рассылки внешне не отличается от обычного индивидуального адреса. Если отправитель не знает, что *birders* — это список рассылки, он вполне может подумать, что он посылает письмо лично некоему профессору по имени Gabriel O. Birders.

Чтение электронной почты

Обычно при запуске пользовательский агент просматривает содержимое почтового ящика пользователя на предмет наличия новой почты. Затем он может объявить пользователю число новых сообщений в почтовом ящике или отобразить по одной строке сведений о каждом письме, после чего перейти в режим ожидания команды пользователя.

В качестве примера работы пользовательского агента рассмотрим типичный сценарий. Запустив пользовательский агент, пользователь запрашивает краткую сводку о своей почте. На экране при этом появляется список писем (см. табл. 7.3). Каждая строка соответствует одному полученному письму. В данном примере в почтовом ящике содержится восемь сообщений.

Таблица 7.3. Пример отображения содержимого почтового ящика

#	Флаги	Размер	Отправитель	Тема
1	K	1030	asw	Изменение в системе MINIX
2	KA	6348	vovka	Не все Вовки так уж противны
3	KF	4519	Amy N. Wong	Запрос сведений
4		1236	bal	Биоинформатика
5		104110	kaashoek	Материалы по одноранговым сетям
6		1223	Frank	Re: Вы рассмотрите мой запрос на грант?
7		3110	guido	Наша статья принята
8		1204	dmr	Re: посещение моего студента

Каждая отображаемая строка содержит несколько полей, извлеченных из конверта или заголовка соответствующего сообщения. В простой системе электронной почты список отображаемых полей встроен в программу. В более сложных системах пользователь может выбрать отображаемые поля, а настройки пользователя будут храниться в специальном файле, называющемся **профилем пользователя**. В данном примере первое отображаемое поле — номер сообщения. Второе поле, *Flags* (флаги) может содержать флаг *K*, означающий, что сообщение не является новым, уже было прочитано и хранится в почтовом ящике; флаг *A*, означающий, что на данное сообщение уже был отправлен ответ; и/или флаг *F*, означающий, что сообщение было переадресовано кому-то еще. Возможно также использование и других флагов.

Третье поле одержит размер сообщения в байтах, а в четвертом поле указывается отправитель сообщения. Поскольку значение этого поля просто извлекается из заголовка сообщения, это поле может содержать имена, полные имена, инициалы, имена регистрации в системе, а также все, что отправитель захочет указать в качестве своего имени. Наконец, поле *Subject* (тема) содержит краткое изложение содержания сообщения. Пользователи, забывающие заполнять поле *Subject*, часто обнаруживают, что их письма читаются респондентами далеко не в первую очередь.

После того как программа отобразила заголовки, пользователь может выполнить одну из нескольких команд: чтение, удаление письма и т. д. Старые системы с текстовым интерфейсом обычно управлялись с помощью односимвольных команд, таких как T (вывести сообщение), A (создать ответ), D (удалить сообщение) и F (переслать). Более современные системы имеют графический интерфейс. Обычно пользователь выбирает сообщение с помощью мыши, затем щелкает на значке, соответствующем определенному действию (выводу сообщения, созданию ответа, удалению и переадресации).

Электронная почта сильно изменилась с тех пор, когда она представляла собой простую передачу файлов. Пользовательские агенты с развитым набором услуг позволяют управляться с огромными потоками почты. Для всех, кому приходится получать и отправлять тысячи писем в год, такие программы просто незаменимы.

Форматы сообщений

Перейдем теперь от рассмотрения пользовательского интерфейса к формату самих сообщений электронной почты. Сначала мы рассмотрим основной ASCII-формат электронного письма стандарта RFC 822. Затем мы познакомимся с мультимедийным расширением этого стандарта.

RFC 822

Сообщения состоят из примитивного конверта (описанного в RFC 821), нескольких полей заголовка, пустой строки и, наконец, тела сообщения. Каждое поле заголовка (логически) состоит из одной строки ASCII-текста, содержащей имя поля, двоеточие и (в большинстве случаев) значение поля. RFC 822 был создан несколько десятилетий назад, и в нем нет четкого разграничения конверта и заголовка. Хотя частично стандарт был пересмотрен в RFC 2822, целиком обновить его было невозможно, поскольку RFC 822 уже был очень широко распространен. Обычно пользовательский агент формирует сообщение и передает его агенту передачи сообщений, который с помощью одного из полей заголовка создает конверт нового вида, представляющий собой некую старомодную смесь сообщения и конверта.

Основные поля заголовка, связанные с транспортировкой сообщения, перечислены в табл. 7.4. Поле *To:* содержит DNS-адрес основного получателя. Возможно наличие и нескольких получателей. В поле *Cc:* указываются адреса дополнительных получателей. С точки зрения качества доставки, никакой разницы между основным и дополнительными получателями нет. Разница между ними чисто психологическая и может быть важна для людей, но совершенно не существует для почтовой системы. Термин *Cc:* (carbon copy — экземпляр, сделанный «под копирку») несколько устарел, так как при работе с компьютерами копировальная бумага вообще-то не используется, тем не менее, он прочно обосновался

в электронной почте. Поле *Bcc:* (Blind carbon copy — слепая копия) аналогично предыдущему, с той разницей, что в последнем случае строка этого поля не видна получателям (как основному, так и дополнительным). Это свойство позволяет рассылать одно письмо одновременно нескольким получателям так, что получатели не будут знать, что это письмо послано еще кому-либо кроме них.

Таблица 7.4. Поля заголовка стандарта RFC 822, связанные с транспортировкой сообщения

Поле	Значение
To:	Электронный адрес (адреса) основного получателя (получателей)
Cc:	Электронный адрес (адреса) дополнительного получателя (получателей)
Bcc:	Электронный адрес (адреса) слепой копии
From:	Автор (авторы) сообщения
Sender:	Электронный адрес отправителя
Received:	Строка, добавляемая каждым агентом передачи сообщений на протяжении маршрута
Return-Path:	Может быть использовано для идентификации обратного пути к отправителю

Следующие два поля, *From:* и *Sender:*, сообщают, соответственно, кто составил и отправил сообщение. Это могут быть разные люди. Например, написать письмо может руководитель предприятия, а отослать — его секретарша. В этом случае руководитель будет числиться в поле *From:*, а секретарша — в поле *Sender:*. Поле *From:* является обязательным, тогда как поле *Sender:* может быть опущено, если его содержимое не отличается от содержимого поля *From:*. Эти поля нужны на случай, если сообщение доставить невозможно и об этом следует проинформировать отправителя. Кроме того, по адресам, указанным в этих полях, может быть оправлен ответ.

Строка, содержащая поле *Received:*, добавляется каждым агентом передачи сообщений на пути следования сообщения. В это поле помещаются идентификатор агента, дата и время получения сообщения, а также другая информация, которая может быть использована для поиска неисправностей в системе маршрутизации.

Поле *Return-Path:* добавляется последним агентом передачи сообщений. Предполагалось, что это поле будет сообщать, как добраться до отправителя. Теоретически, эта информация может быть собрана из всех полей *Received:* заголовка (кроме имени почтового ящика отправителя), однако на практике оно редко заполняется подобным образом и обычно просто содержит адрес отправителя.

Помимо полей, показанных в табл. 7.4, сообщения стандарта RFC 822 могут также содержать широкий спектр полей заголовка, используемых пользовательским агентом или самим пользователем. Наиболее часто используемые поля заголовка приведены в табл. 7.5. Информации в таблице достаточно, чтобы понять назначение полей, поэтому мы не станем рассматривать их все подробно.

Таблица 7.5. Некоторые поля, используемые в заголовке сообщения стандарта RFC 822

Поле	Значение
Date:	Дата и время отправки сообщения
Reply-to:	Электронный адрес, на который следует присылать ответ
Message-Id:	Уникальный номер для последующей ссылки на это сообщение
In-Reply-To:	Идентификатор Message-Id сообщения, в ответ на которое посылается это сообщение
References:	Другие важные ссылки (идентификаторы Message-Id)
Keywords:	Ключевые слова, выбираемые пользователем
Subject:	Краткое изложение темы сообщения для отображения в одной строке

Поле *Reply-to:* иногда используется в случае, если ни составитель письма, ни его отправитель не хотят получать на него ответ. Например, управляющий отделом сбыта пишет письмо, информирующее клиента о новом продукте. Это письмо отправляется его секретарем, но в поле *Reply-to:* указан адрес менеджера отдела продаж, который может ответить на вопросы и принять заказы.

В документе RFC 822 открыто сказано, что пользователям разрешается изобретать собственные заголовки для своих нужд при условии, что эти заголовки начинаются со строки *X-*. Гарантируется, что в будущем никакие стандартные заголовки не будут начинаться с этих символов, чтобы избежать конфликтов между официальными и частными заголовками. Иногда умники-студенты включают поля вроде *X-Fruit-of-the-Day:* (сегодняшний плод) или *X-Disease-of-the-Week:* (недуг недели), использование которых вполне законно, хотя их смысл и не всегда понятен.

После заголовков идет тело самого сообщения. Пользователь может разместить в нем все, что ему угодно. Некоторые люди завершают свои послания сложными подписями, включающими рисунки, созданные из ASCII-символов, популярными и малоизвестными цитатами, политическими заявлениями и разнообразными объявлениями (например, «Корпорация АБВ не несет ответственности за высказанное выше мнение. Собственно, она даже не в силах постичь его»).

MIME — многоцелевые расширения электронной почты в сети Интернет

На заре существования сети ARPANET электронная почта состояла исключительно из текстовых сообщений, написанных на английском языке и представленных символами ASCII. Для подобного применения стандарта RFC 822 было вполне достаточно: он определял формат заголовков, но оставлял содержимое сообщения полностью на усмотрение пользователей. На сегодняшний день такой подход уже не удовлетворяет пользователей, привыкших к Интернету. Требуется обеспечить возможность оправления сообщений со следующими характеристиками.

1. Сообщения на языках с надстрочными знаками (например, на французском, немецком, испанском и т. д.).

2. Сообщения на языках, использующих алфавиты, отличные от латинского (например, на иврите или русском).
3. Сообщения на языках без алфавитов (например, китайском, японском, корейском).
4. Сообщения, вообще не являющиеся текстом (например, аудио или видео).

Решение было предложено в документе RFC 1341, а более новая редакция была опубликована в RFC 2045–2049. Это решение, получившее название **MIME** (Multipurpose Internet Mail Extensions, — многоцелевые расширения электронной почты в Интернете), широко применяется в настоящий момент. Далее приведено его описание. Дополнительную информацию о наборе стандартов MIME см. в RFC.

Основная идея стандартов MIME — продолжить использование формата RFC 822, но с добавлением структуры к телу сообщения и с определением правил кодировки не-ASCII-сообщений. Не отклоняясь от стандарта RFC 822, MIME-сообщения могут передаваться с помощью обычных почтовых программ и протоколов. Все, что нужно изменить, — это отправляющие и принимающие программы, которые пользователи могут создать для себя сами.

Стандартами MIME определяются пять новых заголовков сообщения, приведенных в табл. 7.6. Первый заголовок просто информирует пользовательского агента, получающего сообщение, что тот имеет дело с сообщением MIME, а также сообщает ему номер версии MIME, используемой в этом сообщении. Если сообщение не содержит такого заголовка, то оно считается написанным на английском языке и обрабатывается соответствующим образом.

Таблица 7.6. Заголовки RFC 822, добавленные MIME

Заголовок	Описание
MIME-Version:	Указывает версию стандартов MIME
Content-Description:	Описание содержимого. Строка обычного текста, информирующая о содержимом сообщения
Content-Id:	Уникальный идентификатор
Content-Transfer-Encoding:	Указывает способ кодировки тела сообщения для его передачи
Content-Type:	Тип и формат содержимого сообщения

Заголовок *Content-Description* представляет собой ASCII-строку, информирующую о том, что находится в сообщении. Этот заголовок позволяет пользователю принять решение о том, нужно ли ему декодировать и читать сообщение. Если в строке сказано: «Фотография тушканчика Барбары», а получивший это сообщение не является любителем тушканчиков, то, вероятнее всего, он сразу удалит это сообщение, а не станет перекодировать его в цветную фотографию высокого разрешения.

Заголовок *Content-Id* содержит идентификатор содержимого сообщения. В нем используется тот же формат, что и в стандартном заголовке *Message-Id:*.

Заголовок *Content-Transfer-Encoding* сообщает о способе упаковки тела сообщения для его передачи по сети, которая может возражать против символов, отличных от букв, цифр и знаков препинания. Разработано пять схем (имеется возможность добавления новых схем). Простейшая из них заключается в передаче просто ASCII-текста. Символы ASCII используют 7 разрядов и могут передаваться напрямую протоколом электронной почты при условии, что строка не превышает 1000 символов.

Следующая по простоте схема аналогична предыдущей, но использует 8-разрядные символы, то есть все значения байта от 0 до 255 включительно. Такая схема кодировки нарушает (исходный) протокол электронной почты Интернета, но используется в некоторых частях Интернета, в которых реализовано некоторое расширение исходного протокола. Хотя объявление о способе кодирования не делает его использование автоматически законным, открытое объявление может, по крайней мере, в случае чего объяснить неправильную работу почтовых агентов. Сообщения, использующие 8-разрядную кодировку, также должны соблюдать правило о максимальной длине строки.

Еще хуже обстоит дело с сообщениями в двоичной кодировке. К ним относятся произвольные двоичные файлы, которые не только используют все 8 разрядов в байте, но еще и не соблюдают ограничение на 1000 символов в строке. К этой категории относятся исполняемые программные файлы. Не дается никакой гарантии, что эти двоичные сообщения будут доставлены корректно, но, тем не менее, очень многие пользователи все равно пересылают их друг другу.

Корректный способ кодирования двоичных сообщений состоит в использовании кодировки **base64** (64-символьная кодировка), иногда называемой **ASCII armor** (ASCII-оплетка). При использовании данного метода группы по 24 бита разбиваются на четыре 6-разрядные единицы, каждая из которых посылается в виде обычного разрешенного ASCII-символа. В этой кодировке 6-разрядный символ 0 кодируется ASCII-символом «A», 1 — ASCII-символом «B» и т. д. Затем следуют 26 строчных букв — это уже 10 разрядов, и наконец, + и / для кодирования 62 и 63 соответственно. Последовательности == и = говорят о том, что последняя группа содержит только 8 или 16 бит соответственно. Символы перевода строки и возврата каретки игнорируются, поэтому их можно вставлять в любом месте для того, чтобы строки выглядели не слишком длинными. Таким способом можно передать любой двоичный код.

Для сообщений, почти полностью состоящих из символов ASCII, но с небольшими включениями не-ASCII-символов, подобный метод несколько неэффективен. Вместо него лучше применять кодировку **quoted-printable** (цитируемое печатное кодирование). Это просто 7-битный ASCII, в котором символы, соответствующие значениям ASCII-кода свыше 127, кодируются знаком равенства, за которым следуют две шестнадцатеричных цифры — ASCII-код символа.

Итак, двоичные данные следует посылать в кодировке Base64 или quoted-printable. Когда имеются веские причины не использовать эти методы, можно указать в заголовке *Content-Transfer-Encoding:* свою кодировку.

Последний заголовок в табл. 7.6 представляет наибольший интерес. Он указывает тип тела сообщения. В документе RFC 2045 определены семь типов содержимого сообщений, каждый из которых распадается на несколько подтипов. Подтип отделяется от типа косой чертой, например,

```
Content-Type: video/mpeg
```

Подтип должен быть явно указан в заголовке; подтипов по умолчанию нет. Начальный список типов и подтипов, определенный в документе RFC 2045, приведен в табл. 7.7. С тех пор к ним было добавлено много новых типов и подтипов. Этот список пополняется всякий раз при возникновении соответствующей необходимости.

Таблица 7.7. Типы стандарта MIME и подтипы, определенные в RFC 2045

Тип	Подтип	Описание
Text	Plain	Неформатированный текст
	Enriched	Текст с включением простых команд форматирования
Image	Gif	Неподвижное изображение в формате GIF
	Jpeg	Неподвижное изображение в формате JPEG
Audio	Basic	Слышимый звук
Video	Mpeg	Видеофильм в формате MPEG
Application	Octet-stream	Неинтерпретируемая последовательность байтов
	Postscript	Документ для печати в формате PostScript
Message	Rfc822	Сообщение MIME RFC 822
	Partial	Сообщение разбито на части для передачи
	External-body	Само сообщение должно быть получено по сети
Multipart	Mixed	Независимые части в указанном порядке
	Alternative	То же сообщение в другом формате
	Parallel	Части сообщения следует просматривать одновременно
	Digest	Каждая часть является законченным сообщением стандарта RFC 822

Рассмотрим перечисленные в таблице типы сообщений. Тип *text* означает обычный текст. Комбинация *text/plain* служит для обозначения обычного текстового сообщения, которое может быть отображено на экране компьютера сразу после получения. Для этого не требуется дополнительной обработки или перекодировки. Это значение поля заголовка позволяет передавать обычные сообщения в MIME с добавлением небольшого количества дополнительных заголовков.

Подтип *text/enriched* позволяет включать в текст простой язык разметки документа. Этот язык обеспечивает системно-независимый способ выделять участки текста жирным или наклонным стилями, использовать шрифты самых разных размеров и цветов, отступы, выравнивание, верхние и нижние индексы и формировать простой макет страницы. В основе этого языка разметки лежит язык SGML (Standard Generalized Markup Language — стандартный обобщенный язык разметки), на базе которого был также создан язык HTML (HyperText Markup Language), применяемый в WWW. Например, сообщение

```
<bold> Момент </bold> настал, - сказал <italic> моряк </italic> ...
```

будет отображаться как

Момент настал, — сказал *моряк* ...

Интерпретация зависит от принимающей сообщение системы. Если стили «жирный» и «курсив» доступны, они будут применены. Если нет, для выделения можно использовать подчеркивание, мигание, инверсную печать, выделение другим цветом и т. д. Разные системы могут применять и применяют свои стили.

Когда веб-технологии стали популярны, был добавлен (в RFC 2854) новый тип *text/html*, который позволил пересылать веб-страницы в теле письма RFC 822. В RFC 3023 определен подтип для расширяемого языка разметки страниц, *text/xml*. Далее в этой главе мы рассмотрим HTML и XML.

Следующим типом MIME является *image*. Он используется для передачи неподвижных изображений. На сегодняшний день существует множество различных форматов хранения и передачи изображений, как с использованием сжатия, так и без него. Два формата — GIF и JPEG — встроены практически во все браузеры, однако существует еще множество других, которые, надо полагать, вскоре дополнят этот список.

Типы *audio* и *video* предназначены, соответственно, для передачи звука и двигающегося изображения. Обратите внимание на то, что подтип *video* включает только визуальную информацию, а не звуковую дорожку. Если необходимо передать по электронной почте видеофильм со звуком, то, возможно, придется посылать видеоряд и звуковую дорожку отдельно друг от друга. Это зависит от системы кодирования. Первым видеоформатом, который был определен стандартом MIME, стал формат со скромным названием MPEG (Motion Pictures Experts Group — экспертная группа по вопросам движущегося изображения).

Тип *application* (приложение) предназначен для всех форматов, требующих внешней обработки, не обеспечиваемой ни одним из других типов. Тип *octet-stream* (байтовый поток) представляет собой просто последовательность никак не обрабатываемых байтов. Получив такой поток, пользовательский агент должен, вероятно, предложить пользователю сохранить его в виде файла и запросить для этого файла имя. Последующая обработка файла целиком зависит от пользователя.

Подтип *postscript* означает язык PostScript, созданный компанией Adobe Systems и широко используемый для описания страниц, предназначенных для печати. Многие принтеры имеют встроенные интерпретаторы языка PostScript. Хотя пользовательский агент может для отображения полученных PostScript-файлов просто вызвать внешний интерпретатор языка PostScript, подобные действия, вообще говоря, небезопасны. PostScript является полноценным языком программирования. При достаточном количестве свободного времени и некоторой склонности к самоистязанию на языке PostScript можно написать компилятор языка C или систему управления базами данных. Отображение документа на языке PostScript осуществляется программой на языке PostScript, содержащейся в этом сообщении. Помимо отображения текста, эта программа способна читать, изменять или удалять файлы пользователя, а также может обладать еще целым рядом неприятных побочных эффектов.

Тип *message* позволяет помещать одно сообщение в другое. Это может быть полезно для переадресации письма. Если внутри одного сообщения заключено полное сообщение стандарта RFC 822, следует использовать подтип *rfc822*.

Подтип *partial* позволяет разбивать сообщение на отдельные части и передавать их отдельно (например, в случае, когда инкапсулированное сообщение является слишком длинным). Параметры обеспечивают восстановление сообщения получателем из отдельных фрагментов в правильном порядке.

Подтип *external-body* (внешнее тело) может применяться для очень длинных сообщений (таких как видеофильмы). Вместо того чтобы помещать MPEG-файл в тело письма, в нем сообщается FTP-адрес, по которому пользовательский агент может получить этот файл тогда, когда нужно. Это особенно удобно в случае рассылки по списку рекламных роликов. При этом пользователям, не желающим просматривать ролик, не придется скачивать его по сети в свой почтовый ящик. Даже страшно подумать, что было бы, если бы по электронной почте стали рассылать спам в виде рекламных видеороликов.

Наконец, тип *multipart* позволяет составлять сообщение из нескольких частей, при этом начало и конец каждой части отчетливо указываются. Подтип *mixed* позволяет создавать сообщение из частей различных форматов. В случае применения подтипа *alternative*, напротив, каждая часть должна содержать одно и то же сообщение, но в другом виде или другой кодировке. Например, сообщение может быть послано в виде простого ASCII-текста, а также в форматах RTF и PostScript. Грамотно созданный пользовательский агент, получив такое сообщение, сначала попытается отобразить его в формате PostScript. Если это по какой-либо причине невозможно, тогда производится попытка отобразить часть в формате RTF. Если и это невозможно, отображается ASCII-текст. Части следует располагать в порядке увеличения сложности, чтобы даже старые (до MIME) пользовательские агенты смогли отобразить сообщение хотя бы в виде простого ASCII-текста.

Подтип *alternative* также может использоваться для сообщений, посылаемых одновременно на разных языках. В этом контексте знаменитый розеттский камень, найденный в Египте, может считаться одним из ранних вариантов сообщений типа *multipart/alternative*.

Мультимедийный пример приведен в листинге 7.2. В данном случае поздравление с днем рождения посылается одновременно в виде текста и в виде песни. Если у получателя есть возможность воспроизвести звуковой файл *birthday.snd*, пользовательский агент скачает этот файл с указанного адреса и воспроизведет его. В противном случае на экране получателя в полной тишине отобразится текст сообщения. Части письма разделены двойными дефисами, за которыми следует (определяемая пользователем) строка, указанная как значение параметра *boundary* (граница).

Листинг 7.2. Сообщение, состоящее из RTF-текста и альтернативы в виде звукового файла

```
From: elinor@abc.com
To: carolyn@xyz.com
MIME-Version: 1.0
Message-Id: <0704760941.AA00747@abc.com>
Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
Subject: Земля обошла вокруг Солнца целое число раз
Это преамбула. Пользовательский агент игнорирует ее. Ку-ку.
```



```
--qwertyuiopasdfghjklzxcvbnm
Content-Type: text/enriched
Happy birthday to you
Happy birthday to you
Happy birthday dear <bold> Carolyn </bold>
Happy birthday to you
--qwertyuiopasdfghjklzxcvbnm
Content-Type: message/external-body;
access-type="anon-ftp";
site="bicycle.abc.com";
directory="pub";
name="birthday.snd"
content-type: audio/basic
content-transfer-encoding: base64
--qwertyuiopasdfghjklzxcvbnm--
```

Обратите внимание: заголовок *Content-Type* трижды встречается в данном сообщении. На верхнем уровне он указывает, что сообщение состоит из нескольких частей. Для каждой части он сообщает ее тип и подтип. Наконец, в теле второй части сообщения он указывает пользовательскому агенту тип внешнего файла. Чтобы подчеркнуть это различие, мы использовали в последнем случае строчные символы, хотя для всех заголовков регистр символа не имеет значения. Для внешнего тела в формате, отличном от 7-разрядного ASCII, также требуется заголовок *content-transfer-encoding*.

Для сообщений, состоящих из отдельных частей, существует еще два подтипа. Подтип *parallel* используется в том случае, когда требуется одновременный «просмотр» всех частей сообщения. Например, часто бывает так, что в фильмах звуковой канал отделен от изображения, однако эти два канала удобнее воспроизводить одновременно, а не последовательно.

Наконец, подтип *digest* используется, когда много сообщений упаковываются в одно сообщение. Например, какая-нибудь дискуссионная группа в Интернете может собирать сообщения от подписчиков, а затем высылать их в виде единого сообщения типа *multipart/digest*.

Пересылка писем

Система пересылки писем занимается доставкой электронных сообщений от отправителя получателю. Для этого проще всего установить транспортное соединение от машины-источника до машины-приемника, а затем просто передать по нему сообщение. Вначале мы познакомимся с тем, как это осуществляется в действительности, после чего рассмотрим несколько ситуаций, в которых подобный подход не работает, и обсудим способы разрешения возникающих в связи с этим проблем.

SMTP — простой протокол электронной почты

В Интернете для доставки электронной почты машина-источник устанавливает TCP-соединение с портом 25 машины-приемника. Этот порт прослушивается почтовым демоном, и их общение происходит с помощью протокола **SMTP** (Simple

Mail Transfer Protocol — простой протокол электронной почты). Этот демон принимает входящие соединения и копирует сообщения из них в соответствующие почтовые ящики. Если письмо невозможно доставить, отправителю возвращается сообщение об ошибке, содержащее первую часть этого письма.

Протокол SMTP представляет собой простой ASCII-протокол. Установив TCP-соединение с портом 25, передающая машина, выступающая в роли клиента, ждет запроса принимающей машины, работающей в режиме сервера. Сервер начинает диалог с того, что посылает текстовую строку, содержащую его идентификатор и сообщающую о его готовности (или неготовности) к приему почты. Если сервер не готов, клиент разрывает соединение и повторяет попытку позднее.

Если сервер готов принимать почту, клиент объявляет, от кого поступила почта и кому она предназначается. Если получатель почты существует, сервер дает клиенту добро на пересылку сообщения. Затем клиент посылает сообщение, а сервер подтверждает его получение. Контрольные суммы не проверяются, так как транспортный протокол TCP обеспечивает надежный байтовый поток. Если у отправителя есть еще почта, она также отправляется. После передачи всей почты в обоих направлениях соединение разрывается. Пример диалога клиента и сервера при передаче сообщения из листинга 7.2 показан в листинге 7.3. Строки, посланные клиентом, помечены буквой *C:*, а посланные сервером — *S:*.

Листинг 7.3. Передача сообщения от elinor@abc.com для carolyn@xyz.com

```
S: 220 xyz.com служба SMTP готова
C: HELO abc.com
S: 250 xyz.com приветствует abc.com
C: MAIL FROM: <elinor@abc.com>
S: 250 подтверждаю отправителя
C: RCPT TO: <carolyn@xyz.com>
S: 250 подтверждаю получателя
C: DATA
S: 354 Отправляйте письмо; конец письма обозначается строкой, состоящей из символа "."
C: From: elinor@abc.com
C: To: carolyn@xyz.com
C: MIME-Version: 1.0
C: Message-Id: <0704760941.AA00747@abc.com>
C: Content-Type: multipart/alternative; boundary=qwertyuiopasdfghjklzxcvbnm
C: Subject: Земля обошла вокруг Солнца целое число раз
C:
C: Это преамбула. Пользовательский агент игнорирует ее. Ку-ку.
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: text/richtext
C:
C: Happy birthday to you
C: Happy birthday to you
C: Happy birthday dear <bold> Carolyn </bold>
C: Happy birthday to you
C:
C: --qwertyuiopasdfghjklzxcvbnm
C: Content-Type: message/external-body;
```



```
C: access-type="anon-ftp";
C: site="bicycle.abc.com";
C: directory="pub";
C: name="birthday.snd"
C:
C: content-type: audio/basic
C: content-transfer-encoding: base64
C: --qwertyuiopasdfghjklzxcvbnm
C: .
S: 250 сообщение принято
C: QUIT
S: 221 xyz.com разрываю соединение
```

Несколько комментариев к листингу 7.3. Сначала клиент, естественно, посылает приветствие серверу. Таким образом, первая команда клиента выглядит как *HELO*, что представляет собой наиболее удачный из двух вариантов сокращения слова *HELLO* до четырех символов. Зачем все эти команды было нужно сокращать до четырех букв, сейчас уже никто не помнит.

В нашем примере сообщение должно быть послано только одному получателю, поэтому используется только одна команда *RCPT* (сокращение от слова recipient — получатель). Использование этой команды несколько раз позволяет посылать одно сообщение нескольким получателям. Каждое из них подтверждается или отвергается индивидуально. Несмотря на то, что попытки пересылки сообщения некоторым получателям оказываются неудачными (например, из-за отсутствия адресатов), это сообщение все равно может быть доставлено остальным адресатам, числящимся в списке рассылки.

Наконец, хотя синтаксис четырехсимвольных команд строго определен, синтаксис ответов не столь строг. Правила определяют только числовой код в начале строки. Все, что следует за этим кодом, может считаться комментарием и зависит от конкретной реализации протокола.

Чтобы лучше понять, как работают SMTP и другие рассмотренные в этой главе протоколы, попробуйте сами поработать с ними. В любом случае, для начала найдите машину, подключенную к Интернету. В системе UNIX наберите в командной строке:

```
telnet mail.isp.com 25
```

подставив вместо *mail.isp.com* DNS-имя почтового сервера провайдера. В системе Windows щелкните на кнопке Пуск, затем на кнопке Выполнить и наберите команду в диалоговом окне. В результате выполнения этой команды будет установлено telnet-соединение (то есть соединение TCP) с портом 25 данной машины. Как было показано в табл. 6.3, порт 25 является SMTP-портом. В ответ на введенную команду вы получите что-то вроде этого:

```
Trying 192.30.200.66...
Connected to mail.isp.com
Escape character is '^]'.
220 mail.isp.com Smail #74 ready at Thu, 25 Sept 2002 13:26 +0200
```

Первые три строки посылаются telnet и поясняют для вас происходящее. Последняя строка посылается сервером SMTP удаленной машины и сообщает о го-

товности к общению с вашей машиной и приему почты. Чтобы узнать о доступных командах, наберите

```
HELP
```

Начиная с этого момента, возможен обмен последовательностями команд, показанными в листинге 7.3. Начинаться общение должно с команды клиента HELO.

Стоит отметить, что использование строк ASCII-текста в качестве команд не случайно. Большинство протоколов Интернета работают именно таким образом. Применение ASCII-текста упрощает тестирование и отладку протоколов. Тестирование можно производить, набирая команды вручную, как было показано ранее. Легко читаются выведенные в ответ сообщения.

Несмотря на то, что протокол SMTP определен довольно четко, все же могут возникать определенные проблемы. Одна из них связана с длиной сообщений. Некоторые старые реализации не поддерживали сообщения длиннее 64 Кбайт. Еще одна проблема связана с тайм-аутами. Если таймеры клиента и сервера настроены на разное время ожидания, один из них может внезапно разорвать соединение, в то время как противоположная сторона будет продолжать передачу. Наконец, иногда могут возникать бесконечные «почтовые штормы». Например, если хост 1 хранит список рассылки *A*, а хост 2 — список рассылки *B* и они содержат записи друг о друге, то письмо, посланное по одному из списков, будет создавать бесконечный объем трафика, пока кто-нибудь не заметит это.

Для решения некоторых из этих проблем был разработан расширенный протокол SMTP, **ESMTP**. Он описан в RFC 2821. Клиенты, желающие использовать его, должны начинать сессию связи с посылки приветствия EHLO вместо HELO. Если команда не принимается сервером, значит, сервер поддерживает только обычный протокол SMTP и клиенту следует работать в обычном режиме. Если же EHLO принято, значит, установлена сессия ESMTP и возможна работа с новыми параметрами и командами.

Доставка сообщений

До сих пор мы предполагали, что все пользователи работают на машинах, способных посылать и получать электронную почту. Как мы уже знаем, электронная почта доставляется, когда отправитель устанавливает TCP-соединение с получателем и посылает по нему сообщения. Такая модель прекрасно работала тогда, когда все хосты сети ARPANET (а позднее — Интернет) были, по сути, постоянно на линии и могли принимать TCP-соединения.

Однако с появлением пользователей, связывающихся с провайдерами с помощью модема, такой подход перестал оправдывать себя. Проблема вот в чем: что будет, если Элинор захочет отправить письмо Кэролайн, а та в данный момент не работает в Интернете? Получается, что Элинор не сможет установить TCP-соединение с Кэролайн, следовательно, невозможно будет запустить протокол SMTP, и Кэролайн так и не получит поздравление с днем рождения.

Одно из решений заключается в создании агента передачи сообщений на машине провайдера, который бы принимал и хранил почту для своих пользователей в их почтовых ящиках. Поскольку такой агент может быть на линии постоянно, электронная почта может отправляться ему круглосуточно.

POP3

К сожалению, такое решение создает новую проблему: как пользователю забрать свою почту у агента передачи сообщений провайдера? Ответ таков: следует создать специальный протокол, который позволил бы пользовательскому агенту (на машине клиента) соединиться с агентом передачи сообщений провайдера (на машине провайдера) и скопировать хранящуюся для него почту. Одним из таких протоколов является **POP3** (Post Office Protocol v. 3 — почтовый протокол, 3-я версия), определенный в документе RFC 1939.

Ситуация, при которой доставка осуществляется в условиях постоянного соединения с Интернетом отправителя и получателя, показана на рис. 7.5, *а*. Иллюстрация ситуации, в которой отправитель находится в текущий момент на линии, а приемник — нет, приведена на рис. 7.5, *б*.

