модули должны входить основные идентификационные данные об определенном фрагменте аппаратуры, а также управляющая информация о сете-

вых интерфейсах и протоколах устройства. По состоянию на 2006 год существовало более 200 стандартных и еще больше нестандартных (фирменных) MIB-модулей отдельных производителей. При таком количестве стандартов группе IETF требовался способ идентификации и именования стандартных MIB-модулей, а также конкретных управляемых объектов внутри модулей. Вместо того чтобы начинать с нуля, группа IETF воспользовалась стандартизированной структурой идентификации (именования) объектов, уже разработанной международной организацией по стандартизации (ISO). Как это часто случается, у ISO были «большие планы» относительно стандартизированной структуры идентификации объектов. ISO намеревалась идентифицировать все стандартизированные объекты (например, форматы данных, протоколы или фрагменты информации) всех сетей, независимо от сетевых стандартов (например, IETF, ISO, IEEE или ANSI), производителя оборудования или владельца сети. Цель была действительно грандиозная! Разработанная ISO структура идентификации объектов является частью языка для описания объектов ASN. 1[248], которому посвящен раздел 9.4. В этой всеобъемлющей структуре именования у стандартизированных MIB-модулей есть свой уютный уголок, как будет показано ниже.

Как видно на рис. 9.3, объекты в структуре именования ISO именуются иерархическим образом. Обратите внимание, что каждой ветви дерева присвоено как имя, так и номер (показанный в скобках). Таким образом, каждый узел дерева идентифицируется последовательностью имен или чисел, определяющих путь от корня дерева к его узлу. Интересная, но несколько недоработанная и неофициальная утилита для обхода путей на дереве идентификации объектов (работающая на основе информации, предоставляемой добровольцами) находится в источнике OID Repository[375].

На вершине дерева располагаются две главные организации по стандартизации, имеющие дело с ASN.1 — ISO и ITU-T, а также ветвь, отражающая плоды их совместных усилий. Под узлом ISO мы обнаруживаем узлы для всех стандартов ISO (1.0), а также для изданных организациями по стандартизации различных стран-членов ISO (1.2). Хотя это и не показано на рис. 9.3, США также присутствует на этом дереве, в узле 1.2.840, под которым располагаются такие организации, как IEEE и ANSI, а также стандарты отдельных компаний, например RSA (1.2.840.11359) и Microsoft (1.2.840.113556). Под узлом корпорации Microsoft помещается узел для Microsoft File Formats (1.2.840.113556.4) со стандартами форматов файлов для различных продуктов корпорации

Microsoft, таких как Word (1.2.840.113556.4.2). Но в данной книге нас, прежде всего, интересуют компьютерные сети (а *не* файлы Microsoft Word), поэтому рассмотрим ветвь 1.3, на которой «зреют» стандарты, изданные организациями, признаваемыми ISO. К ним относятся Министерство обороны США (6), под узлом которого мы обнаружим Интернет-стандарты, фонд Open Software Foundation (22), ассоциация авиалиний SITA (69), NATO (57) и многие другие.

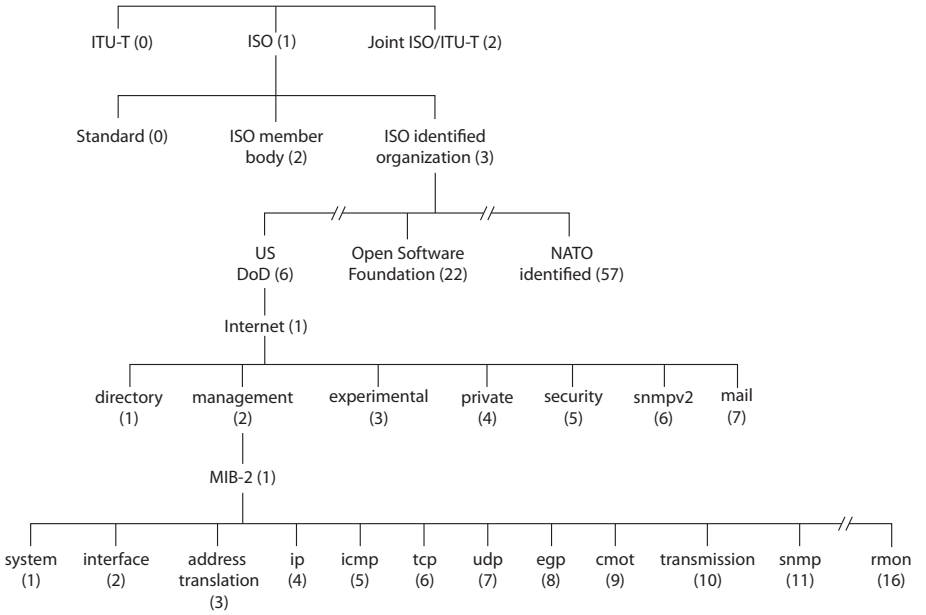


Рис. 9.3. Дерево идентификаторов объектов языка ASN.1

Под узлом `Internet` (1.3.6.1) располагается список из семи категорий. В ветви `private` (1.3.6.1.4) находится список названий и корпоративных кодов более чем 4000 частных компаний, зарегистрированных в уполномоченной организации по назначению номеров Интернета (Internet Assigned Numbers Authority, IANA)[221]. Под узлами `management` (1.3.6.1.2) и `MIB-2` (1.3.6.1.2.1) дерева идентификации объектов располагаются определения стандартизированных MIB-модулей. Уф! Долго же нам пришлось добираться до нужного уголка с пространством имен ISO!

Стандартизированные MIB-модули

На нижнем уровне дерева на рис. 9.3 показаны некоторые важные аппаратно-ориентированные MIB-модули (`system` и `interface`),

а также модули, ассоциированные с основными Интернет-протоколами. В стандарте RFC 5000[549] перечислены все стандартизированные MIB-модули по состоянию на 2008 год. Хотя относящиеся к MIB-модулям документы RFC написаны довольно скучным и сухим языком, будет полезно (подобно тому как полезно питаться овощами) рассмотреть определения некоторых MIB-модулей, чтобы лучше понять, какого рода информация содержится в модуле.

Табл. 9.2. Управляемые объекты из системной группы MIB-2

Идентификатор объекта	**Имя**	**Тип**	**Описание**[441]
1.3.6.1.2.1.1.1	`sysDescr`	OCTET STRING	Полное название и версия типа аппаратного обеспечения системы, операционной системы и сетевого программного обеспечения
1.3.6.1.2.1.1.2	`sysObjectID`	OBJECT IDENTIFIER	Назначаемый производителем идентификатор объекта, обеспечивающий простой и точно выраженный способ определить, что представляет собой управляемый объект
1.3.6.1.2.1.1.3	`sysUpTime`	TimeTicks	Время (в сотых долях секунды) с момента последней инициализации системы
1.3.6.1.2.1.1.4	`sysContact`	OCTET STRING	Человек, отвечающий за управление данным узлом, и его адрес или телефон
1.3.6.1.2.1.1.5	`sysName`	OCTET STRING	Административно назначаемое название управляемого узла. По соглашению, это полное имя домена
1.3.6.1.2.1.1.6	`sysLocation`	OCTET STRING	Физическое расположение узла
1.3.6.1.2.1.1.7	`sysServices`	Integer32	Код, обозначающий набор служб, доступных этому узлу: физических (например, для повторителя), канальных (например, для моста), служб Интернета (например, для IP-шлюза), сквозных (например, для хоста), приложений

Управляемые объекты под узлом system содержат общую информацию об управляемых устройствах; системные MIB-объекты должны поддерживать все управляемые устройства. В табл. 9.2 перечислены управляемые объекты группы system в соответствии со стандартом RFC 1213[441], а в табл. 9.3 — управляемые объекты MIB-модуля для протокола UDP.

Табл. 9.3. Некоторые управляемые объекты из модуля MIB-2 UDP

Идентификатор объекта	Имя	Тип	Описание (из стандарта RFC 4113)
1.3.6.1.2.1.7.1	udpInDatagrams	Counter32	Суммарное количество UDP-дейтаграмм, доставленных UDP-пользователям
1.3.6.1.2.1.7.2	udpNoPorts	Counter32	Суммарное количество полученных UDP-дейтаграмм, для которых нет приложения в порту получателя
1.3.6.1.2.1.7.3	udpInErrors	Counter32	Суммарное количество полученных UDP-дейтаграмм, которые не могут быть доставлены по причинам, не имеющим отношения к отсутствию приложения в порту получателя
1.3.6.1.2.1.7.4	udpOutDatagrams	Counter32	Суммарное количество UDP-дейтаграмм, отправленных данным объектом

9.3.3. Операции и транспортное соответствие протокола SNMP

Протокол SNMPv2 (Simple Network Management Protocol — простой протокол администрирования вычислительной сети версии 2), описанный в RFC 3416, применяется для передачи MIB-информации между управляющими объектами и агентами, работающими от их имени. Чаще всего протокол SNMP используется в **режиме запрос-ответ**, в котором управляющий SNMPv2-объект посылает запрос SNMPv2-агенту, тот получает запрос, выполняет определенное действие и посылает ответ на запрос. Как правило, запрос требуется, чтобы узнать (получить) или изменить (установить) значения MIB-объекта, ассоциированные с управляемым объектом.

Кроме того, протокол SNMP может применяться агентом для отправки управляющему объекту сообщений, передаваемых без запросов

и называемых **прерывающими сообщениями**. Они используются для уведомления управляющего объекта об исключительных ситуациях, возникших в результате изменения значений MIB-объектов. В разделе 9.1 было показано, что администратор вычислительной сети может быть заинтересован в получении прерывающих сообщений, например в случае выхода интерфейса из строя, или определенного уровня перегрузки, или какого-либо еще важного события. Не забывайте о важных компромиссах, принимаемых для согласования между режимом запрос-ответ и режимом прерываний (см. упражнения в конце главы).

Протоколом SNMPv2 определены семь типов сообщений, как правило, называемых протокольными блоками данных (**ПБД**) (табл. 9.4). Формат ПБД показан на рис. 9.4.

- ПБД `GetRequest`, `GetNextRequest` и `GetBulkRequest` передаются управляющим объектом агенту для запроса значения одного или нескольких MIB-объектов на устройстве, управляемом агентом. Идентификаторы MIB-объектов указываются в переменной части ПБД. ПБД `GetRequest`, `GetNextRequest` и `GetBulkRequest` отличаются степенью детализации запрашиваемых данных. При помощи ПБД `GetRequest` можно запросить произвольное количество MIB-значений; для работы со списком или таблицей MIB-объектов можно воспользоваться несколькими ПБД `GetNextRequest`. ПБД `GetBulkRequest` позволяет получить большой блок данных, избегая накладных расходов, связанных с отправкой нескольких сообщений `GetRequest` или `GetNextRequest`. Во всех трех случаях агент отвечает ПБД `Response`, в котором содержатся идентификаторы объектов и ассоциированные с ними значения.
- ПБД `SetRequest` используется управляющим объектом для установки значения одного или нескольких MIB-объектов в управляемом устройстве. Агент отвечает ПБД `Response`, в поле состояния ошибки которой помещается значение `noError` (без ошибки) в подтверждение того факта, что значение действительно установлено.
- ПБД `InformRequest` используется управляющим объектом для передачи другому управляющему объекту MIB-информации. Получающий объект отвечает ПБД `Response`, в поле состояния ошибки которой помещается значение `noError` (без ошибки), подтверждающее факт получения ПБД `InformRequest`.
- Последним типом ПБД протокола SNMPv2 является прерывающее сообщение. Такие сообщения генерируются асинхронно, то есть

не в ответ на запрос, а в ответ на событие, уведомление о котором необходимо управляемому объекту. В RFC 1907 определяются типы прерываний, включая холодный или горячий запуск устройства, включение или отключение канала связи, исчезновение соседа, неудачно проведенная аутентификация. Управляющее устройство не обязано отвечать на полученное прерывающее сообщение

Табл. 9.4. Типы ПБД в протоколе SNMPv2

Тип ПБД SNMPv2 PDU	Отправитель — получатель	Описание
`GetRequest`	Менеджер — агенту	Получает значение одного или более экземпляров объектов MIB
`GetNextRequest`	Менеджер — агенту	Получает значение следующего экземпляра MIB в списке или в таблице
`GetBulkRequest`	Менеджер — агенту	Получает значения большого блока данных, например значения большой таблицы
`InformRequest`	Менеджер — менеджеру	Информирует удаленное управляющее устройство об удаленных относительно этого устройства MIB-значениях
`SetRequest`	Менеджер — агенту	Задает значение одного или нескольких экземпляров объектов MIB
`Response`	Агент — менеджеру или менеджер — менеджеру	Генерируется в ответ на `GetRequest`, `GetNextRequest`, `GetBulkRequest`, `SetRequest PDU` или `InformRequest`
`SNMPv2-Trap`	Агент — менеджеру	Информирование менеджера о возникшем прерывании

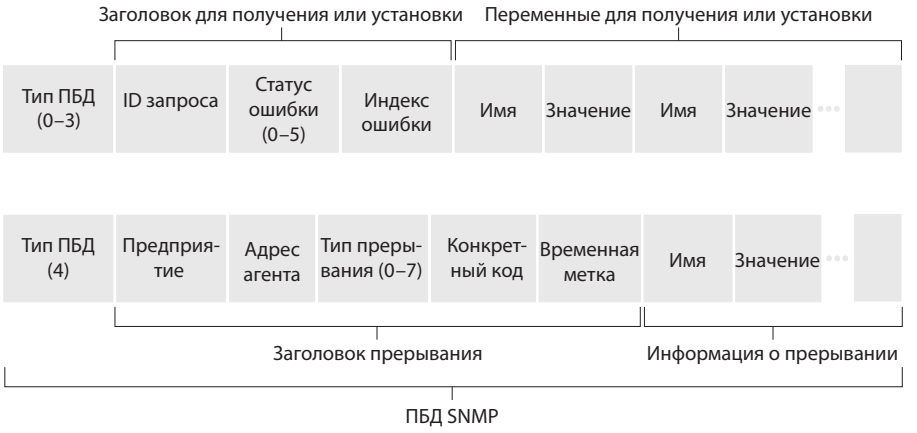


Рис. 9.4. Формат ПБД для протокола SNMP

Учитывая, что протокол SNMPv2 работает по принципу «запрос-ответ», необходимо отметить, что, хотя ПБД могут переноситься при помощи различных транспортных протоколов, как правило, для этой цели используются UDP-дейтаграммы. В самом деле то, что протокол UDP является «предпочтительным транспортом», утверждается в RFC 3417. Поскольку UDP представляет собой транспортный протокол с негарантированной доставкой, нет уверенности, что запрос или ответ дойдет до адресата. Поле идентификатора запроса ПБД используется управляющим объектом для нумерации запросов, посылаемых агентам. Это значение помещается в ответное сообщение агента. Таким образом, поле идентификатора запроса позволяет управляющему объекту обнаружить потерянный запрос или ответ. Решение о том, повторять или нет запрос, на который не получен ответ в установленный срок, принимает управляющий объект. В частности, стандартом SNMP не предписывается какая-либо специальная процедура повторной передачи и даже сама ее необходимость. Там только сказано, что управляющий объект «должен действовать ответственно в отношении частоты и длительности повторных передач». Само собой, это заставляет крепко задуматься над тем, как ведут себя «ответственные» протоколы.

9.3.4. Безопасность и администрирование

Разработчики протокола SNMPv3 заявляют, что он может рассматриваться как протокол SNMPv2 с дополнительными возможностями в области безопасности и администрирования[508]. Действительно, между протоколами SNMPv3 и SNMPv2 есть отличия, но нигде они не заметны так, как в областях безопасности и администрирования. Центральная роль безопасности в протоколе SNMPv3 особенно важна, так как из-за недостатка адекватных механизмов безопасности протокол SNMP применялся в основном не для контроля, а для мониторинга (например, ПБД `SetRequest` в протоколе SNMPvl почти не используется).

Функциональность протокола SNMP к третьей версии выросла, однако вместе с ней, к сожалению, увеличился и объем документации. Чего стоит один лишь факт существования документа RFC 3411[509], в котором «описывается архитектура для описания архитектуры управления SNMP»! От «архитектуры для описания архитектуры» может закружиться голова, но цель RFC 3411 заключается в создании общего языка описания функциональности и действий, предпринимаемых

SNMPv3-агентом или управляющим объектом. Архитектура SNMPv3-объекта проста, и ее изучение поможет нам лучше понять протокол SNMP.

К так называемым **SNMP-приложениям** относятся генератор команд, получатель уведомлений, буферный передатчик (эти три элемента, как правило, присутствуют в управляющем объекте), ответчик команд, генератор уведомлений (эти два элемента, как правило, присутствуют в агенте) и др. Генератор команд создает ПБД `GetRequest`, `GetNextRequest`, `GetBulkRequest` и `SetRequest`, которые мы рассматривали в подразделе 9.3.3, а также обрабатывает ответы, получаемые на эти ПБД. Ответчик команд (приложение, отвечающее на команды) выполняется на агенте. Он получает, обрабатывает и отвечает (с помощью сообщения `Response`) на приходящие ПБД `GetRequest`, `GetNextRequest`, `GetBulkRequest` и `SetRequest`. Работающий на агенте генератор уведомлений генерирует ПБД `Trap`. Эти ПБД принимаются и обрабатываются получателем уведомлений на управляющем объекте. Буферный передатчик занимается перенаправлением запросов, уведомлений и ответных ПБД.

Посылаемый SNMP-приложением ПБД, прежде чем попасть к соответствующему транспортному протоколу, проходит через «движок» протокола SNMP. На рис. 9.5 показано, как создаваемый генератором команд ПБД сначала попадает в модуль диспетчеризации, где определяется версия протокола SNMP. Затем ПБД обрабатывается системой подготовки сообщений, где к ней добавляется заголовок сообщения, содержащий версию протокола SNMP, идентификатор и размер сообщения. Если требуется шифрование или аутентификация, то в заголовок также включаются соответствующие поля (детали см. в документе RFC 3411[509]). Наконец, SNMP-сообщение (созданный приложением ПБД плюс заголовок) передается соответствующему транспортному протоколу. Как правило, для передачи SNMP-сообщений используются протокол UDP (то есть SNMP-сообщение помещается в поле полезной нагрузки UDP-дейтаграммы) и порт 161. Для прерывающих сообщений используется порт 162.

Итак, мы выяснили, что SNMP-сообщения предназначаются не только для мониторинга, но и для контроля (например, с помощью команды `SetRequest`) сетевых элементов. Очевидно, злоумышленник, способный интерпретировать SNMP-сообщения, а также генерировать собственные SNMP-пакеты, может причинить немало неприятностей

в сети. Таким образом, важно, чтобы SNMP-сообщения пересылались в зашифрованном виде. Удивительно, но подобающее внимание вопросу безопасности было уделено лишь в самой последней версии протокола SNMP. В протоколе SNMPv3 применяется модель так называемой **пользовательской безопасности** (User-Based Security, USM)[510]. В этом случае такая информация, как значение ключа или привилегии доступа, ассоциируется с пользователем, который идентифицируется по имени и паролю.