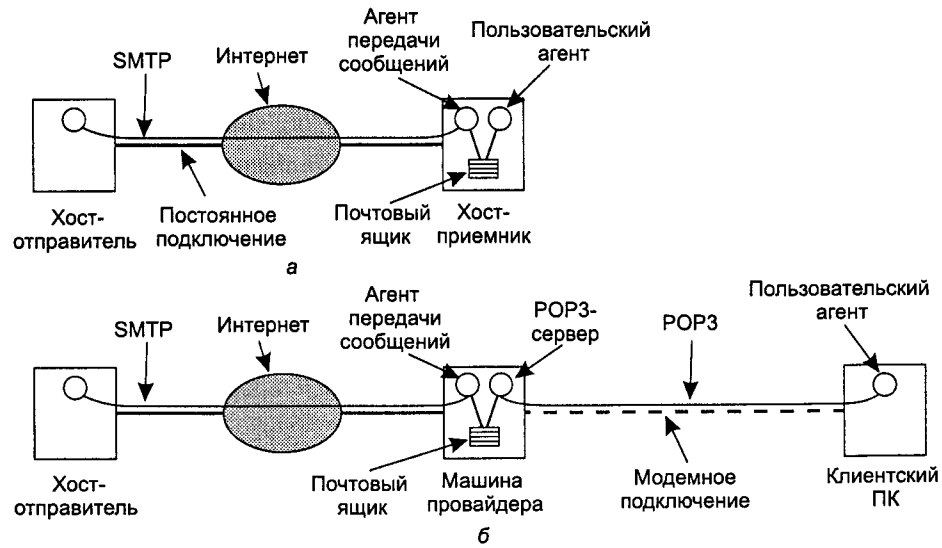


Рис. 7.5. Отправка и прием почты, когда приемник постоянно находится в подключенном состоянии и пользовательский агент работает на одной машине с агентом передачи сообщений (*а*); прием почты при модемном соединении получателя с провайдером (*б*)

Протокол POP3 начинает свою работу, когда пользователь запускает почтовый редактор. Последний дозванивается до провайдера (если только машина уже не находится в подключенном состоянии) и устанавливает TCP-соединение с агентом передачи сообщений с использованием порта 110. После установки соединения протокол POP3 проходит три последовательных состояния.

1. Авторизация.
2. Транзакции.
3. Обновление.

Авторизация связана с процессом входа пользователя в систему. В состоянии транзакций пользователь забирает свою почту и может пометить ее для удаления из почтового ящика. В состоянии обновления происходит удаление помеченной корреспонденции.

Можно посмотреть, как все это происходит, набрав команду вида

```
telnet mail.isp.com 110,
```

где *mail.isp.com* следует заменить на DNS-имя почтового сервера провайдера. Telnet устанавливает TCP-соединение с портом 110, прослушиваемым POP3-сервером. После установки TCP-соединения сервер посылает ASCII-сообщение, объявляя о своем присутствии. Обычно оно начинается с +OK, затем следует комментарий. Возможный сценарий после установки TCP-соединения показан в листинге 7.4. Как и раньше, строки, начинающиеся с C:, говорят о том, что данная команда исходит от клиента (пользователя), а начинающиеся с S: — что это сообщения сервера (агента передачи сообщений на машине провайдера).

Листинг 7.4. Получение трех сообщений по протоколу POP3

```
S: +OK POP3-сервер готов
C: USER carolyn
S: +OK
C: PASS vegetables
S: OK вход в систему произведен
C: LIST
S: 1 2505
S: 2 14302
S: 3 8122
S: .
C: RETR 1
S: (отправляет сообщение 1)
C: DELE 1
C: RETR 2
S: (отправляет сообщение 2)
C: DELE 2
C: RETR 3
S: (отправляет сообщение 3)
C: DELE 3
C: QUIT
S: +OK Конец соединения с POP3-сервером
```

В состоянии авторизации клиент должен сообщить имя пользователя и пароль. После успешного входа в систему клиент может послать команду LIST для запроса списка писем, хранящихся в почтовом ящике. Каждая строка списка соответствует одному письму, в ней указываются его номер и размер. Точка является признаком конца списка.

После этого пользователь может запросить сообщения командой RETR и пометить их для удаления командой DELE. После получения (и, возможно, установки меток удаления) всех писем пользователь посылает команду QUIT для завершения состояния транзакций и входа в состояние обновления. После удаления сервером всех сообщений он посылает ответ и разрывает TCP-соединение.

Несмотря на то, что протокол POP3 действительно поддерживает возможность получения одного или нескольких писем и оставления их на сервере, большинство программ обработки электронной почты просто скачивают все письма и опустошают почтовый ящик на сервере. Такие действия означают, что реально хранится только одна копия писем — на жестком диске пользователя. Если с ним что-то случается, корреспонденция пропадает безвозвратно.

Теперь подведем небольшие итоги того, как происходит работа с электронной почтой клиентов провайдера. Элинор создает сообщение для Кэролайн с помощью редактора электронной почты (то есть пользовательского агента) и щелкает на значке, чтобы отослать его. Программа передает письмо агенту передачи сообщений на хосте Элинор. Агент передачи сообщений видит, что письмо адресовано carolyn@xyz.com, и использует DNS для поиска записи *MX* для *xyz.com* (где *xyz.com* — провайдер Кэролайн). В ответ на запрос возвращается DNS-имя почтового сервера *xyz.com*. Агент передачи сообщений после этого снова обращается к DNS (например, используя gethostbyname): на этот раз ему нужно найти IP-адрес этой машины. Затем с помощью порта 25 найденной машины устанавливается TCP-соединение с SMTP-сервером. Передавая команды SMTP, аналогичные показанным в листинге 7.3, агент пересылает сообщение в почтовый ящик для Кэролайн и разрывает TCP-соединение.

Через некоторое время Кэролайн загружает свой компьютер, соединяется с провайдером и запускает программу электронной почты. Та устанавливает TCP-соединение через порт 110 POP3-сервера, работающего на машине провайдера. Имя DNS или IP-адрес этой машины обычно указывается при установке программы электронной почты либо его получают у провайдера. После установки TCP-соединения почтовая программа Кэролайн запускает протокол POP3 для копирования содержимого почтового ящика на локальный жесткий диск. При этом происходит обмен командами, аналогичными показанным в листинге 7.4. По окончании передачи электронной почты TCP-соединение разрывается. На самом деле в тот же момент можно разорвать и соединение с провайдером, поскольку вся почта уже находится на жестком диске у Кэролайн. Конечно, чтобы отправить ответ на письма, Кэролайн придется снова соединяться с провайдером, поэтому не всегда пользователи разрывают соединение сразу после загрузки почты.

IMAP

Пользователю, имеющему одну учетную запись у одного провайдера и всегда соединяющемуся с провайдером с одной и той же машины, вполне достаточно протокола POP3. Этот протокол и используется повсеместно благодаря его простоте и надежности. Однако в компьютерной индустрии есть такое незыблемое правило: если имеется нечто, что работает безупречно, всегда найдется некто, который захочет снабдить это нечто дополнительными возможностями (и тем самым снабдить его ошибками). Так произошло и с электронной почтой. У многих пользователей есть одна учетная запись в учебном заведении или на работе, но они хотят иметь доступ к ней и из дома, и с работы (учебы), и во время поездок (с портативного компьютера), и из интернет-кафе во время так называемого

отпуска. Хотя POP3 и предоставляет возможность разрешения такой ситуации (так как с его помощью все могут получить всю хранящуюся почту), но проблема в том, что корреспонденция пользователя очень быстро распространится более или менее случайным образом по всем машинам, с которых он получает доступ в Интернет, и некоторые из этих машин могут даже не принадлежать этому пользователю.

Это неудобство привело к созданию альтернативного протокола доставки сообщений, **IMAP** (Interactive Mail Access Protocol — протокол интерактивного доступа к электронной почте), определенного в RFC 2060. В отличие от протокола POP3, который подразумевает, что пользователь будет очищать почтовый ящик после каждого контакта с провайдером и будет работать с почтой в отключенном режиме, протокол IMAP предполагает, что вся почта будет оставаться в почтовых ящиках на сервере неограниченно долго. IMAP обладает широким набором механизмов для чтения сообщений или даже частей сообщений. Такое свойство полезно при использовании медленных модемов, поскольку можно прочесть только текстовую часть письма, к которому приложены большие видео- и аудиофрагменты. Поскольку основное предположение состоит в том, что пользователь не будет копировать на свой компьютер письма, в IMAP входят также инструменты для создания, удаления и других видов управления почтовыми ящиками, размещающимися на сервере. Таким образом, пользователь может завести собственный почтовый ящик для каждого лица, с которым ведется переписка, и переносить сообщения из почтового ящика для всех входящих писем в эти персональные ящики.

Протокол IMAP обладает разнообразными возможностями, например, способностью упорядочивать почту не по порядку ее поступления, как показано в табл. 7.3, а по атрибутам писем (например, «сначала дайте мне письмо от Бобби»). В отличие от POP3, IMAP может заниматься как доставкой исходящей почты от пользователя в направлении места назначения, так и доставлять входящую почту пользователя.

В целом стиль протокола IMAP подобен POP3, пример работы которого показан в листинге 7.4. Различаются они количеством команд — в IMAP их десятки. Сервер IMAP прослушивает порт 143. Сравнение протоколов POP3 и IMAP приведено в табл. 7.8. Следует заметить, что не все провайдеры и не все программы работы с электронной почтой поддерживают оба протокола. Поэтому, выбирая программу и провайдера, следует выяснить, могут ли они работать хоть с одним из этих протоколов, и если да, то с какими именно протоколами.

Таблица 7.8. Сравнение протоколов POP3 и IMAP

Свойство	POP3	IMAP
Где определен	RFC 1939	RFC 2060
Используемый порт TCP	110	143
Место хранения почты	ПК пользователя	Сервер
Способ чтения почты	В автономном режиме	В подключенном режиме
Требуемое время нахождения на линии	Небольшое	Большое

Свойство	POP3	IMAP
Использование ресурсов сервера	Минимальное	Значительное
Поддержка нескольких почтовых ящиков	Отсутствует	Есть
Кто делает резервные копии почтовых ящиков	Пользователь	Провайдер
Удобство для мобильных пользователей	Нет	Да
Контроль загружаемой почты пользователем	Низкий	Полный
Возможность частичной загрузки сообщений	Нет	Есть
Наличие проблем с нехваткой места на диске	Нет	Есть
Простота реализации	Да	Нет
Популярность	Широкая	Растет

Особенности доставки

Независимо от того, куда доставляется почта — напрямую на рабочую станцию пользователя или на удаленный сервер, — многие системы предоставляют средства дополнительной обработки поступающей почты. Особенно большую ценность для пользователей представляет возможность устанавливать **фильтры** — наборы правил, выполняющихся при получении нового письма или при запуске пользовательского агента. Каждое правило определяет некоторое условие и действие при его выполнении. Например, правило может гласить, что любое сообщение от начальника следует помещать в почтовый ящик номер 1, любое сообщение, пришедшее от кого-либо из группы друзей, следует помещать в почтовый ящик номер 2, а любое сообщение, содержащее определенные неприятные слова в поле *Subject*, следует удалять без вопросов.

Некоторые провайдеры предоставляют фильтры для борьбы со спамом. Эти фильтры автоматически классифицируют приходящие сообщения как нормальные или как спам (нежелательная реклама) и сохраняют каждое письмо в соответствующем ящике. Обычно такие фильтры определяют спам по адресу отправителя, если он является известным распространителем нежелательной почты. Затем анализируется поле *Subject*. Если сотни пользователей только что получили письмо с одинаковыми темами, возможно, это спам. Известны и другие методы выявления спама.

Еще одна предоставляемая услуга — возможность (временной) переадресации приходящей почты по другому адресу. Этим адресом может быть даже компьютер, управляемый коммерческой пейджинговой службой, передающей пользователю на пейджер, по радио или через спутник строку *Subject* каждого сообщения.

Еще одной часто используемой услугой доставки писем является возможность установки специального **каникулярного демона**. Это программа, отсылающая в ответ на приходящие письма сообщение типа

```
Привет! Я в отпуске. Вернусь 24 августа. Желаю веселых каникул!
```

В подобных ответах можно также указывать способы решения срочных проблем, помещать адреса людей, которые могут решить специфические проблемы, и т. п. Большинство каникулярных демонов обычно формируют список лиц, которым они посылали подобные заранее заготовленные ответы, и воздерживаются от отправки такого ответа повторно тому же лицу. Грамотно написанные демоны также проверяют, не было ли полученное сообщение прислано по списку рассылки, и если да, то вообще не отвечают на него. (Люди, посылающие сообщения в большие списки рассылки в летние месяцы, обычно не любят получать в ответ сотни сообщений с подробностями планов на отпуск каждого адресата.)

Автор однажды испытал на себе одну из экстремальных форм обработки электронной почты, когда послал сообщение человеку, утверждающему, что он получает по 600 писем в день. Его личность будет сохранена в тайне, в противном случае по меньшей мере половина читателей этой книги также пошлют ему по письму. Будем условно называть его Джоном.

Джон установил на своей машине почтовый робот, просматривающий каждое прибывающее письмо, проверяя, от нового ли оно отправителя. Если да, то робот посылает стандартный ответ, разъясняющий, что Джон более не в состоянии сам читать всю присылаемую ему почту. Вместо того чтобы отвечать каждому в отдельности, он создал личный FAQ-документ (Frequently Asked Questions — часто задаваемые вопросы). Обычно подобные документы формируются не отдельными людьми, а конференциями.

В качестве ответов на типичные вопросы Джон сообщает свои адрес, номера факса и телефона и объясняет, как связаться с его компанией. Он разъясняет, как можно организовать его лекцию, и сообщает, где можно получить его статьи и другие документы. В этом перечне также даются ссылки на программы, которые он написал, конференцию, которую он ведет, стандарт, который он редактировал, и т. д. Возможно, такие меры и являются необходимыми, хотя, не исключено, что личный перечень типичных вопросов является просто символом исключительного статуса.

Веб-почта

Нельзя под конец не сказать нескольких слов о веб-почте. Некоторые веб-сайты, например Hotmail и Yahoo!, предоставляют электронную почту всем желающим. Работает эта система следующим образом. Имеются обычные агенты передачи сообщений, прослушивающие порт 25 в ожидании входящих SMTP-соединений. Чтобы связаться, например, со службой Hotmail, вам необходимо получить DNS-запись *MX*. Это можно сделать, например, набрав в командной строке UNIX

```
host -a -v hotmail.com
```

Если предположить, что почтовый сервер называется *mx10.hotmail.com*, то для установки TCP-соединения, по которому можно привычным образом обмениваться командами SMTP, следует набрать:

```
telnet mx10.hotmail.com 25
```


Итак, ничего необычного в этой процедуре нет, если не принимать во внимание то, что большие серверы часто бывают заняты и иногда приходится предпринимать несколько попыток установки TCP-соединения.

Интересен здесь вопрос доставки электронной почты. Обычно, когда пользователь заходит на веб-страницу почтового сервера, он видит форму, в которой ему нужно заполнить поля *Имя пользователя* и *Пароль*. После того как он щелкает на кнопке *Войти*, пароль и имя пользователя отправляются на сервер, где проверяется их правильность. Если вход в систему осуществлен корректно, сервер находит почтовый ящик пользователя и строит список, подобный табл. 7.3, только оформленный в виде веб-страницы на HTML. Эта веб-страница отсылается на браузер клиента. Пользователь может щелкать на многих элементах страницы, выполняющих функции работы с почтовым ящиком, — чтения, удаления писем и т. д.

Всемирная паутина (WWW)

Всемирная паутина (WWW, World Wide Web) — это архитектура, являющаяся основой для доступа к связанным между собой документам, находящимся на миллионах машин по всему Интернету. За 10 лет своего существования из средства распространения информации на тему физики высоких энергий она превратилась в приложение, о котором миллионы людей с разными интересами думают, что это и есть «Интернет». Огромная популярность этого приложения стала следствием цветного графического интерфейса, благодаря которому даже новички не встречают затруднений при его использовании. Кроме того, Всемирная паутина предоставляет огромное количество информации практически по любому вопросу, от африканских муравьедов до яшмового фарфора.

Всемирная паутина была создана в 1989 году в Европейском центре ядерных исследований CERN (Conseil Européen pour la Recherche Nucleaire) в Швейцарии. В этом центре есть несколько ускорителей, на которых большие группы ученых из разных европейских стран занимаются исследованиями в области физики элементарных частиц. В эти команды исследователей часто входят ученые из пяти-шести и более стран. Эксперименты очень сложны, для их планирования и создания оборудования требуется несколько лет. Программа Web (паутина) появилась в результате необходимости обеспечить совместную работу находящихся в разных странах больших групп ученых, которым нужно было пользоваться постоянно меняющимися отчетами о работе, чертежами, рисунками, фотографиями и другими документами.

Изначальное предложение, создать паутину из связанных друг с другом документов пришло от физика центра CERN Тима Бернерс-Ли (Tim Berners-Lee) в марте 1989 года. Первый (текстовый) прототип заработал спустя 18 месяцев. В декабре 1991 году на конференции Hypertext'91 в Сан-Антонио в штате Техас была произведена публичная демонстрация.

Эта демонстрация, сопровождаемая широкой рекламой, привлекла внимание других ученых. Марк Андрессен (Marc Andreessen) в университете Иллинойса

начал разработку первого графического браузера, Mosaic. Программа увидела свет в феврале 1993 года и стала настолько популярной, что год спустя ее автор Марк Андрессен решил сформировать собственную компанию Netscape Communications Corp., чьей целью была разработка клиентов, серверов и другого программного обеспечения для веб-приложений. Когда в 1995 году корпорация Netscape получила известность, инвесторы, полагая, очевидно, что появилась еще одна корпорация типа Microsoft, заплатили за пакет ее акций 1,5 млрд. долларов. Это было тем более удивительно, что номенклатура продукции компании состояла всего из одного наименования, компания имела устойчивый пассивный баланс, а в своих проспектах корпорация Netscape объявила, что в обозримом будущем не рассчитывает на получение прибыли. В течение последующих трех лет между Netscape Navigator и Internet Explorer от Microsoft развернулась настоящая «война браузеров». Разработчики с той и с другой стороны в безумном порыве пытались напичкать свои программы как можно большим числом функций (а следовательно, и ошибок) и в этом превзойти соперника. В 1998 году America Online купила корпорацию Netscape Communications за 4,2 млрд долларов, положив таким образом конец весьма непродолжительному независимому существованию компании.

В 1994 году CERN и Массачусетский технологический институт (M.I.T., Massachusetts Institute of Technologies) подписали соглашение об основании **WWW-консорциума** (World Wide Web Consortium, иногда применяется сокращение **W3C**) — организации, цель которой заключалась в дальнейшем развитии приложения Web, стандартизации протоколов и поощрении взаимодействия между отдельными сайтами. Бернерс-Ли стал директором консорциума. Хотя о Всемирной паутине уже написано очень много книг, лучшее место, где вы можете получить самую свежую информацию о ней, это сама Всемирная паутина. Домашнюю страницу консорциума можно найти по адресу http://www.w3.org. На этой странице заинтересованный читатель найдет ссылки на другие страницы, содержащие информацию обо всех документах консорциума и о его деятельности.

Представление об архитектуре

С точки зрения пользователя Всемирная паутина состоит из огромного собрания документов, расположенных по всему миру. Документы обычно называют для краткости просто **страницами**. Каждая страница может содержать ссылки (указатели) на другие связанные с ней страницы в любой точке мира. Пользователи могут следовать по ссылке (например, просто щелкнув на ней мышью), при этом страница, на которую указывает ссылка, загружается и появляется в окне браузера. Этот процесс можно повторять бесконечно. Идея страниц, связанных между собой гиперссылками (**гипертекст**), была впервые пророчески предложена в 1945 году, задолго до появления Интернета, Ванневаром Бушем (Vannevar Bush), профессором из Массачусетского университета, занимавшимся электротехникой.

Страницы просматриваются специальной программой, называемой **браузером**. Самыми популярными браузерами являются программы Internet Explorer и Netscape. Браузер предоставляет пользователю запрашиваемую страницу, интерпретирует ее текст и содержащиеся в нем команды форматирования для вывода страницы на экран. Как и большинство веб-страниц, страница, показанная на рис. 7.6, *а* начинается с заголовка, содержит некоторую информацию и заканчивается адресом электронной почты организации, отвечающей за состояние этой страницы. Строки текста, представляющие собой ссылки на другие страницы, называются **гиперссылками**. Они обычно выделяются либо подчеркиванием, либо другим цветом, либо и тем и другим. Чтобы перейти по ссылке, пользователь перемещает указатель мыши в выделенную область, при этом меняется форма указателя, и щелкает на ней. Хотя существуют и текстовые браузеры, как, например, Lynx, они не так популярны, как графические, поэтому мы сконцентрируем наше внимание на последних. Следует заметить, что также разрабатываются браузеры, управляемые голосом.

ДОБРО ПОЖАЛОВАТЬ НА ВЕБ-СТРАНИЦУ УНИВЕРСИТЕТА ВОСТОЧНОГО ПОДУНКА

- Территория университета
  - Информация о приеме в университет
  - Карта территории университета
  - Как добраться до университета
  - Студенчество университета UEP
- Факультеты университета
  - Факультет психологии животных
  - Факультет альтернативных наук
  - Факультет микробиотической кулинарии
  - Факультет нетрадиционных наук
  - Факультет традиционных наук

Webmaster@eastrodunk.edu

а

ФАКУЛЬТЕТ ПСИХОЛОГИИ ЖИВОТНЫХ

- Информация для будущих студентов
- Персонал
  - Преподаватели факультета
  - Аспиранты
  - Неакадемический штат
- Исследовательские проекты
- Вакансии
- Наши самые популярные курсы
  - Обращение с травоядными животными
  - Управление лошадьми
  - Переговоры с вашей зверушкой
  - Сооружение удобной собачьей будки
- Полный список курсов

Webmaster@animalpsyc.eastrodunk.edu

б

Рис. 7.6. Веб-страница (*а*); страница, отображаемая при щелчке на строке Факультет психологии животных (*б*)

Пользователи, интересующиеся факультетом психологии животных, могут получить подробную информацию о нем, щелкнув мышью на подчеркнутой строке

с его названием. При этом браузер получит из сети страницу, на которую указывает соответствующая ссылка, и отобразит эту страницу (показана на рис. 7.6, *б*) на экране. Подчеркнутые строки этой страницы содержат ссылки на другие страницы, и т. д. Каждая новая страница, на которую указывает ссылка, может располагаться как на той же самой машине, что и первая страница, так и на компьютере, расположенном на противоположном конце Земли. Для пользователя это не заметно. Браузер добывает запрашиваемые страницы без участия пользователя. Если пользователь вернется к странице, на которой он уже был, то использованные им ссылки могут быть выделены другим цветом (или подчеркнуты пунктиром), что позволяет отличать их от тех, на которых пользователь еще не щелкал мышью. Обратите внимание что щелчок мышью на любой не подчеркнутой строке (например, *Campus Information*) не вызовет никакого эффекта. Это просто текст, не связанный ни с какой страницей.

Основной принцип работы Паутины показан на рис. 7.7. Браузер отображает веб-страницу на клиентской машине. Когда пользователь щелкает на строке, которая является ссылкой на страницу, расположенную на сервере *abcd.com*, браузер следует по этой гиперссылке. Реально при этом на *abcd.com* отправляется служебное сообщение с запросом страницы. Получив страницу, браузер показывает ее. Если на этой странице содержится гиперссылка на страницу с сервера *xyz.com*, то браузер обращается с запросом к *xyz.com*, и так далее до бесконечности.

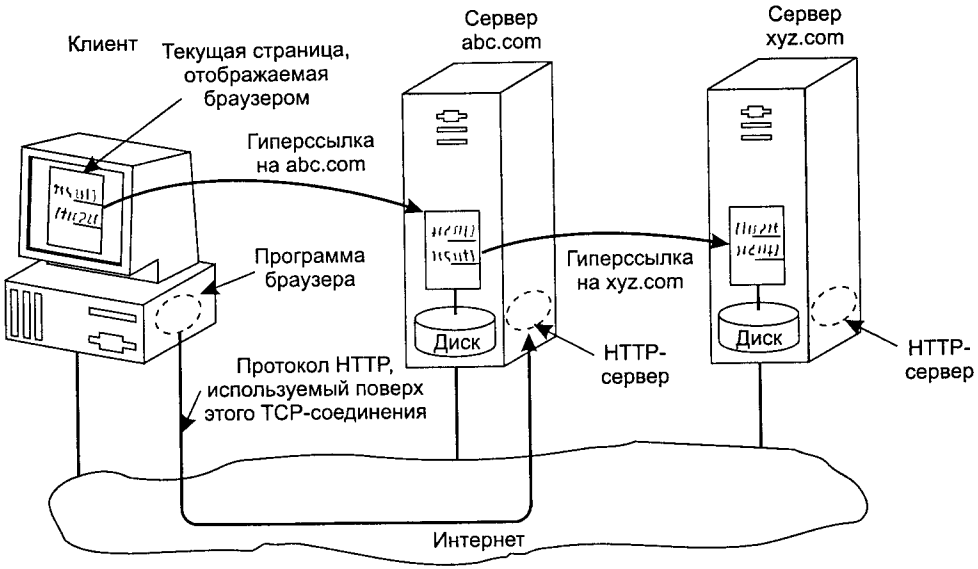


Рис. 7.7. Части Всемирной паутины

Сторона клиента

Давайте теперь более детально рассмотрим сторону клиента, основываясь на рис. 7.7. По сути дела, браузер — это программа, которая может отображать веб-страницы и распознавать щелчки мыши на элементах активной страницы. При выборе элемента браузер следует по гиперссылке и получает с сервера запрашиваемую страницу. Следовательно, гиперссылка внутри документа должна быть устроена так, чтобы она могла указывать на любую страницу Всемирной паутины. Страницы именуются с помощью **URL** (Uniform Resource Locator — унифицированный указатель информационного ресурса). Типичный указатель выглядит так: http://www.abcd.com/products.html.

Более подробно про унифицированные указатели URL будет рассказано далее в этой главе. Пока что достаточно знать, что URL состоит из трех частей: имени протокола (*http*), DNS-имени машины, на которой расположена страница (www.abcd.com), и (обычно) имени файла, содержащего эту страницу (*products.html*). Между щелчком пользователя и отображением страницы происходят следующие события.

Когда пользователь щелкает мышью на гиперссылке, браузером выполняется ряд действий, приводящих к загрузке страницы, на которую указывает ссылка. Предположим, пользователь, блуждая по Интернету, находит ссылку на документ, рассказывающий про интернет-телефонию, а конкретно — на домашнюю страницу ITU, расположенную по адресу http://www.itu.org/home/index.html. Рассмотрим каждое действие, происходящее после выбора этой ссылки.

1. Браузер определяет URL (по выбранному элементу страницы).
2. Браузер запрашивает у службы DNS IP-адрес *www.itu.org*.
3. DNS дает ответ 156.106.192.32.
4. Браузер устанавливает TCP-соединение с 80-м портом машины 156.106.192.32.
5. Браузер отправляет запрос на получение файла */home/index.html*.
6. Сервер *www.itu.org* высылает файл */home/index.html*.
7. TCP-соединение разрывается.
8. Браузер отображает весь текст файла */home/index.html*.
9. Браузер получает и выводит все изображения, прикрепленные к данному файлу.
10. Многие браузеры отображают текущее выполняемое ими действие в строке состояния внизу экрана. Это позволяет пользователю понять причину низкой производительности: например, не отвечает служба DNS или сервер или просто сильно перегружена сеть при передаче страницы.
11. Для отображения каждой новой страницы браузер должен понять ее формат. Чтобы все браузеры могли отображать любые страницы, они пишутся на стандартизованном языке HTML, описывающем веб-страницы. Более детально мы рассмотрим его далее.

Несмотря на то что браузер, по сути дела, представляет собой интерпретатор HTML, большинство браузеров оснащаются многочисленными кнопками и функциями, облегчающими навигацию по Всемирной паутине. У многих браузеров есть кнопки для возврата на предыдущую страницу и перехода на следующую страницу (последняя доступна только в том случае, если пользователь уже возвращался назад), а также кнопка для прямого перехода на домашнюю страницу

пользователя. Большинство браузеров поддерживают в меню команды для установки закладки на текущей странице и отображения списка закладок, что позволяет попадать на любую страницу при помощи всего одного щелчка мышью. Страницы также могут быть сохранены на диске или распечатаны на принтере. Кроме того, доступны многочисленные функции управления отображением страницы и установки различных настроек пользователя.

Помимо обычного текста (не подчеркнутого) и гипертекста (подчеркнутого), веб-страницы могут также содержать значки, рисунки, чертежи и фотографии. Все они могут быть связаны ссылкой с другой страницей. Щелчок на элементе, содержащем ссылку, также вызовет смену текущей страницы, отображаемой браузером. С большими изображениями, такими как фотографии или карты, может быть ассоциировано несколько ссылок, при этом следующая отображаемая страница будет зависеть от того, на каком месте изображения произведен щелчок мышью.

Далеко не все страницы написаны исключительно с применением HTML. Например, страница может состоять из документа в формате PDF, значка в формате GIF, фотографии в JPEG, звукозаписи в формате MP3, видео в MPEG или чего-то еще в одном из сотни форматов. Поскольку стандартные HTML-страницы могут иметь ссылки на любые файлы, у браузера возникает проблема обработки страницы, которую он не может интерпретировать.

Вместо того чтобы наращивать возможности и размеры браузеров, встраивая в них интерпретаторы для различных типов файлов (количество которых быстро растет), обычно применяется более общее решение. Когда сервер возвращает в ответ на запрос какую-либо страницу, вместе с ней высылается некоторая дополнительная информация о ней. Эта информация включает MIME-тип страницы (см. табл. 7.7). Страницы типа *text/html* выводятся браузером напрямую, как и страницы некоторых других встроенных типов. Если же для данного MIME-типа внутренняя интерпретация невозможна, браузер определяет, как выводить страницу, по своей таблице MIME-типов. В данной таблице в соответствие каждому типу ставится программа просмотра.

Существуют два способа отображения: с помощью подключаемого модуля (**plug-in**) или вспомогательных приложений. Подключаемый модуль представляет собой особый код, который браузер извлекает из специального каталога на жестком диске и устанавливает в качестве своего расширения, как показано на рис. 7.8, *а*. Поскольку подключаемые модули работают внутри браузера, у них есть доступ к текущей странице, вид которой они могут изменять. После завершения своей работы (обычно это связано с переходом пользователя на другую страницу) подключаемый модуль удаляется из памяти браузера.

Каждый браузер имеет набор процедур, которые должны реализовывать все подключаемые модули. Это нужно для того, чтобы браузер мог обращаться к последним. Например, существует стандартная процедура, с помощью которой базовый код браузера передает подключаемому модулю данные для отображения. Набор этих процедур образует интерфейс подключаемого модуля и является специфичным для каждого конкретного браузера.

Кроме того, браузер предоставляет подключаемому модулю определенный набор своих процедур. Среди них в интерфейс браузера обычно включаются процедуры распределения и освобождения памяти, вывода сообщений в строке статуса браузера и опроса параметров браузера.

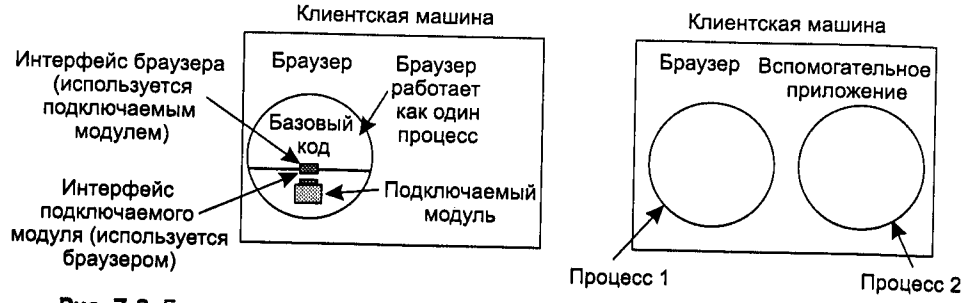


Рис. 7.8. Браузер с подключаемым модулем (*а*); вспомогательное приложение (*б*)

Перед использованием подключаемого модуля его нужно установить. Этот процесс подразумевает, что пользователь копирует с веб-сайта производителя модуля файл установки. В Windows он обычно представляет собой самораспаковывающийся архив ZIP с расширением .exe. Если дважды щелкнуть на таком файле, запускается небольшая программа, включенная в начало архива. Она распаковывает подключаемый модуль и копирует его в соответствующий каталог, известный браузеру. Затем она регистрирует MIME-тип, обрабатываемый модулем, и ассоциирует этот тип с модулем. В системах UNIX установочный файл зачастую представляет собой основной сценарий, осуществляющий копирование и регистрацию.

Вторым способом расширения возможностей браузера является использование **вспомогательных приложений**. Вспомогательное приложение — это полноценная программа, работающая как отдельный процесс. Это показано на рис. 7.8, *б*. Поскольку она никак не связана с браузером, между ними отсутствует какой бы то ни было интерфейс. Вместо этого обычно вспомогательное приложение вызывается с параметром, представляющим собой имя временного файла, содержащего данные для отображения. Получается, что браузер можно настроить на обработку практически любого типа файлов, не внося в него никаких изменений. Современные веб-серверы часто содержат сотни комбинаций типов и подтипов файлов. Новые типы файлов появляются всякий раз при установке новых программ.

Вспомогательные приложения не обязательно связаны только с MIME-типом *application* (приложение). Например, Adobe Photoshop будет работать с *image/x-photoshop*, а RealOne Player может поддерживать *audio/mp3*.

В Windows каждая устанавливаемая на компьютер программа регистрирует типы, которые она хочет поддерживать. Такой механизм приводит к конфликту, когда несколько программ могут обрабатывать один и тот же подтип, например, *video/mpg*. Конфликт разрешается следующим образом: программа, которая регистрируется последней, затирает своей записью существующую ассоциацию (тип MIME, вспомогательное приложение) для тех типов, которые она готова обрабатывать самостоятельно. Следствием этого является то, что каждая уста-

навливаемая программа может изменить метод отображения браузером некоторых типов.

В UNIX этот процесс обычно не является автоматическим. Пользователь должен вручную обновлять конфигурационные файлы. Такой подход приводит к уменьшению числа неожиданных сюрпризов, но при этом у пользователя появляются дополнительных заботы.

Браузеры могут работать и с локальными файлами, не запрашивая информацию с удаленных серверов. Поскольку локальные файлы не сообщают свои MIME-типы, браузеру нужно каким-то хитрым образом определить, какой подключаемый модуль или вспомогательное приложение использовать для типов, отличных от встроенных (таких, как *text/html* или *image/jpeg*). Для поддержки локальных файлов вспомогательные модули и приложения должны быть ассоциированы как с расширениями файлов, так и с типами MIME. При стандартных настройках попытка открыть файл foo.pdf приведет к его открытию в Acrobat, а файл bar.doc будет открыт в Word. Некоторые браузеры для определения типа MIME используют не только данные о типе MIME, но и расширение файла и даже данные, взятые из самого файла. В частности, Internet Explorer по возможности старается ориентироваться на расширение файла, а не на тип MIME.

Здесь также возможны конфликты, поскольку многие программы страстно желают поддерживать, например, .mpg. Профессиональные программы при установке обычно выводят флажки, позволяющие выбрать поддерживаемые типы MIME и расширения. Таким образом, пользователь может выбрать то, что ему требуется, и таким образом избежать случайного затирания существующих ассоциаций. Программы, нацеленные на массового потребителя, полагают, что большинство пользователей понятия не имеют о типах MIME и просто захватывают все что могут, совершенно не обращая внимания на ранее установленные программы.

Расширение возможностей браузера по поддержке новых типов файлов — это удобно, но может также привести к возникновению некоторых проблем. Когда Internet Explorer получает файл с расширением exe, он думает, что это исполняемая программа и никаких вспомогательных средств для нее не требуется. Очевидно, следует просто запустить программу. Однако такой подход может оказаться серьезной дырой в системе защиты информации. Злоумышленнику требуется лишь создать нехитрый сайт с фотографиями, скажем, знаменитых киноактеров или спортсменов и поставить ссылки на вирусы. Один-единственный щелчок мышкой на фотографии может в этом случае привести к запуску непредсказуемой и, возможно, опасной программы, которая будет действовать на машине пользователя. Для предотвращения подобных нежелательных ситуаций Internet Explorer можно настроить на избирательный запуск неизвестных программ, однако не все пользователи в состоянии справиться с настройкой браузера.

В UNIX похожая проблема может возникнуть с основными сценариями, однако при этом требуется, чтобы пользователь сознательно стал устанавливать вспомогательную программу. К счастью, это процесс довольно сложный, и «случайно» установить что-либо практически невозможно (немногие могут сделать это и преднамеренно).

Сторона сервера

О стороне клиента сказано уже достаточно много. Поговорим теперь о стороне сервера. Как мы уже знаем, когда пользователь вводит URL или щелкает на гиперссылке, браузер производит структурный анализ URL и интерпретирует часть, заключенную между *http://* и следующей косой чертой, как имя DNS, которое следует искать. Вооружившись IP-адресом сервера, браузер устанавливает TCP-соединение с портом 80 этого сервера. После этого отсылается команда, содержащая оставшуюся часть URL, в которой указывается имя файла на сервере. Сервер возвращает браузеру запрашиваемый файл для отображения.

В первом приближении веб-сервер напоминает сервер, представленный в листинге 6.1. Этому серверу, как и настоящему веб-серверу, передается имя файла, который следует найти и отправить. В обоих случаях в основном цикле сервер выполняет следующие действия:

1. Принимает входящее TCP-соединение от клиента (браузера).
2. Получает имя запрашиваемого файла.
3. Получает файл (с диска).
4. Возвращает файл клиенту.
5. Разрывает TCP-соединение.

Современные веб-серверы обладают более широкими возможностями, однако существенными в их работе являются именно перечисленные шаги.

Проблема данного подхода заключается в том, что каждый запрос требует обращения к диску для получения файла. В результате число обращений к веб-серверу за секунду ограничено максимальной скоростью обращений к диску. Среднее время доступа к высокоскоростному диску типа SCSI составляет около 5 мс, то есть сервер может обрабатывать не более 200 обращений в секунду. Это число даже меньше, если часто запрашиваются большие файлы. Для крупных веб-сайтов это слишком мало.

Очевидным способом решения проблемы является кэширование в памяти n последних запрошенных файлов. Прежде чем обратиться за файлом к диску, сервер проверяет содержимое кэша. Если файл обнаруживается в кэше, его можно сразу выдать клиенту, не обращаясь к диску. Несмотря на то, что для эффективного кэширования требуются большие объемы памяти и некоторое дополнительное время на проверку кэша и управление его содержимым, суммарный выигрыш во времени почти всегда оправдывает эти накладные расходы и стоимость.

Следующим шагом, направленным на повышение производительности, является создание многопоточных серверов. Одна из реализаций подразумевает, что сервер состоит из входного модуля, принимающего все входящие запросы, и k обрабатывающих модулей, как показано на рис. 7.9. Все $k + 1$ потоков принадлежат одному и тому же процессу, поэтому у обрабатывающих модулей есть доступ к кэшу в адресном пространстве процесса. Когда приходит запрос, входящий модуль принимает его и создает краткую запись с его описанием. Затем запись передается одному из обрабатывающих модулей. Другая возможная реализация подразумевает отсутствие входного модуля; все обрабатывающие модули пыта-

ются получить запросы, однако здесь требуется блокирующий протокол, помогающий избежать конфликтов.

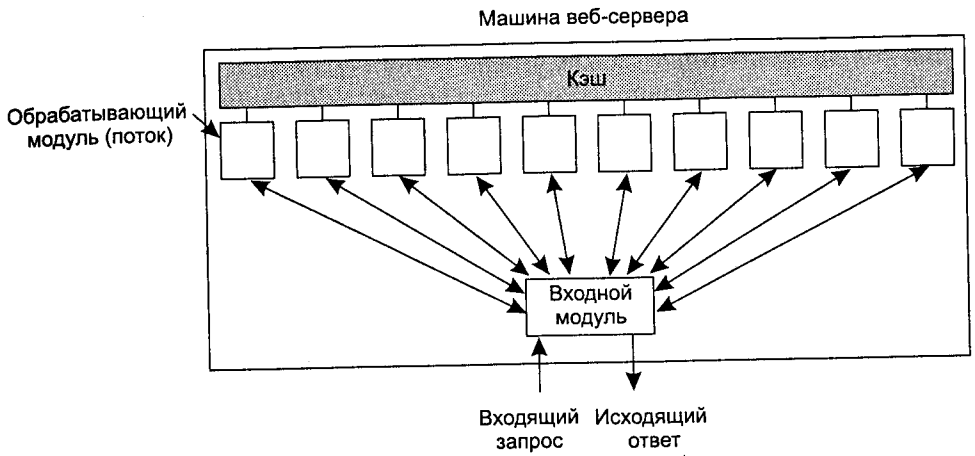


Рис. 7.9. Многопоточный веб-сервер с входным и обрабатывающими модулями

Обрабатывающий модуль вначале проверяет кэш на предмет нахождения там нужных файлов. Если они там действительно есть, он обновляет запись, включая в нее указатель на файл. Если искомого файла в кэше нет, обрабатывающий модуль обращается к диску и считывает файл в кэш (при этом, возможно, затирая некоторые хранящиеся там файлы, чтобы освободить место). Считанный с диска файл попадает в кэш и отсылается клиенту.

Преимущество такой схемы заключается в том, что пока один или несколько обрабатывающих модулей заблокированы в ожидании окончания дисковой операции (при этом такие модули не потребляют мощности центрального процессора), другие модули могут активно обрабатывать захваченные ими запросы. Разумеется, реального повышения производительности за счет многопоточной схемы можно достичь, только если установить несколько дисков, чтобы в каждый момент времени можно было обращаться более чем к одному диску. Имея k обрабатывающих модулей и k дисков, производительность можно повысить в k раз по сравнению с однопоточным сервером и одним диском.

Теоретически, однопоточный сервер с k дисками тоже должен давать прирост производительности в k раз, однако программирование и администрирование такой схемы оказывается очень сложным, так как в этом случае невозможно использование обычных блокирующих системных вызовов READ для чтения с диска. Многопоточные серверы такого ограничения не имеют, поскольку READ будет блокировать только тот поток, который осуществил системный вызов, а не весь процесс.

Современные веб-серверы выполняют гораздо больше функций, чем просто прием имен файлов и отправка файлов. На самом деле, реальная обработка каждого запроса может оказаться довольно сложной. По этой причине на многих серверах каждый обрабатывающий модуль выполняет серии действий. Входной

модуль передает каждый входящий запрос первому доступному модулю, который обрабатывает его путем выполнения некоторого подмножества указанных далее шагов в зависимости от того, что именно требуется для данного запроса:

- вычисление имени запрашиваемой веб-страницы;
- регистрация клиента;
- осуществление контроля доступа для клиента;
- осуществление контроля доступа для веб-страницы;
- проверка кэша;
- получение запрошенной страницы с диска;
- определение типа MIME для включения этой информации в ответ клиенту;
- аккуратное выполнение различных дополнительных задач;
- возвращение ответа клиенту;
- добавление записи в журнал активности сервера.

Шаг 1 необходим, потому что входящий запрос может и не содержать реального имени файла в виде строкового литерала. Например, URL может быть вот таким: http://www.cs.vu.nl. Здесь имя файла отсутствует. Этот URL необходимо дополнить неким именем файла по умолчанию. К тому же современные браузеры могут указывать язык пользователя по умолчанию (например, итальянский или английский), что позволяет серверу выбирать веб-страницу на соответствующем языке, если таковая существует. Вообще говоря, расширение имени — задача не такая уж тривиальная, как может показаться, поскольку существует множество соглашений об именовании файлов.

Шаг 2 состоит в проверке идентификационных данных клиента. Это нужно для отображения страниц, недоступных для широкой публики. Мы обсудим один из способов такой проверки далее в этой главе.

Шаг 3 проверяет наличие каких-либо ограничений, накладываемых на данного клиента и его местоположение. На шаге 4 проверяются ограничения на доступ к запрашиваемой странице. Если определенный файл (например, .htaccess) присутствует в том же каталоге, что и нужная страница, он может ограничивать доступ к файлу. Например, можно установить доступ к странице только для сотрудников компании.

Шаги 5 и 6 включают в себя получение страницы. Во время выполнения шага 6 должна быть обеспечена возможность одновременного чтения с нескольких дисков.

Шаг 7 связан с определением типа MIME, исходя из расширения файла, первых нескольких байтов, конфигурационного файла или каких-то иных источников. Шаг 8 предназначен для различных задач, таких как построение профиля пользователя, сбор статистики и т. д.

На шаге 9 наконец отсылается результат, что фиксируется в журнале активности сервера на шаге 10. Последний шаг требуется для нужд администрирования. Из подобных журналов можно впоследствии узнать ценную информацию

о поведении пользователей — например, о том, в каком порядке люди посещают страницы на сайте.

Если приходит слишком много запросов в секунду, центральный процессор может перестать справляться с их обработкой вне зависимости от того, сколько дисков параллельно работают на сервере. Решается эта проблема установкой на сервере нескольких узлов (компьютеров). Их полезно укомплектовывать реплицированными (содержащими одинаковую информацию) дисками во избежание ситуации, когда узким местом становится дисковый накопитель. В результате возникает многомашинная система, организованная в виде **серверной фермы** (рис. 7.10). Входной модуль по-прежнему принимает входящие запросы, однако распределяет их на сей раз не между потоками, а между центральными процессорами, снижая тем самым нагрузку на каждый компьютер. Отдельные машины сами по себе могут быть многопотоковыми с конвейеризацией, как и в рассматриваемом ранее случае.

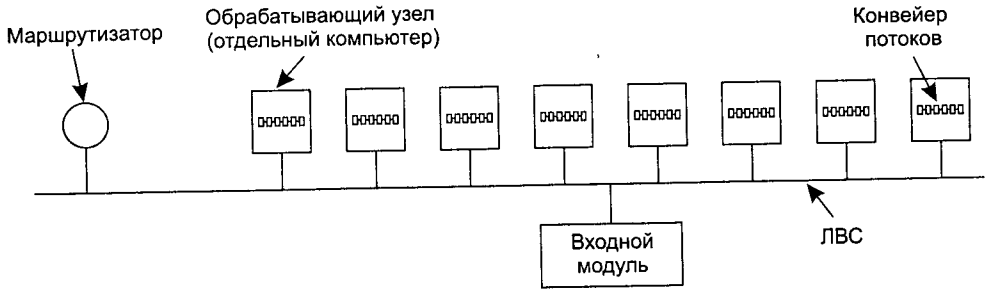


Рис. 7.10. Серверная ферма

Одна из проблем, связанных с серверными фермами, заключается в отсутствии общего кэша — каждый обрабатывающий узел обладает собственной памятью. Эта проблема может быть решена установкой дорогостоящей мультипроцессорной системы с разделяемой памятью, однако существует и более дешевый способ. Он заключается в том, что входной модуль запоминает, на какой узел он посылал запросы конкретных страниц. Последующие запросы тех же страниц он сможет тогда направлять на те же узлы. Таким образом, получается, что каждый узел специализируется по своему набору страниц; и отпадает необходимость хранения одних и тех же файлов в кэшах разных компьютеров.

Другая проблема, возникающая при использовании серверных ферм, состоит в том, что TCP-соединение клиента заканчивается на входном модуле, то есть ответ в любом случае должен пройти через входной модуль. Эта ситуация показана на рис. 7.11, *а*. Здесь как входящий запрос (1), так и исходящий ответ (2) проходят через входной модуль. Иногда для обхода этой проблемы применяется хитрость под названием **передача TCP**. Суть ее в том, что TCP-соединение продлевается до конечного (обрабатывающего) узла, и он может самостоятельно отправить ответ напрямую клиенту (рис. 7.11, *б*). Эта передача соединения для клиента незаметна.

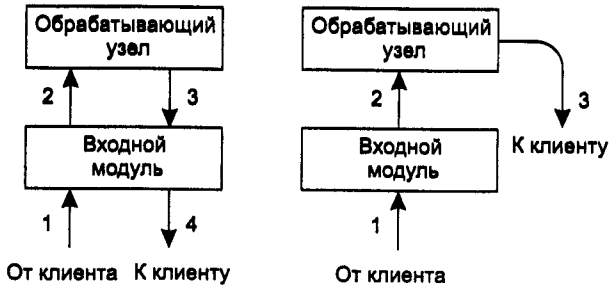


Рис. 7.11. Обычный запрос-ответный обмен (*а*); обмен запросами и ответами при передаче TCP (*б*)

URL — унифицированные указатели информационных ресурсов

Мы несколько раз упоминали о том, что веб-страницы могут быть связаны между собой ссылками. Пора познакомиться с тем, как эти ссылки реализованы. Уже при создании Паутины было очевидно, что для реализации ссылок с одних страниц на другие необходим механизм именования и указания расположения страниц. В частности, прежде чем выводить выбранную страницу на экран, нужно узнать ответы на три следующих вопроса.

1. Как называется эта страница?
2. Где она расположена?
3. Как получить к ней доступ?

Если бы каждой странице можно было присвоить уникальное имя, то в идентификации страниц не было бы никакой неоднозначности. Тем не менее, проблему бы это не решило. Для примера проведем параллель между страницами и людьми. В Соединенных Штатах почти у всех граждан есть номер карточки социального страхования, представляющий собой уникальный идентификатор, так как нет двух людей с одинаковым номером. Тем не менее, зная только номер карточки социального страхования, нет способа узнать адрес владельца и, конечно, нельзя определить, следует ли писать этому гражданину по-английски, по-испански или по-китайски. Во Всемирной паутине проблемы, в принципе, те же самые.

В результате было принято решение идентифицировать страницы способом, решающим сразу все три проблемы. Каждой странице назначается унифицированный указатель информационного ресурса (**URL**, Uniform Resource Locator), который служит уникальным именем страницы. URL состоят из трех частей: протокола (также называемого **схемой**), DNS-имени машины, на которой расположена страница, и локального имени, единственным образом идентифицирующего страницу в пределах этой машины (обычно это просто имя файла). Например, веб-сайт факультета, на котором работает автор, содержит несколько видеофрагментов об университете и городе Амстердаме. Унифицированный указатель страницы с видео выглядит следующим образом:

http://www.cs.vu.nl/video/index-en.html

Этот URL состоит из трех частей: протокола (*http*), DNS-имени хоста (*www.cs.vu.nl*) и имени файла (*video/index-en.html*). Отдельные части URL-указателя разделяются специальными знаками пунктуации. Имя файла представляет собой относительный путь по отношению к веб-каталогу *cs.vu.vu.nl.*

У сайтов могут быть сокращенные имена для ускоренного доступа к определенным файлам. Скажем, при отсутствии в URL имени файла может выводиться главная (домашняя) страница сайта. Если имя файла заканчивается именем каталога, то из него по умолчанию выбирается файл с именем *index.html.* Наконец, имя *~user/* может соответствовать WWW-каталогу пользователя, причем может быть также задано имя файла по умолчанию, например, *index.html.* Так, на домашнюю страницу автора можно попасть по адресу

http://www.cs.vu.nl/~ast/

несмотря на то, что действительное имя файла (*index.html*) отличается от указанного.

Теперь надо понять, как работает гипертекст. Чтобы на неком участке текста браузер мог реагировать на щелчок мыши, при написании веб-страницы нужно обозначить два элемента: отображаемый на экране текст ссылки и URL страницы, которая должна стать текущей при щелчке мышью. Синтаксис такой команды будет пояснен далее в этой главе.

При выборе ссылки браузер с помощью службы DNS ищет имя хоста. Зная IP-адрес хоста, браузер устанавливает с ним TCP-соединение. По этому соединению с помощью указанного протокола браузер посылает имя файла, содержащего страницу. Вот, собственно, и все. Назад по соединению передается страница.

Такая схема является открытой в том смысле, что она позволяет использовать разные протоколы для доставки информационных единиц разного типа. Определены URL-указатели для других распространенных протоколов, понимаемые многими браузерами. Слегка упрощенные формы наиболее употребительных URL-указателей приведены в табл. 7.9.

Таблица 7.9. Некоторые распространенные URL-указатели

Имя	Применение	Пример
http	Гипертекст (HTML)	http://www.cs.vu.nl/~ast/
ftp	FTP	ftp://ftp.cs.vu.nl/pub/minix/README
file	Локальный файл	file:////usr/suzanne/prog.c
news	Телеконференция	news:comp.os.minix
news	Статья новостей	news:AA0134223112@cs.utah.edu
gopher	Gopher	gopher://gopher.tc.umn.edu/11/Libraries
mailto	Отправка электронной почты	mailto:JohnUser@acm.org
telnet	Удаленный терминал	telnet://www.w3.org:80

Кратко рассмотрим этот список. Протокол *http* является родным языком Всемирной паутины, на нем разговаривают веб-серверы. **HTTP** — это сокращение, которое расшифровывается как HyperText Transfer Protocol (протокол передачи гипертекста). Более подробно мы рассмотрим его далее в этой главе.

Протокол *ftp* используется для доступа к файлам по FTP — протоколу передачи файлов по Интернету. За двадцать лет своего существования он достаточно хорошо укоренился в сети. Многочисленные FTP-серверы по всему миру позволяют пользователям в любых концах Интернета регистрироваться на сервере и скачивать разнообразные файлы, размещенные на сервере. Всемирная паутина здесь не вносит особых изменений. Она просто упрощает доступ к FTP-серверам и работу с файлами, ибо само по себе FTP имеет несколько загадочный интерфейс (однако более мощный, чем HTTP: например, он позволяет пользователю машины *A* передать файл с машины *B* на машину *C*).

К локальному файлу также можно обратиться как к веб-странице, либо используя протокол *file*, либо просто указав имя файла. Такой подход напоминает FTP, но не требует наличия сервера. Разумеется, он работает только с локальными файлами, а не с расположенными на удаленных терминалах.

Задолго до появления Интернета появилась система групп новостей USENET. Она состоит примерно из 30 000 конференций, в которых миллионы людей обсуждают широкий круг вопросов, отправляя и читая сообщения, связанные с тематикой данной конференции. Протокол *news* позволяет пользователю вызывать на экран статью с новостями, как если бы она была обычной веб-страницей. Это означает, что веб-браузер легким движением руки превращается в элегантную программу чтения новостей. На самом деле, благодаря кнопкам и пунктам меню многих браузеров чтение новостей USENET становится даже удобнее, чем с помощью специальных программ чтения сетевых новостей.

Для протокола *news* поддерживается два формата URL-указателей. Первый формат указывает телеконференцию, и с его помощью можно получить список новых статей с указанного заранее сайта новостей. Второй формат позволяет получить конкретную статью по ее идентификатору, например, *AA0134223112@cs.utah.edu.* Для получения этой статьи с заранее настроенного сайта браузер использует протокол NNTP (Network News Transfer Protocol — сетевой протокол передачи новостей). Мы изучим NNTP в этой книге, однако надо понимать, что это нечто вроде SMTP, они весьма похожи даже по стилю.

Протокол *gopher* используется системой Gopher, разработанной в университете штата Миннесота и получившей свое название от университетской спортивной команды «Golden Gophers» («Золотые суслики»). (Гоферами называют уроженцев штатов Миннесота, Арканзас и Флорида. Кроме того, на американском сленге это слово означает «добывать», «копать», «искать».) Система Gopher появилась в Интернете на несколько лет раньше Всемирной паутины. Концептуально они похожи: и та, и другая представляют собой схему поиска и получения информации, хранящейся на различных серверах, однако система Gopher поддерживала только тексты и не поддерживала изображений. Сейчас ее можно считать полностью устаревшей, используется она крайне редко.

Последние два протокола не занимаются имитацией получения веб-страниц, но они также полезны. Протокол *mailto* позволяет пользователю посылать электронную почту из веб-браузера. Например, в некоторых браузерах для этого нужно щелкнуть на кнопке OPEN и ввести URL-указатель, состоящий из слова *mailto:*, за которым следует почтовый адрес получателя. В ответ в большинстве

браузеров откроется специальная форма, содержащая поля для редактирования темы письма и других заголовков, а также окно для ввода текста самого письма.

С помощью протокола *telnet* можно установить в подключенном режиме соединение с удаленным компьютером. Он используется так же, как и программа Telnet, что неудивительно, так как большинство браузеров просто вызывают саму программу Telnet как вспомогательное приложение.

Итак, URL-указатели позволяют пользователям не только путешествовать по Всемирной паутине, но и работать с FTP-серверами, BBS, Gopher-серверами, электронной почтой и регистрироваться на удаленных серверах с помощью программы Telnet. Все эти ресурсы оказываются доступны при помощи всего одной программы — веб-браузера, что очень удобно. Если бы отцом этой идеи не был ученый-физик, она стала бы, вероятно, самой убедительной рекламой какой-нибудь компании, специализирующейся на выпуске программного обеспечения.

Несмотря на все перечисленные достоинства, все продолжающийся рост популярности Всемирной паутины выявил один врожденный недостаток URL-схемы. URL указывает на определенный хост. Часто запрашиваемые по сети страницы было бы лучше дублировать и хранить копии в удаленных концах сети, чтобы снизить сетевой трафик. Беда в том, что URL-указатели не предоставляют возможности для ссылки на страницу без указания ее точного адреса. Нельзя сказать: «Мне нужна страница abc, и мне все равно, где вы ее раздобудете». Для решения задачи репликации страниц проблемная группа проектирования Интернета IETF (Internet Engineering Task Force) работает над системой **URN** (Uniform Resource Name — универсальное имя ресурса). Универсальное имя ресурса URN можно считать обобщенным URL-указателем. В настоящее время этот вопрос находится в стадии исследования, хотя уже предложен синтаксис, описанный в RFC 2141.

Принцип отсутствия состояний и cookie-файлы

Как мы уже неоднократно повторяли, Всемирная паутина изначально не может находиться в каком-либо состоянии. Отсутствует понятия сеанса связи. Браузер клиента посылает запрос на сервер и получает в ответ файл. После этого сервер забывает о том, что он когда-либо видел этого клиента.

Вначале, когда Паутина использовалась исключительно для получения документов, предназначенных для широкой публики, такая модель была адекватна. Но по мере наращивания функциональности стали появляться различные проблемы. К примеру, на некоторых сайтах требуется регистрация пользователей; а иногда получение информации с сайта является платным. Возникает вопрос различения запросов от зарегистрированных пользователей и от всех остальных. Вторым примером является электронная коммерция. Как серверу собрать информацию о состоянии корзины пользователя, если тот время от времени пополняет ее содержимое? Третий пример — веб-порталы типа Yahoo. Пользователям предоставляется возможность настраивать вид начальной страницы (например, так, чтобы им сразу показывались последние новости о любимой спортивной команде или курсы ценных бумаг). Однако как сервер сможет корректно отобразить страницу, если он не знает, с каким пользователем он имеет дело?

На первый взгляд может показаться, что решение очевидно: сервер может различать пользователей по их IP-адресам. Однако эта идея не работает. Во-первых, многие пользователи работают на компьютерах с разделяемыми ресурсами (например, сотрудники компании), и с помощью IP-адреса можно идентифицировать лишь компьютер, а не пользователя. Во-вторых, что еще хуже, многие провайдеры используют NAT, поэтому все исходящие пакеты от тысяч клиентов имеют один и тот же IP-адрес.

Для решения этой проблемы фирмой Netscape был предложен сильно раскритикованный метод **cookie-файлов**. Такое название произошло от старинного программистского сленгового словечка. Программа вызывала процедуру и получала взамен нечто, что могло понадобиться впоследствии для выполнения какой-либо задачи. Это «нечто» и называлось cookie. В этом смысле файловый дескриптор в UNIX и дескриптор объектов в Windows можно рассматривать как cookie. Эти специальные маркеры позднее были формализованы в RFC 2109.

Когда пользователь запрашивает страницу, сервер может снабдить свой ответ дополнительной информацией, которая может включать в себя cookie, то есть маркер, представляющий собой маленький (до 4 Кбайт) файл (или строку). Браузеры сохраняют полученные маркеры в специальном каталоге на жестком диске, если только пользователь не отключил данную функцию. Итак, cookie — это файлы или строки, а не исполняемые программы. В принципе, в них может содержаться вирус, но поскольку они рассматриваются всего лишь как информационные данные, нет какой-либо официальной возможности для активации вирусов и нанесения ущерба системе. Однако у хакера всегда есть возможность, используя ошибки в браузере, заставить активироваться вирусы, содержащиеся в cookie.

В маркерах cookie может содержаться до пяти полей, как показано в табл. 7.10. Поле *Домен* содержит имя домена, с которого пришел маркер. Предполагается, что браузеры проверяют тот факт, что серверы не лгут относительно имен доменов. Каждый домен может хранить не более 20 маркеров, связанных с одним клиентом. Поле *Путь* содержит путь в структуре каталогов на сервере, указывающий те части дерева каталогов, которые могут использовать маркер. Часто в этом поле содержится знак «/», означающий, что доступно дерево целиком.

Таблица 7.10. Примеры cookie

Домен	Путь	Содержимое	Годен до	Защищенный
toms-casino.com	/	CustomerID=497793521	15-10-02 17:00	Да
joes-store.com	/	Cart=1-00501;1-07031;2-13721	11-10-02 14:22	Нет
aportal.com	/	Prefs=Stk;SUNW+ORCL;Spt:Jets	31-12-10 23:59	Нет
sneaky.com	/	UserID=362723910	31-12-12 23:59	Нет

Поле *Содержимое* имеет вид *имя* = *значение*. Как *имя*, так и *значение* могут быть совершенно произвольными, на усмотрение сервера. В этом поле хранится основная информация, которую несет в себе маркер.

Поле *Годен до* указывает срок годности маркера. Если это поле отсутствует, браузер отбрасывает cookie сразу после выхода из программы. Такой маркер называется **неустойчивым**. Если же указано время и дата, то о таком маркере говорят, что он **устойчивый**. Он хранится до тех пор, пока не выйдет срок годности. Время, указываемое в поле *Годен до*, — гринвичское. Чтобы удалить cookie с жесткого диска клиента, сервер просто посылает его заново, указывая вышедший срок годности.

Наконец, поле *Защищенный* может быть установлено для индикации того, что браузер может вернуть его только на защищенный сервер. Это свойство используется в электронной коммерции, банковском деле и других приложениях, в которых важна защита информации.

Итак, мы узнали о получении маркеров. Как же они используются? Непосредственно перед отправкой браузером запроса на получение страницы на какой-нибудь веб-сайт проверяется каталог с cookie. Ищутся маркеры, пришедшие с того домена, на который отправляется запрос. Все найденные маркеры отправляются вместе с запросом. Получив их, сервер может интерпретировать содержащуюся в них информацию так, как ему нужно.

Рассмотрим возможные применения cookie. В табл. 7.9 первый cookie был прислан доменом *toms-casino.com* и используется для идентификации клиента. Если через неделю тот же клиент возвращается на сайт, чтобы просадить очередную сумму денег, браузер отправляет cookie, так что сервер узнает старого знакомого. Вооружившись идентификатором пользователя, сервер может найти запись о нем в базе данных и использовать эту информацию для генерации соответствующей веб-страницы. В зависимости от азартности игрока ему может быть предложен покер, таблица результатов сегодняшних скачек или просто торговый автомат.

Второй cookie-маркер пришел с *joes-store.com*. Сценарий в этом случае заключается в том, что клиент бродит по магазину, выбирая себе покупки. Найдя что-нибудь привлекательное, он щелкает на значке товара. При этом сервер создает cookie-маркер, содержащий количество и идентификатор заказанного товара, и отсылает его клиенту. Клиент продолжает бродить по электронному магазину, и в ответ на каждый новый запрос страницы ему отсылается cookie. По мере накопления выбранных товаров дополняется информация в cookie. В таблице показано, что в корзине клиента содержатся три вида товаров, причем заказано два экземпляра товара третьего вида. Наконец, когда пользователь щелкает *ПЕРЕЙТИ К РАСЧЕТАМ*, cookie, содержащий теперь уже полную информацию о покупках, отсылается вместе с запросом на сервер. Таким образом, серверу точно известно, какие товары заказал клиент.

Третий cookie-маркер прибыл с веб-портала. Когда пользователь щелкает на ссылке на портал, браузер отсылает ему cookie, в котором говорится о том, что надо показать страницу, содержащую котировки акций Sun Microsysems и Oracle, а также результаты футбольного матча New York Jets. Так как максимальный размер cookie-файла равен 4 Кбайт, то остается еще много места для более детальной настройки страницы. Например, в нее можно включить сводку погоды, специальные предложения, заголовки статей в крупных газетах и т. п.

Cookie могут использоваться и для нужд самого сервера. Например, с их помощью можно отслеживать число различных посетителей сайта, узнавать, сколько страниц просмотрел каждый из них, и составлять по этим данным статистику. Когда на сервер приходит первый запрос от пользователя, вместе с ним, разумеется, не высылается никакой маркер. Поэтому сервер отсылает обратно cookie со значением счетчика, равным 1. Последующие переходы между страницами сайта уже будут сопровождаться отсылкой cookie. Всякий раз счетчик будет инкрементироваться и отсылаться пользователю. Таким образом, по счетчикам можно узнать, сколько пользователей покинуло сайт, просмотрев только первую страницу, сколько посетителей просматривают по две страницы, и т. д.

Cookie-файлы могут использоваться и не по прямому назначению. Теоретически, они могут отсылаться только на тот сайт, которым они были порождены, однако они могут попадать и в руки хакеров, использующих многочисленные ошибки браузеров. Поскольку некоторые сайты, посвященные электронной коммерции, указывают в cookie номера кредитных карт, нежелательное их использование может нанести серьезный ущерб.

Есть и противоположный вариант использования cookie — для незаметного сбора информации о сайтах, наиболее часто посещаемых данным клиентом. Это делается так. Рекламное агенство, скажем, «Черный Рекламщик», связывается с крупнейшими веб-сайтами и размещает на них рекламные баннеры своих корпоративных клиентов, за что сайту выплачиваются денежные взносы. Вместо того, чтобы предоставлять сайту GIF или JPEG с рекламой, ему дается URL, который следует поместить на всех страницах. Каждый из этих URL содержит уникальный идентификатор в виде имени файла, например:

http://www.sneaky.com/382674902342.gif

Когда пользователь впервые посещает страницу *P*, содержащую такую рекламу, браузер, как водится, принимает HTML-файл. Просматривая его, браузер находит ссылку на изображение на *www.sneaky.com*. Разумеется, он отправляет запрос на получение изображения. Вместе с GIF приходит cookie с уникальным идентификатором пользователя, 362723910 (см. табл. 7.9). «Черный Рекламщик» отмечает, таким образом, тот факт, что пользователь с таким идентификатором посетил страницу *P*. Это делается очень просто, так как ссылка на запрошенный файл (*382674902342.gif*) существует, на самом деле, только на странице *P*. Конечно, одна и та же реклама может располагаться на тысячах разных страниц, но каждая из них имеет свое имя файла. При этом за выпуск каждого экземпляра рекламная компания может взимать с заказчика небольшую сумму.

Затем пользователь может оказаться на другой странице, содержащей баннер от «Черного Рекламщика». Скачав HTML-файл с сервера, браузер видит ссылку на изображение с именем, скажем, http://www.sneaky.com/493654919923.gif и запрашивает его. Поскольку с домена *sneaky.com* уже был получен cookie, браузер отсылает его обратно с идентификатором пользователя. Так «Черный Рекламщик» (ЧР) узнает о том, что пользователь посетил вторую страницу с его рекламой.