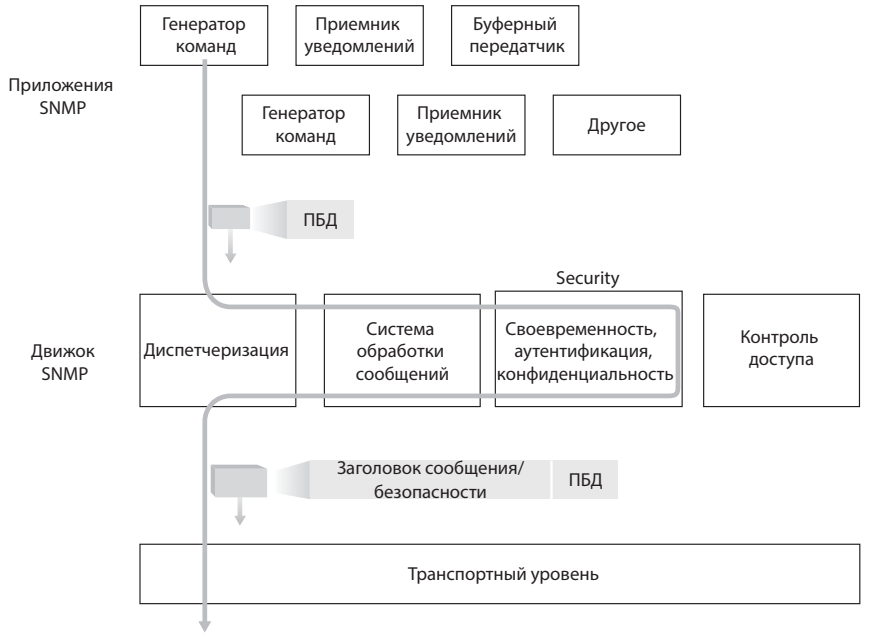


Рис. 9.5. Движок и приложения SNMP

Протокол SNMPv3 предоставляет шифрование, аутентификацию, защиту от атак повторного воспроизведения (см. раздел 8.3), а также контроль доступа.

- *Шифрование*. ПБД протокола SNMP могут быть зашифрованы по стандарту DES в режиме сцепления блоков шифротекста. Обратите внимание, что, поскольку DES представляет собой систему шифрования с разделяемым ключом, секретный ключ должен быть известен получателю зашифрованных данных.
- *Аутентификация*. Протокол SNMP использует коды аутентификации сообщений (MAC), изученные в разделе 8.3.1, обеспечивая с их

помощью как аутентификацию, так и защиту от подделки сообщений[534]. Как вы помните, при работе с MAC-кодами и отправитель, и получатель должны знать общий секретный ключ.

- *Защита от атак повторного воспроизведения*. Как уже отмечалось в главе 8, для защиты от атак повторного воспроизведения могут использоваться одноразовые номера. Соответствующий метод применяется и в протоколе SNMPv3. Чтобы гарантировать, что полученное сообщение представляет собой не повторное воспроизведение какого-либо ранее посланного, получатель требует, чтобы отправитель в каждое сообщение включал определенное значение, основанное на счетчике *получателя*. Этот счетчик отражает интервал времени с момента последней перезагрузки управляющего сетевого программного обеспечения получателя, а также суммарное число перезагрузок с момента его последнего конфигурирования. До тех пор пока значение счетчика в полученном сообщении находится в определенных допустимых пределах, сообщение принимается и может быть расшифровано и/или аутентифицировано[510].
- *Контроль доступа*. Протокол SNMPv3 обеспечивает видовую модель контроля доступа[511], то есть определяет управляющую информацию, которая может запрашиваться и/или изменяться пользователями. SNMP-объект хранит информацию о правах доступа и политиках в локальной базе данных конфигурации (Local Configuration Datastore, LCD). К фрагментам этой базы данных предоставляется доступ как к управляемым объектам[511], и, таким образом, ими можно управлять дистанционно с помощью протокола SNMP.

9.4. Язык ASN.1

В этой книге мы рассмотрели множество интересных вопросов, относящихся к компьютерным сетям. Раздел о языке ASN.1 (Abstract Syntax Notation One — нотация абстрактного синтаксиса, версия 1), вероятно, нельзя отнести к числу самых занимательных. Как и овощная диета, изучение языка ASN.1, говорят, может оказаться полезным. ASN.1 представляет собой стандарт, выпущенный ISO и используемый в ряде протоколов Интернета, относящихся в основном к сфере администрирования вычислительной сети. Например, в разделе 9.3 было показано, что MIB-переменные протокола SNMP неразрывно связаны с языком ASN.1. Поэтому, несмотря на сухость материала о языке ASN.1 в данном разделе, мы надеемся, читатель примет на веру утверждение о его важности.

Чтобы мотивировать читателя на изучение этой темы, проведем следующий мысленный эксперимент. Предположим, мы имеем возможность надежно копировать данные из памяти нашего компьютера в память удаленного компьютера. Решит ли это проблему связи? Ответ на этот вопрос зависит от того, что мы называем «проблемой связи». Очевидно, подобное идеальное копирование из памяти в память обеспечит передачу битов и байтов между компьютерами. Но означает ли оно, что программное обеспечение, работающее на принимающем компьютере, обратившись к данным, получит те же значения, которые хранятся в памяти передающего компьютера? Оказывается, что этот вопрос не так прост, как может показаться. Дело в том, что на компьютерах с различной архитектурой, с различными операционными системами и разными компиляторами используются разные соглашения для хранения и представления данных. Если данные должны передаваться с одного компьютера на другой, проблему представления данных необходимо решить.

Рассмотрим эту проблему на примере простого фрагмента программы на языке C. Как может эта структура располагаться в памяти?

```
struct {
  char code;
  int x;
  } test;
test.x = 259;
test.code = 'a';
```

Слева на рис. 9.6 показано возможное расположение данных в гипотетической архитектуре: на символ `a` выделяется один байт, за которым следует 16-разрядное слово, содержащее целое значение 259, начиная со старшего байта. Расположение той же структуры данных на другом компьютере показано в правой части рис. 9.6. Следом за символом `a` располагается целое значение, хранящееся, начиная с младшего байта и к тому же выровненное по границам слов (по четным байтам). Очевидно, при простом копировании памяти с одного компьютера на другой и при использовании одного и того же определения структуры для доступа к хранимым значениям мы увидим на двух компьютерах совершенно разные результаты.

Тот факт, что в компьютерах с разной архитектурой используются разные внутренние форматы хранения данных, представляет сложную проблему. Формат с прямым порядком следования байтов (самым пер-

вым хранится самый старший байт) применяется в процессорах SPARC и Motorola, тогда как формат с обратным порядком следования байтов (самым первым хранится самый младший байт) используется в процессорах фирмы Intel и процессорах Alpha компаний DEC/ Compaq. В англоязычной литературе эти два формата называются «тупоконечным» и «остроконечным» по аналогии со спором лилипутов из книги Джонатана Свифта «Путешествие Гулливера», которые никак не могли договориться, с какой стороны разбивать вареные яйца. В книге этот спор привел к бунтам и гражданской войне.

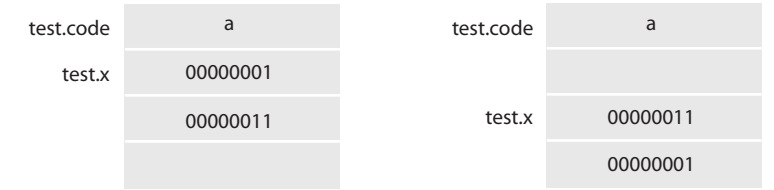


Рис. 9.6. Два разных варианта компоновки данных в двух различных архитектурах

Итак, разные компьютеры могут по-разному хранить и представлять данные. Как же должен решать эту проблему сетевой протокол? Например, если SNMP-агент собирается послать ответное сообщение, содержащее целое значение счетчика полученных UDP-дейтаграмм, в каком формате следует ему представить это целое число? Один из вариантов — пересылать байты в том же порядке, в котором они хранятся в управляющем объекте. Другой вариант — пересылать байты без преобразования формата, то есть так, как они хранятся в передающем компьютере. В любом случае может потребоваться, чтобы отправитель или получатель узнал вариант целочисленного представления данных на машине-партнере.

Третий вариант заключается в использовании независимого от машины, операционной системы и языка программирования метода описания целых чисел и данных других типов (иными словами, в задействовании языка описания данных), а также правил пересылки по сети каждого из этих типов данных. При получении данных определенного типа они принимаются в известном формате, и могут быть сохранены в любом формате, специфичном для данной машины. Этот вариант применяется в языке SMI, который мы рассматривали в разделе 9.3 и в языке ASN.1. В терминах ISO эти два стандарта описывают **службу представления**, то есть службу передачи и преобразования информации из одного машинного формата в другой. Пример проблемы представления

информации, взятый из реального мира, показан на рис. 9.7. Ни один из получателей сообщения не понимает смысла сказанного. Как показано на рис. 9.8, эта проблема может быть решена при помощи выражения идеи на общепонятном (для службы представления) независимом от конкретного участника языке. Информация сначала отсылается получателю, а потом «переводится» на понятный ему язык.

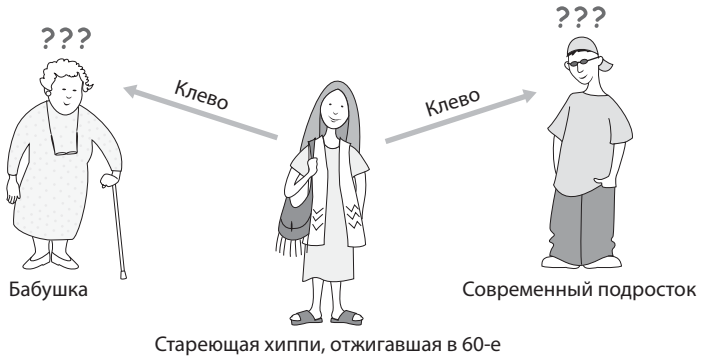


Рис. 9.7. Проблема представления

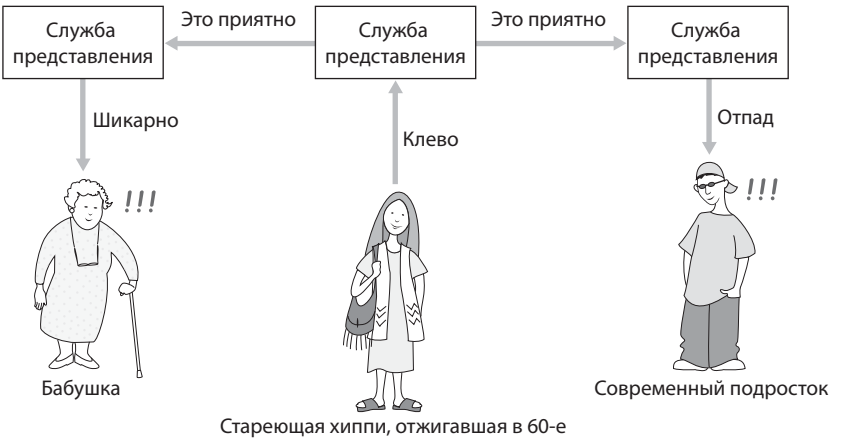


Рис. 9.8. Проблема представления решена

В табл. 9.5 перечислены некоторые типы данных языка ASN.1. Вспомним, что мы уже встречали типы INTEGER, OCTET STRING и OBJECT IDENTIFIER при обсуждении языка SMI. Поскольку здесь мы не ставим перед собой цель дать подробное введение в язык ASN.1,

отсылаем читателей к изучению стандартов, а также к печатной или онлайновой версии книги Лармуфа[310], где рассмотрены типы и конструкторы языка ASN.1, в частности, SEQUENCE и SET, позволяющие описывать структурированные типы данных.

Табл. 9.5. Некоторые типы данных языка ASN.1

Метка	Тип	Описание
1	BOOLEAN	Значение «истина» (true) или «ложь» (false)
2	INTEGER	Любое целое число
3	BITSTRING	Список из одного или более битов
4	OCTET STRING	Список из одного или более байтов
5	NULL	Значение отсутствует
6	OBJECT IDENTIFIER	Имя из стандартного дерева именования языка ASN.1; см. раздел 9.2.2
9	REAL	Значение с плавающей точкой

Помимо языка определения данных ASN.1 предоставляет **основные правила кодирования** (Basic Encoding Rules, **BER**), описывающие способ передачи по сети экземпляров объектов, определенных с помощью языка ASN.1. Правилами BER для кодирования передаваемых данных принят так называемый **метод TLV** (Type, Length, Value — тип, длина, значение). То есть для каждого элемента данных по сети пересылаются код его типа, длина и значение, именно в таком порядке. Такое соглашение при пересылке данных облегчает их интерпретацию при получении.

На рис. 9.9 показан простой пример пересылки двух элементов данных. В этом примере отправитель хочет переслать символьную строку smith, за которой следует целое число 259 (в двоичном представлении это 00000001 00000011, то есть сначала идет байт со значением 1, а за ним байт со значением 3, так как используется прямой порядок следования байтов). Первый байт в передаваемом потоке имеет значение 4, указывающее на то, что следом идут данные типа OCTET STRING. Во втором байте потока передается длина строки байтов типа OCTET STRING, в данном случае это число 5. Третьим байтом потока начинается строка байтов типа OCTET STRING длины 5. В нем содержится ASCII-представление символа *s*. Вслед за символьной строкой передается второй элемент данных, начинающийся с байта 2 (тип INTEGER), еще одного байта 2 (длина целого числа) и самого числа 259 с прямым порядком следования байтов.

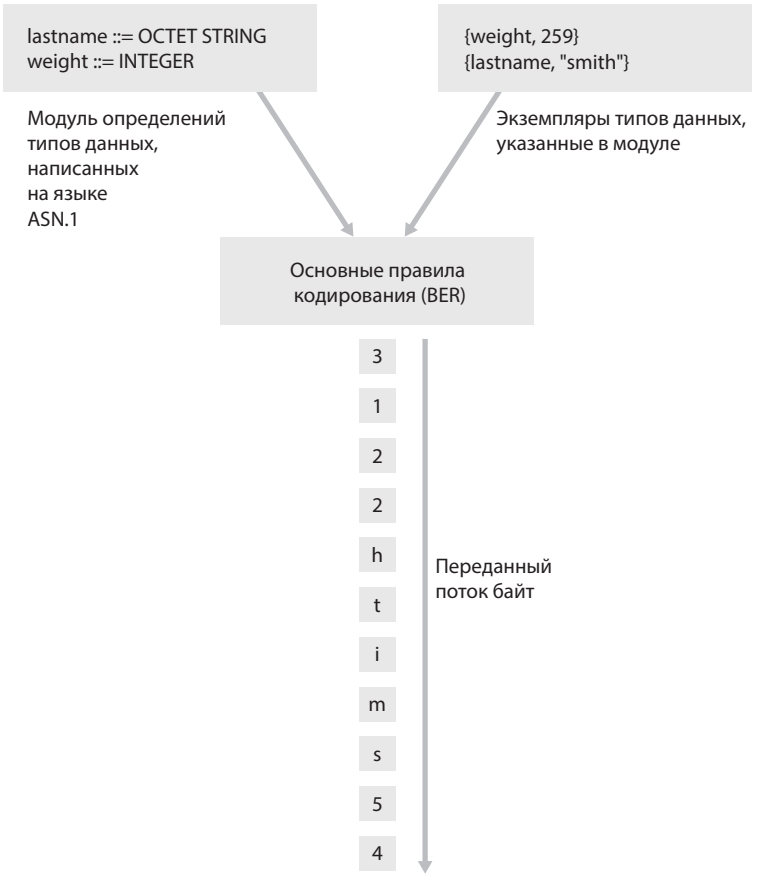


Рис. 9.9. Пример кодирования в системе BER

В нашем обсуждении мы рассмотрели лишь небольшое и простое подмножество языка ASN.1. Дополнительные сведения о языке ASN.1 можно найти в следующих источниках: документ, посвященный стандартам ASN.1[248], онлайновая книга о модели OSI[311], а также сайты, посвященные языку ASN.1, в частности, OSS 2012[377] и OID Repository[375].

9.5. Заключение

Итак, наше изучение администрирования вычислительной сети (и компьютерных сетей в целом) закончено!

В этой последней главе, посвященной администрированию вычислительной сети, мы начали работу с рассказа об инструментальных

средствах, нужных администратору вычислительной сети для мониторинга, тестирования, опроса, конфигурирования, анализа, оценки и контроля сети. Необходимость средств управления была проиллюстрирована на примерах управления такими системами, как электростанции, самолеты и коммерческие организации. Мы видели, что архитектура систем администрирования вычислительной сети состоит из пяти ключевых компонентов: во-первых, администратора вычислительной сети, во-вторых, множества удаленных (от администратора вычислительной сети) устройств, в-третьих, базы управляющей информации (MIB) на этих устройствах, содержащей данные о состоянии и работе устройств, в-четвертых, удаленных агентов, сообщающих администратору MIB-информацию и функционирующих под контролем администратора вычислительной сети, и, в-пятых, протокола для связи между администратором вычислительной сети и удаленными устройствами.

Затем мы углубились в детали архитектуры управляющей информации Интернета и, в частности, в детали протокола SNMP. Было показано, как в протоколе SNMP реализованы упомянутые пять ключевых компонентов администрирования вычислительной сети. Мы изучили MIB-объекты, язык определения данных для описания MIB-объектов (язык SMI) и сам протокол SNMP. Было отмечено, что языки SMI и ASN.1 неразрывно связаны друг с другом, и язык ASN. 1 играет ключевую роль на уровне представления в семиуровневой модели ISO/OSI. Затем мы кратко рассмотрели язык ASN.1. Кроме того, мы отметили необходимость преобразования форматов данных при передаче их с одной машины на другую. Некоторые сетевые архитектуры явно поддерживают данную функцию на уровне представления, однако в стеке протоколов Интернета такой уровень отсутствует.

Также следует заметить, что мы намеренно не стали рассматривать многие вопросы администрирования вычислительной сети, например обнаружение, исправление и предотвращение неисправностей, управление службами. Эти темы также важны, но для их освещения потребовалась бы еще одна книга. Ссылки на нужную литературу и стандарты можно найти в файле среди примеров к книге.

СПИСОК ЛИТЕРАТУРЫ

Замечание об электронных ресурсах. Ниже мы указали ряд ссылок на вебстраницы и документы, существующие только в электронном виде, а также на другие источники, которые не публиковались в печатных изданиях (если, конечно, нам удавалось найти такие ссылки). Мы не указывали электронные адреса для публикаций, имеющихся в сборниках с конференций и в научных журналах, поскольку такие работы обычно легко находятся через форму поиска на сайте соответствующей конференции, либо имеются в электронных библиотеках. Так, статьи со всех конференций и семинаров ACM SIGCOMM доступны на сайте www.acm.org/sigcomm. Все приведенные ниже ссылки были проверены и работали на момент написания книги, но в настоящее время некоторые из них могут оказаться битыми.