Со временем ЧР может составить подробное описание пристрастий пользователя, при этом вовсе не обязательно, чтобы тот щелкал на баннерах. Конечно, остается неизвестным имя пользователя (хотя имеется IP-адрес, и этого может

оказаться достаточно для вычисления имени с помощью баз данных). Однако стоит пользователю указать свое имя на одном из сайтов, сотрудничающих с ЧР, как появляется возможность составить и продать целое веб-досье на пользователя. Продажа таких досье оказывается делом настолько прибыльным, что ЧР выгодно сотрудничать с максимально возможным количеством сайтов и собирать как можно больше информации. Самое коварное во всем этом то, что большинство пользователей даже не подозревают о том, что за ними ведется слежка, и даже считают себя в полной безопасности, поскольку никогда не щелкают ни на каких баннерах.

И если «Черный Рекламщик» хочет стать «Суперчерным Мегарекламщиком», его объявления не должны выглядеть как обычные классические баннерные ссылки. «Объявление» размером в 1 пиксел, сливающееся по цвету с задним фоном страницы (то есть невидимое), будет иметь ровно такой же эффект при слежении за пользователями: браузер будет запрашивать gif-изображение размером 1×1 пиксел и отправлять обратно cookie.

Для самоуспокоения некоторые пользователи настраивают свои браузеры так, чтобы они отвергали любые cookie. Однако это может породить проблемы при работе с «честными» сайтами, которым действительно необходимо обмениваться с пользователями cookie-маркерами. Для решения этой проблемы иногда устанавливают программы, занимающиеся поеданием cookie. Они анализируют все приходящие маркеры и принимают либо отвергают их в зависимости от выбора пользователя (например, задается список сайтов, которым можно доверять). Это дает пользователю возможность детального контроля принимаемых cookie-файлов. Современные браузеры, такие как Mozilla (www.mozilla.org), часто имеют встроенные средства пользовательского контроля cookie.

Статические веб-документы

Основная идея Всемирной паутины состоит в перемещении веб-страниц от сервера клиенту. Простейшие веб-страницы являются статическими, то есть это просто файлы, размещенные на каком-либо сервере и ожидающие востребования. В этом контексте даже видео может быть статической страницей, поскольку это всего лишь файл. В этом разделе мы подробно рассмотрим статические веб-страницы. В следующем разделе нам предстоит изучение динамического наполнения страниц.

HTML — язык разметки веб-страниц

Веб-страницы на сегодняшний день пишутся на языке **HTML** (HyperText Markup Language). С помощью HTML можно размещать на веб-страницах текст, графику, а также указатели на другие страницы. Он является языком разметки, то есть языком, описывающим способ форматирования документа. Термин «разметка» (markup) восходит к тем дням, когда технический редактор с помощью специальной разметки указывал типографу (это такой человек когда-то был), какой шрифт использовать для печати документа. Таким образом, языки разметки содержат подробные команды форматирования. Например, в языке HTML, команда **означает начало участка текста, печатаемого полужирным шрифтом, а** означает конец такого участка. Преимущество языка разметки перед языком, не имеющим явных команд форматирования, заключается в том, что браузеры для отображения страниц, написанных на этом языке, программируются довольно просто: браузер должен понимать и выполнять содержащиеся в тексте команды разметки. Среди других популярных примеров языков разметки — языки TeX и troff.

С помощью встроенных стандартизированных команд разметки в HTML-файлах становится возможным читать и переформатировать любую веб-страницу веб-браузером. Способность изменять форматирование важно, так как должна быть возможность просматривать веб-страницу, созданную на экране с установленным разрешением 1600×1200 точек при 24 битах на точку, на экране с разрешением, например, 640×320 точек при 8 битах на точку.

Далее мы приведем краткий обзор языка HTML, просто чтобы дать о нем представление. Хотя, в принципе, можно создавать HTML-документы с помощью стандартных текстовых редакторов, и многие так и делают, также есть возможность использовать специальные HTML-редакторы, берущие на себя большую часть работы (за счет снижения возможностей пользователя детально контролировать получаемый результат).

Веб-страница состоит из заголовка и тела. Вся страница размещается между командами форматирования, называемыми в языке HTML **тегами**, <html> и </html>. Впрочем, большинство браузеров правильно отобразят страницу и в отсутствие этих тегов. Как видно из рис. 7.12, *а*, заголовок веб-страницы заключен в скобки тегов <head> и </head>, а тело располагается между тегами <body> и </body>. Команды внутри тегов называют **директивами**. Большинство HTML-тегов имеют такой формат, то есть <something> помечает начало чего-либо, а </something> — его конец. Большинство браузеров предоставляют возможность просмотра исходного HTML-кода веб-страниц (пункт меню View Source или нечто подобное).

Регистр символов в тегах не имеет значения. Например, <head> и <HEAD> означают одно и то же, однако новый стандарт требует использования исключительно строчных букв. Формат самого HTML-текста, то есть расположение строк и т. д., не имеет значения. Программы обработки HTML-текстов игнорируют лишние пробелы и переносы строк, так как они все равно форматируют текст так, чтобы он помещался в заданной области отображения. Соответственно для того чтобы исходные HTML-документы легче читались, в них можно добавлять произвольное количество знаков табуляции, пробелов и символов переноса строк. И наоборот, для разделения абзацев в тексте в исходный HTML-текст недостаточно вставить пустую строку, так как она просто игнорируется браузером. В этом случае необходимо явное использование специального тега.

Некоторые теги могут иметь (именованные) параметры, называемые **атрибутами**. Например:

```
<img src="abc" alt="foobar">
```

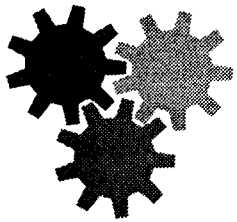
представляет собой тег <img> с атрибутом src, которому присвоено значение «abc», и атрибутом alt, которому присвоено значение «foobar». Для каждого тега стан-

дарт HTML устанавливает список допустимых атрибутов и их значение. Поскольку все атрибуты являются именованными, их порядок не имеет значения.

```
<HTML> <HEAD> <TITLE> Корпорация СООБЩЕСТВО ШТУЧЕК. </TITLE> </HEAD>
<BODY> <H1> Добро пожаловать на страницу компании СООБЩЕСТВО ШТУЧЕК. </H1>
<IMG SRC="http://www.widget.com/images/logo.gif" ALT="AWI Logo"> <BR>
Мы рады приветствовать вас на домашней странице корпорации <B> СООБЩЕСТВО ШТУЧЕК</B>
Мы надеемся, что <I> вы </I> найдете здесь всю необходимую вам информацию.
<P>Ниже приведены ссылки на информацию о нашей замечательной продукции.
Вы можете сделать заказ по Интернету, по телефону или по факсу. <HR>
<H2> Информация о продукции </H2>
<UL> <LI> <A HREF="http://widget.com/products/big"> Большущие штуковины </A>
<LI> <A HREF="http://widget.com/products/little"> Малюсенькие штучки </A>
</UL>
<H2> Номера телефонов </H2>
<UL> <LI> телефон: 1-800-WIDGETS
<LI> факс: 1-415-765-4321
</UL> </BODY> </HTML>
```

а

Добро пожаловать на страницу компании СООБЩЕСТВО ШТУЧЕК



Мы рады приветствовать вас на домашней странице корпорации СООБЩЕСТВО ШТУЧЕК.
Мы надеемся, что вы найдете здесь всю необходимую вам информацию.

Ниже приведены ссылки на информацию о нашей замечательной продукции.
Вы можете сделать заказ по Интернету, по телефону или по факсу.

Информация о продукции
- Огромные штуковины
- Малюсенькие штучки

Номера телефонов
- 1-800-WIDGETS
- 1-415-765-4321

б

Рис. 7.12. Пример веб-страницы на HTML (*а*); форматированная страница (*б*)

Формально при написании HTML-документов должен использоваться набор символов Latin-1 международного стандарта ISO 8859-1, но для пользователей, чьи клавиатуры поддерживают только ASCII-символы, для ввода специальных символов, таких как, например, é, могут использоваться специальные управляющие последовательности символов. Эти последовательности должны начинаться со знака амперсанда и заканчиваться точкой с запятой. Например, `&egrave;` означает символ è, а `&eacute;` — символ é. Так как сами символы <, > и & оказываются зарезервированными, для их отображения в тексте также применяются управляющие последовательности: `<` (less than — знак «меньше»), `>` (greater than — знак «больше») и `&` (ampersand — амперсанд).

Главным пунктом заголовка является название страницы, располагающееся между тегами <title> и </title>. В него можно поместить также некоторую мета-информацию. Оно не отображается на странице, а используется некоторыми браузерами для того, чтобы помечать окно страницы.

Рассмотрим некоторые другие команды языка HTML, используемые в примере на рис. 7.12, и другие, приведенные в табл. 7.11. Заголовки в примере задаются тегами вида <h*n*>, где *n* — цифра от 1 до 6. <h1> является самым важным заголовком, <h6> — наименее важным. Как это отобразить на экране, зависит от браузера. Обычно заголовки с меньшими номерами отображаются более крупными шрифтами. Браузер может также выделять различные заголовки различными цветами. Обычно заголовки <h1> выводятся на экран крупным полужирным шрифтом и выделяются, по меньшей мере, одной пустой строкой над и под заголовком.

Таблица 7.11. Наиболее часто используемые HTML-теги. У некоторых из них могут быть дополнительные параметры

Тег	Описание
<html> ... </html>	Объявляет веб-страницу на языке HTML
<head> ... </head>	Определяет границы заголовка страницы
<title> ... </title>	Определяет границы неотображаемого названия страницы
<body> ... </body>	Определяет границы тела страницы
<h*n*> ... </h*n*>	Определяет границы заголовка уровня n
...	Маркирует блок текста, печатаемого полужирным шрифтом
...	Маркирует блок текста, печатаемого курсивом
...	Помечает начало и конец центрированного по горизонтали текста
...	Помечает начало и конец неупорядоченного списка
...	Помечает начало и конец упорядоченного списка
<menu> ... </menu>	Помечает границы меню
...	Маркирует начало и конец пункта меню
 	Разрыв (перевод строки)
<p>	Начало абзаца
<hr>	Горизонтальная линейка
<img src="...">	Загрузка изображения
 ... 	Определяет гиперссылку

Теги и обозначают, соответственно, полужирный шрифт (boldface) и курсив (italics). Если браузер не может отобразить полужирный шрифт или курсив, он должен применить какой-нибудь другой способ выделения, например, использовать другой цвет или инверсное отображение символов.

Язык HTML предоставляет несколько механизмов создания списков, включая вложенные списки. Тег <ul> (unordered list) начинает неупорядоченный спи-

сок. Отдельные пункты, помеченные исходном тексте тегом <li>, изображаются с маркером абзаца (обычно крупной черной точкой, •) перед собой. Тег <ol> (ordered list) означает начало упорядоченного списка. При его использовании абзацы, помеченные тегом <li>, автоматически нумеруются браузером. У списков, организованных при помощи тегов <ul> и <ol>, одинаковый синтаксис (за исключением открывающих и закрывающих тегов списков) и похожее поведение.

Теги <br>, и <hr> применяются для обозначения границ между различными участками текста. Точный формат может быть определен в *таблице стилей* (см. ниже), ассоциированной со страницей. Тег
 просто вставляет разрыв строки. Обычно браузеры не вставляют пустую строку после тега <br>. Тег , напротив, начинает новый абзац, перед которым может быть вставлена пустая строка и, возможно, добавлен отступ. (Тег отмечающий конец абзаца, существует, но на практике почти не используется. Большинство составителей HTML-страниц даже не знают о его существовании.) Наконец, тег <hr> прерывает строку и рисует на экране горизонтальную линию.

Язык HTML позволяет включать в веб-страницу изображения. Тег <img> указывает, что в данной позиции страницы должно быть загружено изображение. У этого тега может быть несколько параметров. Параметр src задает URL изображения. Стандартом HTML не определяются графические форматы. На практике все браузеры поддерживают файлы форматов GIF и JPEG. Браузеры могут поддерживать любые другие форматы, но эта свобода оказывается палкой о двух концах. Если пользователь привыкнет к браузеру, поддерживающему, скажем, формат файлов BMP, он может включить их в свои веб-страницы, а затем обнаружить, что остальные браузеры просто игнорируют всю его замечательную работу.

У тега <img> может быть еще несколько параметров. Параметр align управляет выравниванием изображения относительно текста. Он может принимать значения *top* (верх), *middle* (центр), *bottom* (низ). Параметр alt предоставляет текст, отображаемый вместо изображения, если пользователь запретил вывод изображений. Параметр ismap является флагом, указывающим, является ли данное изображение активной картой.

И наконец, мы подошли к гиперссылкам, использующим пару тегов <a> (anchor — якорь) и . У этого тега также могут быть различные параметры, из которых следует отметить href (гиперссылка, URL) и name (имя гиперссылки). Текст, располагающийся между тегами <a> и , отображается на экране. Если этот текст выбирается, браузер открывает страницу, на которую указывает гиперссылка. Между тегами <a> и можно также размещать изображение (тег <img>). В этом случае, если пользователь щелкнет на изображении, будет произведен переход по ссылке.

Для примера рассмотрим следующий фрагмент HTML-текста:

```
<a href=http://www.nasa.gov> Домашняя страница NASA </a>
```

При отображении страницы с этим фрагментом на экране появляется следующая строка:

Домашняя страница NASA

Если затем пользователь щелкнет мышью на этом тексте, браузер обратится по сети к указанному URL (http://www.nasa.gov) и попытается получить там веб-страницу и отобразить ее на экране.

В качестве второго примера рассмотрим следующую строку:

```
<a href=http://www.nasa.gov> <img src="shuttle.gif" alt=NASA"> </A>
```

При отображении этой страницы должно быть видно изображение (челночный воздушно-космический аппарат). Щелчок мышью на этом изображении будет иметь тот же результат, что и щелчок на подчеркнутом тексте в предыдущем примере. Если пользователь запретит автоматическое отображение изображений, вместо него будет показан текст «NASA».

Тег <a> может содержать параметр name, что позволяет создать гиперссылку посреди текста, на которую можно ссылаться из другого места этой же страницы. Например, некоторые веб-страницы начинаются с оглавления, состоящего из «локальных» гиперссылок. Щелчок мышью на пункте оглавления позволяет быстро переместиться в нужное место страницы.

HTML продолжает развиваться. В первых двух версиях не существовало таблиц, они были добавлены только в HTML 3.0. HTML-таблица состоит из нескольких **строк**, каждая из которых состоит из нескольких **ячеек**. Ячейка может содержать широкий спектр данных, включая текст, изображения и даже другие таблицы. Ячейки могут объединяться вместе, например, заголовок таблицы может охватывать несколько столбцов. Контроль составителей страниц над внешним видом таблиц ограничен. Последнее слово в таких вопросах, как выравнивание, стили рамок и границы ячеек, остается за браузерами.

Реализация таблицы на языке HTML показана в листинге 7.5, а ее возможное отображение браузером — на рис. 7.13. Этот пример демонстрирует несколько основных возможностей HTML-таблиц. Таблицы начинаются с тега <table>. Для описания основных свойств таблицы может быть предоставлена дополнительная информация.

Листинг 7.5. HTML-таблица

```
<html>
<head> <title> Пример страницы с таблицей </title> </head>
<body>
<table border=all rules=all>
<caption> Некоторые различия html версий </caption>
<col align=left>
<col align=center>
<col align=center>
<col align=center>
<tr> <th> Аспект <th> HTML 1.0 <th>HTML 2.0 <th> HTML 3.0 <th>HTML 4.0</tr>
<tr> <th> Гиперссылки <td> x <td> x <td> x<td> x </tr>
<tr> <th> Изображения <td> x <td> x <td> x<td> x</tr>
<tr> <th> Списки <td> x <td> x <td> x<td> x</tr>
<tr> <th> Активные карты и изображения <td> <td>x<td> x<td> x</tr>
<tr> <th> Формы <td> <td> x <td> x <td> x</tr>
<tr> <th> Математические выражения <td> <td> <td> x <td> x </tr>
<tr> <th> Панели инструментов <td> <td> <td> x <td> x </tr>
```

```
<tr> <th> Таблицы <td> <td> <td> x <td> x </tr>
<tr> <th> Доступность <td> <td> <td> <td> x </tr>
<tr> <th> Внедренные объекты <td> <td> <td> <td> x </tr>
<tr> <th> Скрипты <td> <td> <td> <td> x </tr>
</table>
</body>
</html>
```

Некоторые различия версий HTML

Аспект	HTML 1.0	HTML 2.0	HTML 3.0	HTML 4.0
Гиперссылки	X	X	X	X
Изображения	X	X	X	X
Списки	X	X	X	X
Активные карты и изображения		X	X	X
Формы		X	X	X
Математические выражения			X	X
Панели инструментов			X	X
Таблицы			X	X
Доступность				X
Внедренные объекты				X
Скрипты				X

Рис. 7.13. Возможное отображение таблицы браузером

С помощью тега <caption> можно задать надпись над таблицей. Каждая строка таблицы начинается с тега <tr> (table row — строка таблицы). Отдельные ячейки помечаются тегом <th> (table header — заголовок таблицы) или <td> (table data — данные таблицы). Эти два тега позволяют создавать, соответственно, строки заголовков таблицы и обычные строки, что и показано в примере.

В таблицах также могут использоваться различные другие теги. С их помощью можно устанавливать параметры выравнивания содержимого ячеек по вертикали и горизонтали, задавать параметры границ ячеек, объединять ячейки в группы и многое другое.

HTML 4.0 отличается от предыдущих версий некоторыми новыми свойствами. Они включают в себя специальные методы доступа для людей с ограниченными возможностями, внедрение объектов (обобщение тега <img>, позволившее включать в состав страниц не только изображения, но и другие объекты), поддержку языков написания сценариев (скриптов), что дало толчок к развитию динамических страниц, и т. д.

Если веб-сайт достаточно велик по размерам и сложен настолько, что над ним работает целая группа программистов, желательно обеспечить более или менее схожий дизайн всех страниц. Эта проблема решается при помощи **таблиц стилей**. При их использовании отдельные страницы оформляются не реальными (физическими) стилями (например, курсивом или полужирным шрифтом), а логическими. Среди них могут быть, например, <dn> (Определение), (Слабое выделение), (Сильное выделение), <var> (Программная переменная). Логические стили определяются в таблицах стилей, ссылка на которые ставится в начале кода каждой страницы. Таким образом можно придать всем страницам единый вид. Если веб-мастер решит изменить стиль и отображать его полужирным шрифтом величиной 18 пунктов радикального розового цвета вместо курсива величиной 14 пунктов синего цвета, ему необходимо будет лишь поменять одно определение в таблице стилей, чтобы изменения распространились на весь сайт. Таблицы стилей можно сравнить с файлами в языке C, включаемыми с помощью директивы #include, — там также изменение одного макроопределения влечет за собой изменение во всех программных файлах, использующих данный заголовочный файл.

Формы

Первая версия языка HTML фактически обеспечивала лишь одностороннюю связь. Пользователи могли получать страницы от поставщиков информации, но отправлять информацию обратно было довольно трудно. По мере того, как все больше коммерческих организаций начали использовать Всемирную паутину, потребность в двустороннем трафике стала возрастать. Так, многим компаниям потребовалась возможность принимать заказы на продукцию с помощью своих веб-страниц. Производители программного обеспечения хотели продавать по сети свои программы, для чего требовалась возможность заполнения регистрационных карточек. Компаниям, предоставляющим поиск информации в Паутине, было нужно, чтобы пользователи могли ввести ключевые слова поиска.

В результате в HTML 2.0 были включены **формы**. Формы могут содержать кнопки и поля для ввода текста, позволяющие пользователям делать выбор или вводить необходимую информацию, которую затем можно отсылать владельцу страницы. Для этой цели используется тег <input>. У него могут быть различные параметры, определяющие размер, назначение и другие свойства отображаемого окна. Наиболее часто используемыми формами являются пустые поля для ввода текста, флажки, переключатели, активные карты и кнопки подтверждения. В примере, приведенном в листинге 7.6, показаны некоторые из перечисленных форм. Отображение этой HTML-страницы браузером показано на рис. 7.14.

Листинг 7.6. HTML-текст для бланка заказа

```
<html>
<head> <title> Бланк заказа клиента awi </title> </head>
<body>
<h1> Бланк заказа штуковины </h1>
<form action="http://www.widget.com/cgi-bin/widgetorder" method=post>
Имя <input name="customer" size=60> <p>
Адрес <input name="address" size=58> <p>
Город <input name="city" size=21>
Штат <input name="state" size=4>
Страна <input name="country" size=10> <p>
№ кредитной карты <input name="cardno" size=10>
Срок действия <input name="expires" size=4>
m/c <input name="cc" type=radio value="mastercard">
visa <input name="cc" type=radio value="visacard"> <p>
Размер штуковины:  большой <input name="product" type=radio value="дорогая">
маленький <input name="product" type=radio value="дешевая">
```

```
Доставка  <input name="express" type=checkbox> <p>
<input type=submit value="отправить заказ"> <p>
Благодарим вас за то, что вы заказали у нас штуковины фирмы awi. Это правильный выбор!
</form>
</body>
</html>
```

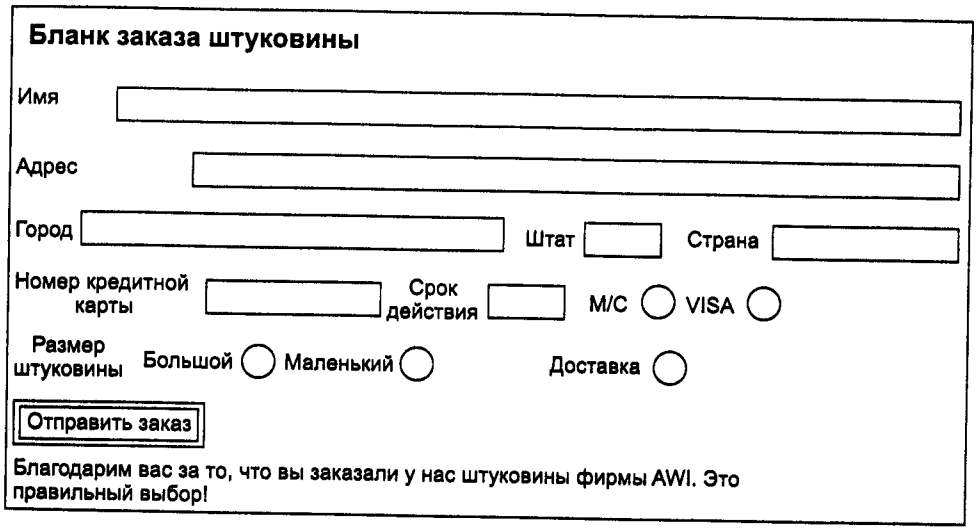


Рис. 7.14. Форматированная страница

Начнем изучение форм с этого примера. Как и все формы, она заключена между тегами <form> и </form>. Текст, не заключенный в теги, просто отображается. Внутри формы разрешено использование всех обычных тегов (например, <b>). В данной форме используются три типа окон для ввода данных.

Окно первого типа следует за текстом «Имя». Ширина этого окна 46 символов. Предполагается, что пользователь введет здесь свое имя, которое будет храниться в виде текстовой строки в переменной customer для последующей обработки. Тег <p> указывает браузеру, что последующий текст следует отображать с новой строки, даже если в текущей строке еще достаточно места. С помощью этого и других тегов автор страницы может управлять внешним видом бланка на экране.

В следующих окнах формы спрашивается адрес заказчика, то есть улица, город, штат и страна. Между этими полями не вставляются теги <p>, поэтому браузер по возможности пытается отобразить их все в одной строке. С точки зрения браузера, этот абзац представляет собой просто шесть отдельных элементов — три строки, перемежающиеся тремя окнами. Он отображает их друг за другом, слева направо, переходя на новую строку по мере необходимости. Таким образом, вполне возможно, что на экране с разрешением 1600×1200 точек все три строки и соответствующие им окна поместятся в одну строку, тогда как на экране с разрешением 1024×768 точек они будут разбиты на две строки. В худшем случае слово «Страна» может оказаться в конце одной строки, а соответствующее окно ввода — в начале следующей строки.

В следующей строке у пользователя запрашивается номер кредитной карты и срок ее действия. Передавать номера кредитных карт по Интернету следует только в том случае, если приняты все соответствующие меры предосторожности. Более подробно этот аспект будет обсуждаться в главе 8.

Следом за датой истечения срока действия кредитной карты мы обнаруживаем новые для нас элементы управления — переключатели. Они используются, когда требуется выбрать только один вариант из нескольких. Они напоминают кнопки на автомагнитолах, служащие для быстрого доступа к заданным радиостанциям. Браузер отображает эти элементы управления таким образом, что пользователь может включать их щелчком мыши на них (или с помощью клавиатуры). Переключатели этого типа всегда объединяются в группы. При этом включение одного переключателя автоматически выключает все остальные переключатели этой группы. Внешний вид переключателя зависит от используемого графического интерфейса. В пункте бланка «Размер штуковины» также используются два переключателя. Группы переключателей определяются по значению параметра name (имя) тега <input>. Специальных скобок из тегов вроде <radiobutton> ... </radiobutton> для определения групп переключателей не предусмотрено.

Параметр value указывает, какая кнопка была нажата пользователем. В зависимости от того, какую кредитную карту выберет пользователь для своих расчетов, переменной cc будет присвоено значение текстовой строки «mastercard» или «visacard».

Следом за двумя наборами переключателей в бланке используется элемент управления типа checkbox (флажок). Он похож на переключатель предыдущего типа, так как тоже может находиться в одном из двух состояний (установлен/сброшен), но не объединяется в группы и включается и выключается щелчком мыши на нем независимо от других элементов управления того же типа. Например, заказывая пиццу на веб-странице компании Electropizza, пользователь может выбрать *и* сардины, *и* лук, *и* ананас (если у него крепкий желудок), но ему не удастся одновременно выбрать маленькую, среднюю и большую пиццу. Выбор приправы к пицце будет представлен в виде трех отдельных элементов управления типа checkbox, тогда как выбор размера пиццы будет реализован с помощью набора переключателей.

Если список, из которого предстоит сделать выбор пользователю, очень длинный, то использование переключателей несколько неудобно. В этом случае можно использовать теги <select> и </select> для определения списка альтернатив. При этом семантика переключателей сохраняется, если только не используется параметр multiple, превращающий список альтернатив в набор независимо устанавливаемых и сбрасываемых флажков. Некоторые браузеры отображают пункты списка между тегами <select> и </select> как всплывающее меню.

Итак, мы рассмотрели два встроенных типа тега <input>: radio и checkbox. На самом деле, мы уже познакомились и с третьим типом этого тега, то есть с типом text. Мы не заметили этого, потому что этот тип является типом по умолчанию,

поэтому параметр type = text можно не указывать. Еще два возможных значения параметра type: password и textarea. Окно password аналогично окну text, с той разницей, что при вводе текста в окне password символы не отображаются на экране. Окно textarea отличается от окна text тем, что может содержать несколько строк текста.

Возвращаясь к примеру на рис. 7.14, мы, наконец, добираемся до последнего использованного в этом примере элемента управления — кнопки submit (подтверждение). Когда пользователь щелкает мышью на этой кнопке, заполненный им бланк отсылается обратно на компьютер, на котором размещена эта веб-страница. Как и все остальные рассмотренные типы, submit является зарезервированным словом, интерпретируемым браузером. Строка параметра value (значение) в данном случае содержит надпись на кнопке. В принципе, все элементы управления, образуемые с помощью тега <input>, могут иметь параметр value. В окнах ввода текста содержимое параметра value отображается в окне редактирования, и пользователь может редактировать или удалить его. Элементы управления checkbox и radio также могут быть инициализированы с помощью специального служебного слова checked («выбрано». Дело в том, что value просто отображает текст, но не отображает предпочитаемый выбор.).

Когда пользователь нажимает кнопку submit, браузер упаковывает всю собранную информацию в одну большую строку и отправляет ее на сервер для обработки. Поля с данными разделяются амперсандами (&), а вместо пробелов ставятся знаки +. В нашем примере такая строка, отсылаемая на сервер, может выглядеть так, как показано ниже (вы видите две строки, а не одну, только из-за недостаточной ширины бумажного листа):

```
customer=John+Doe&address=100+Main+St.&city=White+Plains&state=NY&country=USA&cardno=1
234567890&expires=6/98&cc=mastercard&product=cheap&express=on
```

Это сообщение отправляется на сервер в виде одной текстовой строки. Если флажок элемента управления checkbox сброшен, соответствующая ему переменная опускается. Сервер сам решает, что ему делать с полученной строкой. Мы обсудим это позднее.

XML и XSL

Язык HTML — с формами или без оных — никак не определяет структуру веб-страниц. Он смешивает содержимое страницы и описание средств ее форматирования. По мере роста популярности электронной коммерции и других приложений появлялась все более очевидная необходимость в структурировании веб-страниц и отделении содержимого от форматирования. Например, поисковая программа, обещающая найти в Мировой паутине книгу или компакт-диск по самой выгодной цене, должна проанализировать множество страниц, находя нужное наименование и цену. Если страница написана на обычном HTML, такой программе будет очень тяжело определить, где указано название товара, а где — его цена.

По этой причине консорциум WWW (W3C) предложил расширение HTML, позволяющее структурировать страницы для облегчения их автоматической обработки. Для целей было создано два языка. Первый, **XML** (eXtensible Markup Language — Расширяемый язык разметки веб-страниц), описывает структурированное содержимое страниц, а второй, **XSL** (eXtensible Style Language — расширяемый язык стилей), описывает форматирование независимо от содержимого. И о том, и о другом можно говорить очень долго, поэтому нам приходится ограничиться лишь поверхностным описанием идей, лежащих в основе этих языков.

Рассмотрим документ XML, представленный в листинге 7.7. В нем определяется структура book_list, представляющая собой список книг. Под каждую книгу отведено три поля: название, автор и год издания. Эти структуры чрезвычайно просты. Разрешается иметь структуры с повторяющимися полями (например, несколько полей с именами авторов), необязательными полями (например, название прилагающегося компакт-диска), а также альтернативные поля (например, URL магазина, если книга еще есть в продаже, и URL аукциона, если весь тираж уже распродан).

В приведенном примере каждое поле является неделимой сущностью, однако разрешается разделять поля на подполя. Например, поле, содержащее имя автора, может быть — для улучшения возможностей поиска и форматирования — организовано следующим образом:

```
<author>
<first_name> Эндрю </first_name>
<last_name> Таненбаум </last_name>
</author>
```

Итак, любое поле может иметь подполя неограниченной вложенности.

Код, представленный в листинге 7.7, делает лишь одно: определяет список из трех книг. Ничего не говорится о том, как должна выглядеть веб-страница на экране. Информация о форматировании страницы берется из другого файла, book_list.xsl, содержащего определения XSL. Реально данный файл представляет собой таблицу стилей, в которой оговаривается вид страницы. (Существуют и альтернативы таблицам стилей, позволяющие, например, преобразовывать XML в HTML, однако обсуждение этой темы выходит за рамки этой книги.)

Листинг 7.7. Простой пример на XML

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="book_list.xsl"?>

<book_list>
<book>
<title> Компьютерные сети, 4 изд. </title>
<author> Эндрю С. Таненбаум </author>
<year> 2003 </year>
</book>

<book>
<title> Современные операционные системы, 2 изд. </title>
<author> Эндрю С. Таненбаум </author>
<year> 2001 </year>
</book>

<book>
```

```
<title> Архитектура компьютера, 4 изд. </title>
<author> Эндрю С. Таненбаум </author>
<year> 1999 </year>
</book>

</book_list>
```

Пример XSL-файла для форматирования страницы из листинга 7.7 приведен в листинге 7.8. За некоторыми необходимыми объявлениями, включающими, например, URL используемого стандарта XSL, следуют теги, первыми из которых являются <html> и <body>. С этого начинается любая обычная веб-страница. Затем следует определение таблицы, включающее заголовки трех столбцов. Обратите внимание на то, что в дополнение к тегам <th> поставлены закрывающие теги </th>. Раньше нам было все равно, есть они или нет. Однако спецификации XML и XSL куда строже, чем HTML. Оговаривается, что страницы с синтаксическими ошибками должны отвергаться браузерами в любом случае, даже если они в состоянии понять, что имел в виду разработчик страницы. Браузер, отображающий синтаксически некорректный код XML или XSL, будет сам по себе признан некорректным при первом же тестировании на совместимость со стандартами. Однако браузерам разрешается выявлять ошибочные места. Такие драконовские меры нужны для борьбы с несметным числом небрежно написанных страниц, которые появились в Сети за последние годы.

Листинг 7.8. Таблица стилей на XSL

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl=http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">

<html>
<body>

<table border="2">
<tr>
<th>Название</th>
<th>Автор</th>
<th>Год</th>
</tr>

<xsl:for-each select="book_list/book">
<tr>
<td><xsl:value-of select="title"/> </td>
<td><xsl:value-of select="author"/> </td>
<td><xsl:value-of select="year"/> </td>
</tr>
</xsl:for-each>
</table>

</body>
</html>
</xsl:template>
</xsl:stylesheet>
```


Выражение

```
<xsl:for-each select="book_list/book">
```

аналогично подобному выражению на языке С. С его помощью запускается цикл (ограниченный тегами <xsl:for-each>). На каждую книгу приходится одна итерация этого цикла. И каждая итерация выдает пять строк: <tr>, название, автор, год и тег </tr>. По окончании цикла выводятся закрывающие теги </body> и </html>. Результат интерпретации браузером этой таблицы стилей такой же, как если бы это была обычная страница, содержащая таблицу. Однако благодаря такому формату анализирующая программа сможет по XML-файлу легко найти, например, книги, изданные после 2000 года. Надо отметить, что, хотя наш XSL-файл сдержит нечто вроде цикла, веб-страницы на XML и XSL все равно остаются статическими, поскольку они содержат лишь инструкции, указывающие браузеру, как отображать страницу. Тем же, в принципе, занимается и HTML. Разумеется, чтобы интерпретировать XML и XSL, браузер должен поддерживать эти языки. На сегодняшний день, впрочем, большинство браузеров имеют такую возможность. До сих пор не очень понятно, заменит ли XSL традиционные таблицы стилей.

Мы не показали этого в нашем примере, но XML позволяет разработчику веб-страницы определять структуры заранее в специальном файле. Такие файлы определений затем можно подключать для построения сложных страниц. Дополнительную информацию, касающуюся этого и многих других свойств XML и XSL, можно найти в любой из многочисленных книг, посвященных этой теме. Например, в (Livingston, 2000; Williamson, 2001).

Перед тем как закончить наш краткий рассказ об XML и XSL, будет нелишним прокомментировать идеологическую борьбу между консорциумом WWW и сообществом веб-дизайнеров. Изначальной целью HTML было определение именно *структуры* документа, а вовсе не его *внешнего вида*. Например, строка

```
<h1>Фотографии Натальи</h1>
```

сообщает браузеру о том, что следует выделить заголовок, однако ничего не говорит о его гарнитуре, размере или цвете. Все эти детали реализует браузер по своему усмотрению: у него есть преимущество, состоящее в том, что он знает свойства конкретного монитора (например, сколько на нем точек). Однако дизайнерам веб-страниц в какой-то момент захотелось получить тотальный контроль над видом создаваемых страниц. Тогда были добавлены новые теги, уточняющие, как должен выглядеть документ. Например,

```
<font face="Helvetica" size="24" color="red"> Фотографии Натальи </font>
```

Были добавлены также методы точного позиционирования элементов на экране. Проблема, присущая такому подходу, заключается в том, что такие страницы не обладают свойством переносимости. Они могут замечательно смотреться на браузере у создателя, однако на другом браузере, другой версии того же браузера или просто на экране с другим разрешением могут выглядеть совершеннейшей кашей. Одна из задач XML состояла в попытке вернуться к истокам, когда определялась только структура, а не внешний вид документа. Вместе с тем, XSL

позволяет управлять тем, как выглядят страницы. Оба языка, впрочем, порой используются не по назначению. Следует иметь это в виду.

XML можно использовать не только для описания веб-страниц. Все чаще он используется в качестве языка для связи между прикладными программами. В частности, **SOAP** (Simple Object Access Protocol — простой протокол доступа к объектам) предоставляет возможность выполнения удаленных вызовов процедур между приложениями способом, независимым от языка и системы. Клиент формирует запрос в виде сообщения XML и отсылает его на сервер по описываемому далее протоколу HTML. Сервер отсылает назад ответ, представляющий собой форматированное XML-сообщение. Таким образом могут общаться приложения, работающие на разнородных платформах.

XHTML — расширенный язык разметки гипертекста

К языку HTML постоянно предъявляются новые требования. Многие представители этой индустрии чувствуют, что в будущем большинство устройств, связанных со Всемирной паутиной, будут представлять собой не ПК, а беспроводные портативные устройства типа PDA. У таких мини-компьютеров нет столь большого объема памяти, чтобы работать с большими браузерами, обладающими сложной эвристикой, с помощью которой они пытаются отображать синтаксически некорректные страницы. Таким образом, следующей версией после HTML 4 должен стать язык, отличающийся крайне высокой требовательностью. Он называется не HTML 5, а **XHTML**, поскольку, по сути дела, представляет собой HTML 4, приведенный к стандарту XML. Под этим мы подразумеваем, что теги типа <h1> не имеют существенного значения. Чтобы добиться от такого тега того эффекта, который он производит в HTML 4, необходимо написать определение на XSL в отдельном файле. XHTML — это новый веб-стандарт, который рекомендуется использовать при создании любых веб-страниц для достижения максимальной переносимости на другие платформы и независимости отображения от браузера.

Между XHTML и HTML 4 существует шесть существенных и множество мелких различий. Во-первых, страницы и браузеры стандарта XHTML должны работать в строгом соответствии со стандартом. Низкопробные страницы уже отжили свой век. Это свойство унаследовано из XML.

Во-вторых, все теги и атрибуты должны быть написаны строчными буквами. Так, тег <HTML> будет считаться некорректным в XHTML. Необходимо писать <html>. Аналогично, некорректной записью считается такая: <img SRC="pic001.jpg"> Она содержит имя атрибута, написанное заглавными буквами, а это запрещено.

В-третьих, всегда должны присутствовать закрывающие теги, даже для </p>. Если у тега не может быть естественного закрывающего тега (например, <br>, <hr>, <img>), то перед закрывающей скобкой тега следует ставить косую черту. Например

```
<img src="pic001.jpg" />
```

В-четвертых, все значения атрибутов должны указываться в двойных кавычках. Вот пример неправильного использования тега:

```
<img src=pic001.jpg height=500 />
```


Число 500 должно быть заключено в двойные кавычки, как и имя JPEG-файла.

В-пятых, свойство вложенности тегов должно использоваться корректно. В прошлом это не требовалось, важно было только получить ожидаемый результат. Раньше вполне легальным было написать:

```
<center> <b> Летние фотографии </center> </b>
```

В XHTML это запрещено. Закрывающие теги должны быть написаны строго в обратном порядке по отношению к открывающим тегам.

В-шестых, в каждом документе должен быть указан его тип. Мы имели возможность в этом убедиться на примере листинга 7.8. Все серьезные и мелкие изменения, которые происходят в стандартах, обсуждаются на сайте www.w3.org.

Динамические веб-документы

Все идеи, рассматривавшиеся до сих пор, соответствуют модели, показанной в листинге 6.1: клиент сообщает серверу имя файла, а тот в ответ возвращает файл. В первые годы существования Всемирной паутины все ее содержимое и в самом деле было статическим (просто файлы). Однако в последние годы в Сети появляется все больше динамических объектов, то есть таких, которые создаются по требованию, а не хранятся постоянно на диске. Автоматическое создание объектов может происходить как на стороне сервера, так и на стороне клиента. Рассмотрим оба случая по порядку.

Динамическая генерация содержимого веб-страниц на стороне сервера

Чтобы понять, зачем вообще нужна динамическая генерация веб-страниц на стороне сервера, рассмотрим использование форм, описанных ранее. Когда пользователь заполняет поля формы и нажимает кнопку *Submit* (Подтверждение), серверу отправляется сообщение, содержащее в себе данные, предоставленные пользователем. Это сообщение не содержит имя запрашиваемого файла. Требуется, чтобы оно было передано для обработки программе или скрипту. Обычно обработка подразумевает использование пользовательских данных для поиска по базе данных на серверном диске и создание HTML-страницы, содержимое которой зависит от результатов этого поиска. Затем страница отсылается клиенту. Например, в приложении для электронной коммерции нажатие кнопки *ПЕРЕЙТИ К РАСЧЕТАМ* приводит к тому, что браузер возвращает на сервер cookie-файл с содержимым корзины клиента. На сервере при этом должны запуститься определенные программы или скрипты, в задачу которых входят обработка cookie и генерация HTML-страницы. Отправляемая клиенту HTML-страница может содержать форму со списком товаров, положенных в корзину, и адрес доставки вместе с запросом подтверждения заказа и предложением выбрать одну из возможных форм оплаты. Этапы обработки информации, полученной из HTML-формы, показаны на рис. 7.15.

Традиционный способ работы с формами и другими видами интерактивных веб-страниц связан с использованием системы **CGI** (Common Gateway Interface —

общий шлюзовой интерфейс). Это стандартизованный интерфейс, позволяющий веб-серверам общаться с прикладными программами и скриптами, разрешающими вводить данные (например, в формы) и в ответ генерировать HTML-страницы. Обычно такие прикладные программы представляют собой скрипты, написанные на языке описания сценариев Perl, поскольку писать их проще и быстрее, чем программы (по крайней мере, если вы умеете программировать на Perl). Существует договоренность, в соответствии с которой эти скрипты должны размещаться в каталоге CGI-BIN, который доступен при помощи URL. Иногда вместо Perl используется другой язык написания скриптов, Python.

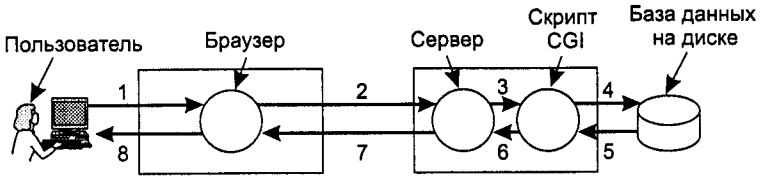


Рис. 7.15. Этапы обработки информации, полученной из формы

В качестве примера работы CGI рассмотрим случай, когда продукция компании «Великие Штуковины» приходит без гарантийного талона. Вместо этого клиенту предлагается зарегистрироваться в Интернете по адресу www.grwd.com. Там имеется ссылка:

Щелкните здесь для регистрации приобретения

Ссылка эта может указывать, например, на сценарий на языке Perl, расположенный по адресу www.grwd.com/cgi-bin/reg.perl. При запуске этого сценария без параметров обратно отсылается HTML-страница, содержащая регистрационную форму. Когда пользователь заполняет ее и нажимает кнопку *Submit*, скрипту передается сообщение, содержащее указанные им значения. Вид этого сообщения традиционен при работе с формами. Что происходит дальше, предугадать нетрудно. Perl-скрипт анализирует параметры, создает в базе данных запись о новом клиенте, отсылает назад HTML с регистрационным номером и телефоном службы поддержки. Понятно, что это не единственный способ обработки форм, однако он является наиболее распространенным. О создании CGI-скриптов и программировании на Perl написано много книг. Среди них стоит отметить (Hanegan, 2001; Lash, 2002; Meltzer и Michalski, 2001).

Динамическое создание веб-страниц на стороне сервера может быть реализовано не только с помощью CGI-скриптов. Существует еще один распространенный способ, который заключается во внедрении небольших скриптов в HTML-страницы. Они выполняются на сервере, в их задачу входит генерирование страницы. Популярным инструментом для написания таких скриптов является **PHP** (Hypertext Preprocessor — гипертекстовый препроцессор). При его использовании требуется, чтобы сервер понимал PHP (точно так же, как браузер должен понимать XML, чтобы интерпретировать страницы, написанные на одноименном языке). Обычно серверы предполагают, что у файлов страниц, написанных на PHP, расширение php, а не htm или html.

В листинге 7.9 приведен пример маленького скрипта на PHP; он должен работать на любом сервере, если на нем установлен гипертекстовый препроцессор. Он состоит из пр. вычного обрамления на HTML, а сам скрипт содержится внутри тега <?php ... ?>. Вся его работа заключается в создании страницы, сообщающей всю известную информацию о запустившем его браузере. Браузеры обычно отправляют кое-какие данные о себе вместе с запросами (и любыми прикладными cookie-файлами). Эти данные сохраняются в переменной HTTP_USER_AGENT. Если этот листинг сохранить в файле test.php в веб-каталоге компании ABCD, то пользователь, набрав URL www.abcd.com/test.php, сможет получить страницу, из которой он узнает, какие браузер, язык и операционную систему он использует.

Листинг 7.9. Пример страницы на HTML с внедренным PHP-скриптом

```
<html>
<body>

<h2>А я вот что про тебя знаю:</h2>
<?php echo $HTTP_USER_AGENT ?>

</body>
</html>
```

PHP особенно хорошо подходит для обработки форм — с его помощью она осуществляется даже проще, чем путем написания CGI-скрипта. Пример обработки формы показан в листинге 7.10, *а*. Здесь мы видим обычную HTML-страницу с формой. Непривычно выглядит только первая строка, в которой указывается, что скрипт action.php должен быть запущен для обработки параметров после заполнения пользователем формы и нажатия кнопки подтверждения. Форма в этом примере состоит из двух текстовых полей ввода, в одном из которых запрашивается имя клиента, а в другом — его возраст. По окончании работы пользователя с формой на сервер отсылается стандартная строка, пример которой мы уже видели ранее. Эта строка обрабатывается, из нее извлекаются значения переменных name и age. Затем начинает свою работу скрипт action.php, показанный в листинге 7.10, *б*. Он генерирует ответ. Работа скрипта заключается в исполнении php-команд. Если пользователь предоставил данные «Харриет» и «24», то ему будет прислан HTML-файл, код которого показан в листинге 7.10, *в*. Как видите, обработка форм с помощью PHP производится элементарно.

Несмотря на простоту использования, PHP — это мощный язык программирования, ориентированный на предоставление интерфейса между Всемирной паутиной и серверными базами данных. В PHP есть переменные, строки, массивы и большинство управляющих структур, присущих языку C, однако ввод-вывод гораздо мощнее, чем обычный printf. PHP имеет открытый исходный код и распространяется бесплатно.

Листинг 7.10. Веб-страница с формой (а); PHP-скрипт для обработки формы (б); результат работы PHP-скрипта при исходных данных «Харриет» и «24» соответственно (в)

```
<html>
<body>
<form action="action.php" method="post">
```

```
<p> Введите свое имя: <input type="text" name="name"> </p>
<p> Введите свой возраст: <input type="text" name="age"> </p>
<input type="submit">
</form>
</body>
</html>
```

(а)

```
<html>
<body>
<h1> Ответ: </h1>
Привет, <?php echo $name; ?>!
Предсказываю: в следующем году тебе будет <?php echo $age+1; ?>
</body>
</html>
```

(б)

```
<html>
<body>
<h1> Ответ: </h1>
Привет, Харриет!
Предсказываю: в следующем году тебе будет 25
</html>
</body>
```

(в)

PHP был разработан специально под сервер Apache, который также обладает открытым исходным кодом и является самым распространенным веб-сервером в мире. Более подробную информацию по PHP можно найти в (Valade, 2002).

Итак, мы знаем уже два различных способа генерации динамических HTML-страниц: с помощью CGI-скриптов и внедрения PHP. Есть еще третий метод, называемый **JSP** (JavaServer Pages — страницы сервера Java). Он в целом схож с PHP и отличается только тем, что динамическая часть программируется на языке Java. Файлы страниц, написанных с помощью JSP, имеют одноименное расширение: .jsp. Еще один метод создания динамических страниц — **ASP** (Active Server Pages — активные серверные страницы). Это ответ Microsoft на PHP и JSP. В качестве языка динамического веб-программирования используется собственный язык написания скриптов, созданный Microsoft, — Visual Basic Script. Соответственно, файлы страниц, написанных с использованием этого метода, имеют расширение .asp. Вопрос выбора между PHP, JSP и ASP в основном политический (открытый код *Sun* против кода *Microsoft*). С точки зрения технологий все эти методы вполне сравнимы по возможностям.

Весь набор методов создания динамических страниц иногда называют **динамическим HTML** (DHTML).

Создание динамических веб-страниц на стороне клиента

Скрипты CGI, PHP, JSP и ASP решают вопросы обработки форм и взаимодействия с базами данных, расположенными на сервере. Они могут принимать входящую информацию из форм, осуществлять поиск по одной или нескольким базам данных и в качестве результата генерировать HTML-страницы. Но ни один из этих методов не позволяет напрямую взаимодействовать с пользователем, например, реагировать на движения мышкой. Для этих целей необходимы скрипты, внедренные в HTML-страницы и выполняющиеся не на серверной, а на клиентской машине. Начиная с HTML 4.0, появилась возможность включать скрипты такого типа с помощью тега <script>. Наиболее популярный язык написания сценариев для клиентской стороны — это JavaScript. Его мы вкратце и рассмотрим далее.

Итак, JavaScript — это язык написания сценариев, использующий идеи, *крайне* отдаленно напоминающие язык программирования Java. Но JavaScript — это не Java по определению. Как и другие языки написания скриптов, он очень высокоуровневый. Так, одной строкой JavaScript можно создать диалоговое окно, войти в цикл ожидания пользовательского ввода и сохранить полученную строку в переменной. Столь высокий уровень языка идеально подходит для разработки интерактивных веб-страниц. С другой стороны, тот факт, что JavaScript не стандартизован и мутирует быстрее, чем мушка-дрозофила в рентгеновском луче, сильно усложняет написание платформонезависимых программ. Надо, впрочем, надеяться, что рано или поздно этот язык дойдет до более или менее устойчивого состояния.

Пример программы на JavaScript показан в листинге 7.11. Как и в листинге 7.10, *а*, программа создает форму с запросом имени и возраста пользователя и гениальным образом предсказывает, исходя из этих данных, каков будет возраст человека в следующем году. Тело скрипта почти такое же, как и в примере PHP. Основная разница состоит в объявлении кнопки *Submit* и определении присваивания в этом объявлении. Оператор присваивания сообщает браузеру о том, что в случае нажатия кнопки необходимо запустить скрипт response и передать ему форму в качестве параметра.

Совершенно по-новому здесь объявляется функция response. Объявление находится в заголовке HTML-файла, который обычно хранит информацию о заголовках, цвете фона и т. п. Функция извлекает из формы значение поля name и сохраняет его в виде строки в переменной person. Также извлекается значение поля age. Оно приводится к целочисленному типу с помощью функции eval, затем к значению добавляется 1, и результат сохраняется в years. После этого документ открывается для записи, в него записываются четыре строки (для этого используется метод writeln), и документ закрывается. Документ представляет собой HTML-страницу, как видно по многочисленным тегам HTML. Браузер выводит готовый документ на экран.

Листинг 7.11. Применение JavaScript для обработки формы

```
<html>
<head>
<script language="javascript" type="text/javascript">
function response(test_form) {
var person = test_form.name.value;
var years = eval(test_form.age.value) + 1;
document.open()
document.writeln("<html> <body>");
```

```
document.writeln("Привет, " + person + "!<br>");
document.writeln("Предсказываю: в следующем году тебе будет " + years +".");
document.writeln("</body> </html>");
document.close();
}
</script>
</head>

<body>
<form>
Введите свое имя: <input type="text" name="name">
<p>
Введите свой возраст: <input type="text" name="age">
<p>
<input type="button" value="Подтверждение" onclick="response(this.form)">
</form>
</body>
</html>
```

Важно понимать, что обрабатываются программы, показанные в листингах 7.10 и 7.11, совершенно по-разному, несмотря на их внешнее сходство. Что происходит в первом случае (листинг 7.10)? После того как пользователь нажимает кнопку *Submit*, браузер собирает всю введенную информацию в одну длинную строку и отправляет ее на тот сервер, с которого пришла страница. Сервер видит имя PHP-файла и запускает его. PHP-скрипт создает новую HTML-страницу, которая отсылается браузеру для отображения. Что касается второго случая (листинг 7.11), то после нажатия кнопки *Submit* браузер сам выполняет действия функции JavaScript, содержащейся на странице. Вся работа производится локально, внутри браузера. С сервером никакого взаимодействия не осуществляется. Как следствие, результат появляется практически мгновенно, тогда как при использовании PHP задержка прибытия страницы с результатом может составлять несколько секунд. Разница между скриптами, работающими на стороне сервере и на стороне клиента, показана на рис. 7.16. Показаны шаги, выполняемые в каждом случае. В обоих случаях все эти этапы производятся после вывода формы на экран. Шаг 1 состоит в приеме данных от пользователя. Затем следует их обработка, и вот здесь имеются значительные различия.

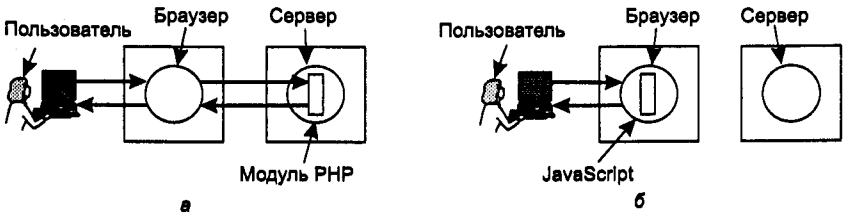


Рис. 7.16. PHP-скрипт на стороне сервера (*а*); сценарий на JavaScript на стороне клиента (*б*)

Эти различия вовсе не означают, что JavaScript «лучше», чем PHP. Просто у них различные сферы применения. PHP (а с ним неявно и JSP, и ASP) применяется тогда, когда необходимо взаимодействовать с удаленной базой данных. JavaScript используют тогда, когда требуется взаимодействие с пользователем в пределах его компьютера. Разумеется, возможно (и довольно часто осуществляется) одновременное использование PHP и JavaScript, хотя они и не могут, например, обрабатывать одно и то же событие (типа нажатия кнопки) или производить одно и то же действие.

JavaScript — это полноценный язык программирования, ничуть не слабее по возможностям, чем C или Java. В нем есть понятия переменных, строк, массивов, объектов, функций и всех привычных управляющих структур. К тому же он обладает множеством полезных свойств, специфичных для веб-страниц, включая возможность управления окнами и фреймами, установки и получения cookie, работы с формами и гиперссылками. Пример программы на JavaScript с рекурсивной функцией приведен в листинге 7.12.

Листинг 7.12. Программа на JavaScript для подсчета и вывода факториалов

```
<html>
<head>
<script language="javascript" type="text/javascript">

function response(test_form) {
function factorial(n) {if (n == 0) return 1; else return n * factorial(n - 1);}
var r=eval(test_form.number.value); //r=введенный аргумент
document.myform.mytext.value = "Результаты:\n";
for (var i = 1;i<=r;i++)//Вывести одну строку от 1 до r
document.myform.mytext.value += (i + "!= "+factorial(i) + "\n");
}
</script>
</head>

<body>
<form name="myform">
Введите число: <input type="text" name="number">
<input type="button" value="Подсчет таблицы факториалов"
onclick="response(this.form)">
<p>
<textarea name="mytext" rows="25" cols="50"> </textarea>
</form>
</body>
</html>
```

С помощью JavaScript можно также отслеживать появление мыши над определенными объектами на странице. На многих веб-страницах с JavaScript можно заметить, что при наведении курсора мыши на какой-нибудь текст или изображение что-нибудь происходит. Часто при этом меняется изображение или вдруг выскакивает меню. Такое поведение легко программируется на JavaScript и несколько оживляет веб-страницы. Пример приведен в листинге 7.13.

Листинг 7.13. Интерактивная страница, отвечающая на движения мышью

```
<html>
<head>
<script language="javascript" type="text/javascript">
```

рам не приходится беспокоиться о потерянных, дублированных, слишком длинных сообщения и подтверждениях. Все это обеспечивается протоколом TCP.

В HTTP 1.0 после установки соединения посылался один запрос, на который приходил один ответ. После этого TCP-соединение разрывалось. В то время типичная веб-страница целиком состояла из HTML-текста, и такой способ взаимодействия был адекватным. Однако прошло несколько лет, и в странице оказалось множество значков, изображений и других украшений. Очевидно, что установка TCP-соединения для передачи одного значка нерациональна и слишком дорога.

Это соображение привело к созданию протокола HTTP 1.1, который поддерживал **устойчивые соединения**. Это означало, что появилась возможность установки TCP-соединения, отправки запроса, получения ответа, а затем передачи и приема дополнительных запросов и ответов. Таким образом, снизились накладные расходы, возникавшие при постоянных установках и разрывах соединения. Стало возможным также конвейеризировать запросы, то есть отправлять запрос 2 еще до прибытия ответа на запрос 1.

Методы

Несмотря на то что HTTP был разработан специально для использования в веб-технологиях, он был намеренно сделан более универсальным, чем это было необходимо, так как рассчитывался на будущее применение в объектно-ориентированных приложениях. По этой причине в дополнение к обычным запросам веб-страниц были разработаны специальные операции, называемые **методами**. Они обязаны своим существованием технологии SOAP. Каждый запрос состоит из одной или нескольких строк ASCII, причем первое слово является именем вызываемого метода. Встроенные методы перечислены в табл. 7.12. Помимо этих общих методов, у различных объектов могут быть также свои специфические методы. Имена методов чувствительны к регистру символов, то есть метод *GET* существует, а *get* — нет.

Таблица 7.12. Встроенные методы HTTP-запросов

Метод	Описание
GET	Запрос чтения веб-страницы
HEAD	Запрос чтения заголовка веб-страницы
PUT	Запрос сохранения веб-страницы
POST	Добавить к именованному ресурсу (например, к веб-странице)
DELETE	Удалить веб-страницу
TRACE	Отобразить входящий запрос
CONNECT	Зарезервирован для будущего использования
OPTIONS	Опрос определенных параметров

Метод *GET* запрашивает у сервера страницу (под которой в общем случае подразумевается объект, но на практике это обычно просто файл), закодированную согласно стандарту MIME. Большую часть запросов к серверу составляют именно запросы *GET*. Вот самая типичная форма *GET*:

```
GET filename HTTP/1.1,
```

где `filename` указывает на запрашиваемый ресурс (файл), а `1.1` — на используемую версию протокола.

Метод *HEAD* просто запрашивает заголовок сообщения, без самой страницы. С помощью этого метода можно узнать время последнего изменения страницы для сбора индексной информации или просто для проверки работоспособности данного URL.

Метод *PUT* является противоположностью метода *GET*: он не читает, а записывает страницу. Этот метод позволяет создать набор веб-страниц на удаленном сервере. Тело запроса содержит страницу. Она может быть кодирована с помощью MIME. В этом случае строки, следующие за командой *PUT*, могут включать различные заголовки, например, *Content-Type* или заголовки аутентификации, подтверждающие права абонента на запрашиваемую операцию.

Метод *POST* несколько напоминает метод *PUT*. Он также содержит URL, но вместо замены имеющихся данных новые данные «добавляются» (в неком общем смысле) к уже существующим. Это может быть публикация сообщения в конференции или добавление файла к электронной доске объявлений BBS. На практике ни *PUT*, ни *POST* широко не применяются.

Метод *DELETE*, что неудивительно, удаляет страницу. Как и в методе *PUT*, здесь особую роль могут играть аутентификация и разрешение на выполнение этой операции. Даже при наличии у пользователя разрешения на удаление страницы нет никакой гарантии, что метод *DELETE* удалит страницу, так как даже при согласии удаленного HTTP-сервера сам файл может оказаться защищенным от изменения или перемещения.

Метод TRACE предназначен для отладки. Он приказывает серверу отослать назад запрос. Этот метод особенно полезен, когда запросы обрабатываются некорректно и клиенту хочется узнать, что за запрос реально получает сервер.

Метод *CONNECT* в настоящее время не используется. Он зарезервирован для будущего применения.

Метод *OPTIONS* позволяет клиенту узнать у сервера о его свойствах или о свойствах какого-либо конкретного файла.

В ответ на каждый запрос от сервера поступает ответ, содержащий строку состояния, а также, возможно, дополнительную информацию (например, веб-страницу или ее часть). Строка состояния может содержать трехразрядный код состояния, сообщающий об успешном выполнении запроса или о причинах неудачи. Первый разряд предназначен для разделения всех ответов на пять основных групп, как показано в табл. 7.13. Коды, начинающиеся с 1 (1xx), на практике используются редко. Коды, начинающиеся с 2, означают, что запрос был обработан успешно и данные (если их запрашивали) отосланы. Коды 3xx сообщают клиенту о том, что нужно попытать счастья в другом месте — используя либо другой URL, либо свой собственный кэш.

Таблица 7.13. Группы кодов состояния, содержащиеся в ответах сервера

Код	Значение	Примеры
1xx	Информация	100 — сервер согласен обрабатывать запросы клиента
2xx	Успех	200 — запрос успешно обработан; 204 — содержимое отсутствует
3xx	Перенаправление	301 — страница перемещена; 304 — кэшированная страница все еще доступна
4xx	Ошибка клиента	403 — ошибка доступа; 404 — страница не найдена
5xx	Ошибка сервера	500 — внутренняя ошибка сервера; 503 — попробуйте еще раз позднее

Коды, начинающиеся с 4, означают, что запрос по какой-либо причине, связанной с клиентом, потерпел неудачу: например, была запрошена несуществующая страница или сам запрос был некорректен. Наконец, коды 5xx сообщают об ошибках сервера, возникших либо вследствие ошибки программы, либо из-за временной перегрузки.

Заголовки сообщений

За строкой запроса (например, содержащей название метода *GET*) могут следовать другие строки с дополнительной информацией. Они называются **заголовками запросов**. Эту информацию можно сравнить с параметрами, предоставляемыми при вызове процедуры. В свою очередь, ответы могут содержать **заголовки ответов**. Некоторые заголовки могут встречаться и там, и там. Наиболее важные из них перечислены в табл. 7.14.

Таблица 7.14. Некоторые заголовки сообщений протокола HTTP

Заголовок	Тип	Содержимое
User-Agent	Запрос	Информация о браузере и его платформе
Accept	Запрос	Тип страниц, поддерживаемых клиентом
Accept-Charset	Запрос	Поддерживаемые клиентом наборы символов
Accept-Encoding	Запрос	Поддерживаемые клиентом типы кодирования
Accept-Language	Запрос	Естественные языки, понимаемые клиентом
Host	Запрос	Имя DNS-сервера
Authorization	Запрос	Список персональных идентификаторов клиента
Cookie	Запрос	Отправка ранее принятого cookie-файла на сервер
Date	Запрос/Ответ	Дата и время отправки сообщения
Upgrade	Запрос/Ответ	Протокол, на который хочет переключиться отправитель
Server	Ответ	Информация о сервере
Content-Encoding	Ответ	Тип кодирования содержимого (например, gzip)
Content-Language	Ответ	Естественный язык, используемый на странице
Content-Length	Ответ	Размер страницы в байтах
Content-Type	Ответ	Тип MIME страницы

Заголовок	Тип	Содержимое
Last-Modified	Ответ	Время и дата внесения последних изменений в страницу
Location	Ответ	Команда клиенту на пересылку его запроса по другому адресу
Accept-Ranges	Ответ	Сервер готов принимать запросы на страницы указанного размера
Set-Cookie	Ответ	Сервер хочет, чтобы клиент сохранил cookie

Заголовок *User-Agent* позволяет клиенту информировать сервер о версии своего браузера, операционной системы или предоставлять другую информацию о себе. В листинге 7.9 мы видели, что сервер каким-то волшебным образом получал эти данные и мог при необходимости использовать их в PHP-скрипте. Как раз с помощью заголовка *User-Agent* клиент и сообщил серверу о себе.

Четыре заголовка, начинающиеся с *Accept*, сообщают серверу о типах информации, которые он готов принять (если их набор ограничен). Первый приведенный в таблице заголовок определяет типы MIME, которые будут корректно приняты клиентом (например, *text/html*). Заголовок *Accept-Charset* сообщает о том, какой набор символов клиент хотел бы видеть (например, ISO-8859 или Unicode-1-1). В заголовке *Accept-Encoding* речь идет о приемлемых методах сжатия (например, *gzip*). Наконец, *Accept-Language* сообщает, на каком языке клиент готов читать документы (например, на испанском). Если сервер имеет возможность выбирать из нескольких страниц, он подберет наиболее подходящий для клиента вариант в соответствии с полученной информацией. Если запрос удовлетворить невозможно, возвращается код ошибки, и запрос считается неудавшимся.

Заголовок *Host* описывает сервер. Его значение берется из URL. Этот заголовок обязателен. Почему? Потому что некоторые IP-адреса могут обслуживать несколько имен DNS одновременно, и серверу необходимо каким-то образом различать, кому передавать запрос.

Заголовок *Authorization* требуется в тех случаях, когда запрашивается защищенная страница. С его помощью клиент может подтвердить свои права на просмотр запрашиваемой страницы.

Несмотря на то, что cookie описываются в RFC 2109, а не в RFC 2616, для их описания существуют два заголовка. В частности, заголовок *Cookie* применяется клиентом при возвращении на сервер cookie-файла, который ранее был послан какой-либо машиной из домена сервера.

Заголовок *Date* может применяться как в запросах, так и в ответах. Он содержит время и дату отправки сообщения.

Заголовок *Upgrade* может использоваться для облегчения перехода на будущие (возможно, несовместимые с предыдущими) версии протокола HTTP. Он позволяет клиенту объявлять о поддерживаемых им протоколах, а серверу — объявлять о применяемых им протоколах.

А теперь мы подошли к заголовкам, которые может устанавливать только сервер при создании ответов на запросы. Первый из них, *Server*, позволяет серверу сообщать информацию о себе. При желании он может указать некоторые свои параметры.

Следующие четыре заголовка, начинающиеся с *Content-*, дают серверу возможность описать свойства посылаемой им страницы.

Заголовок *Last-modified* содержит дату и время внесения последних изменений в отправляемую страницу. Он играет важную роль при кэшировании страницы.

Заголовок *Location* вставляется сервером для информирования клиента о том, что стоит попробовать осуществить свой запрос повторно по другому URL. Такая ситуация может возникать при «переезде» страницы или тогда, когда несколько URL ссылаются на одну и ту же страницу (возможно, на «зеркало» страницы, расположенное на другом сервере). Этот трюк нередко применяется теми компаниями, главная веб-страница которых прописана в домене *.com*, однако клиенты перенаправляются с нее на национальные или региональные страницы, имеющие свои IP-адреса или написанные на более приемлемом для клиента языке.

Если страница очень велика по размеру, клиент может не захотеть принимать ее сразу целиком. Некоторые серверы могут принимать запросы, ограничивающие размеры страниц, отсылаемых за один раз. Если страница оказывается слишком большой, она будет разбита на более мелкие единицы и выслана в несколько приемов. Заголовок *Accept-Ranges* сообщает о том, что сервер готов поддерживать такие запросы частей страниц.

Set-cookie — это второй заголовок, относящийся к cookie-маркерам. Если этот заголовок установлен сервером, предполагается, что, увидев его, клиент сохранит у себя cookie и вернет его вместе со следующим запросом на сервер.