Замечание о стандартах RFC: копии стандартов RFC публикуются на многих сайтах. Официальный орган, занимающийся управлением этими стандартами называется «Редакция RFC в составе общества Интернета» (RFC Editor of the Internet Society), сайт данной редакции — www.rfc-editor.org. На этом сайте вы можете найти любой стандарт RFC по названию, номеру, авторам. Также система покажет вам номера обновлений к этому стандарту. Стандарты RFC могут обновляться или устаревать. Рекомендуем читать их на сайте rfc.com.ru.

1. 3Com Corp., White paper: Understanding IP addressing: Everything you ever wanted to know, www.apnic.net/__data/assets/pdf_file/0020/8147/501302.pdf
2. Third Generation Partnership Project, www.3gpp.org.
3. 3GPP, TS 23.002: Network Architecture: Digital Cellular Telecommunications System (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE, www.3gpp.org/ftp/Specs/html-info/23002.htm
4. P. Albitz and C. Liu, DNS and BIND, O'Reilly & Associates, Petaluma, CA, 1993.
5. N. Abramson, The Aloha System–Another Alternative for Computer Communications, Proc. 1970 Fall Joint Computer Conference, AFIPS Conference, p. 37, 1970.
6. N. Abramson, Development of the Alohanet, IEEE Transactions on Information Theory, Vol. IT-31, No. 3 (Mar. 1985), pp. 119–123.
7. N. Abramson, The Alohanet — Surfing for Wireless Data, IEEE Communications Magazine, Vol. 47, No. 12, pp. 21–25.
8. H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, A. Donnelly, Symbiotic Routing in Future Data Centers, Proc. 2010 ACM SIGCOMM.
9. V. K. Adhikari, S. Jain, Y. Chen, Z. L. Zhang, Vivisecting YouTube: An Active Measurement Study, Technical Report, University of Minnesota, 2011.
10. V. K. Adhikari, Y. Gao, F. Hao, M. Varvello, V. Hilt, M. Steiner, Z. L. Zhang, Unreeling Netflix: Understanding and Improving Multi-CDN Movie Delivery, Technical Report, University of Minnesota, 2012.
11. A. Afanasyev, N. Tilley, P. Reiher, L. Kleinrock, Host-to-Host Congestion Control for TCP, IEEE Communications Surveys & Tutorials, Vol. 12, No. 3, pp. 304–342.

12. S. Agarwal, J. Lorch, Matchmaking for Online Games and Other Latency-sensitive P2P Systems, Proc. 2009 ACM SIGCOMM.

13. J. S. Ahn, P. B. Danzig, Z. Liu, and Y. Yan, Experience with TCP Vegas: Emulation and Experiment, Proc. 1995 ACM SIGCOMM (Boston, MA, Aug. 1995), pp. 185–195.

14. Akamai homepage, www.akamai.com

15. A. Akella, S. Seshan, A. Shaikh, An Empirical Evaluation of Wide-Area Internet Bottlenecks, Proc. 2003 ACM Internet Measurement Conference (Miami, FL, Nov. 2003).

16. S. Akhshabi, A. C. Begen, C. Dovrolis, An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP, Proc. 2011 ACM Multimedia Systems Conf.

17. I. Akyildiz, D. Gutierrex-Estevez, E. Reyes, The Evolution to 4G Cellular Systems, LTE Advanced, Physical Communication, Elsevier, 3 (2010), 217–244.

18. Alcatel-Lucent, Introduction to Evolved Packet Core, downloads.lightreading.com/wplib/alcatellucent/ALU_WP_Intro_to_EPC.pdf.

19. M. Al-Fares, A. Loukissas, A. Vahdat, A Scalable, Commodity Data Center Network Architecture, Proc. 2008 ACM SIGCOMM.

20. M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data Center TCP (DCTCP), Proc. 2010 ACM SIGCOMM.

21. E. Allman, The Robustness Principle Reconsidered: Seeking a Middle Ground, Communications of the ACM, Vol. 54, No. 8 (Aug. 2011), pp. 40–45.

22. J. B. Andersen, T. S. Rappaport, S. Yoshida, Propagation Measurements and Models for Wireless Communications Channels, IEEE Communications Magazine, (Jan. 1995), pp. 42–49.

23. M. Andrews, M. Shepherd, A. Srinivasan, P. Winkler, F. Zane, Clustering and Server Election Using Passive Monitoring, Proc. 2002 IEEE INFOCOM.

24. S. Androutsellis-Theotokis, D. Spinellis, A Survey of Peer-to-Peer Content Distribution Technologies, ACM Computing Surveys, Vol. 36, No. 4 (Dec. 2004), pp. 335–371.

25. C. Aperjis, M.J. Freedman, R. Johari, Peer-Assisted Content Distribution with Prices, Proc. ACM CoNEXT'08 (Madrid, Dec. 2008).

26. G. Appenzeller, I. Keslassy, N. McKeown, Sizing Router Buffers, Proc. 2004 ACM SIGCOMM (Portland, OR, Aug. 2004).

27. G. R. Ash, Dynamic Routing in Telecommunications Networks, McGraw Hill, New York, NY, 1998.

28. The Address Supporting Organization home page, www.aso.icann.org

29. AT&T, AT&T High Speed Internet Business Edition Service Level Agreements, www.att.com/gen/general?pid=6622

30. Atheros Communications Inc. Atheros AR5006 WLAN Chipset Product Bulletins, www.atheros.com/pt/AR5006Bulletins.htm

31. B. Augustin, B. Krishnamurthy, W. Willinger, IXPs: Mapped? Proc. Internet Measurement Conference (IMC), November 2009.

32. E. Ayanoglu, S. Paul, T. F. La Porta, K. K. Sabnani, R. D. Gitlin, AIRMAIL: A Link-Layer Protocol for Wireless Networks, ACM ACM/Baltzer Wireless Networks Journal, 1: 47–60, Feb. 1995.

33. A. Bakre, B. R. Badrinath, I-TCP: Indirect TCP for Mobile Hosts, Proc. 1995 Int. Conf. on Distributed Computing Systems (ICDCS) (May 1995), pp. 136–143.

34. H. Balakrishnan, V. Padmanabhan, S. Seshan, R. Katz, A Comparison of Mechanisms for Improving TCP Performance Over Wireless Links, IEEE/ACM Transactions on Networking Vol. 5, No. 6 (Dec. 1997).

35. H. Balakrishnan, F. Kaashoek, D. Karger, R. Morris, I. Stoica, Looking Up Data in P2P Systems, Communications of the ACM, Vol. 46, No. 2 (Feb. 2003), pp. 43–48.

36. M. Baldauf, S. Dustdar, F. Rosenberg, A Survey on Context-Aware Systems, Int. J. Ad Hoc and Ubiquitous Computing, Vol. 2, No. 4 (2007), pp. 263–277.

37. H. Ballani, P. Francis, S. Ratnasamy, A Measurement-based Deployment Proposal for IP Anycast, Proc. 2006 ACM Internet Measurement Conf.

38. H. Ballani, P. Costa, T. Karagiannis, Ant Rowstron, Towards Predictable Datacenter Networks, Proc. 2011 ACM SIGCOMM.

39. P. Baran, On Distributed Communication Networks, IEEE Transactions on Communication Systems, Mar. 1964. Rand Corporation Technical report with the same title (Memorandum RM-3420-PR, 1964). www.rand.org/publications/RM/RM3420/

40. J. Bardwell, You Believe You Understand What You Think I Said . . . The Truth About 802.11 Signal And Noise Metrics: A Discussion Clarifying Often-Misused 802.11 WLAN Terminologies, www.connect802.com/download/techpubs/2004/you_believe_D100201.pdf

41. P. Barford, N. Duffield, A. Ron, J. Sommers, Network Performance Anomaly Detection and Localization, Proc. 2009 IEEE INFOCOM (Apr. 2009).

42. P. Baronti, P. Pillai, V. Chook, S. Chessa, A. Gotta, Y. Hu, Wireless Sensor Networks: A Survey on the State of the Art and the 802.15.4 and ZigBee Standards, Computer Communications, Vol. 30, No. 7 (2007), pp. 1655–1695.

43. S. A. Basset and H. Schulzrinne, An analysis of the Skype peer-to-peer Internet Telephony Protocol, Proc. 2006 IEEE INFOCOM (Barcelona, Spain, Apr. 2006).

44. BBC news online A Small Slice of Design, Apr. 2001, news.bbc.co.uk/2/hi/science/nature/1264205.stm

45. BBC, Multicast, www.bbc.co.uk/multicast/

46. N. Beheshti, Y. Ganjali, M. Ghobadi, N. McKeown, G. Salmon, Experimental Study of Router Buffer Sizing, Proc. ACM Internet Measurement Conference (October 2008, Vouliagmeni, Greece).

47. P. Bender, P. Black, M. Grob, R. Padovani, N. Sindhushayana, A. Viterbi, CDMA/HDR: A bandwidth-efficient high-speed wireless data service for nomadic users, IEEE Commun. Mag., Vol. 38, No. 7 (July 2000) pp. 70–77.

48. T. Berners-Lee, CERN, Information Management: A Proposal, Mar. 1989, May 1990. www.w3.org/History/1989/proposal.html

49. T. Berners-Lee, R. Cailliau, A. Luotonen, H. Frystyk Nielsen, A. Secret, The World-Wide Web, Communications of the ACM, Vol. 37, No. 8 (Aug. 1994), pp. 76–82.

50. D. Bertsekas, R. Gallagher, Data Networks, 2nd Ed., Prentice Hall, Englewood Cliffs, NJ, 1991.

51. P. Biddle, P. England, M. Peinado, B. Willman, The Darknet and the Future of Content Distribution, 2002 ACM Workshop on Digital Rights Management, (Nov. 2002, Washington, D.C.) crypto.stanford.edu/DRM2002/darknet5.doc

52. E. W. Biersack, Performance evaluation of forward error correction in ATM networks, Proc. 1999 ACM SIGCOMM (Baltimore, MD, Aug. 1992), pp. 248–257.

53. Internet Software Consortium page on BIND, www.isc.org/downloads/bind/

54. C. Bisdikian, An Overview of the Bluetooth Wireless Technology, IEEE Communications Magazine, No. 12 (Dec. 2001), pp. 86–94.

55. M. Bishop, Computer Security: Art and Science, Boston: Addison Wesley, Boston MA, 2003.

56. U. Black, ATM Volume I: Foundation for Broadband Networks, Prentice Hall, 1995.

57. U. Black, ATM Volume II: Signaling in Broadband Networks, Prentice Hall, 1997.

58. M. Blumenthal, D. Clark, Rethinking the Design of the Internet: the End-to-end Arguments vs. the Brave New World, ACM Transactions on Internet Technology, Vol. 1, No. 1 (Aug. 2001), pp. 70–109.

59. G. V. Bochmann, C. A. Sunshine, Formal methods in communication protocol design, IEEE Transactions on Communications, Vol. 28, No. 4 (Apr. 1980) pp. 624–631.

60. J-C. Bolot, T. Turletti, A rate control scheme for packet video in the Internet, Proc. 1994 IEEE INFOCOM, pp. 1216–1223.

61. J-C. Bolot, A. Vega-Garcia, Control Mechanisms for Packet Audio in the Internet, Proc. 1996 IEEE INFOCOM, pp. 232–239.

62. S. Bradner, A. Mankin, IPng: Internet Protocol Next Generation, Addison-Wesley, Reading, MA, 1996.

63. L. Brakmo, L. Peterson, TCP Vegas: End to End Congestion Avoidance on a Global Internet, IEEE Journal of Selected Areas in Communications, Vol. 13, No. 8 (Oct. 1995), pp. 1465–1480.

64. L. Breslau, E. Knightly, S. Shenker, I. Stoica, H. Zhang, Endpoint Admission Control: Architectural Issues and Performance, Proc. 2000 ACM SIGCOMM (Stockholm, Sweden, Aug. 2000).

65. B. Bryant, Designing an Authentication System: A Dialogue in Four Scenes, web.mit.edu/kerberos/www/dialogue.html

66. V. Bush, As We May Think, The Atlantic Monthly, July 1945. www.theatlantic.com/unbound/flashbks/computer/bushf.htm

67. J. Byers, M. Luby, M. Mitzenmacher, A. Rege, A digital fountain approach to reliable distribution of bulk data, Proc. 1998 ACM SIGCOMM (Vancouver, Canada, Aug. 1998), pp. 56–67.

68. CableLabs homepage, www.cablelabs.com

69. CacheLogic homepage, www.cachelogic.com

70. M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, J. van der Merwe, Design and implementation of a Routing Control Platform, Proc. Networked Systems Design and Implementation (May 2005).

71. M. Caesar, J. Rexford, BGP Routing Policies in ISP Networks, IEEE Network Magazine, Vol. 19, No. 6 (Nov. 2005).

72. M. Casado, M. Freedman, J. Pettit, J. Luo, N. Gude, N. McKeown, S. Shenker, Rethinking Enterprise Network Control, IEEE/ACM Transactions on Networking (ToN), Vol. 17, No. 4 (Aug. 2009), pp. 1270–1283.

73. C. Caldwell, The Prime Pages, www.utm.edu/research/primes/prove.

74. N. Cardwell, S. Savage, T. Anderson, Modeling TCP Latency, Proc. 2000 IEEE INFOCOM (Tel-Aviv, Israel, Mar. 2000).

75. Center for Collaborative Adaptive Sensing of the Atmosphere, www.casa.umass.edu

76. M. Casado, M. Freedman, J. Pettit, J. Luo, N. McKeown, S. Shenker, Ethane: Taking Control of the Enterprise, Proc. 2007 ACM SIGCOMM (Kyoto, Japan, Aug. 2007).

77. S. Casner, S. Deering, First IETF Internet Audiocast, ACM SIGCOMM Computer Communications Review, Vol. 22, No. 3 (July 1992), pp. 92–97.

78. Ceiva homepage, www.ceiva.com/

79. Center for Embedded Network Sensing, www.cens.ucla.edu/

80. V. Cerf, R. Kahn, A Protocol for Packet Network Interconnection, IEEE Transactions on Communications Technology, Vol. COM-22, No. 5, pp. 627–641.

81. CERT, Advisory 2001–09: Statistical Weaknesses in TCP/IP Initial Sequence Numbers, www.cert.org/advisories/CA-2001-09.html

82. CERT, CERT Advisory CA-2003-04 MS-SQL Server Worm, www.cert.org/advisories/CA-2003-04.html

83. CERT Coordination Center, www.cert.org/advisories

84. CERT, Packet Filtering for Firewall Systems, www.cert.org/tech_tips/packet_filtering.html

85. CERT, Advisory CA-96.21: TCP SYN Flooding and IP Spoofing Attacks, www.cert.org/advisories/CA-1998-01.html

86. H. J. Chao, C. Lam, E. Oki, Broadband Packet Switching Technologies — A Practical Guide to ATM Switches and IP Routers, John Wiley & Sons, 2001.

87. C. Zhang, P. Dunghel, D. Wu, K. W. Ross, Unraveling the BitTorrent Ecosystem, IEEE Transactions on Parallel and Distributed Systems, Vol. 22, No. 7 (July 2011).

88. G. Chen, D. Kotz, A Survey of Context-Aware Mobile Computing Research, Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, Nov. 2000. www.cs.dartmouth.edu/reports/TR2000-381.pdf

89. K.-T. Chen, C.-Y. Huang, P. Huang, C.-L. Lei, Quantifying Skype User Satisfaction, Proc. 2006 ACM SIGCOMM (Pisa, Italy, Sept. 2006).

90. K. Chen, C. Guo, H. Wu, J. Yuan, Z. Feng, Y. Chen, S. Lu, W. Wu, Generic and Automatic Address Configuration for Data Center Networks, Proc. 2010 ACM SIGCOMM.

91. Y. Chen, S. Jain, V. K. Adhikari, Z. Zhang, Characterizing Roles of Front-End Servers in End-to-End Performance of Dynamic Content Distribution, Proc. 2011 ACM Internet Measurement Conference (Berlin, Germany, Nov. 2011).

92. T. Chenoweth, R. Minch, S. Tabor, Wireless Insecurity: Examining User Security Behavior on Public Networks, Communications of the ACM, Vol. 53, No. 2 (Feb. 2010), pp. 134–138.

93. B. Cheswick, H. Burch, S. Branigan, Mapping and Visualizing the Internet, Proc. 2000 Usenix Conference (San Diego, CA, June 2000).

94. D. Chiu, R. Jain, Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks, Computer Networks and ISDN Systems, Vol. 17, No. 1, pp. 1–14. www.cs.wustl.edu/~jain/papers/cong_av.htm

95. M. Christiansen, K. Jeffay, D. Ott, F. D. Smith, Tuning Red for Web Traffic, IEEE/ACM Transactions on Networking, Vol. 9, No. 3 (June 2001), pp. 249–264.

96. Y. Chu, S. Rao, S. Seshan, H Zhang, A Case for End System Multicast, IEEE J. Selected Areas in Communications, Vol 20, No. 8 (Oct. 2002), pp. 1456–1471.

97. S. Chuang, S. Iyer, N. McKeown, Practical Algorithms for Performance Guarantees in Buffered Crossbars, Proc. 2005 IEEE INFOCOM.

98. C. Cicconetti, L. Lenzini, A. Mingozi, K. Eklund, Quality of Service Support in 802.16 Networks, IEEE Network Magazine (Mar./Apr. 2006), pp. 50–55.

99. Cisco Systems Inc., Cisco XR 12000 Series and Cisco 12000 Series Routers, www.cisco.com/en/US/products/ps6342/index.html

100. Cisco Systems Inc., Catalyst 8500 Campus Switch Router Architecture, www.cisco.com/c/en/us/support/switches/catalyst-8540-csr-switch/model.html

101. Cisco Visual Networking Index: Forecast and Methodology, 2010–2015, White Paper, 2011.

102. Cisco 2012, Data Centers, www.cisco.com/c/en/us/solutions/data-center-virtualization/analyst_reports.html

103. Cisco Systems Inc., How NAT Works, academy.delmar.edu/courses/download/ciscoios/nat_howitworks.pdf

104. Cisco Systems Inc., Advanced QoS Services for the Intelligent Internet, cs.uccs.edu/~cchow/pub/contentsw/paper/qos/qos_wp.pdf.gz

105. Cisco Systems Inc., Congestion Management Overview, www.cisco.com/en/US/docs/ios/12_2/qos/configuration/guide/qcfconmg.html

106. Cisco Systems Inc, Multiservice Switches, www.cisco.com/warp/public/cc/pd/si/index.shtml

107. Cisco Systems Inc., Defining Strategies to Protect Against TCP SYN Denial of Service Attacks, www.cisco.com/en/US/tech/tk828/technologies_tech_note09186a00800f67d5.shtml

108. Cisco, Visual Networking Index, www.cisco.com/web/solutions/sp/vni/vni_forecast_highlights/index.html

109. D. Clark, The Design Philosophy of the DARPA Internet Protocols, Proc. 1988 ACM SIGCOMM (Stanford, CA, Aug. 1988).

110. I. Clarke, T. W. Hong, S. G. Miller, O. Sandberg, B. Wiley, Protecting Free Expression Online with Freenet, IEEE Internet Computing (Jan.–Feb. 2002), pp. 40–49.