Пример использования HTTP

Поскольку HTTP является текстовым протоколом, взаимодействие с сервером посредством терминала (который в данном случае выступает как противоположность браузеру) можно организовать достаточно просто. Необходимо лишь установить TCP-соединение с портом 80 сервера. Читателю предоставляется возможность самому посмотреть, как работает этот сценарий (предпочтительнее запускать его в системе UNIX, поскольку некоторые другие системы могут не отображать статус соединения). Итак, последовательность команд такова:

```
telnet www.ietf.org 80 >log

GET /rfc.html HTTP/1.1
Host: www.ietf.org

close
```

Эта последовательность команд устанавливает telnet-соединение (то есть TCP-соединение) с портом 80 веб-сервера IETF, расположенного по адресу www.ietf.org. Результат сеанса связи записывается в файл log, который затем можно просмотреть. Далее следует команда *GET*. Указывается имя запрашиваемого файла и протокол передачи. Следом идет обязательная строка с заголовком *Host*. Пустая строка, которая находится за ней, также обязательна. Она сигнализирует серверу о том, что заголовки запросов закончились. Командой `close` (это команда программы telnet) соединение разрывается.

Файл журнала соединения, log, может быть просмотрен с помощью любого текстового редактора. Он должен начинаться примерно так, как показано в листинге 7.14, если только на сайте IETF за это время не произошли какие-нибудь изменения.

Листинг 7.14. Начало вывода файла www.ietf.org/rfc.html

```
Trying 4.17.168.6...
Connected to www.ietf.org
Escape character is '^]'.
HTTP/1.1 200 OK
Date: Wed, 08 May 2002 22:54:22 GMT
Server: Apache/1.3.20 (Unix) mod_ssl/2.8.4 OpenSSL/0.9.5a
Last-Modified: Mon, 11 Sep 2000 13:56:29 GMT
ETag: "2a79d-c8b-39bce48d"
Accept-Ranges: bytes
Content-Length: 3211
Content-Type: text/html
X-Pad: предотвращает ошибки браузеров

<html>
<head>
<title>IETF RFC Page</title>

<script language="javascript">
function url() {
var x = document.form1.number.value
if (x.length == 1) {x = "000" + x}
if (x.length == 2) {x = "00" + x}
if (x.length == 3) {x = "0" + x}
document.form1.action = "/rfc/rfc" + x + ".txt"
document.form1.submit
}
</script>

</head>
```

Первые три строки в этом листинге созданы программой telnet, а не удаленным сайтом. А вот строка, начинающаяся с `HTTP/1.1`, — это уже ответ IETF, говорящий о том, что сервер желает общаться с вами при помощи протокола HTTP/1.1. Далее следует ряд заголовков и, наконец, само содержимое запрашиваемого файла. Мы уже видели все заголовки, кроме *ETag*, который является уникальным идентификатором страницы, связанным с кэшированием, и *X-Pad* — нестандартного заголовка, помогающего бороться с ошибками браузеров.

Повышение производительности

Популярность Всемирной паутины стала настоящей бедой для нее. Серверы, маршрутизаторы и каналы связи все чаще оказываются перегруженными. Многие уже стали называть WWW (World Wide Web) «Всемирным ожиданием» (World Wide Wait). Проблема нескончаемых задержек привела ученых к необхо-

димости разработки методов повышения производительности. Далее мы обсудим три из них: кэширование, репликацию серверов и сети с доставкой содержимого.

Кэширование

Довольно простым способом повышения производительности является сохранение ранее загружавшихся страниц на случай их повторного запроса. Этот метод особенно эффективен при работе с часто посещаемыми страницами (такими как www.yahoo.com или www.cnn.com). Сохранение веб-страниц «про запас» для последующего использования называется **кэшированием**. Обновление кэша является обычной процедурой для некоторого процесса, называемого сервером-посредником, или **прокси** . Чтобы иметь возможность использовать метод кэширования, браузер должен быть настроен на обращение к посреднику, а не к реальному серверу, на котором хранится страница. Если у сервера-посредника есть нужная страница, она сразу же возвращается пользователю. В противном случае ее придется получить с сервера, добавить в кэш для будущего использования и только после этого предоставить пользователю.

С кэшированием связаны два важных вопроса.

1. Кто должен заниматься кэшированием?
2. Сколько времени страницы должны храниться в кэше?

На первый вопрос есть несколько ответов. На отдельных персональных компьютерах часто имеется прокси, поэтому поиск ранее запрошенных страниц происходит быстро. В корпоративной ЛВС прокси-сервер обычно устанавливается на машине с разделяемыми ресурсами, и если один из клиентов данной ЛВС запросил страницу с сервера, то другой может получить ее уже из кэша сервера-посредника (прокси). Прокси-серверы часто устанавливают у себя провайдеры с целью повышения скорости доступа для всех своих клиентов. Нередко все эти кэши работают одновременно, поэтому запрос вначале отправляется на локальный прокси-сервер. Если там страница не обнаружена, запрос передается на прокси-сервер ЛВС. Не найдя у себя запрашиваемую страницу, последний обращается к прокси-серверу провайдера. На этом этапе страница уже должна быть получена в любом случае: либо она берется из кэша, либо приходит с веб-сервера. Схема с несколькими кэшами, работающими последовательно, называется **иерархическим кэшированием**. Возможная реализация этого метода показана на рис. 7.18.

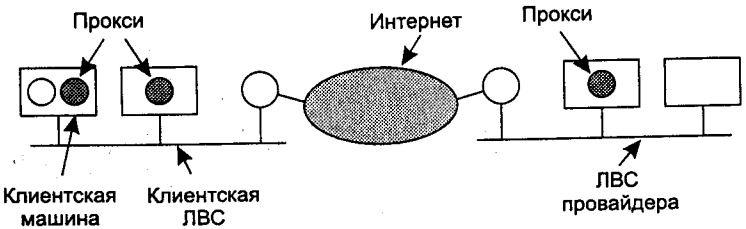


Рис. 7.18. Иерархическое кэширование с тремя серверами-посредниками

Вопрос о сроке хранения кэшированных страниц решается несколько хитрее. Некоторые страницы не кэшируются вообще. Это касается, например, страниц, содержащих списки 50 самых котируемых акций, цены на которые меняются каждую секунду. В случае кэширования пользователь получал бы *устаревшие* данные. С другой стороны, если на бирже в какой-то день торги не проводятся, эта страница может оставаться актуальной в течение нескольких часов или даже дней, до начала следующих торгов. Таким образом, необходимость в кэшировании каждой отдельно взятой страницы может сильно варьироваться с течением времени.

Ответ на вопрос, в какой момент удалять страницы из кэша, зависит от того, насколько свежими хочет видеть их пользователь (поскольку они сохраняются на диске, проблемы нехватки места обычно не возникают). Если сервер-посредник выбрасывает страницы из кэша слишком быстро, он вряд ли вернет устаревшую страницу, однако и эффективность такого кэша будет низкой. Если же хранить данные слишком долго, эффективность будет достигаться в основном за счет предоставления уже никому не нужной, устаревшей информации.

Решается этот философский вопрос с использованием двух подходов. Первый подразумевает эвристический анализ для принятия решения о сроке хранения страницы. Обычно он основывается на значении заголовка *Last-Modified* (см. табл. 7.13). Если страница подвергалась изменениям час назад, она будет храниться в кэше также в течение часа. Если же последние изменения были внесены в страницу год назад, очевидно, она содержит довольно стабильную информацию (например, список греческих и римских богов), и ее можно хранить в кэше в течение года, резонно предполагая, что за это время ничего на ней не изменится. Несмотря на то, что такой эвристический анализ работает весьма успешно, все же время от времени из кэша извлекаются устаревшие страницы.

Другой подход является более дорогим, но и более надежным в смысле исключения возможности хранения в кэше устаревших страниц. Используются особенности RFC 2616, имеющие отношение к управлению кэшем. Одной из самых полезных особенностей является наличие заголовка *If-Modified-Since*, который сервер-посредник может посылать веб-серверу. В нем указывается страница, состояние которой хочет выяснить прокси, а также время внесения последних изменений (значение заголовка *Last-Modified* на странице). Если страница не подвергалась изменениям, сервер отошлет обратно короткое сообщение *Not Modified* («Изменений нет», код 304, см. табл. 7.12). Это будет означать, что прокси может использовать хранящуюся в кэше страницу. Если же на странице произошли какие-либо изменения, сервер пришлет ее обновленную версию. Такой подход требует обмена запросами и ответами, но если страница еще не устарела, ответ сервера будет очень коротким.

Два указанных подхода можно комбинировать. В течение первого интервала времени ΔT после получения страницы прокси просто извлекает запрошенную страницу из кэша. По прошествии определенного промежутка времени прокси использует *If-Modified-Since* для проверки состояния страницы. Выбор ΔT подразумевает некоторый эвристический анализ, базирующийся на знании времени последних изменений на странице.

Динамические веб-страницы (например, созданные PHP-скриптом) не должны кэшироваться вообще никогда, так как их содержимое переменное по определению. Для этого, а также для некоторых других специальных случаев предусмотрен механизм, с помощью которого сервер может информировать все серверы-посредники на пути к клиенту о том, что не следует использовать текущую страницу без запроса ее актуальности. Этот же механизм может применяться для страниц, которые могут меняться довольно часто. В RFC 2616 определен также ряд других механизмов управления кэшированием.

Еще один подход к повышению производительности называется **упреждающим кэшированием**. Получая страницу с веб-сервера, прокси-сервер может проверить ее на наличие гиперссылок. Если таковые имеются, он может запросить и поместить в кэш страницы, на которые указывают гиперссылки, просто на тот случай, если они понадобятся пользователю. Этот метод может помочь уменьшить время доступа к последующим запросам, однако может привести к загрузке линий связи при передаче страниц, которые, возможно, так и не будут запрошены.

Понятно, что кэширование во Всемирной паутине реализуется далеко не тривиально. Эту тему можно было бы обсуждать еще долго. На самом деле, ей посвящены отдельные книги, например (Rabinovich и Spatscheck, 2002; Wessels, 2001). Однако нам пора двигаться дальше.

Репликация серверов

Кэширование — это технология повышения производительности, применяемая на стороне клиента. Существуют также методы, связанные со стороной сервера. Один из наиболее распространенных способов повышения производительности Сети заключается в репликации содержимого серверов на нескольких машинах, расположенных в разных точках земного шара. Такие серверы с одинаковым содержимым иногда называются **зеркалами**. Репликация часто применяется на корпоративных сайтах, где главная страница содержит несколько изображений и ссылок на региональные подразделения сайтов. Пользователь может выбрать сайт, находящийся на ближайшем сервере. Начиная с этого момента, все его взаимодействие с данным сайтом будет производиться через этот сервер.

Реплицированные сайты обычно отличаются статичностью. Компания решает, где расположить зеркала, решает вопрос с серверами в каждом из регионов и создает более или менее информативные страницы на каждом зеркале (возможно, убирая изображения падающих снежинок с сервера на Майами и фотографии пляжных лежаков с сервера в центре континента). Зеркала обычно сохраняются неизменными в течение многих месяцев или даже лет.

К сожалению, Всемирной паутине присущ феномен, известный под названием **внезапная давка**, когда тихий, неприметный, никому не известный сайт мгновенно оказывается в центре всеобщего внимания и интереса. Например, до 6 ноября 2000 года веб-сайт государственного секретаря штата Флорида, www.dos.state.fl.us, рассказывал немногочисленным посетителям о последних заседаниях и давал информацию о регистрации нотариальной деятельности во Флориде. Однако 7 ноября 2000 года, когда президентство в США вдруг стало целиком зависеть от нескольких тысяч спорных голосов в горстке округов Флориды, этот сайт внезапно стал одним из пяти наиболее посещаемых в мире. Надо ли говорить о том, что он просто не мог справиться с такой нагрузкой и практически задохнулся от наплыва посетителей.

Понадобился какой-то способ автоматической репликации веб-сайтов при угрозе массовой атаки посетителей. Причем количество зеркал в разных точках Земли должно удовлетворять спросу. Эти реплицированные сайты должны пережить волну высокого трафика, после чего большинство из них можно спокойно удалить с серверов. Для этого у сайта должна быть предварительная договоренность с какой-нибудь крупной хостинговой компанией о создании зеркал при необходимости и об оплате реально занимаемых в каждый момент времени ресурсов.

Более гибкая стратегия заключается в создании динамической постраничной репликации, зависящей от местоположения источника трафика. Некоторые исследования в этой области описаны в книгах (Pierre и др., 2001; Pierrre и др., 2002).

Сети доставки содержимого

Чем отличается капиталистический мир от всех иных формаций? Тем, что в нем всегда найдется человек, который извлечет выгоду даже из Всемирного ожидания. Как? Очень просто. Компании, называющие себя **CDN** (Content Delivery Networks — сети доставки содержимого), договариваются с поставщиками данных (например, музыкальными сайтами, газетами и другими компаниями, заинтересованными в быстрой доставке содержимого своих веб-сайтов) и за скромную плату предлагают им обеспечивать эффективную доставку их данных конечным пользователям. После подписания контракта компания передает CDN содержимое своего сайта для предварительной обработки (вкратце обсуждается далее) и последующего распространения.

После этого CDN договаривается с большим числом провайдеров, обещая хорошо заплатить за размещение в их ЛВС удаленно управляемой копии сайта с ценной для пользователя информацией. Для провайдера это не только способ получения дополнительного дохода, но и отличный ход в конкурентной борьбе: ведь клиенты этого провайдера смогут получать информацию CDN с очень хорошей скоростью. Провайдеры, отказывающиеся от предложений CDN, не только не получают «бесплатный сыр», но и проигрывают в качестве предоставляемых услуг по сравнению с провайдерами, сотрудничающими с сетями доставки содержимого. В таких условиях отказ от подписания контракта с CDN оказывается просто безумием. Крупнейшие сети доставки содержимого работают более чем с 10 000 серверами, разбросанными по всему миру.

Тысячекратное копирование содержимого, очевидно, существенно повышает производительность Сети. Тем не менее, вначале необходимо заставить систему работать, а для этого требуется способ переадресации запроса пользователя на ближайший сервер CDN. Желательно, чтобы его местоположение совпадало с провайдером. Эта переадресация должна происходить без изменения DNS или какой-либо другой части стандартной инфраструктуры Интернета. Далее в не-

сколько упрощенном виде описывается принцип работы Akamai — самой большой CDN.

Процесс начинается с передачи поставщиком информации своего веб-сайта в CDN. Каждая полученная страница проходит в сети доставки содержимого предварительную обработку. При этом существующие URL заменяются модифицированными. Идея, стоящая за этой стратегией, заключается в том, что обычно сайт состоит из некоторого числа маленьких страниц (обычного HTML-текста), на которых расположены ссылки на большие файлы (аудио- и видеозаписи, изображения). Модифицированные HTML-страницы остаются на своих прежних местах (на сервере поставщика информации), а на серверы CDN уходят только файлы больших размеров.

Рассмотрим работу этой схемы на примере веб-страницы фирмы «Пушистые фильмы» (листинг 7.15, *а*). После предварительной обработки она превращается в страницу, описанную в листинге 7.15, *б*, и размещается на сервере «Пушистых фильмов» по адресу www.furryvideo.com/index.html.

Когда пользователь набирает URL www.furryvideo.com, служба DNS возвращает IP-адрес веб-сайта фирмы, позволяя получить заглавную страницу сайта самым обычным образом — с сервера «Пушистых фильмов». Если же пользователь переходит куда-либо по гиперссылке с этой страницы, браузер через службу DNS обращается к серверу *cdn-server.com*. Затем на соответствующий IP-адрес браузером отправляется HTTP-запрос, в качестве ответа на который ожидается получение видеофайла в формате MPEG.

Однако этого не происходит по той простой причине, что сервер *cdn-server.com* не содержит никаких данных. А содержит их подставной HTTP-сервер сети доставки содержимого. По имени файла и названию сервера он определяет, какая страница запрашивается и кому из поставщиков информации она принадлежит. Кроме того, анализируется IP-адрес входящего запроса, и по базе данных определяется возможное местоположение пользователя. Вооружившись этой информацией, он устанавливает, какой из серверов CDN способен предоставить наилучшее качество обслуживания. Решение принять не так уж просто, поскольку близкое географическое расположение не обязательно означает близость в терминах сетевой топологии. Кроме того, даже оптимальный по расстоянию сервер в данный момент может быть слишком сильно загружен. Приняв решение, *cdn-server.com* отсылает ответ, возвращая код 301 и заголовок *Location*, содержащий URL файла, расположенного на одном из подведомственных серверов CDN. Предположим, что этот URL выглядит так: www.CDN-0420.com/furryvideo/bears.com. Браузер обрабатывает этот адрес стандартным образом, в результате чего пользователь получает MPEG-файл.

Листинг 7.15. Исходная веб-страница (а); та же страница после обработки CDN (б)

```
<html>
<head><title>Пушистые фильмы</title></head>
<body>
<h1> Список пушистых фильмов</h1>
<p>Бесплатные примеры:</p>
<a href="bears.mpg">Актуальные проблемы медведей</a><br>
<a href="bunnies.mpg">Забавные кролики</a><br>
<a href="mice.mpg">Мышки-норушки</a><br>
</body>
</html>
```

(а)

```
<html>
<head><title>Пушистые фильмы</title></head>
<body>
<h1> Список пушистых фильмов</h1>
<p>Бесплатные примеры:</p>
<a href="http://cdn-server.com/furryvideo/bears.mpg"> Актуальные проблемы
медведей</a><br>
<a href=" http://cdn-server.com/furryvideo/bunnies.mpg"> Забавные кролики</a><br>
<a href=" http://cdn-server.com/furryvideo/mice.mpg"> Мышки-норушки</a><br>
</body>
</html>
```

(б)

Все этапы описанного процесса показаны на рис. 7.19. На первом шаге определяется IP-адрес www.furryvideo.com. С сервера самым обычным образом загружается и выводится на экран HTML-страница. На ней размещены три гиперссылки на *cdn-server* (см. листинг 7.15, *б*). Пользователь выбирает, скажем, первую из них. Ищется ее DNS-адрес (шаг 5), который затем возвращается пользователю (шаг 6). При отправке запроса файла *bears.mpg* на *cdn-server* (шаг 7) клиенту возвращается просьба переадресовать свой запрос на сервер *CDN-0420.com* (шаг 8). Если он следует совету (шаг 9), ему выдается файл из кэша прокси-сервера (шаг 10). Вся эта система работает благодаря шагу 8, когда подставной HTTP-сервер перенаправляет запрос пользователя на сервер-посредник, расположенный максимально близко от клиента.

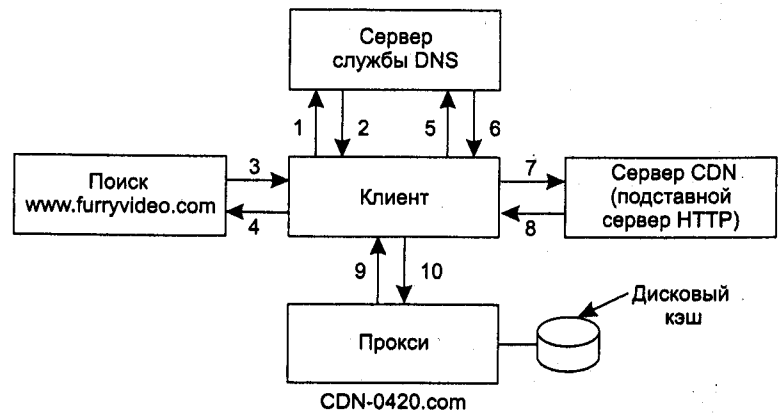


Рис. 7.19. Этапы поиска URL при использовании CDN

Сервер CDN, на который клиент обычно перенаправляется, чаще всего представляет собой прокси с кэшем большого размера, в который заранее загружают-

ся наиболее важные данные. Если, несмотря на эти меры, пользователь все же запрашивает файл, отсутствующий в кэше, он загружается с настоящего сервера поставщика информации и после этого сохраняется в кэше для последующего использования. Организация CDN-сервера в виде прокси, а не в виде точной копии исходного сервера позволяет сэкономить дисковое пространство, сократить время предварительной загрузки и улучшить другие показатели производительности.

Более полную информацию, касающуюся сетей доставки содержимого, можно найти в (Hill, 2002; Rabinovich и Spatcheck, 2002).

Беспроводная Паутина

В последнее время наблюдается все более широкий интерес к маленьким портативным устройствам, способным иметь доступ к Всемирной паутине с помощью беспроводных соединений. На самом деле, первые шаги в этом направлении уже сделаны. Несомненно, в ближайшие годы произойдет много изменений в этой области, однако все же стоит рассмотреть некоторые существующие ныне идеи, связанные с беспроводными технологиями в Web. Это поможет осознать, на какой стадии развития мы находимся сейчас и что может ждать нас впереди. Рассмотрим две технологии беспроводных глобальных веб-систем, успешно продвигающиеся на современном рынке: WAP и i-mode.

WAP — беспроводный протокол распространения данных через Интернет

Когда Интернет и мобильная связь стали общепринятыми явлениями, возникла идея объединения этих двух технологий в мобильном телефоне со встроенным экраном, с помощью которого пользователь смог бы получить доступ ко Всемирной паутине и электронной почте. Впервые эта идея была выдвинута консорциумом, основанным компаниями Nokia, Ericsson, Motorola и phone.com (бывшая Unwired Planet) и состоящим из сотен сотрудников. Система получила название **WAP** (Wireless Application Protocol — беспроводной прикладной протокол).

Устройством, поддерживающим WAP, может быть достаточно современный мобильный телефон, PDA или же ноутбук, не обладающий никакими голосовыми возможностями. Спецификация поддерживает все эти, а также другие устройства. Основная идея состоит в использовании уже существующей цифровой беспроводной инфраструктуры. Пользователи могут в буквальном смысле слова звонить на WAP-шлюз по беспроводному каналу и посылать ему запросы на вебстраницы. Шлюз сперва проверяет свой кэш на наличие в ней нужной страницы. Если она действительно там есть, страница, разумеется, отсылается абоненту; в противном случае он загружает ее себе через обычный Интернет. В сущности, это означает, что WAP 1.0 представляет собой систему с коммутацией каналов с очень высокой поминутной оплатой. Неудивительно, что большинство пользователей не пришли в восторг от возможности разглядывания веб-страниц на крошечном экране и поминутной оплаты. Поэтому технология WAP, мягко говоря, потерпела неудачу. Кроме указанных проблем, впрочем, были и другие.

Тем не менее, протокол WAP и его главный конкурент i-mode (описан далее) базируются на сходных технологиях, и не исключено, что WAP 2.0 еще ждет большой успех. Так как WAP 1.0 был первенцем среди технологий беспроводного Интернета, стоит рассказать о нем хотя бы вкратце.

WAP представляет собой стек протоколов доступа ко Всемирной паутине, оптимизированных под соединения с низкой пропускной способностью, использующие беспроводные устройства с медленными процессорами, небольшой памятью и маленьким экраном. Понятно, что эти требования значительно отличаются от случая настольных ПК. Эти отличия отразились на протоколах. Уровни стека протоколов показаны на рис. 7.20

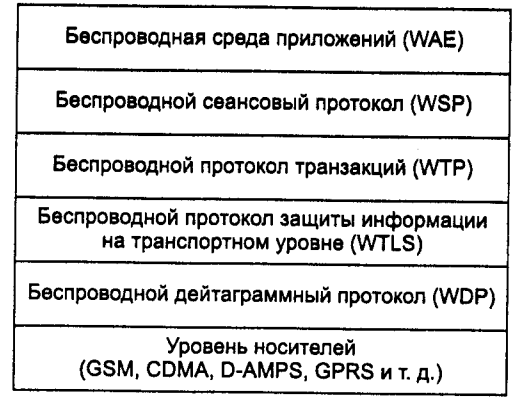


Рис. 7.20. Стек протоколов WAP

На нижнем уровне поддерживаются все существующие мобильные системы, включая GSM, D-AMPS и CDMA. Скорость передачи данных в WAP 1.0 составляет 9600 бит/с. Над уровнем носителей находится дейтаграммный протокол, **WDP** (Wireless Datagram Protocol — беспроводной дейтаграммный протокол). По сути дела, это UDP. Следом идет уровень обеспечения защиты информации, совершенно необходимый в сетях, где информация передается в эфире. Используемый здесь WTLS представляет собой часть Netscape SSL — системы, которую мы будем рассматривать в главе 8. Над ним располагает уровень транзакций, на котором происходит управление запросами и ответами. Возможны варианты разной степени надежности. Этот уровень заменяет TCP, который не используется в беспроводных соединениях по причине низкой эффективности. Затем идет сеансовый уровень, подобный HTTP/1.1, однако отличающийся от него некоторыми ограничениями и расширениями, соответствующими применению. Наконец, верхний уровень — это микробраузер (WAE).

Помимо высокой стоимости услуг есть еще один аспект, который, несомненно, повлиял на низкую популярность WAP: эта технология не использует HTML. Вместо этого на верхнем уровне (WAE) используется специальный язык разметки **WML** (Wireless Markup Language — язык разметки для беспроводного Интернета), являющийся одним из приложений XML. По этой причине, в принципе,

WAP-устройство может получать доступ только к страницам, преобразованным в WML. Понимание того, что это сильно ограничивает область применения, а следовательно, и значимость WAP, привело к разработке фильтров, конвертирующих HTML-страницы в WML-страницы «на лету». Вся эта система изображена на рис. 7.21.

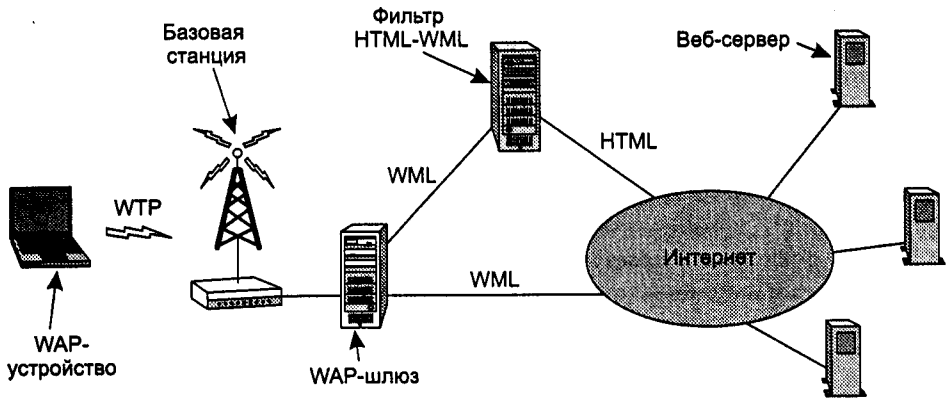


Рис. 7.21. Архитектура WAP

Надо отдать должное тому, что технология WAP, в общем-то, шла впереди своего времени. Когда она была впервые запущена, стандарт XML был почти неизвестен за пределами консорциума W3C, поэтому всезнающая пресса сделала далеко идущий вывод: ТЕХНОЛОГИЯ WAP НЕСОВМЕСТИМА С HTML. Более корректно было бы говорить о том, что ТЕХНОЛОГИЯ WAP УЖЕ ВНЕДРИЛА НОВЫЙ СТАНДАРТ HTML. Однако ущерб репутации был нанесен, и исправить положение оказалось так тяжело, что WAP 1.0 так и не стал популярным. Мы еще вернемся к этой технологии после рассмотрения ее главного соперника.

I-mode

В то время как многоотраслевой консорциум производителей телекоммуникационных систем и компьютерного оборудования выковывал новый открытый стандарт, использующий новейшую версию HTML, в Японии велись собственные разработки. Японка Мари Мацунага (Mari Matsunaga) изобрела альтернативный подход к беспроводным веб-технологиям и назвала его **i-mode** (information-mode — режим передачи информации). Она смогла убедить дочерние компании бывшей японской телефонной монополии в том, что ее подход является прогрессивным, и в феврале 1999 года NTT DoCoMo (буквально: Вездесущая японская телефонно-телеграфная компания) запустила новую услугу в Японии. Через три года у нее было уже более 35 миллионов абонентов по всей Японии, которые получили доступ к 40 000 специальных веб-сайтов i-mode. Телекоммуникационные компании по всему миру были удивлены огромным финансовым успехам, особенно в свете

того факта, что путь WAP оказался практически тупиковым. Что же это за система, как она работает? Далее мы ответим на эти вопросы.

Технология i-mode включает в себя три основных компонента: новую систему передачи данных, новый тип телефонных аппаратов и новый язык для создания веб-страниц. Система передачи состоит из двух отдельных сетей: существующей мобильной сети с коммутацией каналов (нечто вроде D-AMPS) и новой сети с коммутацией пакетов, специально разработанной для i-mode. В голосовом режиме используется традиционная сеть с коммутацией пакетов и поминутной оплатой. Что касается i-mode, то в этом режиме используется сеть с коммутацией пакетов, которая постоянно находится во включенном состоянии (как ADSL или кабельный Интернет), поэтому понятие повременной оплаты отсутствует. Взимается оплата, пропорциональная количеству посланных пакетов. В настоящее время использовать две сети одновременно невозможно.

Аппараты выглядят как мобильные телефоны с маленькими экранами. NNT DoCoMo постоянно рекламирует устройства i-mode, утверждая, что они являются более хорошими мобильными телефонами, чем беспроводные веб-терминалы, хотя на самом деле они как раз ими и являются. При этом пользователей, даже не знающих о том, что они имеют постоянное подключение к Интернету, большинство. По их мнению, принадлежащие им устройства i-mode представляют собой просто мобильные телефоны с расширенным набором функций. С такой моделью услуги i-mode вполне согласуется тот факт, что пользователь не имеет возможности программировать телефонный аппарат, хотя в них встроен некий аналог ПК образца 1995 года с Windows 95 или UNIX.

При включении телефона i-mode выводится список категорий официально предоставляемых услуг. Насчитывается более тысячи услуг, разбитых примерно на 20 категорий. Каждая услуга, представляющая собой маленький специализированный веб-сайт, поддерживается одной из независимых компаний. В основной пакет услуг входят такие категории, как электронная почта, новости, погода, спорт, игры, покупки, карты, гороскопы, развлечения, путешествия, путеводители, мелодии звонков, рецепты, азартные игры, доступ к банковским счетам, ставки акций. Такого рода услуги рассчитаны прежде всего на подростков и двадцатилетних молодых людей, которым обычно нравятся электронные безделушки, особенно если они имеют модную раскраску. Одно то, что более 40 компаний занимаются продажей такой «серьезной» продукции, как мелодии звонков, говорит о многом. Самой популярной услугой остается, разумеется, электронная почта, с помощью которой можно отправлять сообщения размером до 500 байт. Это, конечно, большой прогресс по сравнению с SMS (Short Message Service — Служба коротких сообщений), где размер сообщений не должен превышать 160 байт. Игры также популярны.

Существует более 40 000 веб-сайтов i-mode, однако для доступа к ним нужно набирать их URL, а не выбирать пункт меню. Можно сказать, что официальный список услуг — это некий интернет-портал, позволяющий выбрать один из сайтов с помощью пункта меню, без необходимости ввода URL.

NNT DoCoMo жестко контролирует официальные услуги. Чтобы попасть в список, услуга должна удовлетворять ряду известных критериев. Например,

«услуга не должна негативно влиять на общество», «в японско-английских словарях должно быть достаточно много слов», «услуги, предоставляющие мелодии для звонков, должны обновлять набор мелодий как можно чаще», а также «ни одна услуга не должна вести себя вызывающе или каким-либо мешать деятельности NNT DoCoMo» (Frengle, 2002). Остальные 40 000 сайтов могут делать что хотят, — это их личное дело.

Модель i-mode-бизнеса столь разительно отличается от привычного Интернета, что, наверное, стоит пояснить ее отдельно. Базовая ежемесячная абонентская плата составляет несколько долларов в месяц. Поскольку существует также плата за трафик, в базовый пакет услуг включено лишь небольшое число пакетов, которые можно передать бесплатно. Впрочем, пользователь может выбрать тариф с более высоким уровнем бесплатного трафика. Оплата в расчете на один пакет может резко снижаться при предоплате 10 Мбайт в месяц вместо 1 Мбайт. Дополнительный трафик можно оплачивать прямо через Интернет.

Для того чтобы пользоваться данной услугой, нужно стать ее абонентом. Это не слишком интеллектуальное действие: надо лишь щелкнуть на пункте меню и ввести свой PIN-код. Обычно ежемесячная плата за пользование большинством официальных услуг составляет $1–2 в месяц. NNT DoCoMo добавляет нужную сумму к счету за пользование телефоном. 91 % этой суммы уходит поставщику услуги, а 9 % телефонная компания оставляет себе. Если услуга, не являющаяся официальной, имеет миллион клиентов, ей необходимо каждый месяц рассылать миллион счетов, примерно по $1 каждый. Если же услуга входит в разряд официальных, все операции по счетам осуществляет NNT DoCoMo, производя, в частности, ежемесячное перечисление $910 000 на банковский счет услуги. Избавление от этих забот является хорошим стимулом для того, чтобы поставщики услуг становились официальными. Это выгодно и телефонной компании, поскольку сотрудничество с поставщиками официальных услуг ведет к повышению прибыли. Кроме того, доступ к официальной услуге осуществляется через меню, что на руку ее поставщику, поскольку пользователю найти такой сайт гораздо проще. В счет, предъявляемый абоненту, входит стоимость телефонных переговоров, плата за i-mode, услуги сайтов i-mode, а также за дополнительный трафик.

Несмотря на невероятный успех технологии i-mode в Японии, перспективы ее распространения в Европе и США до сих пор остаются неясными. Условия и предпосылки, существующие в это стране, во многом отличаются от западных.

Во-первых, большинство потенциальных клиентов на Западе (например, подростки, студенты колледжей, бизнесмены) имеют дома персональные компьютеры с большими мониторами и обеспечены доступом в Интернет со скоростью по крайней мере 56 Кбит/с, а чаще с гораздо более высокой. В Японии лишь немногие имеют дома персональные компьютеры с доступом в Интернет, частично из-за недостатка места для них, частично из-за заоблачных цен телефонных компаний на услуги местной связи (что-нибудь вроде $700 за установку линии и $1,50 за час местных разговоров). Для большинства японцев i-mode — это единственный способ доступа в Интернет.