111. D. Cohen, Issues in Transnet Packetized Voice Communication, Proc. Fifth Data Communications Symposium (Snowbird, UT, Sept. 1977), pp. 6–13.

112. B. Cohen, Incentives to Build Robustness in BitTorrent, First Workshop on the Economics of Peer-to-Peer Systems (Berkeley, CA, June 2003).

113. Cookie Central homepage, www.cookiecentral.com/n_cookie_faq.htm

114. X. Zhang, J. Liu, J., B. Li, and T.-S. P. Yum, CoolStreamingDONet/: A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming, Proc. 2005 IEEE INFOCOM (Miami, FL, Mar. 2005).

115. T. H. Cormen, Introduction to Algorithms, 2nd Ed., MIT Press, Cambridge, MA, 2001.

116. B. Crow, I. Widjaja, J. Kim, P. Sakai, IEEE 802.11 Wireless Local Area Networks, IEEE Communications Magazine (Sept. 1997), pp. 116–126.

117. J. Crowcroft, Z. Wang, A. Smith, J. Adams, A Comparison of the IETF and ATM Service Models, IEEE Communications Magazine (Nov./Dec. 1995), pp. 12–16.
118. J. Crowcroft, M. Handley, I. Wakeman, Internetworking Multimedia, Morgan-Kaufman, San Francisco, 1999.
119. A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, DevoFlow: Scaling Flow Management for High-Performance Networks, Proc. 2011 ACM SIGCOMM.
120. M. A. Cusumano, D. B. Yoffie, Competing on Internet Time: Lessons from Netscape and its Battle with Microsoft, Free Press, New York, NY, 1998.
121. E. Dahlman, B. Gudmundson, M. Nilsson, J. Sköld, UMTS/IMT-2000 Based on Wideband CDMA, IEEE Communications Magazine (Sept. 1998), pp. 70–80.
122. J. N. Daigle, Queuing Theory for Telecommunications, Addison-Wesley, Reading, MA, 1991.
123. Y. Dalal, R. Metcalfe, Reverse Path Forwarding of Broadcast Packets, Communications of the ACM, Vol. 21, No. 12 (Dec. 1978), pp. 1040–1048.
124. B. Davie and Y. Rekhter, MPLS: Technology and Applications, Morgan Kaufmann Series in Networking, 2000.
125. G. Davies, F. Kelly, Network Dimensioning, Service Costing, and Pricing in a Packet-Switched Environment, Telecommunications Policy, Vol. 28, No. 4, pp. 391–412.
126. Digital Equipment Corporation, In Memoriam: J. C. R. Licklider 1915–1990, SRC Research Report 61, Aug. 1990. www.memex.org/licklider.pdf
127. J. DeClercq, O. Paridaens, Scalability Implications of Virtual Private Networks, IEEE Communications Magazine, Vol. 40, No. 5 (May 2002), pp. 151–157.
128. A. Demers, S. Keshav, S. Shenker, Analysis and Simulation of a Fair Queuing Algorithm, Internetworking: Research and Experience, Vol. 1, No. 1 (1990), pp. 3–26.
129. D. Denning (Editor), P. Denning (Preface), Internet Besieged: Countering Cyberspace Scofflaws, Addison-Wesley, Reading, MA, 1997.
130. IETF Dynamic Host Configuration working group, datatracker.ietf.org/wg/dhc/charter/
131. P. Dhungel, K. W. Ross, M. Steiner., Y. Tian, X. Hei, Xunlei: Peer-Assisted Download Acceleration on a Massive Scale, Passive and Active Measurement Conference (PAM) 2012, Vienna, 2012.
132. W. Diffie, M. E. Hellman, New Directions in Cryptography, IEEE Transactions on Information Theory, Vol IT-22 (1976), pp. 644–654.
133. S. N. Diggavi, N. Al-Dhahir, A. Stamoulis, R. Calderbank, Great Expectations: The Value of Spatial Diversity in Wireless Networks, Proceedings of the IEEE, Vol. 92, No. 2 (Feb. 2004).
134. J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, B. Weihl, Globally Distributed Content Delivert, IEEE Internet Computing (Sept.–Oct. 2002).
135. Y. Ding, Y. Du, Y. Hu, Z. Liu, L. Wang, K. W. Ross, A. Ghose, Broadcast Yourself: Understanding YouTube Uploaders, Proc. 2011 ACM Internet Measurement Conference (Berlin).
136. C. Diot, B. N. Levine, B. Lyles, H. Kassem, D. Balensiefen, Deployment Issues for the IP Multicast Service and Architecture, IEEE Network, Vol. 14, No. 1 (Jan./Feb. 2000) pp. 78–88.

137. M. Dischinger, A. Haeberlen, K. Gummadi, S. Saroiu, Characterizing residential broadband networks, Proc. 2007 ACM Internet Measurement Conference, pp. 24–26.

138. X. Dmitiropoulos, D. Krioukov, M. Fomenkov, B. Huffaker, Y. Hyun, KC Claffy, G. Riley, AS Relationships: Inference and Validation, ACM Computer Communication Review (Jan. 2007).

139. Data-over-cable service interface specifications: Radio-frequency interface specification. ITU-T J.112, 2004.

140. Data-Over-Cable Service Interface Specifications, DOCSIS 3.0: MAC and Upper Layer Protocols Interface Specification, CM-SP-MULPIv3.0-I16-110623, 2011.

141. M. Dodge, An Atlas of Cyberspaces, www.cybergeography.org/atlas/isp_maps.html

142. M. Donahoo, K. Calvert, TCP/IP Sockets in C: Practical Guide for Programmers, Morgan Kaufman, 2001.

143. J. R. Douceur, The Sybil Attack, First International Workshop on Peer-to-Peer Systems (IPTPS '02) (Cambridge, MA, Mar. 2002).

144. DSL Forum homepage, www.dslforum.org/

145. P. Dhungel, D. Wu, B. Schonhorst, K.W. Ross, A Measurement Study of Attacks on BitTorrent Leechers, 7th International Workshop on Peer-to-Peer Systems (IPTPS 2008) (Tampa Bay, FL, Feb. 2008).

146. R. Droms, T. Lemon, The DHCP Handbook (2nd Edition), SAMS Publishing, 2002.

147. J. Edney and W. A. Arbaugh, Real 802.11 Security: Wi-Fi Protected Access and 802.11i, Addison-Wesley Professional, 2003.

148. W. K. Edwards, R. Grinter, R. Mahajan, D. Wetherall, Advancing the State of Home Networking, Communications of the ACM, Vol. 54, No. 6 (June 2011), pp. 62–71.

149. K. Eklund, R. Marks, K. Stanswood, S. Wang, IEEE Standard 802.16: A Technical Overview of the Wireless MAN Air Interface for Broadband Wireless Access, IEEE Communications Magazine (June 2002), pp. 98–107.

150. H. Ellis, The Story of Non-Secret Encryption, jya.com/ellisdoc.htm

151. Ericsson, LTE–An Introduction, www.ericsson.com/res/docs/2011/lte_an_introduction.pdf

152. Ericsson, The Evolution of Edge, www.iwpc.org/Workshop_Folders/08_03_GSM_EDGE_Extensions/3107_The_evolution_of_EDGE_A.pdf

153. D. Estrin, M. Handley, A. Helmy, P. Huang, D. Thaler, A Dynamic Bootstrap Mechanism for Rendezvous-Based Multicast Routing, Proc. 1998 IEEE INFOCOM (New York, NY, Apr. 1998).

154. J. Falkner, M. Piatek, J.P. John, A. Krishnamurthy, T. Anderson, Profiling a Million Sser DHT, Proc. 2007 ACM Internet Measurement Conference.

155. C. Faloutsos, M. Faloutsos, P. Faloutsos, What Does the Internet Look Like? Empirical Laws of the Internet Topology, Proc. 1999 ACM SIGCOMM (Boston, MA, Aug. 1999).

156. N. Farrington, G. Porter, S. Radhakrishnan, H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, A. Vahdat, Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers, Proc. 2010 ACM SIGCOMM.

157. N. Feamster, J. Winick, J. Rexford, A Model for BGP Routing for Network Engineering, Proc. 2004 ACM SIGMETRICS (New York, NY, June 2004).
158. N. Feamster, H. Balakrishnan, Detecting BGP Configuration Faults with Static Analysis, NSDI (May 2005).
159. M. Feldman J. Chuang, Overcoming Free-Riding Behavior in Peer-to-peer Systems, ACM SIGecom Exchanges (July 2005).
160. D. Feldmeier, Fast Software Implementation of Error Detection Codes, IEEE/ACM Transactions on Networking, Vol. 3, No. 6 (Dec. 1995), pp. 640–652.
161. Federal Information Processing Standard, Secure Hash Standard, FIPS Publication 180-1. www.itl.nist.gov/fipspubs/fip180-1.htm
162. S. Floyd, K. Fall, Promoting the Use of End-to-End Congestion Control in the Internet, IEEE/ACM Transactions on Networking, Vol. 6, No. 5 (Oct. 1998), pp. 458–472.
163. S. Floyd, M. Handley, J. Padhye, J. Widmer, Equation-Based Congestion Control for Unicast Applications, Proc. 2000 ACM SIGCOMM (Stockholm, Sweden, Aug. 2000).
164. S. Floyd, A Report on Some Recent Developments in TCP Congestion Control, IEEE Communications Magazine (Apr. 2001).
165. S. Floyd, References on RED (Random Early Detection) Queue Management, www.icir.org/floyd/red.html
166. S. Floyd, V. Jacobson, Synchronization of Periodic Routing Messages, IEEE/ACM Transactions on Networking, Vol. 2, No. 2 (Apr. 1997) pp. 122–136.
167. S. Floyd, TCP and Explicit Congestion Notification, ACM SIGCOMM Computer Communications Review, Vol. 24, No. 5 (Oct. 1994), pp. 10–23.
168. S. Fluhrer, I. Mantin, A. Shamir, Weaknesses in the Key Scheduling Algorithm of RC4, Eighth Annual Workshop on Selected Areas in Cryptography, (Toronto, Canada, Aug. 2002).
169. B. Fortz, M. Thorup, Internet Traffic Engineering by Optimizing OSPF Weights, Proc. 2000 IEEE INFOCOM (Tel Aviv, Israel, Apr. 2000).
170. B. Fortz, J. Rexford, M. Thorup, Traffic Engineering with Traditional IP Routing Protocols, IEEE Communication Magazine (Oct. 2002).
171. C. Fraleigh, F. Tobagi, C. Diot, Provisioning IP Backbone Networks to Support Latency Sensitive Traffic, Proc. 2003 IEEE INFOCOM (San Francisco, CA, Mar. 2003).
172. M. J. Freedman, E. Freudenthal, D. Mazires, Democratizing Content Publication with Coral, USENIX NSDI, 2004.
173. T. Friedman, D. Towsley Multicast Session Membership Size Estimation, Proc. 1999 IEEE INFOCOM (New York, NY, Mar. 1999).
174. J. Frost, BSD Sockets: A Quick and Dirty Primer, world.std.com/~jimf/papers/sockets/sockets.html
175. FTTH Council, NORTH AMERICAN FTTH STATUS–MARCH 31, 2011 (March 2011), www.ftthcouncil.org
176. FTTH Council, 2011 Broadband Consumer Research (June 2011), www.ftthcouncil.org
177. R. G. Gallagher, P. A. Humblet, P. M. Spira, A Distributed Algorithm for Minimum Weight-Spanning Trees, ACM Trans. on Programming Languages and Systems, Vol. 1, No. 5 (Jan. 1983), pp. 66–77.

178. L. Gao, J. Rexford, Stable Internet Routing Without Global Coordination, IEEE/ACM Transactions on Networking, Vol. 9, No. 6 (Dec. 2001), pp. 681–692.

179. L. Garces-Erce, K. W. Ross, E. Biersack, P. Felber, G. Urvoy-Keller, TOPLUS: Topology Centric Lookup Service, Fifth Int. Workshop on Networked Group Communications (NGC 2003) (Munich, Sept. 2003) cis.poly.edu/~ross/papers/TOPLUS.pdf

180. F. C. Gartner, A Survey of Self-Stabilizing Spanning-Tree Construction Algorithms, Technical Report IC/2003/38, Swiss Federal Institute of Technology (EPFL), School of Computer and Communication Sciences, June 10, 2003. ic2.epfl.ch/publications/documents/IC_TECH_REPORT_200338.pdf

181. L. Gauthier, C. Diot, and J. Kurose, End-to-end Transmission Control Mechanisms for Multiparty Interactive Applications on the Internet, Proc. 1999 IEEE INFOCOM (New York, NY, Apr. 1999).

182. A. Girard, Routing and Dimensioning in Circuit-Switched Networks, Addison-Wesley, Reading, MA, 1990.

183. R. Glitho, Contrasting OSI Systems Management to SNMP and TMN, Journal of Network and Systems Management, Vol. 6, No. 2 (June 1998), pp. 113–131.

184. The Gnutella Protocol Specification, v0.4 www9.limewire.com/developer/gnutella_protocol_0.4.pdf

185. David J. Goodman, Wireless Personal Communications Systems, Prentice-Hall, 1997.

186. Google data centers. www.google.com/corporate/datacenter/locations.html

187. W. Goralski, Frame Relay for High-Speed Networks, John Wiley, New York, 1999.

188. W. Goralski, Optical Networking and WDM, Osborne/McGraw-Hill, Berkeley, CA, 2001.

189. A. Greenberg, J. Hamilton, D. Maltz, P. Patel, The Cost of a Cloud: Research Problems in Data Center Networks, ACM Computer Communications Review (Jan. 2009).

190. A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, S. Sengupta, VL2: A Scalable and Flexible Data Center Network, Proc. 2009 ACM SIGCOMM.

191. A. Greenberg, J. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, S. Sengupta, VL2: A Scalable and Flexible Data Center Network, Communications of the ACM, Vol. 54, No. 3 (Mar. 2011), pp. 95–104.

192. T. Griffin, Interdomain Routing Links, www.cl.cam.ac.uk/~tgg22/interdomain/

193. S. Guha, N. Daswani, R. Jain, An Experimental Study of the Skype Peer-to-Peer VoIP System, Proc. Fifth Int. Workshop on P2P Systems (Santa Barbara, CA, 2006).

194. L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, X. Zhang, Measurement, Analysis, and Modeling of BitTorrent-Like Systems, Proc. 2005 ACM Internet Measurement Conference.

195. C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers, Proc. 2009 ACM SIGCOMM.

196. P. Gupta, N. McKeown, Algorithms for Packet Classification, IEEE Network Magazine, Vol. 15, No. 2 (Mar./Apr. 2001), pp. 24–32.

197. Ha, S., Rhee, I., L. Xu, CUBIC: A New TCP-Friendly High-Speed TCP Variant, ACM SIGOPS Operating System Review, 2008.
198. S. Halabi, Internet Routing Architectures, 2nd Ed., Cisco Press, 2000.
199. D. Halperin, T. Heydt-Benjamin, B. Ransford, S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, W. Maisel, Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses, Proc. 29th Annual IEEE Symposium on Security and Privacy (May 2008).
200. D. Halperin, S. Kandula, J. Padhye, P. Bahl, D. Wetherall, Augmenting Data Center Networks with Multi-Gigabit Wireless Links, Proc. 2011 ACM SIGCOMM.
201. A. A. Hanbali, E. Altman, P. Nain, A Survey of TCP over Ad Hoc Networks, IEEE Commun. Surveys and Tutorials, Vol. 7, No. 3 (2005), pp. 22–36.
202. X. Hei, C. Liang, J. Liang, Y. Liu, K. W. Ross, A Measurement Study of a Large-scale P2P IPTV System, IEEE Trans. on Multimedia (Dec. 2007).
203. J. Heidemann, K. Obraczka, J. Touch, Modeling the Performance of HTTP over Several Transport Protocols, IEEE/ACM Transactions on Networking, Vol. 5, No. 5 (Oct. 1997), pp. 616–630.
204. G. Held, Data Over Wireless Networks: Bluetooth, WAP, and Wireless LANs, McGraw-Hill, 2001.
205. O. Hersent, D. Gurle, J-P. Petit, IP Telephony: Packet-Based Multimedia Communication Systems, Pearson Education Limited, Edinburgh, 2000.
206. G. Holland, N. Vaidya, V. Bahl, A Rate-Adaptive MAC Protocol for Multi-Hop Wireless Networks, Proc. 2001 ACM Int. Conference of Mobile Computing and Networking (Mobicom01) (Rome, Italy, July 2001).
207. C.V. Hollot, V. Misra, D. Towsley, W. Gong, Analysis and design of controllers for AQM routers supporting TCP flows, IEEE Transactions on Automatic Control, Vol. 47, No. 6 (June 2002), pp. 945–959.
208. C. Haung, V. Sharma, K. Owens, V. Makam, Building Reliable MPLS Networks Using a Path Protection Mechanism, IEEE Communications Magazine, Vol. 40, No. 3 (Mar. 2002), pp. 156–162.
209. Y. Huang, R. Guerin, Does Over-Provisioning Become More or Less Efficient as Networks Grow Larger?, Proc. IEEE Int. Conf. Network Protocols (ICNP) (Boston MA, November 2005).
210. C. Huang, Jin Li, K.W. Ross, Can Internet VoD Be Profitable?, Proc 2007 ACM SIGCOMM (Kyoto, Aug. 2007).
211. C. Huang, J. Li, A. Wang, K. W. Ross, Understanding Hybrid CDN-P2P: Why Limelight Needs its Own Red Swoosh, Proc. 2008 NOSSDAV, Braunschweig, Germany.
212. C. Huang, N. Holt, Y. A. Wang, A. Greenberg, J. Li, K. W. Ross, A DNS Reflection Method for Global Traffic Management, Proc. 2010 USENIX, Boston.
213. C. Huitema, IPv6: The New Internet Protocol, 2nd Ed., Prentice Hall, Englewood Cliffs, NJ, 1998.
214. G. Huston, Interconnection, Peering, and Settlements–Part I, The Internet Protocol Journal, Vol. 2, No. 1 (Mar. 1999).
215. G. Huston, NAT Anatomy: A Look Inside Network Address Translators, The Internet Protocol Journal, Vol. 7, No. 3 (Sept. 2004).
216. G. Huston, Confronting IPv4 Address Exhaustion, www.potaroo.net/ispcol/2008-10/v4depletion.html

217. G. Huston, G. Michaelson, IPv6 Deployment: Just where are we? www.potaroo.net/ispcol/2008-04/ipv6.html
218. G. Huston, A Rough Guide to Address Exhaustion, The Internet Protocol Journal, Vol. 14, No. 1 (Mar. 2011).
219. G. Huston, Transitioning Protocols, The Internet Protocol Journal, Vol. 14, No. 1 (Mar. 2011).
220. Internet Architecture Board homepage, www.iab.org/
221. Internet Assigned Number Authority homepage, www.iana.org/
222. Internet Assigned Number Authority, Private Enterprise Numbers www.iana.org/assignments/enterprise-numbers
223. Internet Assigned Numbers Authority, Protocol Numbers, www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml
224. IANA Root Zone Database, www.iana.org/domains/root/db/
225. The Internet Corporation for Assigned Names and Numbers homepage, www.icann.org
226. IEC Online Education, Optical Access, www.iec.org/online/tutorials/opt_acc/index.asp
227. IEEE 802 LAN/MAN Standards Committee homepage, www.ieee802.org/
228. IEEE 802.11, 1999 Edition (ISO/IEC 8802-11: 1999) IEEE Standards for Information Technology–Telecommunications and Information Exchange Between Systems–Local and Metropolitan Area Network–Specific Requirements–Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification, standards.ieee.org/getieee802/download/802.11-1999.pdf
229. IEEE, IEEE P802.11–Task Group N–Meeting Update: Status of 802.11n, grouper.ieee.org/groups/802/11/Reports/tgn_update.htm
230. IEEE 802.15 Working Group for WPAN homepage, grouper.ieee.org/groups/802/15/.
231. IEEE 802.15 WPAN Task Group 4, www.ieee802.org/15/pub/TG4.html
232. IEEE, IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems, standards.ieee.org/getieee802/download/802.16-2004.pdf
233. IEEE, IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems, Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1, standards.ieee.org/getieee802/download/802.16e-2005.pdf
234. IEEE, IEEE Standard for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks, standards.ieee.org/getieee802/download/802.1Q-2005.pdf
235. IEEE Std 802.1X-2001 Port-Based Network Access Control, standards.ieee.org/reading/ieee/std_public/description/lanman/802.1x-2001_desc.html
236. IEEE, IEEE 802.3 CSMA/CD (Ethernet), grouper.ieee.org/groups/802/3/
237. IEEE, IEEE 802.5 homepage, www.ieee802.org/5/www8025org/
238. Internet Engineering Task Force homepage, www.ietf.org
239. S. Ihm, V. S. Pai, Towards Understanding Modern Web Traffic, Proc. 2011 ACM Internet Measurement Conference (Berlin).
240. The IMAP Connection, www.imap.org

241. Intel Corp, Intel® 82544 Gigabit Ethernet Controller, www.intel.ru/content/www/ru/ru/intelligent-systems/embedded-processors-which-intel-processor-fits-your-project.html
242. Intel Corp., WiMax Technology, www.intel.com/technology/wimax/index.htm
243. Internet2 Multicast Working Group homepage, www.internet2.edu/multicast/
244. IPv6.com homepage, www.ipv6.com/
245. Internet Systems Consortium homepage, www.isc.org
246. Information Sciences Institute, DoD Standard Internet Protocol, Internet Engineering Note 123 (Dec. 1979), www.isi.edu/in-notes/ien/ien123.txt
247. International Organization for Standardization homepage, International Organization for Standardization, www.iso.org/
248. International Organization for Standardization, X.680: ITU-T Recommendation X.680 (2002) Information Technology–Abstract Syntax Notation One (ASN.1): Specification of Basic Notation. www.itu.int/ITU-T/studygroups/com17/languages/X.680-0207.pdf
249. Asymmetric Digital Subscriber Line (ADSL) Transceivers. ITU-T G.992.1, 1999.
250. Asymmetric Digital Subscriber Line (ADSL) Transceivers–Extended Bandwidth ADSL2 (ADSL2Plus). ITU-T G.992.5, 2003.
251. International Telecommunication Union, ITU-T X.509, The Directory: Public-key and attribute certificate frameworks (August 2005).
252. International Telecommunication Union, The Internet of Things, 2005, www.itu.int/osg/spu/publications/internetofthings/InternetofThings_summary.pdf
253. The ITU homepage, www.itu.int/ru/Pages/default.aspx
254. International Telecommunications Union, ICT Statistics, www.itu.int/ITU-D/icteye/Reports.aspx
255. ITU, Measuring the Information Society, 2011, www.itu.int/ITU-D/ict/publications/idi/2011/index.html
256. ITU, The World in 2010: ICT Facts and Figures, www.itu.int/ITU-D/ict/material/Telecom09_flyer.pdf
257. International Telecommunication Union, Recommendation Q.2931 (02/95) — Broadband Integrated Services Digital Network (B-ISDN)–Digital subscriber signalling system no. 2 (DSS 2)–User-network interface (UNI)–Layer 3 specification for basic call/connection control.
258. S. Iyer, R. Zhang, N. McKeown, Routers with a Single Stage of Buffering, Proc. 2002 ACM SIGCOMM (Pittsburgh, PA, Aug. 2002).
259. S. Iyer, R. R. Kompella, N. McKeown, Designing Packet Buffers for Router Line Cards, IEEE Transactions on Networking, Vol. 16, No. 3 (June 2008), pp. 705–717.
260. V. Jacobson, Congestion Avoidance and Control, Proc. 1988 ACM SIGCOMM (Stanford, CA, Aug. 1988), pp. 314–329.
261. R. Jain, A timeout-based congestion control scheme for window flow-controlled networks, IEEE Journal on Selected Areas in Communications SAC-4, 7 (Oct. 1986).
262. R. Jain, A Delay-Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks, ACM SIGCOMM Computer Communications Review, Vol. 19, No. 5 (1989), pp. 56–71.

263. R. Jain, FDDI Handbook: High-Speed Networking Using Fiber and Other Media, Addison-Wesley, Reading, MA, 1994.

264. R. Jain. S. Kalyanaraman, S. Fahmy, R. Goyal, S. Kim, Tutorial Paper on ABR Source Behavior, ATM Forum/96-1270, Oct. 1996. www.cse.wustl.edu/~jain/atmf/ftp/atm96-1270.pdf

265. S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, D. Towsley, Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP backbone, Proc. 2003 IEEE INFOCOM.

266. P. Ji, Z. Ge, J. Kurose, D. Towsley, A Comparison of Hard-State and Soft-State Signaling Protocols, Proc. 2003 ACM SIGCOMM (Karlsruhe, Germany, Aug. 2003).

267. W. Jiang, J. Lennox, H. Schulzrinne, K. Singh, Towards Junking the PBX: Deploying IP Telephony, NOSSDAV'01 (Port Jefferson, NY, June 2001).

268. D. Jimenez, Outside Hackers Infiltrate MIT Network, Compromise Security, The Tech, Vol. 117, No 49 (Oct. 1997), p. 1, www-tech.mit.edu/V117/N49/hackers.49n.html

269. C. Jin, D. X. We, S. Low, FAST TCP: Motivation, architecture, algorithms, performance, Proc. 2004 IEEE INFOCOM (Hong Kong, March 2004).

270. H. Kaaranen, S. Naghian, L. Laitinen, A. Ahtiainen, V. Niemi, Networks: Architecture, Mobility and Services, New York: John Wiley & Sons, 2001.

271. D. Kahn, The Codebreakers: The Story of Secret Writing, The Macmillan Company, 1967.

272. R. E. Kahn, S. Gronemeyer, J. Burchfiel, R. Kunzelman, Advances in Packet Radio Technology, Proc. 1978 IEEE INFOCOM, 66, 11 (Nov. 1978).

273. A. Kamerman, L. Monteban, WaveLAN-II: A High–Performance Wireless LAN for the Unlicensed Band, Bell Labs Technical Journal (Summer 1997), pp. 118–133.

274. J. Kangasharju, K. W. Ross, J. W. Roberts, Performance Evaluation of Redirection Schemes in Content Distribution Networks, Proc. 5th Web Caching and Content Distribution Workshop (Lisbon, Portugal, May 2000).

275. K. Kar, M. Kodialam, T. V. Lakshman, Minimum Interference Routing of Bandwidth Guaranteed Tunnels with MPLS Traffic Engineering Applications, IEEE J. Selected Areas in Communications (Dec. 2000).

276. P. Karn, C. Partridge, Improving Round-Trip Time Estimates in Reliable Transport Protocols, Proc. 1987 ACM SIGCOMM.

277. M. Karol, M. Hluchyj, A. Morgan, Input Versus Output Queuing on a Space-Division Packet Switch, IEEE Transactions on Communications, Vol. 35, No. 12 (Dec. 1987), pp. 1347–1356.

278. D. Katabi, M. Handley, C. Rohrs, Internet Congestion Control for Future High Bandwidth-Delay Product Environments, Proc. 2002 ACM SIGCOMM (Pittsburgh, PA, Aug. 2002).

279. I. Katzela, M. Schwartz. Schemes for Fault Identification in Communication Networks, IEEE/ACM Transactions on Networking, Vol. 3, No. 6 (Dec. 1995), pp. 753–764.

280. C. Kaufman, R. Perlman, M. Speciner, Network Security, Private Communication in a Public World, Prentice Hall, Englewood Cliffs, NJ, 1995.

281. F. P. Kelly, A. Maulloo, D. Tan, Rate control for communication networks: Shadow prices, proportional fairness and stability, J. Operations Res. Soc., Vol. 49, No. 3 (Mar. 1998), pp. 237–252.
282. T. Kelly, Scalable TCP: improving performance in high speed wide area networks, ACM SIGCOMM Computer Communications Review, Volume 33, No. 2 (Apr. 2003), pp 83–91.
283. K. Kilkki, Differentiated Services for the Internet, Macmillan Technical Publishing, Indianapolis, IN, 1999.
284. H. Kim, S. Rixner, V. Pai, Network Interface Data Caching, IEEE Transactions on Computers, Vol. 54, No. 11 (Nov. 2005), pp. 1394–1408.
285. C. Kim, M. Caesar, J. Rexford, Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises, Proc. 2008 ACM SIGCOMM (Seattle, WA, Aug. 2008).
286. L. Kleinrock, Information Flow in Large Communication Networks, RLE Quarterly Progress Report, July 1961.
287. L. Kleinrock, 1964 Communication Nets: Stochastic Message Flow and Delay, McGraw-Hill, New York, NY, 1964.
288. L. Kleinrock, Queuing Systems, Vol. 1, John Wiley, New York, 1975.
289. L. Kleinrock, F. A. Tobagi, Packet Switching in Radio Channels: Part I–Carrier Sense Multiple-Access Modes and Their Throughput-Delay Characteristics, IEEE Transactions on Communications, Vol. 23, No. 12 (Dec. 1975), pp. 1400–1416.
290. L. Kleinrock, Queuing Systems, Vol. 2, John Wiley, New York, 1976.
291. L. Kleinrock, The Birth of the Internet, www.lk.cs.ucla.edu/personal_history.html
292. E. Kohler, M. Handley, S. Floyd, DDCP: Designing DCCP: Congestion Control Without Reliability, Proc. 2006 ACM SIGCOMM (Pisa, Italy, Sept. 2006).
293. T. Kolding, K. Pedersen, J. Wigard, F. Frederiksen, P.Mogensen, High Speed Downlink Packet Access: WCDMA Evolution, IEEE Vehicular Technology Society News (Feb. 2003), pp. 4–10.
294. T. Koponen, S. Shenker, H. Balakrishnan, N. Feamster, I. Ganichev, A. Ghodsi, P. B. Godfrey, N. McKeown, G. Parulkar, B. Raghavan, J. Rexford, S. Arianfar, D. Kuptsov, Architecting for Innovation, ACM Computer Communications Review, 2011.
295. J. Korhonen, Introduction to 3G Mobile Communications, 2nd ed., Artech House, 2003.
296. J. Koziol, Intrusion Detection with Snort, Sams Publishing, 2003.
297. B. Krishnamurthy, and J. Rexford, Web Protocols and Practice: HTTP/ 1.1, Networking Protocols, and Traffic Measurement, Addison-Wesley, Boston, MA, 2001.
298. B. Krishnamurthy, C. Wills, Y. Zhang, On the Use and Performance of Content Distribution Networks, Proc. 2001 ACM Internet Measurement Conference.
299. R. Krishnan, H. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, J. Gao, Moving Beyond End-to-end Path Information to Optimize CDN Performance, Proc. 2009 ACM Internet Measurement Conference.
300. S. Kulkarni, C. Rosenberg, Opportunistic Scheduling: Generalizations to Include Multiple Constraints, Multiple Interfaces, and Short Term Fairness, Wireless Networks, 11 (2005), 557–569.

301. R. Kumar, K.W. Ross, Optimal Peer-Assisted File Distribution: Single and Multi-Class Problems, IEEE Workshop on Hot Topics in Web Systems and Technologies (Boston, MA, 2006).

302. C. Labovitz, G. R. Malan, F. Jahanian, Internet Routing Instability, Proc. 1997 ACM SIGCOMM (Cannes, France, Sept. 1997), pp. 115–126.

303. C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, F. Jahanian, Internet Inter-Domain Traffic, Proc. 2010 ACM SIGCOMM.

304. M. Labrador, S. Banerjee, Packet Dropping Policies for ATM and IP Networks, IEEE Communications Surveys, Vol. 2, No. 3 (Third Quarter 1999), pp. 2–14.

305. M. Lacage, M.H. Manshaei, T. Turletti, IEEE 802.11 Rate Adaptation: A Practical Approach, ACM Int. Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM) (Venice, Italy, Oct. 2004).

306. A. Lakhina, M. Crovella, C. Diot, Diagnosing Network-Wide Traffic Anomalies, Proc. 2004 ACM SIGCOMM.

307. A. Lakhina, M. Crovella, C. Diot, Mining Anomalies Using Traffic Feature Distributions, Proc. 2005 ACM SIGCOMM.

308. T. V. Lakshman, U. Madhow, The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss, IEEE/ACM Transactions on Networking, Vol. 5, No. 3 (1997), pp. 336–350.

309. S. Lam, A Carrier Sense Multiple Access Protocol for Local Networks, Computer Networks, Vol. 4 (1980), pp. 21–32.

310. J. Larmouth, Understanding OSI, International Thomson Computer Press 1996.

311. J. Larmouth, Understanding OSI, www.business.salford.ac.uk/legacy/isi/books/osi/osi.html

312. G. Lawton, Is IPv6 Finally Gaining Ground? IEEE Computer Magazine (Aug. 2001), pp. 11–15.

313. S. LeBlond, C. Zhang, A. Legout, K. W. Ross, W. Dabbous, Exploring the Privacy Limits of Real-Time Communication Applications, Proc. 2011 ACM Internet Measurement Conference (Berlin, 2011).

314. S. LeBlond, C. Zhang, A. Legout, K. W. Ross, W. Dabbous, I Know Where You and What You Are Sharing: Exploiting P2P Communications to Invade Users Privacy, Proc. 2011 ACM Internet Measurement Conference (Berlin).

315. T. Leighton, Improving Performance on the Internet, Communications of the ACM, Vol. 52, No. 2 (Feb. 2009), pp. 44–51.

316. B. Leiner, V. Cerf, D. Clark, R. Kahn, L. Kleinrock, D. Lynch, J. Postel, L. Roberts, S. Woolf, A Brief History of the Internet, www.isoc.org/internet/history/brief.html

317. K. Leung, V. O.K. Li, TCP in Wireless Networks: Issues, Approaches, and Challenges, IEEE Commun. Surveys and Tutorials, Vol. 8, No. 4 (2006), pp. 64–79.

318. L. Li, D. Alderson, W. Willinger, J. Doyle, A First-Principles Approach to Understanding the Internet's Router-Level Topology, Proc. 2004 ACM SIGCOMM (Portland, OR, Aug. 2004).

319. J. Li, M. Guidero, Z. Wu, E. Purpus, T. Ehrenkranz, BGP Routing Dynamics Revisited. ACM Computer Communication Review (April 2007).

320. J. Liang, N. Naoumov, K.W. Ross, The Index Poisoning Attack in P2P File-Sharing Systems, Proc. 2006 IEEE INFOCOM (Barcelona, Spain, April 2006).

321. Y. Lin, I. Chlamtac, Wireless and Mobile Network Architectures, John Wiley and Sons, New York, NY, 2001.

322. N. Liogkas, R. Nelson, E. Kohler, L. Zhang, Exploiting BitTorrent For Fun (But Not Profit), 6th International Workshop on Peer-to-Peer Systems (IPTPS 2006).

323. B. Liu, D. Goeckel, D. Towsley, TCP-Cognizant Adaptive Forward Error Correction in Wireless Networks, Proc. 2002 Global Internet.

324. J. Liu, I. Matta, M. Crovella, End-to-End Inference of Loss Nature in a Hybrid Wired/Wireless Environment, Proc. WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks.

325. Z. Liu, P. Dhungel, Di Wu, C. Zhang, K. W. Ross, Understanding and Improving Incentives in Private P2P Communities, ICDCS (Genoa, Italy, 2010).

326. T. Locher, P. Moor, S. Schmid, R. Wattenhofer, Free Riding in BitTorrent is Cheap, Proc. ACM HotNets 2006 (Irvine CA, Nov. 2006).

327. J. Lui, V. Misra, D. Rubenstein, On the Robustness of Soft State Protocols, Proc. IEEE Int. Conference on Network Protocols (ICNP '04), pp. 50–60.

328. A. Luotonen, Web Proxy Servers, Prentice Hall, Englewood Cliffs, NJ, 1998.

329. D. Lynch, M. Rose, Internet System Handbook, Addison-Wesley, Reading, MA, 1993.

330. M. Macedonia, D. Brutzman, MBone Provides Audio and Video Across the Internet, IEEE Computer Magazine, Vol. 27, No. 4 (Apr. 1994), pp. 30–36.

331. J. Mahdavi, S. Floyd, TCP-Friendly Unicast Rate-Based Flow Control, unpublished note (Jan. 1997).

332. Computer Economics, 2005 Malware Report: The Impact of Malicious Code Attacks, www.computereconomics.com

333. IETF Mobile Ad-hoc Networks (manet) Working Group, www.ietf.org/html.charters/manet-charter.html

334. Z. M. Mao, C. Cranor, F. Boudlis, M. Rabinovich, O. Spatscheck, J. Wang, A Precise and Efficient Evaluation of the Proximity Between Web Clients and Their Local DNS Servers, Proc. 2002 USENIX ATC.

335. MaxMind homepage, www.maxmind.com/app/ip-location

336. P. Maymounkov, D. Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. Proceedings of the 1st International Workshop on Peerto-Peer Systems (IPTPS '02) (Mar. 2002), pp. 53–65.

337. N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, M. Horowitz, The Tiny Tera: A Packet Switch Core, IEEE Micro Magazine (Jan.–Feb. 1997).

338. N. McKeown, A Fast Switched Backplane for a Gigabit Switched Router, Business Communications Review, Vol. 27, No. 12. tiny-tera.stanford.edu/~nickm/papers/cisco_fasts_wp.pdf

339. N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, OpenFlow: Enabling Innovation in Campus Networks, ACM SIGCOMM Computer Communication Review, Vol. 38, No. 2 (Apr. 2008).

340. J. McQuillan, I. Richer, E. Rosen, The New Routing Algorithm for the Arpanet, IEEE Transactions on Communications, Vol. 28, No. 5 (May 1980), pp. 711–719.