Во-вторых, на Западе люди как-то не привыкли платить $1 в месяц за посещение сайта CNN, $1 за посещение Yahoo, еще $1 за Google, и т. д. И это еще не учитывая нескольких долларов за каждый загруженный мегабайт данных. Большинство провайдеров в западных странах взимают со своих клиентов фиксированную ежемесячную плату, не зависящую от реальных объемов использования линии. Надо отметить, что это является достойным ответом на требования пользователя.

В-третьих, большинство японцев пользуются i-mode по дороге в школу или на работу в метро или поезде. В Европе гораздо меньше людей, перемещающихся ежедневно на поезде, а в США таких почти нет. А использовать i-mode дома, имея под рукой компьютер с 17-дюймовым монитором, ADSL со скоростью 1 Мбит/с и огромным количеством свободных мегабайт на диске, как-то странно. Однако кто мог предвидеть, что мобильная связь станет такой популярной? Не исключено, что i-mode еще займет свою нишу на западном рынке телекоммуникаций.

Как мы уже говорили, аппараты i-mode используют уже имеющуюся сеть с коммутацией каналов для передачи речи и новую сеть с коммутацией пакетов для передачи данных. Последняя базируется на CDMA и передает 128-байтные пакеты со скоростью 9600 бит/с. Схема такой сети представлена на рис. 7.22. Телефон общается по беспроводному соединению со шлюзом преобразования протоколов при помощи протокола **LTP** (Lightweight Transport Protocol — упрощенный транспортный протокол). Этот шлюз, в свою очередь, общается с сервером i-mode по волоконно-оптическому соединению с высокой пропускной способностью. Наконец, сервер i-mode соединен со всеми доступными своему пользователю службами. Когда абонент выбирает один из элементов официального списка, отправляется запрос на сервер i-mode, в кэше которого хранится большинство страниц. Это помогает повысить производительность. Запросы страниц, отсутствующих в официальном списке, обходят сервер i-mode и попадают прямиком в Интернет.

В современный телефон обычно встраиваются процессор с частотой порядка 100 МГц, несколько мегабайт постоянной флэш-памяти, около 1 Мбайт оперативной памяти и небольшой экран. Для работы i-mode требуется экран размером, по крайней мере, 72×94 пиксела, однако некоторые высококачественные телефоны оснащены экранами размером 120×160 пикселов. При этом глубина цвета обычно составляет 8 бит (256 цветов). Этого, конечно, мало для отображения фотографий, однако вполне достаточно для векторных рисунков и несложной анимации. Поскольку мышь отсутствует, навигация по экрану осуществляется с помощью стрелок на клавиатуре.

Программная структура i-mode показана на рис. 7.23. На нижнем уровне расположена простая операционная система, управляющая оборудованием. Далее мы видим модуль реализации сетевой коммуникации, в котором применяется протокол LTP (собственный протокол NTT DoCoMo). Над ним расположен простой оконный менеджер, который поддерживает как текстовый режим, так и простой графический (GIF-файлы). Когда размер окна составляет в лучшем случае 120×160 пикселов, этот менеджер не особо перетруждается.

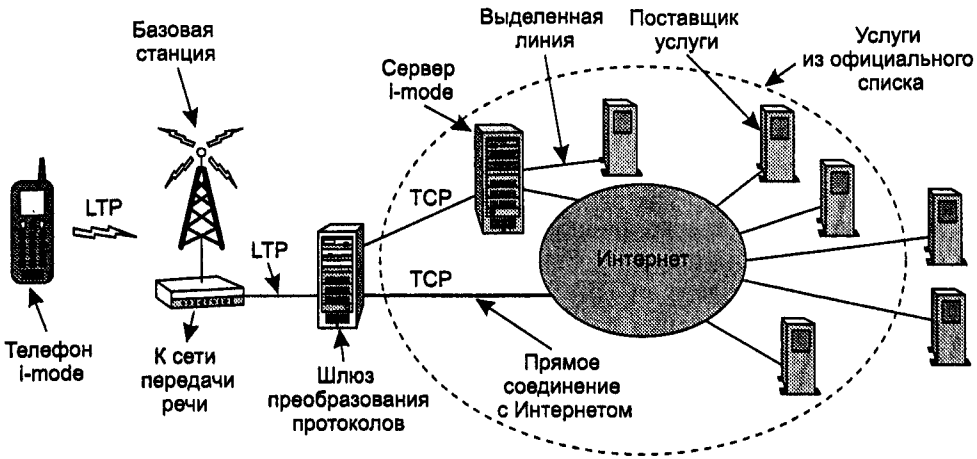


Рис. 7.22. Структура сети передачи данных i-mode и транспортные протоколы

Модуль взаимодействия с пользователем		
Подключаемые модули	Интерпретатор cHTML	Java
Простой оконный менеджер		
Сетевая коммуникация		
Операционная система реального времени		

Рис. 7.23. Программная структура i-mode

Четвертый уровень содержит интерпретатор веб-страниц (то есть браузер). В i-mode используется не полная версия HTML, а только часть этого языка под названием **cHTML** (compact HTML — компактный HTML), имеющий много общего с HTML 1.0. Этот уровень также содержит подключаемые модули и вспомогательные приложения наподобие браузеров обычных ПК. Одним из стандартных вспомогательных приложений является интепретатор для слегка модифицированной версии JVM.

На самом верхнем уровне расположен модуль взаимодействия с пользователем, который, как следует из его названия, реализует взаимодействие «человек—машина».

Рассмотрим язык cHTML чуть более подробно. Как уже говорилось, это почти что HTML 1.0 с некоторыми сокращениями и в то же время добавлениями, позволившими применять его в мобильных телефонах. Он был направлен консорциуму W3C для стандартизации, однако тот не проявил к нему особого интереса, поэтому cHTML пока что так и остается патентованным (в противоположность стандартизованному) продуктом.

В cHTML разрешено использование большинства базовых тегов обычного HTML, включая <html>, <head>, <title>, <body>, <h*n*>, , <ul>, <ol>, <menu>, <li>, <br>, <p>, <hr>, <img>, <form> и <input>. Теги <b> и <i> запрещены.

Для связывания с другими страницами разрешено использование <a>, однако добавляется схема *tel*. В сущности, схема *tel* аналогична *mailto*. При выборе ссылки с *mailto* браузер открывает форму для редактирования и отправки электронной почты адресату, указанному в ссылке. Если же гиперссылка снабжена *tel*, браузер набирает указанный телефонный номер. Так, например, телефонная книжка может содержать простые изображения, связанные с каждым из абонентов. При выборе какого-либо из них браузер может набрать номер абонента. Телефонные URL обсуждаются в RFC 2806.

Браузер cHTML, разумеется, имеет ряд ограничений. Он не поддерживает JavaScript, фреймы, таблицы стилей, цвета фона и фоновые изображения. Также отсутствует поддержка изображений в формате JPEG, поскольку на их распаковку требуется слишком много времени. Java-апплеты разрешены, однако их размер (в настоящее время) ограничен 10 Кбайт, поскольку скорость беспроводных соединений весьма невысока.

Убрав некоторые теги HTML, NTT DoCoMo одновременно добавил некоторые свои. Так, тег <blink> заставляет текст мигать. Добавление этого элемента кажется шагом несколько непоследовательным, если учитывать, что <b> был убран лишь потому, что внешний вид страницы в данном случае большой роли не играет, а <blink> связан не с чем иным, как с внешним видом. Тем не менее, так было сделано. Тег <marquee> позволяет отображать текст на экране в виде бегущей строки (на манер бегущей строки с курсами ценных бумаг на бирже или рекламы на вокзале).

Еще одно новшество — атрибут *align* тега <br>. Это оказалось необходимо, поскольку на экране размером 6 строк по 16 символов велика вероятность того, что слово будет разорвано посередине (как на рис. 7.24, *а*). *Align* позволяет несколько снизить эту вероятность, форматируя текст так, как показано на рис. 7.24, *б*. Интересно, что текст на японском языке в принципе не может страдать от проблемы переноса на новую строку. При отображении иероглифов весь экран разбивается на ячейки размером 9×10 пикселов или 12×12 пикселов, в зависимости от шрифта. В каждой такой ячейке помещается ровно один иероглиф, который означает не меньше, чем слово на любом европейском языке. Разрывы строк между словами разрешены при отображении японского языка всегда.

```
Куда мой мёд дев
аться мог? Ведь
был полнёхонек г
оршок! Он убежат
ь никак не мог
Ведь у него же н
```

а

```
Куда мой мёд
деваться мог?
Ведь был
полнёхонек
горшок! Он
убежать никак н
```

б

Рис. 7.24. Алан Милн сталкивается с проблемой экрана 16×6

Несмотря на то что в японском языке десятки тысяч иероглифов, компания NNT DoCoMo изобрела еще 166 собственных, называемых **emoji**. Они обладают высокой детализацией, а по сути аналогичны смайликам, представленным в табл. 7.2. Среди них есть символы астрологических знаков, пива, гамбургера, парка развлечений, символы дня рождения, мобильного телефона, собаки, кошки, Рождества, разбитого сердца, поцелуя, значки для различных настроений, спящая рожица и, конечно же, символ прекрасного чувства.

Еще один новый атрибут служит для того, чтобы пользователь мог выбрать пункт меню с помощью клавиатуры, что, несомненно, чрезвычайно важно для компьютера без мыши. Пример его использования в cHTML показан в листинге 7.16.

Листинг 7.16. Пример страницы на языке cHTML

```
<html>
<body>
<h1> Выберите:</h1>
<a href="messages.chtml" accesskey="1"> Проверить голосовую почту </a><br>
<a href="mail.chtml" accesskey="2"> Проверить E-mail </a><br>
<a href="games.chtml" accesskey="3"> Игра </a>
</body>
</html>
```

Хотя активность клиентской стороны несколько ограничена, сервер i-mode представляет собой полноценный компьютер, со всеми привычными штучками. Он поддерживает CGI, Perl, PHP, JSP, ASP и вообще все, что поддерживает любой нормальный веб-сервер.

Краткое сравнение первого поколения систем WAP и i-mode приведено в табл. 7.15. Некоторые аспекты сравнения кажутся не слишком значительными, однако зачастую они оказываются весьма важными. Например, у пятнадцатилетних подростков обычно нет кредитных карт, поэтому возможность включения оплаты покупок в электронных магазинах в телефонные счета радикально меняет их отношение к этой технологии. Более подробную информацию об i-mode можно узнать из книг (Frengle, 2002; Vacca, 2002).

Таблица 7.15. Сравнение первого поколения систем WAP и i-mode

Свойство	WAP	i-mode
Определение	Стек протоколов	Услуга
Устройства	Мобильный телефон, PDA, ноутбук	Мобильный телефон
Доступ	Наборный (dial-up)	Постоянное подключение
Базовая сетевая технология	Коммутация каналов	Две: коммутация каналов и коммутация пакетов
Скорость передачи данных	9600 бит/с	9600 бит/с
Экран	Монохромный	Цветной
Язык разметки	WML (Приложение XML)	cHTML
Язык написания сценариев	WMLscript	Отсутствует
Оплата	Поминутная	За трафик

Свойство	WAP	i-mode
Оплата покупок	Кредитной картой	Включение в телефонный счет
Пиктограммы	Отсутствуют	Присутствуют
Стандартизация	Открытый стандарт форума WAP	Собственная разработка NNT DoCoMo
Зона использования	Европа, Япония	Япония
Типичный пользователь	Бизнесмен	Девушка

Второе поколение беспроводных веб-технологий

Технология WAP 1.0, базировавшаяся на известных международных стандартах, должна была стать мощным инструментом для деловых людей, которым приходится часто совершать поездки. Эта идея провалилась. Технология i-mode была создана в качестве игрушки для японских подростков и базировалась на стандартах, совершенно не известных за пределами Японии. И она оказалась чрезвычайно популярной. Что произошло после этого? Обе стороны извлекли уроки из первого опыта создания беспроводных веб-систем. Консорциум WAP убедился в том, что содержимое, как ни странно, имеет значение, и малое число сайтов, написанных на совместимом с системой языке разметки, губительно. Компания NNT DoCoMo осознала, что закрытая система, слишком тесно связанная с крошечными телефонами и японским образом жизни, не является оптимальной в качестве экспортного продукта. Разработчики обеих систем пришли к выводу о том, что необходим открытый стабильный универсальный язык разметки. Он позволил бы представлять множество сайтов в совместимом формате. Наконец-то производители поняли, что войны форматов не способствуют успеху в бизнесе.

Вот-вот должно появиться второе поколение обеих беспроводных веб-технологий. Версия WAP 2.0 появилась первой, поэтому мы рассмотрим ее в качестве примера. Кое-что было унаследовано из WAP 1.0, поскольку некоторые идеи были весьма хороши. Во-первых, WAP может работать на базе разных сетей. В первом поколении использовалась коммутация каналов, однако всегда существовала и остается возможность использования коммутации пакетов. Второе поколение в большей мере обращается к коммутации пакетов. Взять для примера хотя бы GPRS. Во-вторых, WAP всегда поддерживал и продолжает поддерживать широкий спектр устройств, от мобильных телефонов до мощных портативных компьютеров.

WAP 2.0 обладает некоторыми новыми чертами:

1. Пассивная модель приема сообщений наряду с активной моделью опроса сообщений.
2. Поддержка приложений с интегрированной телефонией.
3. Поддержка мультимедийных сообщений.
4. Наличие 264 пиктограмм.
5. Интерфейс с устройствами долговременного хранения информации.
6. Поддержка модулей, подключаемых к браузеру.

Активная модель опроса давно известна: клиент запрашивает страницу и получает ее. Что касается пассивной модели, то она подразумевает получение данных без запроса (например, клиент может получать непрерывный поток сообщений о ставках на ценные бумаги или состоянии трафика).

Технологии передачи данных и речи постепенно сливаются воедино, и WAP 2.0 реализует многие из них. Мы уже имели возможность убедиться в том, что это реально, на примере i-mode, где гиперссылка может представлять собой текст (или изображение), связанный с телефонным номером. Наряду с электронной почтой и телефонией имеется поддержка мультимедийных сообщений.

Невероятная популярность японских смайликов, emoji, не могла оставить равнодушным консорциум WAP, и в WAP 2.0 появились 264 новые пиктограммы. Они разбиты на категории, среди которых есть следующие: животные, приборы, одежда, эмоции, кулинария, человек, пол, карты, музыка, производство, спорт, время, инструменты, транспорт, оружие и погода. Получилось довольно забавно: стандарт описывает только названия пиктограмм (например, «сонная рожица», «крепкие объятия»); сами же картинки не приводятся. Наверное, причиной тому стала боязнь оскорбить представителей какой-нибудь культуры своим пониманием того, как выглядит «сонный человек» или «крепкие объятия». В i-mode такой проблемы не существовало, поскольку система была рассчитана только на одну страну и одну культуру.

Предоставление интерфейса с устройствами долговременного хранения данных не означает, что отныне все телефоны стандарта WAP 2.0 будут снабжаться большим жестким диском. Флэш-память тоже является устройством хранения информации. Например, беспроводная WAP-камера может использовать ее для временного сохранения изображений перед размещением лучших снимков в Интернете.

Наконец, теперь можно расширять возможности браузера за счет подключаемых модулей. Появилась поддержка языков написания скриптов.

Имеются также некоторые технические отличия WAP 2.0. Два наиболее серьезных из них связаны со стеком протоколов и языком разметки. WAP 2.0 продолжает поддерживать как старый стек протоколов WAP (см. рис. 7.20), так и стандартный стек интернет-протоколов TCP и HTTP/1.1. В целях некоторого упрощения кода в TCP были внесены четыре небольших изменения (не угрожающие совместимости): 1) используется фиксированное окно размером 64 Кбайи; 2) отсутствует затяжной запуск; 3) максимальное значение MTU равно 1500 байт; 4) слегка изменен алгоритм пересылки данных. TLS является стандартизованным IETF протоколом транспортного уровня с защитой информации; мы будем рассматривать его в главе 8. Многие устройства, судя по всему, будут поддерживать оба стека протоколов, показанных на рис. 7.25.

Еще одно техническое отличие WAP 2.0 от WAP 1.0 связано с языком разметки. WAP 2.0 поддерживает базовый язык XHTML Basic, предназначенный для малых беспроводных устройств. Так как японская телефонная компания NNT DoCoMo согласилась использовать это же подмножество, веб-дизайнеры могут пользоваться этим форматом, не заботясь о том, чтобы их страницы работали как в фиксированном Интернете, так и на всех беспроводных устройствах. Эти решения должны положить конец войнам форматов, которые были настоящей угрозой развитию беспроводных веб-технологий.

Рис. 7.25. WAP 2.0 поддерживает два стека протоколов

Теперь, наверное, уместно сказать несколько слов об HTML Basic. Он предназначен для мобильных телефонов, телевизоров, PDA, торговых автоматов, пейджеров, автомобильных компьютеров, электронных игр и даже часов. В связи с этим в нем отсутствует поддержка каких бы то ни было таблиц стилей, скриптов, фреймов. Однако большинство стандартных тегов реализовано. Они сгруппированы в 11 модулей. Некоторые являются обязательными, некоторые — нет. Все они определены в XML. Модули и некоторые примеры тегов перечислены в табл. 7.16. Мы не стали перечислять все теги: более полную информацию можно найти на сайте www.w3.org.

Таблица 7.16. Модули и теги XHTML Basic

Модуль	Необходимость	Функция	Примеры тегов
Структура	Да	Структура документа	`body`, `head`, `html`, `title`
Текст	Да	Информация	`br`, `code`, `dfn`, `em`, `hn`, `kbd`, `p`, `strong`
Гипертекст	Да	Гиперссылки	`a`
Списки	Да	Титульный список	`dl`, `dt`, `dd`, `ol`, `ul`, `li`
Формы	Нет	Заполнение форм	`form`, `input`, `label`, `option`, `textarea`
Таблицы	Нет	Прямоугольные таблицы	`caption`, `table`, `td`, `th`, `tr`
Изображения	Нет	Размещение изображений	`img`
Объекты	Нет	Апплеты, карты и т. д.	`object`, `param`

продолжение

Таблица 7.16 (*продолжение*)

Модуль	Необходимость	Функция	Примеры тегов
Метаинформация	Нет	Дополнительная информация	meta
Ссылка	Нет	Аналогично <a>	link
База	Нет	Точка отсчета URL	base

Несмотря на принятое соглашение об использовании XHTML Basic, над WAP и i-mode нависает угроза под кодовым названием *802.11*. Предполагается, что второе поколение беспроводных веб-технологий будет работать на скорости 384 Кбит/с. Это куда лучше, чем 9600 бит/с, однако все же несколько медленнее, чем 11 или 54 Мбит/с (стандарт 802.11). Разумеется, сети 802.11 установлены далеко не везде, однако появляется все больше ресторанов, отелей, магазинов, компаний, аэропортов, автобусных станций, музеев, университетов, больниц и других организаций, устанавливающих у себя базовые станции для своих работников и посетителей. Уже очень скоро настанет момент, когда в городской местности появится новая мода: пройти пару кварталов и зайти в кафе, чтобы выпить чашечку кофе и проверить электронную почту. В различных заведениях и конторах рядом с логотипами принимаемых к оплате кредитных карт уже пора помещать логотип 802.11. Это, несомненно, привлечет клиентов. На картах города (которые, кстати, можно загрузить из Интернета) появятся зеленые и красные зоны: охваченные и не охваченные базовыми станциями 802.11 соответственно. И люди будут перемещаться из одной зоны покрытия в другую, как кочевники, бредущие по пустыне от оазиса к оазису.

Тем не менее, если в ресторанах базовые станции появятся довольно скоро, в сельской местности фермеры, скорее всего, не будут торопиться устанавливать 802.11. Поэтому зоны действия таких сетей будут неоднородными и ограниченными, в основном, центрами городов (из-за небольшого радиуса действия базовых станций — в лучшем случае несколько сотен метров). Это может привести к появлению двухрежимных беспроводных устройств, работающих с сетью 802.11, а при отсутствии оной — с WAP.

Мультимедиа

Беспроводные веб-технологии — это, конечно, замечательное изобретение последних лет, однако далеко не единственное. Для многих не что иное, как мультимедиа, служит чашей святого Грааля сетевых технологий. Слово «мультимедиа» возбуждает и физиков, и лириков, и разработчиков, и коммерсантов. Одни видят в мультимедиа бесконечный источник интересных технических проблем, связанных, например, с доставкой (интерактивного) видео по заказу, другие — не меньший источник прибыли. Поскольку для передачи мультимедийных данных требуется очень высокая пропускная способность, довольно трудно заставить систему работать даже поверх стационарных соединений. Что касается беспроводных технологий, то добиться даже VHS-качества пока что не удается, и на решение этой проблемы потребуется, как минимум, несколько лет. Мы будем рассматривать далее методы передачи мультимедийных данных по проводным сетям.

Буквально мультимедиа означает использование двух или более средств аудиовизуальной информации. Если бы издатель этой книги захотел присоединиться к всеобщей моде пускания пыли в глаза, он мог бы разрекламировать ее как использующую мультимедийные технологии. В конце концов, в ней действительно используются два средства информации: текст и графика (рисунки). Тем не менее, когда употребляется это слово, обычно имеются в виду некие информационные сущности, *длящиеся во времени*, то есть проигрываемые в течение определенного интервала времени, обычно во взаимодействии с пользователем. На практике чаще всего это аудио и видео, то есть звук плюс движущееся изображение.

Несмотря на такое довольно понятное определение, многие, говоря «мультимедиа», имеют в виду только чисто звуковую информацию, например интернет-телефонию или интернет-радио. Строго говоря, они не правы. Более удачным термином в данном случае является словосочетание **потоковая информация** (streaming media), однако мы, поддавшись стадному инстинкту, все же будем именовать аудиоданные, передающиеся в реальном масштабе времени, «мультимедиа». В следующих разделах мы узнаем, как компьютер обрабатывает звук и видео и как он сжимает такого рода данные. Затем мы рассмотрим некоторые сетевые технологии, связанные с мультимедиа. Довольно понятно и в хорошем объеме (три тома) взаимодействие сетевых и мультимедийных технологий описано в (Steinmetz и Nahrstedt, 2002; Steinmetz и Nahrstedt, 2003a; Steinmetz и Nahrstedt, 2003b).

Основы цифровой обработки звука

Звуковая волна представляет собой одномерную акустическую волну (волну давления). Когда такая волна достигает уха, барабанная перепонка начинает вибрировать, вызывая вибрацию тонких костей внутреннего уха, в результате чего в мозг по нерву посылается пульсирующий сигнал. Эта пульсация воспринимается слушателем как звук. Подобным образом, когда акустическая волна воздействует на микрофон, им формируется электрический сигнал, представляющий собой амплитуду звука как функцию времени. Представление, хранение, обработка и передача подобных аудиосигналов — именно эти вопросы рассматриваются при изучении мультимедийных систем.

Человеческое ухо способно слышать сигналы в диапазоне частот от 20 до 20 000 Гц, хотя некоторые животные, например собаки, могут слышать и более высокие частоты. Громкость, воспринимаемая ухом, изменяется логарифмически по отношению к частоте, поэтому сила звука обычно измеряется в логарифмах отношения амплитуд. Единицей измерения служит **децибел (дБ)**:

$$1 \text{ дБ} = 20 \log_{10}(A/B).$$

Если принять нижний порог слышимости (давление около 0,0003 дин/см^2, что равно $3 \cdot 10^{-5}$ Па) для синусоидальной волны частотой 1 кГц за 0 дБ,

то громкость обычного разговора будет соответствовать 50 дБ, а болевой порог наступит при силе звука около 120 дБ, что соответствует отношению амплитуд, равному 1 миллиону.

Человеческое ухо удивительно чувствительно к изменениям звука, длящимся всего несколько миллисекунд. Глаз, напротив, не в состоянии заметить такие кратковременные изменения. Таким образом, флуктуация (джиттер) в несколько миллисекунд при передаче мультимедиа влияет в большей степени на качество звука, чем на качество изображения.

Звуковые волны можно преобразовывать в цифровую форму при помощи **аналого-цифрового преобразователя (АЦП)**. На вход АЦП подается электрическое напряжение, а на выходе формируется двоичное число. На рис. 7.26, *а* показан пример синусоидальной волны. Чтобы представить этот сигнал в цифровом виде, мы можем измерять значения сигнала (отсчеты) через равные интервалы времени ΔT, как показано на рис. 7.26, *б*. Если звуковая волна не является чисто синусоидальной, а представляет собой сумму нескольких синусоидальных волн и самая высокая частота ее составляющих равна f, тогда, согласно теореме Найквиста (см. главу 2), для последующего восстановления сигнала достаточно измерять значения сигнала с частотой дискретизации $2f$. Производить замеры сигнала с большей частотой нет смысла, так как более высокие частоты отсутствуют в сигнале.

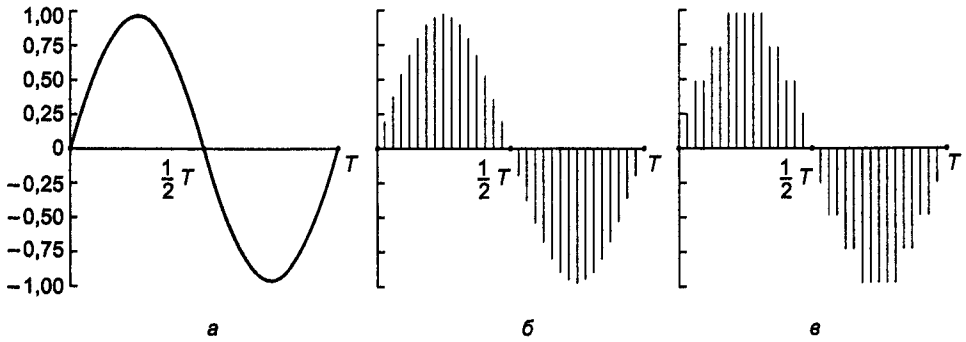


Рис. 7.26. Синусоидальная волна (*а*); дискретизация синусоидальной волны (*б*); квантование отсчетов 4 битами (*в*)

Оцифрованные отсчеты (сэмплы) никогда не бывают точными. Например, отсчеты на рис. 7.26, *в* могут принимать только 9 значений — от −1,00 до +1,00 с шагом 0,25. При 8-битовом квантовании каждый отсчет может принимать одно из 256 различных значений. При 16 битах на отсчет можно кодировать сигнал с еще более высокой точностью, так как каждому значению сигнала можно сопоставить одно из 65 536 различных значений. Ошибка, возникающая в результате неточного соответствия квантованного сигнала, способного принимать конечное число значений, исходному сигналу, называется **шумом квантования**. При недостаточном количестве битов, которыми представляется каждый отсчет сигнала, этот шум может быть настолько велик, что будет различим на слух как искажение исходного сигнала или как посторонние шумы.

Двумя хорошо известными примерами использования цифрового звука являются телефон (если применяются новые цифровые АТС) и аудио-компакт-диски. В кодово-импульсной модуляции, применяемой в телефонной системе, используются восьмибитовые отсчеты, замеряемые 8000 раз в секунду. В Северной Америке и Японии семью битами кодируются данные, а восьмой бит является служебным; в европейских же системах все 8 бит отводятся для данных. Таким образом, скорость передачи данных составляет 56 000 или 64 000 бит/с. При частоте дискретизации в 8 кГц частотные составляющие сигнала выше 4 кГц теряются.

Аудио-компакт-диски содержат звуковой сигнал, оцифрованный с частотой дискретизации 44 100 Гц, в результате чего они могут хранить звуки с частотами до 22 кГц, что воспринимается как достаточно качественный звук людьми, но считается весьма низким качеством среди собак, ценящих хорошую музыку. Каждому отсчету выделяется 16 бит, его значение пропорционально амплитуде сигнала. Обратите внимание на то, что 16-битовый отсчет может принимать всего 65 536 различных значений, хотя измерения показывают, что динамический диапазон человеческого уха составляет около 1 миллиона значений. Таким образом, использование 16 бит на отсчет дает некоторый шум квантования (хотя полный динамический диапазон и не охвачен, качество звучания компакт-дисков обычно не вызывает нареканий). При 44 100 отсчетах в секунду по 16 бит каждый аудио-компакт-диску требуется пропускная способность в 705,6 Кбит/с для монофонического сигнала и 1,411 Мбит/с — для стереофонического. Хотя это и меньше, чем требуется для передачи видеосигнала (см. далее), передача звука в таком (несжатом) формате в реальном времени займет канал T1 почти целиком.

Цифровой звук легко обрабатывается компьютерным программным обеспечением. Существуют десятки программ для персональных компьютеров, позволяющие пользователям записывать, воспроизводить, редактировать, микшировать и хранить звук. Сегодня вся профессиональная звукозапись и редактирование звука осуществляются в цифровом виде.

Музыка представляет собой лишь частный случай звука, хотя и очень важный. Помимо музыки, другим важным частным случаем мультимедиа является передача речи. Ее достаточно осуществлять в диапазоне частот от 600 до 6000 Гц. Речь состоит из гласных и согласных звуков, обладающих разными свойствами. Гласные звуки производятся при открытом голосовом тракте, при этом воздух, проходя через гортань, резонирует с частотой, определяемой размерами и формой голосовой системы и положением языка и челюсти говорящего. Гласные звуки являются почти периодичными с длительностью периода около 30 мс. Согласные звуки производятся при частично блокированном голосовом тракте. Эти звуки имеют не столь регулярную структуру, как гласные.

Некоторые системы воспроизведения и передачи речи используют модели голосовой системы для сведения речи к небольшому набору параметров (например, размерам и формам различных полостей) вместо того, чтобы просто дискретизировать звуковой сигнал речи. Однако рассмотрение устройства этих голосовых кодеров выходит за рамки данной книги.

Сжатие звука

Итак, как мы уже знаем, для передачи звука с качеством аудио-компакт-дисков требуется пропускная способность, равная 1,411 Мбит/с. Понятно, что для практической передачи подобных данных через Интернет требуется значительное сжатие. Для этого были разработаны различные алгоритмы сжатия оцифрованного звука. Одним из самых популярных форматов является аудио-MPEG, имеющий три уровня (разновидности). Самым известным и качественным является **MP3** (MPEG layer 3 — MPEG 3-го уровня). В Интернете можно найти огромное количество записей в MP3, не все из которых на самом деле являются легальными. Это привело к множеству судебных разбирательств, инициированных ущемленными в своих законных правах артистами и обладателями авторских прав. MP3 — это часть стандарта MPEG, предназначенного для сжатия видеосигнала. Методы сжатия движущихся изображений мы рассмотрим позднее в этой главе, а сейчас обратимся к сжатию звука.

Существуют две концепции сжатия звука. При **кодировании формы сигналов** сигнал раскладывается на компоненты при помощи преобразования Фурье. На рис. 2.1, *а* показан пример в виде временной функции и амплитуд, получающихся в результате ее разложения в ряд Фурье. Амплитуда каждого компонента кодируется с минимальными искажениями. Задачей является максимально аккуратная передача формы сигнала с минимально возможной затратой битов.

Другая концепция называется **перцепционным кодированием**. Она основана на некоторых недостатках слухового аппарата человека, позволяющих шифровать сигнал таким образом, что слушатель не ощутит никакой разницы по сравнению с настоящим сигналом, хотя на осциллографе эта разница будет весьма заметна. Наука, на которой базируется перцепционное кодирование, называется **психоакустикой**. Она изучает восприятие звука человеком. Формат MP3 использует перцепционное кодирование.

Ключевым свойством перцепционного кодирования является то, что одни звуки могут **маскировать** другие. Представьте себе, что теплым летним вечером вы медитируете на лужайке, слушая живой концерт для флейты с оркестром. Затем, откуда ни возьмись, появляется бригада рабочих с отбойными молотками в руках, которая начинает вскрывать асфальт на близлежащей улице. Расслышать флейту, к сожалению, уже никто не в состоянии. Нежные звуки, издаваемые ею, подверглись *маскированию* звуками отбойных молотков. Если рассматривать ситуацию с точки зрения передачи данных, то в этот момент достаточно кодировать лишь диапазон частот, в котором работают отбойные молотки, — все равно флейту за этим грохотом не слышно. Способность громких звуков определенного диапазона частот «прятать» более тихие звуки других диапазонов (которые были бы слышны при отсутствии громких звуков) называется **частотным маскированием**. На самом деле, даже после того как рабочие выключат отбойные молотки, слушатели не будут слышать флейту в течение некоторого небольшого периода времени. Это связано с тем, что при появлении очень громкого звука коэффициент усиления человеческого уха резко снизился, и после прекращения работы отбойных молотков требуется время для его возвращения в нормальное состояние. Этот эффект называется **временным маскированием**.

Чтобы перейти от качественного описания этих эффектов к количественным, представим себе проведение некого эксперимента 1. Человек, находящийся в тихом помещении, надевает наушники, соединенные со звуковой картой компьютера. Компьютер генерирует звук (чистую синусоидальную звуковую волну) с частотой 100 Гц, сила которого постепенно возрастает. Испытуемый должен нажать клавишу на клавиатуре, как только он услышит звук. Компьютер запоминает силу звука, при которой была нажата клавиша, и повторяет эксперимент на частотах 200 Гц, 300 Гц и т. д., доходя до верхнего предела слышимых частот. Эксперимент необходимо провести над большим количеством испытуемых. На рис. 7.27, *а* показан график с логарифмическим масштабом на обеих осях, показывающий усредненную зависимость порога слышимости от частоты звука. Наиболее очевидный вывод, который можно сделать при взгляде на эту кривую, состоит в том, что нет никакой необходимости когда бы то ни было кодировать частоты, амплитуда которых ниже порога слышимости. Например, если сила звука на частоте 100 Гц равна 20 дБ, этот звук можно не кодировать, и качество звучания при этом не ухудшится, так как уровень 20 дБ при 100 Гц находится ниже порога слышимости (рис. 7.27, *а*).