341. D. Medhi, D. Tipper (eds.), Special Issue: Fault Management in Communication Networks, Journal of Network and Systems Management, Vol. 5. No. 2 (June 1997).

342. R. M. Metcalfe, D. R. Boggs. Ethernet: Distributed Packet Switching for Local Computer Networks, Communications of the Association for Computing Machinery, Vol. 19, No. 7 (July 1976), pp. 395–404.
343. A. Myers, T. Ng, H. Zhang, Rethinking the Service Model: Scaling Ethernet to a Million Nodes, ACM Hotnets Conference, 2004.
344. IP/MPLS Forum homepage, www.ipmplsforum.org/
345. J. Mirkovic, S. Dietrich, D. Dittrich. P. Reiher, Internet Denial of Service: Attack and Defense Mechanisms, Prentice Hall, 2005.
346. P. V. Mockapetris, K. J. Dunlap, Development of the Domain Name System, Proc. 1988 ACM SIGCOMM (Stanford, CA, Aug. 1988).
347. P. Mockapetris, Sigcomm Award Lecture, video доступно на сайте www.postel.org/sigcomm
348. J. Mogul, TCP offload is a dumb idea whose time has come, Proc. HotOS IX: The 9th Workshop on Hot Topics in Operating Systems (2003), USENIX Association.
349. P. Molinaro-Fernandez, N. McKeown, H. Zhang, Is IP Going to Take Over the World (of Communications)?, Proc. 2002 ACM Hotnets.
350. M. L. Molle, K. Sohraby, A. N. Venetsanopoulos, Space-Time Models of Asynchronous CSMA Protocols for Local Area Networks, IEEE Journal on Selected Areas in Communications, Vol. 5, No. 6 (1987), pp. 956–968.
351. D. Moore, G. Voelker, S. Savage, Inferring Internet Denial of Service Activity, Proc. 2001 USENIX Security Symposium (Washington, DC, Aug. 2001).
352. D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, N. Weaver, Inside the Slammer Worm, 2003 IEEE Security and Privacy Conference.
353. A. Moshchuk, T. Bragin, S. Gribble, H. Levy, A Crawler-based Study of Spyware on the Web, Proc. 13th Annual Network and Distributed Systems Security Symposium (NDSS 2006) (San Diego, CA, Feb. 2006).
354. Motorola, Long Term Evolution (LTE): A Technical Overview, www.motorola.com/staticfiles/Business/Solutions/Industry%20Solutions/Service%20Providers/Wireless%20Operators/LTE/_Document/Static%20Files/6834_MotDoc_New.pdf
355. M. Mouly, M. Pautet, The GSM System for Mobile Communications, Cell and Sys, Palaiseau, France, 1992.
356. J. Moy, OSPF: Anatomy of An Internet Routing Protocol, Addison-Wesley, Reading, MA, 1998.
357. J. Mudigonda, P. Yalagandula, J. C. Mogul, B. Stiekes, Y. Pouffary, NetLord: A Scalable Multi-Tenant Network Architecture for Virtualized Datacenters, Proc. 2011 ACM SIGCOMM.
358. B. Mukherjee, Optical Communication Networks, McGraw-Hill, 1997.
359. B. Mukherjee, Optical WDM Networks, Springer, 2006.
360. R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, A. Vahdat, PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric, Proc. 2009 ACM SIGCOMM.
361. B. Nadel, 4G shootout: Verizon LTE vs. Sprint WiMax, Computerworld, February 3, 2011.
362. E. Nahum, T. Barzilai, D. Kandlur, Performance Issues in WWW Servers, IEEE/ACM Transactions on Networking, Vol 10, No. 1 (Feb. 2002).

363. N. Naoumov, K.W. Ross, Exploiting P2P Systems for DDoS Attacks, Intl Workshop on Peer-to-Peer Information Management (Hong Kong, May 2006),

364. G. Neglia, G. Reina, H. Zhang, D. Towsley, A. Venkataramani, J. Danaher, Availability in BitTorrent Systems, Proc. 2007 IEEE INFOCOM.

365. R. Neumann, Internet Routing Black Hole, The Risks Digest: Forum on Risks to the Public in Computers and Related Systems, Vol. 19, No. 12 (May 1997). catless.ncl.ac.uk/Risks/19.12.html#subj1.1

366. G. Neville-Neil, Whither Sockets? Communications of the ACM, Vol. 52, No. 6 (June 2009), pp. 51–55.

367. A Nicholson, Y. Chawathe, M. Chen, B. Noble, D. Wetherall, Improved Access Point Selection, Proc. 2006 ACM Mobisys Conference (Uppsala Sweden, 2006).

368. H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. W. Lie, C. Lilley, Network Performance Effects of HTTP/1.1, CSS1, and PNG, W3C Document, 1997 (also appears in Proc. 1997 ACM SIGCOM (Cannes, France, Sept 1997), pp. 155–166.

369. National Institute of Standards and Technology, Advanced Encryption Standard (AES), Federal Information Processing Standards 197, Nov. 2001, csrc.nist.gov/publications/fips/fips197/fips-197.pdf

370. National Institute of Standards, Estimating IPv6 & DNSSEC Deployment SnapShots, usgv6-deploymon.antd.nist.gov/snap-all.html

371. Nmap homepage, www.insecure.com/nmap

372. J. Nonnenmacher, E. Biersak, D. Towsley, Parity-Based Loss Recovery for Reliable Multicast Transmission, IEEE/ACM Transactions on Networking, Vol. 6, No. 4 (Aug. 1998), pp. 349–361.

373. National Telecommunications and Information Administration (NTIA), US Department of Commerce, Management of Internet names and addresses, Docket Number: 980212036-8146-02. www.ntia.doc.gov/ntiahome/domainname/6_5_98dns.htm

374. M. O'Dell, Network Front-End Processors, Yet Again, Communications of the ACM, Vol. 52, No. 6 (June 2009), pp. 46–50.

375. OID Repository, www.oid-info.com/

376. International Organization for Standardization homepage, www.iso.org/iso/en/ISOOnline.frontpage

377. OSS Nokalva, ASN.1 Resources, www.oss.com/asn1/

378. J. Padhye, V. Firoiu, D. Towsley, J. Kurose, Modeling TCP Reno Performance: A Simple Model and its Empirical Validation, IEEE/ACM Transactions on Networking, Vol. 8 No. 2 (Apr. 2000), pp. 133–145.

379. J. Padhye, S. Floyd, On Inferring TCP Behavior, Proc. 2001 ACM SIGCOMM (San Diego, CA, Aug. 2001).

380. P. Pan, H. Schulzrinne, Staged Refresh Timers for RSVP, Proc. 2nd Global Internet Conference (Phoenix, AZ, Dec. 1997).

381. A. Parekh, R. Gallagher, A generalized processor sharing approach to flow control in integrated services networks: the single-node case, IEEE/ACM Transactions on Networking, Vol. 1, No. 3 (June 1993), pp. 344–357.

382. C. Partridge, S. Pink, An Implementation of the Revised Internet Stream Protocol (ST-2), Journal of Internetworking: Research and Experience, Vol. 3, No. 1 (Mar. 1992).

383. C. Partridge, et al. A Fifty Gigabit per second IP Router, IEEE/ACM Transactions on Networking, Vol. 6, No. 3 (Jun. 1998), pp. 237–248.
384. A. Pathak, Y. A. Wang, C. Huang, A. Greenberg, Y. C. Hu, J. Li, K. W. Ross, Measuring and Evaluating TCP Splitting for Cloud Services, Passive and Active Measurement (PAM) Conference (Zurich, 2010).
385. V. Paxson, End-to-End Internet Packet Dynamics, Proc. 1997 ACM SIGCOMM (Cannes, France, Sept. 1997).
386. A. Perkins, Networking with Bob Metcalfe, The Red Herring Magazine (Nov. 1994).
387. C. Perkins, O. Hodson, V. Hardman, A Survey of Packet Loss Recovery Techniques for Streaming Audio, IEEE Network Magazine (Sept./Oct. 1998), pp. 40–47.
388. C. Perkins, Mobile IP: Design Principles and Practice, Addison-Wesley, Reading, MA, 1998.
389. C. Perkins, Ad Hoc Networking, Addison-Wesley, Reading, MA, 2000.
390. R. Perlman, Interconnections: Bridges, Routers, Switches, and Internetworking Protocols, 2nd ed., Addison-Wesley Professional Computing Series, Reading, MA, 1999.
391. The International PGP Home Page, www.pgpi.org
392. L. Phifer, The Trouble with NAT, The Internet Protocol Journal, Vol. 3, No. 4 (Dec. 2000), www.cisco.com/web/about/ac123/ac147/ac174/ac182/about_cisco_ipj_archive_article09186a00800c83ec.html
393. M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, A. Venkataramani, Do Incentives Build Robustness in Bittorrent?, Proc. NSDI (2007).
394. M. Piatek, T. Isdal, A. Krishnamurthy, T. Anderson, One hop Reputations for Peer-to-peer File Sharing Workloads, Proc. NSDI (2008).
395. R. Pickholtz, D. Schilling, L. Milstein, Theory of Spread Spectrum Communication–a Tutorial, IEEE Transactions on Communications, Vol. 30, No. 5 (May 1982), pp. 855–884.
396. PingPlotter homepage, www.pingplotter.com
397. D. Piscatello, A. Lyman Chapin, Open Systems Networking, Addison-Wesley, Reading, MA, 1993.
398. Point Topic Ltd., World Broadband Statistics Q1 2008, www.tendencias21.net/attachment/96525/
399. Growth of the BGP Table–1994 to Present, bgp.potaroo.net/
400. PPLive homepage, www.pplive.com
401. Quagga, Quagga Routing Suite, www.quagga.net/
402. J. Quittner, M. Slatalla, Speeding the Net: The Inside Story of Netscape and How it Challenged Microsoft, Atlantic Monthly Press, 1998.
403. IP Intelligence, www.quova.com/
404. C. Raiciu , S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, M. Handley, Improving Datacenter Performance and Robustness with Multipath TCP, Proc. 2011 ACM SIGCOMM.
405. K. K. Ramakrishnan, R. Jain, A Binary Feedback Scheme for Congestion Avoidance in Computer Networks, ACM Transactions on Computer Systems, Vol. 8, No. 2 (May 1990), pp. 158–181.

406. S. Raman, S. McCanne, A Model, Analysis, and Protocol Framework for Soft State-based Communication, Proc. 1999 ACM SIGCOMM (Boston, MA, Aug. 1999).
407. B. Raman, K. Chebrolu, Experiences in using Wi-Fi for Rural Internet in India, IEEE Communications Magazine, Special Issue on New Directions in Networking Technologies in Emerging Economies (Jan. 2007).
408. R. Ramaswami, K. Sivarajan, G. Sasaki, Optical Networks: A Practical Perspective, Morgan Kaufman Publishers, 2010.
409. R. Ramjee, J. Kurose, D. Towsley, H. Schulzrinne, Adaptive Playout Mechanisms for Packetized Audio Applications in Wide-Area Networks, Proc. 1994 IEEE INFOCOM.
410. K. R. Rao and J. J. Hwang, Techniques and Standards for Image, Video and Audio Coding, Prentice Hall, Englewood Cliffs, NJ, 1996.
411. A. S. Rao, Y. S. Lim, C. Barakat, A. Legout, D. Towsley, W. Dabbous, Network Characteristics of Video Streaming Traffic, Proc. 2011 ACM CoNEXT (Tokyo).
412. Robust Audio Tool, www-mice.cs.ucl.ac.uk/multimedia/software/rat/
413. S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A Scalable Content-Addressable Network, Proc. 2001 ACM SIGCOMM (San Diego, CA, Aug. 2001).
414. S. Ren, L. Guo, and X. Zhang, ASAP: an AS-aware peer-relay protocol for high quality VoIP, Proc. 2006 IEEE ICDCS (Lisboa, Portugal, July 2006).
415. E. Rescorla, SSL and TLS: Designing and Building Secure Systems, Addison-Wesley, Boston, 2001.
416. S. Crocker, Host Software, RFC 001 (самый первый RFC!).
417. J. Postel, User Datagram Protocol, RFC 768, Aug. 1980.
418. E. Rosen, Vulnerabilities of Network Control Protocols, RFC 789.
419. J. Postel, Internet Protocol: DARPA Internet Program Protocol Specification, RFC 791, Sept. 1981.
420. J. Postel, Internet Control Message Protocol, RFC 792, Sept. 1981.
421. J. Postel, Transmission Control Protocol, RFC 793, Sept. 1981.
422. J. Postel, NCP/TCP Transition Plan, RFC 801, Nov. 1981.
423. D. C. Plummer, An Ethernet Address Resolution Protocol–or–Converting Network Protocol Addresses to 48 bit Ethernet Address for Transmission on Ethernet Hardware, RFC 826, Nov. 1982.
424. V. Cerf, Packet Satellite Technology Reference Sources, RFC 829, Nov. 1982.
425. J. Postel, J. Reynolds, TELNET Protocol Specification, RFC 854, May 1993.
426. J. Mogul, J. Postel, Internet Standard Subnetting Procedure, RFC 950, Aug. 1985.
427. J. Postel and J. Reynolds, File Transfer Protocol (FTP), RFC 959, Oct. 1985.
428. B. Kantor, P. Lapsley, Network News Transfer Protocol, RFC 977, Feb. 1986.
429. J. Davin, J.D. Case, M. Fedor, M. Schoffstall, A Simple Gateway Monitoring Protocol, RFC 1028, Nov. 1987.
430. P. V. Mockapetris, Domain Names–Concepts and Facilities, RFC 1034, Nov. 1987.
431. P. Mockapetris, Domain Names–Implementation and Specification, RFC 1035, Nov. 1987.
432. C. L. Hendrick, Routing Information Protocol, RFC 1058, June 1988.

433. R. Braden, D. Borman, and C. Partridge, Computing The Internet Checksum, RFC 1071, Sept. 1988.
434. D. Waitzman, C. Partridge, S. Deering, Distance Vector Multicast Routing Protocol, RFC 1075, Nov. 1988.
435. S. Deering, Host Extension for IP Multicasting, RFC 1112, Aug. 1989.
436. R. Braden, Requirements for Internet Hosts–Communication Layers, RFC 1122, Oct. 1989.
437. R. Braden, ed., Requirements for Internet Hosts–Application and Support, RFC-1123, Oct. 1989.
438. D. Oran, OSI IS-IS Intra-Domain Routing Protocol, RFC 1142, Feb. 1990.
439. C. Topolcic, Experimental Internet Stream Protocol: Version 2 (ST-II), RFC 1190, Oct. 1990.
440. J. Mogul, S. Deering, Path MTU Discovery, RFC 1191, Nov. 1990.
441. K. McCloghrie, M. T. Rose, Management Information Base for Network Management of TCP/IP-based internets: MIB-II, RFC 1213, Mar. 1991.
442. S. Deering, ICMP Router Discovery Messages, RFC 1256, Sept. 1991.
443. R. Rivest, The MD4 Message-Digest Algorithm, RFC 1320, Apr. 1992.
444. R. Rivest, The MD5 Message-Digest Algorithm, RFC 1321, Apr. 1992.
445. V. Jacobson, S. Braden, D. Borman, TCP Extensions for High Performance, RFC 1323, May 1992.
446. S. Kent, Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management, RFC 1422.
447. C. Partridge, T. Mendez, W. Milliken, Host Anycasting Service, RFC 1546, 1993.
448. D. Perkins, Requirements for an Internet Standard Point-to-Point Protocol, RFC 1547, Dec. 1993.
449. J. Moy, Multicast Extensions to OSPF, RFC 1584, Mar. 1994.
450. R. Braden, D. Clark, S. Shenker, Integrated Services in the Internet Architecture: an Overview, RFC 1633, June 1994.
451. R. Braden, D. Clark, S. Crocker, C. Huitema, Report of IAB Workshop on Security in the Internet Architecture, RFC 1636, Nov. 1994.
452. W. Simpson (ed.), The Point-to-Point Protocol (PPP), RFC 1661, July 1994.
453. W. Simpson (ed.), PPP in HDLC-Like Framing, RFC 1662, July 1994.
454. J. Reynolds and J. Postel, Assigned Numbers, RFC 1700, Oct. 1994.
455. S. Bradner, A. Mankin, The Recommendations for the IP Next Generation Protocol, RFC 1752, Jan. 1995.
456. Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear, Address Allocation for Private Internets, RFC 1918, Feb. 1996.
457. J. Hawkinson, T. Bates, Guidelines for Creation, Selection, and Registration of an Autonomous System (AS), RFC 1930, Mar. 1996.
458. N. Haller, C. Metz, A One-Time Password System, RFC 1938, May 1996.
459. J. Myers and M. Rose, Post Office Protocol–Version 3, RFC 1939, May 1996.
460. T. Berners-Lee, R. Fielding, H. Frystyk, Hypertext Transfer Protocol — HTTP/1.0, RFC 1945, May 1996.
461. C. Perkins, IP Encapsulation within IP, RFC 2003, Oct. 1996.
462. C. Perkins, Minimal Encapsulation within IP, RFC 2004, Oct. 1996.
463. M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP Selective Acknowledgment Options, RFC 2018, Oct. 1996.

464. K. Hubbard, M. Kosters, D. Conrad, D. Karrenberg, J. Postel, Internet Registry IP Allocation Guidelines, RFC 2050, Nov. 1996.
465. H. Krawczyk, M. Bellare, R. Canetti, HMAC: Keyed-Hashing for Message Authentication, RFC 2104, Feb. 1997.
466. R. Droms, Dynamic Host Configuration Protocol, RFC 2131, Mar. 1997.
467. P. Vixie, S. Thomson, Y. Rekhter, J. Bound, Dynamic Updates in the Domain Name System, RFC 2136, Apr. 1997.
468. W. Simpson, PPP Vendor Extensions, RFC 2153, May 1997.
469. R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, S. Jamin, Resource ReSerVation Protocol (RSVP)–Version 1 Functional Specification, RFC 2205, Sept. 1997.
470. J. Wroclawski, The Use of RSVP with IETF Integrated Services, RFC 2210, Sept. 1997.
471. J. Wroclawski, Specification of the Controlled-Load Network Element Service, RFC 2211, Sept. 1997.
472. S. Shenker, J. Wroclawski, General Characterization Parameters for Integrated Service Network Elements, RFC 2215, Sept. 1997.
473. H. Schulzrinne, A. Rao, R. Lanphier, Real Time Streaming Protocol (RTSP), RFC 2326, Apr. 1998.
474. J. Moy, OSPF Version 2, RFC 2328, Apr. 1998.
475. H. Kummert, The PPP Triple-DES Encryption Protocol (3DESE), RFC 2420, Sept. 1998.
476. G. Malkin, RIP Version 2, RFC 2453, Nov. 1998.
477. S. Deering, R. Hinden, Internet Protocol, Version 6 (IPv6) Specification, RFC 2460, Dec. 1998.
478. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, An Architecture for Differentiated Services, RFC 2475, Dec. 1998.
479. K. McCloghrie, D. Perkins, J. Schoenwaelder, Structure of Management Information Version 2 (SMIv2), RFC 2578, Apr. 1999.
480. K. McCloghrie, D. Perkins, J. Schoenwaelder, Textual Conventions for SMIv2, RFC 2579, Apr. 1999.
481. K. McCloghrie, D. Perkins, J. Schoenwaelder, Conformance Statements for SMIv2, RFC 2580, Apr. 1999.
482. J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, Assured Forwarding PHB Group, RFC 2597, June 1999.
483. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, R. Fielding, Hypertext Transfer Protocol–HTTP/1.1, RFC 2616, June 1999.
484. P. Srisuresh, M. Holdrege, IP Network Address Translator (NAT) Terminology and Considerations, RFC 2663.
485. D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, J. McManus, Requirements for Traffic Engineering Over MPLS, RFC 2702, Sept. 1999.
486. P. Ferguson, D. Senie, Network Ingress Filtering: Defeating Denial of Service Attacks which Employ IP Source Address Spoofing, RFC 2827, May 2000.
487. C. Rigney, S. Willens, A. Rubens, W. Simpson, Remote Authentication Dial In User Service (RADIUS), RFC 2865, June 2000.
488. L. Berger, D. Gan, G. Swallow, P. Pan, F. Tommasi, S. Molendini, RSVP Refresh Overhead Reduction Extensions, RFC 2961, Apr. 2001.

489. B. Wellington, Secure Domain Name System (DNS) Dynamic Update, RFC 3007, Nov. 2000.
490. P. Srisuresh, K. Egevang, Traditional IP Network Address Translator (Traditional NAT), RFC 3022, Jan. 2001.
491. P. Srisuresh, K. Egevang, Traditional IP Network Address Translator (Traditional NAT), RFC 3022, Jan. 2001.
492. E. Rosen, A. Viswanathan, R. Callon, Multiprotocol Label Switching Architecture, RFC 3031, Jan. 2001.
493. E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, A. Conta, MPLS Label Stack Encoding, RFC 3032, Jan. 2001.
494. M. Eder, S. Nag, Service Management Architectures Issues and Review, RFC 3052, Jan. 2001.
495. L. Sanchez, K. McCloghrie, J. Saperia, Requirements for Configuration Management of IP-Based Networks, RFC 3139, June 2001.
496. K. Ramakrishnan, S. Floyd, D. Black, The Addition of Explicit Congestion Notification (ECN) to IP, RFC 3168, Sept. 2001.
497. D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, RSVP-TE: Extensions to RSVP for LSP Tunnels, RFC 3209, Dec. 2001.
498. G. Huston, Commentary on Inter-Domain Routing in the Internet, RFC 3221, Dec. 2001.
499. J. Reynolds, Assigned Numbers: RFC 1700 is Replaced by an On-line Database, RFC 3232, Jan. 2002.
500. B. Davie, A. Charny, J.C.R. Bennet, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, D. Stiliadis, An Expedited Forwarding PHB (Per-Hop Behavior), RFC 3246, Mar. 2002.
501. D. Grossman, New Terminology and Clarifications for Diffserv, RFC 3260, Apr. 2002.
502. J. Rosenberg, H. Schulzrinne, G. Carmarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, SIP: Session Initiation Protocol, RFC 3261, July 2002.
503. J. Boyle, V. Gill, A. Hannan, D. Cooper, D. Awduche, B. Christian, W.S. Lai, Overview and Principles of Internet Traffic Engineering, RFC 3272, May 2002.
504. L. Ong, J. Yoakum, An Introduction to the Stream Control Transmission Protocol (SCTP), RFC 3286, May 2002.
505. J. Boyle, V. Gill, A. Hannan, D. Cooper, D. Awduche, B. Christian, W. S. Lai, Applicability Statement for Traffic Engineering with MPLS, RFC 3346, Aug. 2002.
506. B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan, Internet Group Management Protocol, Version 3, RFC 3376, Oct. 2002.
507. M. Allman, S. Floyd, C. Partridge, Increasing TCP's Initial Window, RFC 3390, Oct. 2002.
508. J. Case, R. Mundy, D. Partain, Introduction and Applicability Statements for Internet Standard Management Framework, RFC 3410, Dec. 2002.
509. D. Harrington, R. Presuhn, B. Wijnen, An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks, RFC 3411, Dec. 2002.
510. U. Blumenthal and B. Wijnen, User-based Security Model (USM) for Version 3 of the Simple Network Management Protocol (SNMPv3), RFC 3414, December 2002.

511. B. Wijnen, R. Presuhn, K. McCloghrie, View-based Access Control Model (VACM) for the Simple Network Management Protocol (SNMP), RFC 3415, Dec. 2002.
512. R. Presuhn, J. Case, K. McCloghrie, M. Rose, S. Waldbusser, Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP), Dec. 2002.
513. R. Bush and D. Meyer, Some internet architectural guidelines and philosophy, RFC 3439, Dec. 2003.
514. J. Jonsson, B. Kaliski, Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1, RFC 3447, Feb. 2003.
515. L. Andersson, G. Swallow, The Multiprotocol Label Switching (MPLS) Working Group Decision on MPLS Signaling Protocols, RFC 3468, Feb. 2003.
516. V. Sharma, Ed., F. Hellstrand, Ed, Framework for Multi-Protocol Label Switching (MPLS)-based Recovery, RFC 3469, Feb. 2003. ftp://ftp.rfc-editor.org/in-notes/rfc3469.txt
517. M. Crispin, Internet Message Access Protocol–Version 4rev1, RFC 3501, Mar. 2003.
518. H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, RFC 3550, July 2003.
519. S. Bhattacharyya (ed.), An Overview of Source-Specific Multicast (SSM), RFC 3569, July 2003.
520. P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko, Diameter Base Protocol, RFC 3588, Sept. 2003.
521. B. Fenner, D. Meyer, Ed., Multicast Source Discovery Protocol (MSDP), RFC 3618, Oct. 2003.
522. S. Floyd, High Speed TCP for Large Congestion Windows, RFC 3649, Dec. 2003.
523. B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz, Ed., Extensible Authentication Protocol (EAP), RFC 3748, June 2004.
524. S. Floyd, T. Henderson, A. Gurtov, The NewReno Modification to TCP's Fast Recovery Algorithm, RFC 3782, Apr. 2004.
525. A. Adams, J. Nicholas, W. Siadak, Protocol Independent Multicast–Dense Mode (PIM-DM): Protocol Specification (Revised), RFC 3973, Jan. 2005.
526. R. Raghunarayan, Ed., Management Information Base for the Transmission Control Protocol (TCP), RFC 4022, Mar. 2005.
527. B. Fenner, J. Flick, Management Information Base for the User Datagram Protocol (UDP), RFC 4113, June 2005.
528. E. Nordmark, R. Gilligan, Basic Transition Mechanisms for IPv6 Hosts and Routers, RFC 4213, Oct. 2005.
529. Y. Rekhter, T. Li, S. Hares, Ed., A Border Gateway Protocol 4 (BGP-4), RFC 4271, Jan. 2006.
530. S. Murphy, BGP Security Vulnerabilities Analysis, RFC 4274, Jan. 2006.
531. Meyer, D. and K. Patel, BGP-4 Protocol Analysis, RFC 4274, January 2006.
532. R. Hinden, S. Deering, IP Version 6 Addressing Architecture, RFC 4291, February 2006.
533. S. Routhier, Ed. Management Information Base for the Internet Protocol (IP), RFC 4293, Apr. 2006.
534. S. Kent, K. Seo, Security Architecture for the Internet Protocol, RFC 4301, Dec. 2005.

535. S. Kent, IP Authentication Header, RFC 4302, Dec. 2005.
536. S. Kent, IP Encapsulating Security Payload (ESP), RFC 4303, Dec. 2005.
537. D. Eastlake, Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH), RFC 4305, Dec. 2005.
538. E. Kohler, M. Handley, S. Floyd, Datagram Congestion Control Protocol (DCCP), RFC 4340, Mar. 2006.
539. A. Conta, S. Deering, M. Gupta, Ed., Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification, RFC 4443, Mar. 2006.
540. T. Dierks, E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.1, RFC 4346, Apr. 2006.
541. S. Waldbusser, Remote Network Monitoring Management Information Base Version 2, RFC 4502, May 2006.
542. K. Zeilenga, Ed., Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names, RFC 4514, June 2006.
543. B. Fenner, M. Handley, H. Holbrook, I. Kouvelas, Protocol Independent Multicast–Sparse Mode (PIM-SM): Protocol Specification (Revised), RFC 4601, Aug. 2006.
544. H. Holbrook, B. Cain, Source-Specific Multicast for IP, RFC 4607, Aug. 2006.
545. M. McBride, J. Meylor, D. Meyer, Multicast Source Discovery Protocol (MSDP) Deployment Scenarios, RFC 4611, Aug. 2006.
546. V. Fuller, T. Li, Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan, RFC 4632, Aug. 2006.
547. R. Stewart, ed., Stream Control Transmission Protocol, RFC 4960, Sept. 2007.
548. W. Eddy, TCP SYN Flooding Attacks and Common Mitigations, RFC 4987, Aug. 2007.
549. RFC editor, Internet Official Protocol Standards, RFC 5000, May 2008.
550. A. Li (ed.), RTP Payload Format for Generic Forward Error Correction, RFC 5109, Dec. 2007.
551. P. Savola, Overview of the Internet Multicast Routing Architecture, RFC 5110, Jan. 2008.
552. D. Simon, B. Aboba, R. Hurst, The EAP-TLS Authentication Protocol, RFC 5216, Mar. 2008.
553. D. Thaler, B. Aboba, What Makes for a Successful Protocol?, RFC 5218, July 2008.
554. J. Klensin, Simple Mail Transfer Protocol, RFC 5321, Oct. 2008.
555. P. Resnick, Ed., Internet Message Format, RFC 5322, Oct. 2008.
556. S. Floyd, M. Handley, J. Padhye, J.Widmer, TCP Friendly Rate Control (TFRC): Protocol Specification, RFC 5348, Sept. 2008.
557. J Rosenberg, A Hitchhiker's Guide to the Session Initiation Protocol (SIP), RFC 5411, Feb. 2009.
558. M. Allman, V. Paxson, E. Blanton, TCP Congestion Control, RFC 5681, Sept. 2009.
559. C. Perkins, Ed., IP Mobility Support for IPv4, Revised, RFC 5944, November 2010.

560. C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, Internet Key Exchange Protocol Version 2 (IKEv2), RFC 5996, Sept. 2010.
561. S. Frankel, S. Krishnan, IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap, RFC 6071, Feb. 2011.
562. A Barth, HTTP State Management Mechanism, RFC 6265, Apr. 2011.
563. V. Paxson, M. Allman, J. Chu, M. Sargent, Computing TCP's Retransmission Timer, RFC 6298, June 2011.
564. I. Rhee, Error Control Techniques for Interactive Low-Bit Rate Video Transmission over the Internet, Proc. 1998 ACM SIGCOMM (Vancouver BC, Aug. 1998).
565. L. Roberts, T. Merril, Toward a Cooperative Network of Time-Shared Computers, AFIPS Fall Conference (Oct. 1966).
566. J. Roberts, Internet Traffic, QoS and Pricing, Proc. 2004 IEEE INFOCOM, Vol. 92, No. 9 (Sept. 2004), pp. 1389–1399.
567. R. Rodrigues, P. Druschel, Peer-to-Peer Systems, Communications of the ACM, Vol. 53, No. 10 (Oct. 2010), pp. 72–82.
568. Rohde and Schwarz, UMTS Long Term Evolution (LTE) Technology Introduction, Application Note 1MA111.
569. R. Rom, M. Sidi, Multiple Access Protocols: Performance and Analysis, Springer-Verlag, New York, 1990.
570. Root Servers homepage, www.root-servers.org/
571. M. Rose, The Simple Book: An Introduction to Internet Management, Revised Second Edition, Prentice Hall, Englewood Cliffs, NJ, 1996.
572. K. W. Ross, Multiservice Loss Models for Broadband Telecommunication Networks, Springer, Berlin, 1995.
573. A. Rowston, P. Druschel, Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems, Proc. 2001 IFIP/ACM Middleware (Heidelberg, Germany, 2001).
574. R. Rivest, A. Shamir, L. Adelman, A Method for Obtaining Digital Signatures and Public-key Cryptosystems, Communications of the ACM, Vol. 21, No. 2 (Feb. 1978), pp. 120–126.
575. RSA Laboratories, How Fast is RSA? www.rsa.com/rsalabs/node.asp?id=2215
576. RSA Laboratories, How large a key should be used in the RSA Crypto system? www.rsa.com/rsalabs/node.asp?id=2218
577. D. Rubenstein, J. Kurose, D. Towsley, Real-Time Reliable Multicast Using Proactive Forward Error Correction, Proceedings of NOSSDAV '98 (Cambridge, UK, July 1998).
578. A. Rubin, White-Hat Security Arsenal: Tackling the Threats, Addison-Wesley, 2001.
579. M. Ruiz-Sánchez, E. Biersack, W. Dabbous, Survey and Taxonomy of IP Address Lookup Algorithms, IEEE Network Magazine, Vol. 15, No. 2 (Mar./Apr. 2001), pp. 8–23.
580. J. Saltzer, D. Reed, D. Clark, End-to-End Arguments in System Design, ACM Transactions on Computer Systems (TOCS), Vol. 2, No. 4 (Nov. 1984).
581. Global Internet Phenomena Report, Spring 2011, www.sandvine.com/downloads/general/global-internet-phenomena/2011/1h-2011-global-internet-phenomena-report.pdf.

582. B. Sardar, D. Saha, A Survey of TCP Enhancements for Last-Hop Wireless Networks, IEEE Commun. Surveys and Tutorials, Vol. 8, No. 3 (2006), pp. 20–34.
583. S. Saroiu, P.K. Gummadi, S.D. Gribble, A Measurement Study of Peer-to-Peer File Sharing Systems, Proc. of Multimedia Computing and Networking (MMCN) (2002).
584. S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, An Analysis of Internet Content Delivery Systems, USENIX OSDI (2002).
585. T. Saydam, T. Magedanz, From Networks and Network Management into Service and Service Management, Journal of Networks and System Management, Vol. 4, No. 4 (Dec. 1996), pp. 345–348.
586. J. Schiller, Mobile Communications 2nd edition, Addison Wesley, 2003.
587. B. Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, John Wiley and Sons, 1995.
588. H. Schulzrinne, A Comprehensive Multimedia Control Architecture for the Internet, NOSSDAV'97 (Network and Operating System Support for Digital Audio and Video) (St. Louis, MO, May 1997).
589. Henning Schulzrinne's RTP site, www.cs.columbia.edu/~hgs/rtp
590. Henning Schulzrinne's RTSP site, www.cs.columbia.edu/~hgs/rtsp
591. Henning Schulzrinne's SIP site, www.cs.columbia.edu/~hgs/sip
592. M. Schwartz, Computer-Communication Network Design and Analysis, Prentice-Hall, Englewood Cliffs, N.J., 1997.
593. M. Schwartz, Information, Transmission, Modulation, and Noise, McGraw Hill, New York, NY 1980.
594. M. Schwartz, Performance Analysis of the SNA Virtual Route Pacing Control, IEEE Transactions on Communications, Vol. 30, No. 1 (Jan. 1982), pp. 172–184.
595. J. Scourias, Overview of the Global System for Mobile Communications: GSM. www.privateline.com/PCS/GSM0.html
596. S. Segaller, Nerds 2.0.1, A Brief History of the Internet, TV Books, New York, 1998.
597. N. Shacham, P. McKenney, Packet Recovery in High-Speed Networks Using Coding and Buffer Management, Proc. 1990 IEEE INFOCOM (San Francisco, CA, Apr. 1990), pp. 124–131.
598. A. Shaikh, R. Tewari, M. Agrawal, On the Effectiveness of DNS-based Server Selection, Proc. 2001 IEEE INFOCOM.
599. P. Sharma, E, Perry, R. Malpani, IP Multicast Operational Network management: Design, Challenges, and Experiences, IEEE Network Magazine (Mar. 2003), pp. 49–55.
600. S. Singh, The Code Book: The Evolution of Secrecy from Mary, Queen of Scotsto Quantum Cryptography, Doubleday Press, 1999.
601. H. Schulzrinne Software Package site, www.cs.columbia.edu/IRT/software/
602. E. Skoudis, L. Zeltser, Malware: Fighting Malicious Code, Prentice Hall, 2004.
603. E. Skoudis, T. Liston, Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses (2nd Edition), Prentice Hall, 2006.
604. Skype homepage, www.skype.com
605. W3C Synchronized Multimedia homepage, www.w3.org/AudioVideo
606. J. Smith, Fighting Physics: A Tough Battle, Communications of the ACM, Vol. 52, No. 7 (July 2009), pp. 60–65.

607. Sourcefire Inc., Snort homepage, www.snort.org/
608. S. J. Solari, Digital Video and Audio Compression, McGraw Hill, New York, NY, 1997.
609. F. Solensky, IPv4 Address Lifetime Expectations, in IPng: Internet Protocol Next Generation (S. Bradner, A. Mankin, ed.), Addison-Wesley, Reading, MA, 1996.
610. J. D. Spragins, Telecommunications Protocols and Design, Addison-Wesley, Reading, MA, 1991.
611. R. Srikant, The Mathematics of Internet Congestion Control, Birkhauser, 2004
612. K. Sripanidkulchai, B. Maggs, and H. Zhang, An analysis of live streaming workloads on the Internet, Proc. 2004 ACM Internet Measurement Conference (Taormina, Sicily, Italy), pp. 41–54.
613. W. Stallings, SNMP, SNMP v2, and CMIP The Practical Guide to Network Management Standards, Addison-Wesley, Reading, MA, 1993.
614. W. Stallings, SNMP, SNMPv2, SNMPv3, and RMON 1 and 2, Addison-Wesley, Reading, MA, 1999.
615. M. Steinder, A. Sethi, Increasing robustness of fault localization through analysis of lost, spurious, and positive symptoms, Proc. 2002 IEEE INFOCOM.
616. W. R. Stevens, Unix Network Programming, Prentice-Hall, Englewood Cliffs, NJ.
617. W. R. Stevens, TCP/IP Illustrated, Vol. 1: The Protocols, Addison-Wesley, Reading, MA, 1994.
618. W.R. Stevens, Unix Network Programming, Volume 1: Networking APIs-Sockets and XTI, 2nd edition, Prentice-Hall, Englewood Cliffs, NJ, 1997.
619. J. Stewart, BGP4: Interdomain Routing in the Internet, Addison-Wesley, 1999.
620. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, Proc. 2001 ACM SIGCOMM (San Diego, CA, Aug. 2001).
621. J. Stone, M. Greenwald, C. Partridge, J. Hughes, Performance of Checksums and CRC's Over Real Data, IEEE/ACM Transactions on Networking, Vol. 6, No. 5 (Oct. 1998), pp. 529–543.
622. J. Stone, C. Partridge, When Reality and the Checksum Disagree, Proc. 2000 ACM SIGCOMM (Stockholm, Sweden, Aug. 2000).
623. W. T. Strayer, B. Dempsey, A. Weaver, XTP: The Xpress Transfer Protocol, Addison-Wesley, Reading, MA, 1992.
624. A. Stubblefield, J. Ioannidis, A. Rubin, Using the Fluhrer, Mantin, and Shamir Attack to Break WEP, Proceedings of 2002 Network and Distributed Systems Security Symposium (2002), pp. 17–22.
625. M. Subramanian, Network Management: Principles and Practice, Addison-Wesley, Reading, MA, 2000.
626. L. Subramanian, S. Agarwal, J. Rexford, R. Katz, Characterizing the Internet Hierarchy from Multiple Vantage Points, Proc. 2002 IEEE INFOCOM.
627. K.Sundaresan, K. Papagiannaki, The Need for Cross-layer Information in Access Point Selection, Proc. 2006 ACM Internet Measurement Conference (Rio De Janeiro, Oct. 2006).
628. A.-J. Su, D. Choffnes, A. Kuzmanovic, and F. Bustamante, Drafting Behind Akamai Proc. 2006 ACM SIGCOMM.

629. K. Suh, D. R. Figueiredo, J. Kurose and D. Towsley, Characterizing and detecting relayed traffic: A case study using Skype, Proc. 2006 IEEE INFOCOM (Barcelona, Spain, Apr. 2006).

630. C. Sunshine, Y. Dalal, Connection Management in Transport Protocols, Computer Networks, North-Holland, Amsterdam, 1978.

631. M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, M. Ammar, Answering What-If Deployment and Configuration Questions with WISE, Proc. 2008 ACM SIGCOMM (Aug. 2008).

632. TechOnLine, Protected Wireless Networks, online webcast tutorial, techonline.com/community/tech_topic/internet/21752

633. R. Teixeira and J. Rexford, Managing Routing Disruptions in Internet Service Provider Networks, IEEE Communications Magazine (Mar. 2006).

634. D. Thaler and C. Ravishankar, Distributed Center-Location Algorithms, IEEE Journal on Selected Areas in Communications, Vol. 15, No. 3 (Apr. 1997), pp. 291–303.

635. Technical History of Network Protocols, Cyclades, www.cs.utexas.edu/users/chris/think/Cyclades/index.shtml

636. Y. Tian, R. Dey, Y. Liu, K. W. Ross, China's Internet: Topology Mapping and Geolocating, IEEE INFOCOM Mini-Conference 2012 (Orlando, FL, 2012).

637. F. Tobagi, Fast Packet Switch Architectures for Broadband Integrated Networks, Proc. 1990 IEEE INFOCOM, Vol. 78, No. 1 (Jan. 1990), pp. 133–167.

638. Tor: Anonymity Online, www.torproject.org

639. R. Torres, A. Finamore, J. R. Kim, M. M. Munafo, S. Rao, Dissecting Video Server Selection Strategies in the YouTube CDN, Proc. 2011 Int. Conf. on Distributed Computing Systems.

640. J. S. Turner Design of a Broadcast packet switching network, IEEE Transactions on Communications, Vol. 36, No. 6 (June 1988), pp. 734–743.

641. B. Turner, 2G, 3G, 4G Wireless Tutorial, blogs.nmscommunications.com/communications/2008/10/2g-3g-4g-wireless-tutorial.html

642. UPnP Forum homepage, www.upnp.org/

643. R. van der Berg, How the 'Net works: an introduction to peering and transit, arstechnica.com/guides/other/peering-and-transit.ars

644. G. Varghese, A. Lauck, Hashed and Hierarchical Timing Wheels: Efficient Data Structures for Implementing a Timer Facility, IEEE/ACM Transactions on Networking, Vol. 5, No. 6 (Dec. 1997), pp. 824–834.

645. S. Vasudevan, C. Diot, J. Kurose, D. Towsley, Facilitating Access Point Selection in IEEE 802.11 Wireless Networks, Proc. 2005 ACM Internet Measurement Conference, (San Francisco CA, Oct. 2005).

646. Verizon, Verizon FiOS Internet: FAQ, forums.verizon.com/t5/FiOS-Internet/bd-p/FiOS_Internet

647. Verizon, Global Latency and Packet Delivery SLA, www.verizonenterprise.com/terms/global_latency_sla.xml

648. D. C. Verma, Content Distribution Networks: An Engineering Approach, John Wiley, 2001.

649. C. Villamizar, C. Song. High performance tcp in ansnet, ACM SIGCOMM Computer Communications Review, Vol. 24, No. 5 (1994), pp. 45–60.

650. A. Viterbi, CDMA: Principles of Spread Spectrum Communication, Addison-Wesley, Reading, MA, 1995.
651. P. Vixie, What DNS Is Not, Communications of the ACM, Vol. 52, No. 12 (Dec. 2009), pp. 43–47.
652. The World Wide Web Consortium, A Little History of the World Wide Web (1995), www.w3.org/History.html
653. I. Wakeman, J. Crowcroft, Z. Wang, D. Sirovica, Layering Considered Harmful, IEEE Network (Jan. 1992), pp. 20–24.
654. M. Waldrop, Data Center in a Box, Scientific American (July 2007).
655. J. Walker, IEEE P802.11 Wireless LANs, Unsafe at Any Key Size; An Analysis of the WEP Encapsulation, Oct. 2000, www.drizzle.com/%7Eaboba/IEEE/0-362.zip
656. D. Wall, Mechanisms for Broadcast and Selective Broadcast, Ph.D. thesis, Stanford University, June 1980.
657. B. Wang, J. Kurose, P. Shenoy, D. Towsley, Multimedia Streaming via TCP: An Analytic Performance Study, Proc. 2004 ACM Multimedia Conference (New York, NY, Oct. 2004).
658. B. Wang, J. Kurose, P. Shenoy, D. Towsley, Multimedia Streaming via TCP: An Analytic Performance Study, ACM Transactions on Multimedia Computing Communications and Applications (TOMCCAP), Vol. 4, No. 2 (Apr. 2008), pp. 16:1–22.
659. G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. S. E. Ng, M. Kozuch, M. Ryan, c-Through: Part-time Optics in Data Centers, Proc. 2010 ACM SIGCOMM.
660. S. Weatherspoon, Overview of IEEE 802.11b Security, Intel Technology Journal (2nd Quarter 2000), download.intel.com/technology/itj/q22000/pdf/art_5.pdf
661. W. Wei, B. Wang, C. Zhang, J. Kurose, D. Towsley, Classification of Access Network Types: Ethernet, Wireless LAN, ADSL, Cable Modem or Dialup?, Proc. 2005 IEEE INFOCOM (Apr. 2005).
662. W. Wei, C. Zhang, H. Zang, J. Kurose, D. Towsley, Inference and Evaluation of Split-Connection Approaches in Cellular Data Networks, Proc. Active and Passive Measurement Workshop (Adelaide, Australia, Mar. 2006).
663. D. X. Wei, C. Jin, S. H. Low, S. Hegde, FAST TCP: Motivation, Architecture, Algorithms, Performance, IEEE/ACM Transactions on Networking (2007).
664. M. Weiser, The Computer for the Twenty-First Century, Scientific American (Sept. 1991): 94–10. www.ubiq.com/hypertext/weiser/SciAmDraft3.html
665. A. White, K. Snow, A. Matthews, F. Monrose, Hookt on fon-iks: Phonotactic Reconstruction of Encrypted VoIP Conversations, IEEE Symposium on Security and Privacy, Oakland, CA, 2011.
666. Wireless Geographic Logging Engine, www.wigle.net
667. R. Williams, A Painless Guide to CRC Error Detection Algorithms, www.ross.net/crc/crcpaper.html
668. C. Wilson, H. Ballani, T. Karagiannis, A. Rowstron, Better Never than Late: Meeting Deadlines in Datacenter Networks, Proc. 2011 ACM SIGCOMM.
669. WiMax Forum, www.wimaxforum.org
670. Wireshark homepage, www.wireshark.org
671. D. Wischik, N. McKeown, Part I: Buffer Sizes for Core Routers, ACM SIGCOMM Computer Communications Review, Vol. 35, No. 3 (July 2005).

672. T. Woo, R. Bindignavle, S. Su, S. Lam, SNP: an interface for secure network programming, Proc. 1994 Summer USENIX (Boston, MA, June 1994), pp. 45–58.

673. L. Wood, Lloyds Satellites Constellations, www.ee.surrey.ac.uk/Personal/L.Wood/constellations/iridium.html

674. J. Wu, Z. M. Mao, J. Rexford, J. Wang, Finding a Needle in a Haystack: Pinpointing Significant BGP Routing Changes in an IP Network, Proc. USENIX NSDI (2005).

675. Xanadu Project homepage, www.xanadu.com/

676. X. Xiao, A. Hannan, B. Bailey, L. Ni, Traffic Engineering with MPLS in the Internet, IEEE Network (Mar./Apr. 2000).

677. H. Xie, Y.R. Yang, A. Krishnamurthy, Y. Liu, A. Silberschatz, P4P: Provider Portal for Applications, Proc. 2008 ACM SIGCOMM (Seattle, WA, Aug. 2008).

678. M. Yannuzzi, X. Masip-Bruin, O. Bonaventure, Open Issues in Interdomain Routing: A Survey, IEEE Network Magazine (Nov./Dec. 2005).

679. R. Yavatkar, N. Bhagwat, Improving End-to-End Performance of TCP over Mobile Internetworks, Proc. Mobile 94 Workshop on Mobile Computing Systems and Applications (Dec. 1994).

680. YouTube 2009, Google container data center tour, 2009.

681. H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, SybilGuard: Defending Against Sybil Attacks via Social Networks, Proc. 2006 ACM SIGCOMM (Pisa, Italy, Sept. 2006).

682. E. Zegura, K. Calvert, M. Donahoo, A Quantitative Comparison of Graph-based Models for Internet Topology, IEEE/ACM Transactions on Networking, Vol. 5, No. 6, (Dec. 1997).

683. L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala, RSVP: A New Resource Reservation Protocol, IEEE Network Magazine, Vol. 7, No. 9 (Sept. 1993), pp. 8–18.

684. L. Zhang, A Retrospective View of NAT, The IETF Journal, Vol. 3, Issue 2 (Oct. 2007).

685. M. Zhang, W. John, C. Chen, Architecture and Download Behavior of Xunlei: A Measurement-Based Study, Proc. 2010 Int. Conf. on Educational Technology and Computers (ICETC).

686. X. Zhang, Y. Xu, Y. Liu, Z. Guo, Y. Wang, Profiling Skype Video Calls: Rate Control and Video Quality, IEEE INFOCOM (Mar. 2012).

687. B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, J. Kubiatowicz, Tapestry: A Resilient Global-scale Overlay for Service Deployment, IEEE Journal on Selected Areas in Communications, Vol. 22, No. 1 (Jan. 2004).

688. H. Zimmerman, OS1 Reference Model-The ISO Model of Architecture for Open Systems Interconnection, IEEE Transactions on Communications, Vol. 28, No. 4 (Apr. 1980), pp. 425–432.

689. P. Zimmermann, Why do you need PGP? www.pgpi.org/doc/whypgp/en/

690. M. Zink, K. Suh, Y. Gu, J. Kurose, Characteristics of YouTube Network Traffic at a Campus Network — Measurements, Models, and Implications, Computer Networks, Vol. 53, No. 4 (2009), pp. 501–514.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Б

В

К

Л

М

Н

О

П

Р

С

Т

Э

Производственно-практическое издание

МИРОВОЙ КОМПЬЮТЕРНЫЙ БЕСТСЕЛЛЕР

Джеймс Куроуз
Кит Росс

КОМПЬЮТЕРНЫЕ СЕТИ
НИСХОДЯЩИЙ ПОДХОД

Директор редакции *Е. Капьёв*
Ответственный редактор *В. Обручев*
Художественный редактор *Е. Мишина*

ООО «Издательство «Э»
123308, Москва, ул. Зорге, д. 1. Тел. 8 (495) 411-68-86.

Өндіруші: «Э» АҚБ Баспасы, 123308, Мәскеу, Ресей, Зорге көшесі, 1 үй.
Тел. 8 (495) 411-68-86.
Тауар белгісі: «Э»
Қазақстан Республикасында дистрибьютор және өнім бойынша арыз-талаптарды қабылдаушының
өкілі «РДЦ-Алматы» ЖШС, Алматы қ., Домбровский көш., 3«а», литер Б, офис 1.
Тел.: 8 (727) 251-59-89/90/91/92, факс: 8 (727) 251 58 12 вн. 107.
Өнімнің жарамдылық мерзімі шектелмеген.
Сертификация туралы ақпарат сайтта Өндіруші «Э»

Сведения о подтверждении соответствия издания согласно законодательству РФ
о техническом регулировании можно получить на сайте Издательства «Э»

Өндірген мемлекет: Ресей
Сертификация қарастырылмаған

Подписано в печать 13.04.2015.
Формат 70x100$^{1}/_{16}$. Печать офсетная. Усл. печ. л. 73,89.
Тираж экз. Заказ

ISBN 978-5-699-78090-7

Оптовая торговля книгами Издательства «Э»:
142700, Московская обл., Ленинский р-н, г. Видное,
Белокаменное ш., д. 1, многоканальный тел.: 411-50-74.

По вопросам приобретения книг Издательства «Э» зарубежными оптовыми покупателями обращаться в отдел зарубежных продаж
International Sales: International wholesale customers should contact Foreign Sales Department for their orders.

По вопросам заказа книг корпоративным клиентам, в том числе в специальном оформлении, *обращаться по тел.: +7 (495) 411-68-59, доб. 2115/2117/2118; 411-68-99, доб. 2762/1234.*

Оптовая торговля бумажно-беловыми и канцелярскими товарами для школы и офиса:
142702, Московская обл., Ленинский р-н, г. Видное-2,
Белокаменное ш., д. 1, а/я 5. Тел./факс: +7 (495) 745-28-87 (многоканальный).

Полный ассортимент книг издательства для оптовых покупателей:
В Санкт-Петербурге: ООО СЗКО, пр-т Обуховской Обороны, д. 84Е.
Тел.: (812) 365-46-03/04.
В Нижнем Новгороде: 603094, г. Нижний Новгород, ул. Карпинского, д. 29, бизнес-парк «Грин Плаза». Тел.: (831) 216-15-91 (92/93/94).
В Ростове-на-Дону: ООО «РДЦ-Ростов», пр. Стачки, 243А.
Тел.: (863) 220-19-34.
В Самаре: ООО «РДЦ-Самара», пр-т Кирова, д. 75/1, литера «Е».
Тел.: (846) 269-66-70.
В Екатеринбурге: ООО«РДЦ-Екатеринбург», ул. Прибалтийская, д. 24а.
Тел.: +7 (343) 272-72-01/02/03/04/05/06/07/08.
В Новосибирске: ООО «РДЦ-Новосибирск», Комбинатский пер., д. 3.
Тел.: +7 (383) 289-91-42.
В Киеве: ООО «Форс Украина», г. Киев,пр. Московский, 9 БЦ «Форум».
Тел.: +38-044-2909944.

Полный ассортимент продукции Издательства «Э» можно приобрести в магазинах «Новый книжный» и «Читай-город».
Телефон единой справочной: 8 (800) 444-8-444.
Звонок по России бесплатный.

В Санкт-Петербурге: в магазине «Парк Культуры и Чтения БУКВОЕД», Невский пр-т, д.46. Тел.: +7(812)601-0-601, www.bookvoed.ru/

Розничная продажа книг с доставкой по всему миру.
Тел.: +7 (495) 745-89-14.

Компьютерные сети

Нисходящий подход

Книга знакомит читателя с фундаментальными основами построения и функционирования компьютерных сетей на примере пятиуровневой архитектуры сети Интернет.

Описаны базовые компоненты компьютерной сети, ключевые подходы к передаче данных в телекоммуникационных сетях, принципы взаимодействия сетей друг с другом, подробно рассмотрены важнейшие службы и протоколы всех уровней сетевой архитектуры. Отдельная глава посвящена беспроводным и мобильным сетям и их особенностям. Большое внимание уделено одной из самых развивающихся сегодня областей – мультимедийным сетевым технологиям, в частности специфике передачи аудио- и видеоданных. Будут затронуты важные аспекты сетевой безопасности и разнообразные принципы, методы и приемы, обеспечивающие безопасный обмен информацией. В заключительной части книги рассмотрены инструменты и средства, которыми необходимо владеть, чтобы грамотно управлять механизмом под названием «компьютерная сеть».

Весь материал книги снабжен интересными примерами, кроме того, читателю доступны дополнительные материалы для выполнения упражнений, а также уникальные файлы-презентации для каждой главы, представляющие собой наглядные и красочные планы представленного в главах материала.

Книга будет особенно полезна всем, кто специализируется в области технологий компьютерных сетей, – от студентов до системных администраторов.

Вы узнаете:

- **как функционируют Интернет и локальные сети**
- **о коммутации пакетов и связанных с этим проблемах**
- **принципы работы прикладного, транспортного, сетевого и канального уровней OSI**
- **как работают сетевые протоколы, включая TCP, UDP и IP**
- **об особенностях беспроводных сетей, таких как Wi-Fi и LTE**
- **о мультимедийных сетевых технологиях: IP-телефонии, потоковом вещании и других**
- **чем отличаются версии протокола аутентификации ap**
- **об основах криптографии**
- **о возможностях обеспечения безопасности с помощью PGP, SSL, IPSec, WEP и прочих технологий**
- **об администрировании сети**

Джеймс Куроуз – заслуженный профессор кибернетики в Массачусетском университете. Он работал главным редактором в журналах «IEEE Transactions on Communications» и «IEEE/ACM Transactions on Networking». Доктор Куроуз является членом организаций IEEE и ACM. Сфера его научных интересов охватывает сетевые протоколы и архитектуру, измерение работы сетей, сенсорные сети, мультимедийные коммуникации, моделирование и оценку производительности. Джеймс Куроуз получил степень доктора философии в области кибернетики в Колумбийском университете.

Кит Росс возглавляет факультет кибернетики в университете штата Нью-Йорк. Сфера научных интересов доктора Росса охватывает проблемы безопасности и конфиденциальности, социальные сети, одноранговые сети, анализ Интернета, потоковое видео, сети распределения контента и стохастическое моделирование. Доктор Росс является членом института IEEE, лауреатом премии за лучшую научную работу Infocom Best Paper за 2009 год, а также обладателем аналогичных наград за 2011 и 2008 годы в области мультимедийных коммуникаций (вручается Телекоммуникационным обществом института IEEE).

ISBN 978-5-699-78090-7

Книга «Компьютерные сети. Нисходящий подход» прекрасно подходит для студентов, изучающих локальные вычислительные сети, и специалистов по телекоммуникациям. Принцип нисходящего изложения формирует перед читателем целостную картину – от общих принципов сетей и технологий до их технической и практической реализации, а различные примеры делают чтение интересным настолько, насколько это возможно для технической литературы.

Дмитрий Кондратьев, директор по бизнес-администрированию, издательство «ЭКСМО»

Дополнительные материалы можно скачать по ссылке:
http://eksmo.ru/upload/Networks_Primers.zip