Теперь рассмотрим эксперимент 2. Пусть компьютер повторяет действия эксперимента 1, но на этот раз на каждую тестовую частоту будет накладываться синусоидальная звуковая волна постоянной амплитуды с частотой, скажем, 150 Гц. Мы обнаружим, что порог слышимости для частот, расположенных вблизи 150 Гц, резко возрастает. Это отражено на графике на рис. 7.27, *б*.

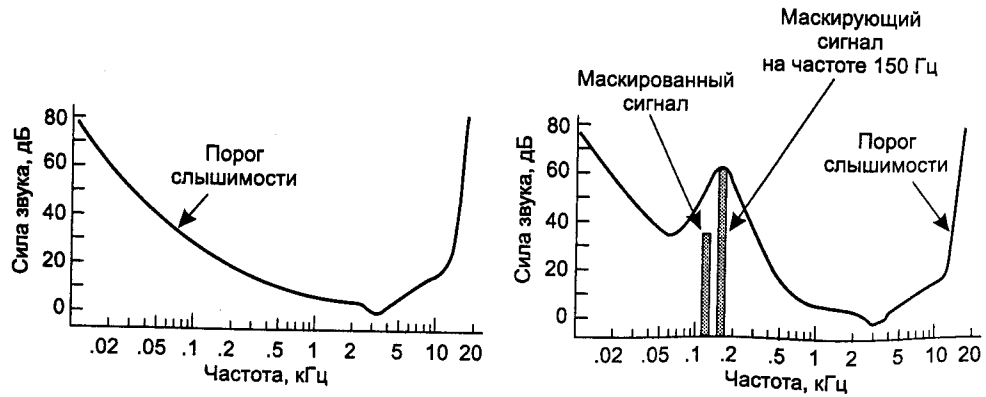


Рис. 7.27. Порог слышимости как функция частоты (*а*); эффект маскирования (*б*)

Из последнего наблюдения можно сделать следующий вывод: зная, какие сигналы маскируются более мощными сигналами на близлежащих частотах, мы можем пренебречь соответствующими частотами и не кодировать их, экономя тем самым биты. Из рис. 7.27, *б* очевидно, что сигналом с частотой 125 Гц можно полностью пренебречь, и никто не заметит разницы. Знание свойств временного маскирования позволяет даже после прекращения звучания громкого сигнала

в каком бы то ни было частотном диапазоне в течение некоторого времени (пока ухо настраивается на меньшую мощность звука) продолжать пренебрегать кодированием этой частоты. Суть алгоритма MP3 состоит в разложении сигнала в ряд Фурье для получения силы звука на каждой из частот с последующей передачей исключительно немаскированных частот, кодируемых минимально возможным числом бит.

Теперь, зная основной принцип, мы можем рассмотреть, как производится само кодирование. Сжатие звука выполняется путем замеров формы сигналов, производимых с частотой 32 000, 44 100 или 48 000 раз в секунду. Замеры могут сниматься по одному или двум каналам в одной из четырех комбинаций:

1. Монофонический звук (один входной поток).
2. Двойной монофонический звук (например, звуковая дорожка на английском и японском).
3. Разъединенное стерео (каждый канал сжимается отдельно).
4. Объединенное стерео (учитывается межканальная избыточность сигнала).

Для начала выбирается желаемая выходная битовая скорость. С помощью алгоритма MP3 можно сжать записанную на компакт-диск стереофоническую запись рок-н-ролла до 96 Кбит/с с потерей качества, едва заметной даже для фанатов рок-н-ролла, не лишенных слуха. Если мы хотим «перегнать в MP3» фортепианный концерт, нам понадобится битовая скорость по крайней мере 128 Кбит/с. Чем обусловлена такая разница? Дело в том, что соотношение сигнал/шум в рок-н-ролле гораздо выше, чем в фортепианном концерте (только в техническом смысле, разумеется). Можно, впрочем, выбрать меньшую битовую скорость и получить более низкое качество воспроизведения.

После этого отсчеты обрабатываются группами по 1152 (что занимает около 26 мс). Каждая группа предварительно проходит через 32 цифровых фильтра, выделяющих 32 частотных диапазона. Одновременно входной сигнал заводится в психоакустическую модель для определения маскирующих частот. Затем каждый из 32 частотных диапазонов преобразуется с целью получения более точного спектрального разрешения.

Следующим шагом является распределение имеющегося запаса бит между частотными диапазонами. При этом большее число бит отводится под диапазон с наибольшей немаскированной спектральной мощностью, меньшее — под немаскируемые диапазоны с меньшей спектральной мощностью, и совсем не отводятся биты под маскируемые диапазоны. Наконец, битовые последовательности шифруются с помощью кода Хаффмана (Huffman), который присваивает короткие коды числам, появляющимся наиболее часто, и длинные — появляющимся редко.

На самом деле, эта тема далеко не исчерпана. Существуют методы шумоподавления, сглаживания сигналов, использования межканальной избыточности (при наличии такой возможности), однако все это, к сожалению, невозможно охватить в рамках нашей книги. Более формально изложенные математические основы этих процессов даются в книге (Pan, 1995).

Потоковое аудио

Перейдем от технологии оцифровки звука к трем сетевым приложениям, использующим ее. Первое, что мы рассмотрим, будет потоковое аудио, то есть прослушивание звукозаписей через Интернет. Это иногда называется «музыкой по заказу». Затем мы познакомимся с интернет-радио и технологией передачи речи поверх IP.

В Интернете есть множество музыкальных веб-сайтов, на многих из которых представлены списки песен, которые пользователь может прослушать, щелкнув на названии. Подобные услуги на некоторых сайтах бесплатны (например, таким образом молодая группа может рекламировать свою деятельность и искать своего слушателя); иногда за прослушивание взимается плата, хотя при этом чаще всего есть возможность бесплатно ознакомиться с композицией (например, первые 15 секунд записи). Наиболее распространенный способ реализации такой системы «музыки по заказу» показан на рис. 7.28.

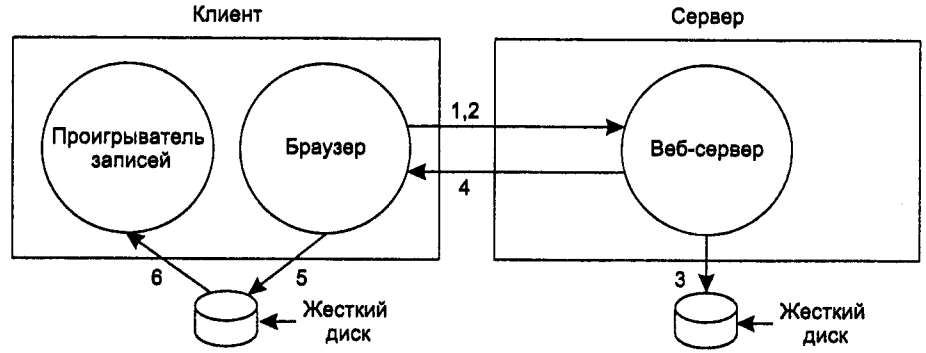


Рис. 7.28. Простейший способ реализации технологии «музыки по заказу» на веб-сайте

Процесс начинается, когда пользователь щелкает на названии песни. Вначале вступает в дело браузер. Первый шаг заключается в установке TCP-соединения с веб-сервером, на который указывает гиперссылка. На втором шаге ему направляется HTTP-запрос *GET*, содержащий заявку на получение файла. Шаги 3 и 4 выполняются сервером: он получает песню (которая представляет собой обычный файл в формате MP3 или каком-нибудь другом) с диска и отсылает ее браузеру. Если размер файла превышает объем памяти сервера, он может извлекать и отправлять его поблочно.

По MIME-типу файла (например, *audio/mp3*) или по расширению браузер определяет способ его воспроизведения. Обычно запускается вспомогательное приложение, ассоциированное с данным типом файлов, например, RealOne Player, Windows Media Player или Winamp. Обычно браузер общается со вспомогательным приложением, записывая информацию, предназначенную для воспроизведения, во временный файл. Поэтому до начала проигрывания весь музыкальный файл будет записан во временный файл на диске (шаг 5). После этого будет запущен проигрыватель, которому браузер передаст имя временного файла. Шаг 6

заключается в поблочном считывании данных из файла и непосредственном воспроизведении музыки.

В принципе, такой подход совершенно корректен, и музыку пользователь услышит. Единственная серьезная проблема заключается в том, что вся запись должна быть предварительно передана по сети. Если музыкальный файл занимает 4 Мбайт (типичный размер аудиофайла с песней в формате MP3) и передается при помощи модема со скоростью 56 Кбит/с, пользователь будет наслаждаться тишиной в течение почти 10 минут, прежде чем услышит музыку. Далеко не все меломаны приходят от этого в восторг. К тому же не стоит забывать, что после окончания данной песни следующую можно будет услышать также только через 10 минут.

Как обойти эту проблему, не внося изменений в работу браузера? Музыкальные сайты пришли к следующей схеме. Файл, связанный гиперссылкой с названием песни, на самом деле является не музыкальным, а **метафайлом**. Метафайл обычно очень короткий, в типичной ситуации он состоит всего лишь из одной текстовой строки, которая выглядит примерно так:

```
rtsp://joes-audio-server/song-0025.mp3
```

Получив такой однострочный файл, браузер записывает его во временный файл, запускает в качестве вспомогательного приложения проигрыватель и передает ему, как обычно, имя временного файла. Проигрыватель видит, что временный файл содержит URL. Он соединяется с сервером *joes-audio-server* и запрашивает песню. Вы, должно быть, уже заметили, что браузер не принимает участия в этом процессе.

В большинстве случаев сервер, указанный в метафайле, не совпадает с веб-сервером, содержащим ссылку на метафайл. Более того, обычно это даже не HTTP-сервер, а специализированный мультимедийный сервер. В приведенном примере этот сервер использует протокол **RTSP**, это становится понятно при взгляде на ссылку — она начинается с названия схемы, *rtsp*. RTSP описан в RFC 2326.

Проигрыватель мультимедиа решает следующие 4 основные задачи:

1. Управление интерфейсом пользователя.
2. Обработка ошибок передачи.
3. Распаковка сжатых аудиоданных.
4. Устранение флуктуации (джиттера).

Большинство современных программ для воспроизведения мультимедиа имеют привлекательный интерфейс. Часто внешний вид панелей управления **(оболочек)** можно менять. Как мы уже сказали, в задачи проигрывателя входит общение с пользователем.

Вторая его задача — обработка ошибок. При передаче музыки в реальном времени редко используется протокол TCP, так как ошибки и повторные передачи могут приводить к недопустимо долгим паузам. Вместо этого передача осуществляется по протоколу типа RTP, который мы изучали в главе 6. Подобно большинству протоколов реального времени, RTP работает поверх UDP, то есть

пакеты могут теряться по пути. Проблему потерянных пакетов решает проигрыватель.

Иногда для упрощения обработки ошибок при передаче музыки применяется принцип чередования. Например, пакет может содержать 220 стереоотсчетов, каждый из которых содержит пару 16-разрядных чисел. Этого вполне достаточно для 5 мс звучания музыки. Однако протокол может посылать в одном пакете все нечетные отсчеты для 10 мс звучания, а в следующем — все четные отсчеты для того же интервала. Таким образом, потеря одного пакета не приведет к возникновению паузы длиной 5 мс, вместо этого будет потерян каждый второй отсчет 10-миллисекундного интервала. Эта утрата может быть восполнена, если проигрыватель произведет интерполяцию, используя предыдущий и следующий отсчеты (относительно каждого потерянного). Таким образом, примерные значения будут восстановлены.

Принцип чередования, используемый для восстановления ошибок, показан на рис. 7.29. Здесь видно, что каждый пакет содержит чередующиеся отсчеты для 10-миллисекундного интервала. В результате потери пакета 3 (см. рисунок) не возникает паузы, а только лишь на некоторое время снижается частота следования отсчетов. Утерянные значения путем интерполяции могут быть восстановлены; это обеспечит непрерывность звучания. Данная конкретная схема работает только с несжатыми отсчетами, однако она показывает, как при помощи хитрого алгоритма кодирования превращать потерянные пакеты не в раздражающие паузы, а в непрерывные интервалы с пониженным качеством. Между тем в RFC 3119 описан метод, позволяющий применять аналогичную процедуру к сжатым аудиоданным.

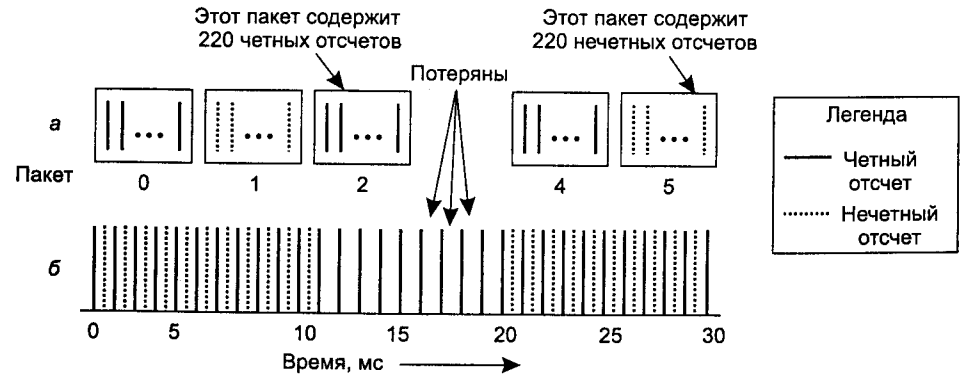


Рис. 7.29. Когда пакеты содержат чередующиеся отсчеты, потеря пакета уменьшает частоту следования отсчетов, но не приводит к возникновению паузы

Третья задача программы-проигрывателя заключается в декодировании сжатых аудиоданных. Несмотря на высокую требуемую интенсивность вычислений, применяемый метод довольно очевиден.

Четвертая задача — устранение флуктуации (джиттера), бича всех систем реального времени. Все системы воспроизведения потокового аудио перед началом

проигрывания буферизируют около 10–15 с звучания, как показано на рис. 7.30. В идеале — сервер должен продолжать заполнять буфер со скоростью, необходимой для проигрывателя, однако в реальности это может и не происходить, поэтому всегда полезно иметь постоянную обратную связь.

Существуют два способа постоянного поддержания буфера в заполненном состоянии. При использовании **выталкивающего сервера** (pull server) проигрыватель отправляет запросы на получение нового блока всегда, когда в буфере есть свободное место. Цель этого метода состоит в том, чтобы загружать в буфер как можно больше данных.

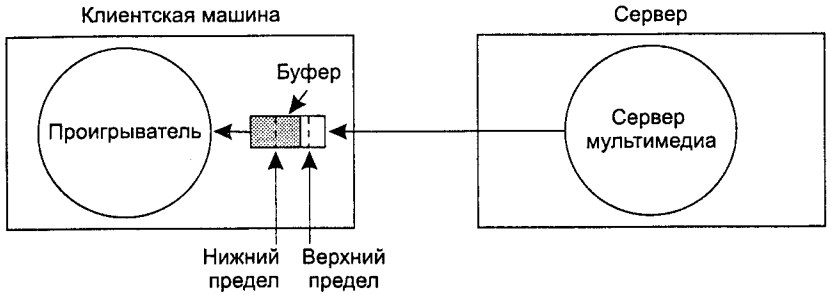


Рис. 7.30. Проигрыватель буферизирует входные данные, принимаемые с мультимедийного сервера, и воспроизводит содержимое буфера, а не данные, поступающие напрямую из сети

Недостатком выталкивающего сервера является наличие необязательных запросов данных. Серверу известно, что он отослал весь файл, так почему же проигрыватель продолжает что-то запрашивать? По этой причине данный метод используется редко.

При использовании **проталкивающего сервера** проигрыватель отправляет запрос *PLAY* (Воспроизведение), и сервер просто отправляет ему данные. При этом возможны два варианта: либо сервер мультимедиа работает со скоростью воспроизведения, либо он работает с более высокой скоростью. В обоих случаях некоторое количество данных буферизируется перед началом проигрывания. Если сервер работает со скоростью воспроизведения, то посылаемые им данные добавляются в конец буфера, а проигрыватель извлекает данные из начала буфера. Пока все работает без сбоев, объем данных, хранимых в буфере, остается постоянным во времени. Это очень простая схема — здесь не требуется пересылать управляющие сообщения в каких-либо направлениях.

Еще один вариант схемы проталкивания основан на том, что сервер выдает данные быстрее, чем реально требуется проигрывателю. Преимуществом является то, что при отсутствии гарантий стабильной скорости сервер имеет возможность наверстать упущенное из-за задержки время. Однако есть опасность переполнения буфера, возникающая вследствие того, что данные «потребляются» медленнее, чем выдаются (а это, в свою очередь, необходимо для устранения возможных пауз при воспроизведении).

Решением этой проблемы является установка проигрывателем **нижнего** и **верхнего пределов** заполнения буфера. Суть в том, что сервер выдает данные

лишь до тех пор, пока буфер не заполнится до верхнего предела. После этого проигрыватель просит сервер приостановить передачу. Поскольку данные будут продолжать прибывать в течение времени передачи запроса приостановки, расстояние между верхним пределом и концом буфера должно быть больше некоторого X — количества данных, которые успеют передаться за этот промежуток времени. X соответствует задержкам в канале, возникающим вследствие ограниченной пропускной способности. После приостановки сервера буфер начнет опустошаться. Когда количество данных в нем достигнет нижнего предела, проигрыватель попросит сервер мультимедиа возобновить передачу. Нижний предел должен быть установлен таким образом, чтобы буфер не опустошался полностью.

Для работы по методу проталкивающего сервера проигрыватель должен осуществлять удаленное управление сервером. Это обеспечивается протоколом RTSP, предоставляющим соответствующий механизм управления. Он определен в RFC 2326. Протокол не предназначен для управления потоком данных, для этого обычно применяется RTP. Основные команды RTSP приведены в табл. 7.17.

Таблица 7.17. Команды RTSP, посылаемые проигрывателем на сервер

Команда	Действие сервера
DESCRIBE	Перечисляет параметры мультимедийных данных
SETUP	Устанавливает логическое соединение между проигрывателем и сервером
PLAY	Начинает отправлять данные клиенту
RECORD	Начинает прием данных от клиента
PAUSE	Приостанавливает передачу данных
TEARDOWN	Удаляет логическое соединение

Интернет-радио

Как только стало возможным передавать потоковое аудио через Интернет, коммерческие радиостанции задумались о том, как бы организовать вещание в Сети (параллельно с вещанием в эфире). Вскоре маленькие радиостанции в колледжах стали передавать сигнал через Интернет. *Студенты* колледжей организовали собственные радиостанции. Действительно, технологии сейчас находятся на том уровне, когда практически каждый может основать собственную радиостанцию. интернет-радио — это совсем молодая область, находящаяся в стадии развития, но о ней стоит сказать пару слов.

Есть два основных подхода к организации радиовещания в Интернете. Первый подразумевает, что передачи предварительно записываются и сохраняются на диске. Слушатели могут получить доступ к архивам радиостанции, выбрать интересующую передачу и загрузить ее себе для прослушивания. В общем-то, это ничем не отличается от потокового аудио, которое мы только что обсуждали. Также возможно сохранять передачи сразу же после их выхода в прямом эфире. В этом случае архив состоит из передач, которые звучали, скажем, полчаса назад или еще меньше. Преимуществом является то, что технически это очень просто

организовать, — используются все те же методы потокового аудио; при этом слушатели могут выбрать любую передачу из архива.

Совсем другой подход связан с *радиовещанием* через Интернет. Некоторые станции организуют параллельное вещание — в эфире и в Сети. Однако появляется все больше станций, работающих исключительно через Интернет. Некоторые технологии, применяемые для передачи потокового аудио, подходят и для живого вещания, однако есть некоторые серьезные различия.

Похожи эти технологии тем, что и там, и там требуется буферизация на стороне пользователя, позволяющая снизить флуктуацию (джиттер). Буферизация 10–15 с звучания до начала проигрывания позволяет сделать вещание непрерывным даже в условиях довольно заметной флуктуации (джиттера) в сети. До тех пор, пока пакеты прибывают раньше, чем они реально нужны, не имеет никакого значения, когда именно они прибывают.

Одно из ключевых *отличий* состоит в том, что потоковое аудио можно выдавать со скоростью, превышающей скорость воспроизведения, поскольку приемник может остановить процесс, когда буфер заполняется до верхнего предела. В принципе, за счет этого появляется время на передачу потерянных пакетов, хотя практически это свойство редко используется. Что касается живого радиовещания, здесь скорость выдачи информации всегда точно соответствует скорости ее создания и воспроизведения.

Еще одно отличие состоит в том, что аудитория радиостанции может исчисляться сотнями или тысячами слушателей, тогда как потоковое аудио рассчитано на двухточечный обмен информацией. В таких условиях, очевидно, интернет-радио может передавать широковещательный сигнал с помощью протоколов RTP/RTSP. Это наиболее эффективный способ работы.

Однако на сегодняшний день интернет-радио работает по-другому. Реально происходит вот что: пользователь устанавливает TCP-соединение с радиостанцией и принимает данные посредством протокола TCP. Конечно, это порождает ряд проблем, таких как остановка передачи при заполнении окна, потеря пакетов с последующей повторной передачей и т. д.

Почему же вместо широковещания по RTP применяется однонаправленная передача по TCP? Есть три причины этого. Во-первых, лишь немногие провайдеры поддерживают широковещание, этот метод передачи используется очень редко. Во-вторых, протокол RTP гораздо менее известен, нежели TCP, а многие радиостанции слишком малы, чтобы иметь в штате профессиональных компьютерщиков. Гораздо проще использовать понятный и популярный протокол TCP, который поддерживается большинством программных продуктов. В-третьих, многие любят слушать радио на работе, то есть за границей брандмауэра. Большинство сетевых администраторов настраивают брандмауэры таким образом, чтобы защитить локальную сеть от нежелательного проникновения в нее извне. Обычно разрешается установка TCP-соединений с удаленного порта 25 (SMTP для электронной почты), прием UDP-пакетов с удаленного порта 53 (DNS), а также установка TCP-соединений с портом 80 (HTTP для Всемирной паутины). Почти все прочие возможности, включая RTP, могут быть заблокированы. Таким образом, единственный способ передать радиосигнал через брандмауэр — это заставить

веб-сайт притвориться HTTP-сервером (по крайней мере, для брандмауэра) и, соответственно, использовать HTTP-серверы, которые общаются по TCP. Такие суровые меры, обеспечивая лишь минимальную защиту информации, зачастую резко снижают эффективность мультимедийных приложений.

Поскольку интернет-радио — это новая среда передачи данных, войны форматов идут полным ходом. RealAudio, Windows Media Audio и MP3 ведут достаточно агрессивную конкуренцию на этом рынке, борясь за право быть доминирующим форматом радиовещания в Интернете. Сейчас появился еще один формат — Vorbis, который технически похож на MP3, но является открытым и не использует патентованные методы, на которые опирается MP3.

Типичная интернет-радиостанция представляет собой веб-сайт, на котором выложены расписание передач, информация о ведущих и множество рекламы. Обычно можно найти один или несколько логотипов, указывающих на поддерживаемые аудиоформаты (или просто надпись «ПРОСЛУШАТЬ», если поддерживается только один формат). Значки с этими логотипами являются гиперссылками на метафайлы, о которых говорилось ранее.

Когда пользователь щелкает на одном из значков, пересылается короткий метафайл. Браузер, используя MIME-тип или расширение файла, определяет подходящее вспомогательное приложение (то есть проигрыватель). Метафайл записывается во временный файл, затем открывается программа-проигрыватель, которой передается имя временного файла. Видя содержащийся в нем URL (обычно со схемой *http* или *rtsp*, что позволяет обойти накладываемые брандмауэром ограничения и одновременно удовлетворить потребности популярных мультимедийных приложений), проигрыватель связывается с сервером и начинает работать как радиоприемник. Кстати говоря, аудиоданные передаются в виде одного потока, поэтому работа по *http* возможна, но только для радио: передавать видео, для которого характерно наличие по крайней мере двух потоков, с помощью *http* не удается — нужно что-нибудь типа *rtsp*.

Еще одной интересной особенностью интернет-радио является то, что практически все желающие, даже студенты, могут организовать собственную радиостанцию. Основные компоненты, необходимые для этого, изображены на рис. 7.31. Базой является обычный персональный компьютер со звуковой картой и микрофоном. Что касается программного обеспечения, то понадобится проигрыватель типа Winamp или Freeamp с подключаемым модулем для записи звука и кодеком выбранного формата (например, MP3 или Vorbis).

Поток аудиоданных, создаваемый станцией, отправляется на большой сервер мультимедиа в Интернете, который занимается распространением этого потока между множеством TCP-соединений. Сервер обычно работает с большим количеством маленьких радиостанций. Ведется список обслуживаемых радиостанций и предоставляется информация о том, какие из них в данных момент вещают. Потенциальные слушатели соединяются с этим сервером, выбирают станцию и получают данные по TCP. Существуют как коммерческие программы, включающие в себя все необходимые компоненты, так и открытые программные средства, такие как icecast. Разумеется, есть серверы, занимающиеся платной поддержкой радиостанций.

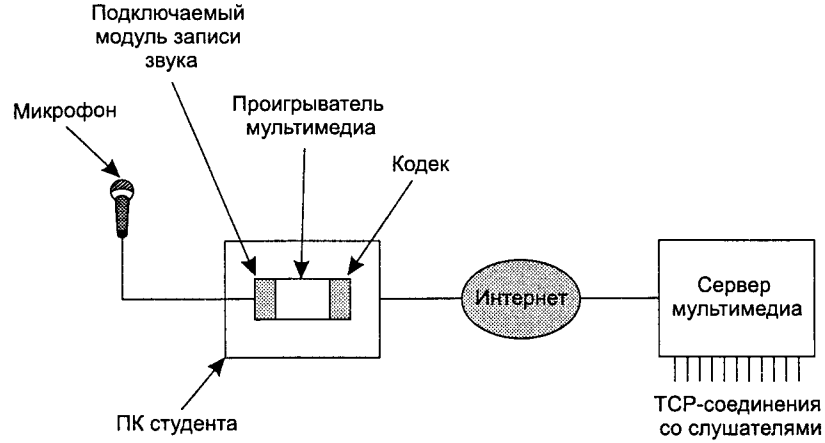


Рис. 7.31. Студенческая радиостанция

Передача речи поверх IP

Когда-то в стародавние времена общественная коммутируемая телефонная сеть была основным средством передачи речи; изредка она использовалась и для передачи данных. Однако с годами объем передаваемых данных все возрастал, и к 1999 году объемы данных и речи в телефонной сети уравнялись (и то, и другое можно измерить количеством бит в секунду, поскольку в глубинах телефонной системы используется цифровое PCM-кодирование). К 2002 году объем информационного трафика стал на порядок больше объема речевого трафика, и его рост (экспоненциальный!) продолжается. Между тем объем речевого трафика сохраняется практически неизменным (прирост составляет около 5 % в год).

В результате этого многие сетевые операторы, использующие системы с коммутацией пакетов, внезапно оказались заинтересованы в передаче речевых данных по сетям передачи данных. Требуемое для этого увеличение пропускной способности минимально, так как масштабы сетей с коммутацией пакетов позволяют передавать информационный трафик. Тем не менее, расходы среднего потребителя на телефонные переговоры могут превышать расходы на Интернет, поэтому сетевые операторы увидели в интернет-телефонии источник больших дополнительных доходов, не требующий прокладки новых кабелей. Так родилась **система передачи речи поверх IP**, или **интернет-телефония**.

H.323

С самого начала всем было понятно, что если каждый производитель станет изобретать собственный стек протоколов, система никогда работать не будет. Во избежание возникновения этой проблемы заинтересованные стороны объединились под покровительством Международного союза телекоммуникаций (ITU) и начали разработку единого стандарта. В 1996 году ITU выпустил рекомендации с индексом H.323 под заголовком «Видеотелефонные системы и оборудование

локальных вычислительных сетей, не предоставляющих гарантированное качество обслуживания». Такое название могло родиться только в телефонной индустрии. Данные рекомендации были пересмотрены в 1998 году, и новый вариант H.323 стал основой построения первых глобальных систем интернет-телефонии.

H.323 скорее дает общее представление об архитектуре систем интернет-телефонии, нежели описывает некий конкретный протокол. В документе можно найти множество ссылок на различные специализированные протоколы кодирования речи, установки соединения, передачи сигналов, данных и т. п., однако их описание не приводится. Общая модель изображена на рис. 7.32. В центре находится **шлюз**, соединяющий Интернет с телефонной сетью. Он поддерживает протокол H.323 со стороны Интернета и протоколы коммутируемой телефонной сети общего пользования с «телефонной» стороны. Устройства коммуникации называются **терминалами**. В локальной вычислительной сети может быть **машина-вратарь**, управляющая конечными узлами, находящимися под ее юрисдикцией (в ее **зоне**).

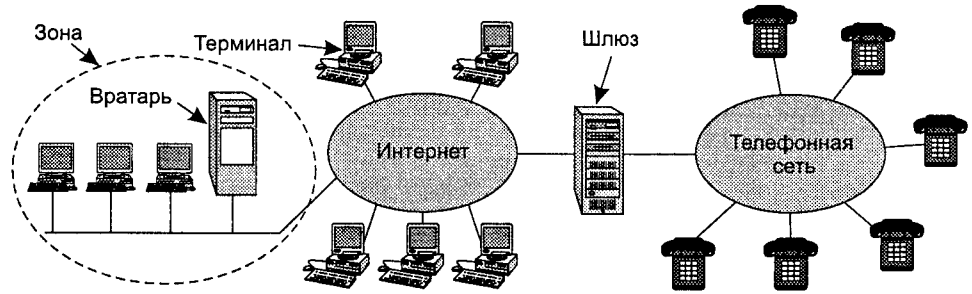


Рис. 7.32. Модель архитектуры H.323 для интернет-телефонии

Работу телефонной сети обеспечивает множество протоколов. Во-первых, необходим протокол кодирования и декодирования речи. Система PCM, которую мы изучали в главе 2, определена в рекомендациях ITU **G.711**. С ее помощью один голосовой канал кодируется 8-битными отсчетами с частотой 8000 раз в секунду. В результате получается 64-килобитный несжатый поток речевых данных. Все системы H.323 обязаны поддерживать G.711. Тем не менее, разрешена (но не является обязательной) поддержка и других протоколов кодирования речи. Они используют иные алгоритмы сжатия и приводят к несколько отличающемуся компромиссу между качеством и использованием пропускной способности. Например, в **G.723.1** берутся блоки по 240 отсчетов (30 мс речи) и используется кодирование с предсказанием, снижающее размер блоков до 24 или 20 байт. На выходе этого алгоритма получается поток со скоростью 6,4 или 5,3 Кбит/с (сжатие в 10 или 12 раз соответственно). Разумеется, качество звучания при этом гораздо ниже. Могут быть реализованы и другие алгоритмы кодирования.

Поскольку разрешено использование нескольких алгоритмов сжатия, необходим отдельный протокол, который позволил бы терминалам договориться об использовании одного из этих протоколов. Такой протокол называется **H.245**. Он

позволяет согласовать также другие параметры соединения, например битовую скорость. RTCP требуется для управления каналами RTP. Кроме того, нужны протоколы для установления и разрыва соединений, обеспечения тонального вызова, генерирования звуков звонков и других стандартных функций телефонной системы. Используется стандарт ITU **Q.931**. Терминалам нужен протокол для ведения переговоров с машиной-вратарем (если такая присутствует в локальной сети). Для этого в системе работает протокол **H.225**. Канал между ПК и вратарем, которым этот протокол управляет, называется каналом **RAS** (Registration/Admission/Status — Регистрация/Доступ/Статус). Он позволяет терминалам, кроме всего прочего, входить в зону и покидать ее, запрашивать и освобождать пропускную способность, обновлять данные о состоянии. Наконец, нужен протокол для непосредственной передачи данных. На этом участке работает RTP. Как обычно, управляется он RTCP. Иерархия всех этих протоколов показана на рис. 7.33.

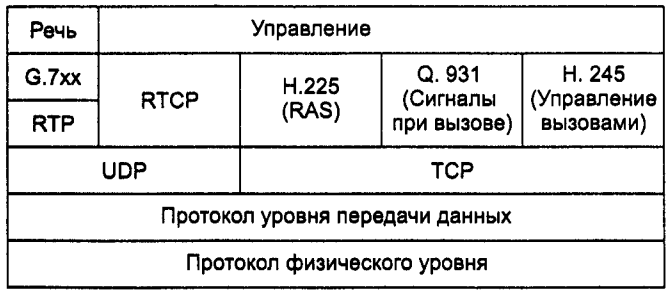


Рис. 7.33. Стек протоколов H.323

Чтобы понять, как эти протоколы взаимодействуют друг с другом, рассмотрим случай ПК, являющегося терминалом локальной сети (с вратарем) и звонящего на удаленный телефон. Вначале компьютеру нужно найти вратаря, поэтому он рассылает широковещательным образом специальный UDP-пакет через порт 1718. Из ответа вратаря ПК узнает его IP-адрес. Теперь компьютер должен зарегистрироваться у вратаря. Для этого он посылает ему сообщение RAS в пакете UDP. После регистрации компьютер обращается к вратарю с просьбой (сообщение доступа RAS) о резервировании пропускной способности. Только после выделения этого ресурса можно начинать установку соединения. Предварительное резервирование пропускной способности позволяет вратарю ограничить число соединений, устанавливаемых на исходящей линии, что, в свою очередь, служит для обеспечения необходимого качества обслуживания.

Теперь ПК устанавливает TCP-соединение с вратарем, чтобы осуществить телефонный звонок. При установлении телефонного соединения используются традиционные протоколы телефонной сети, ориентированные на соединение. Поэтому требуется протокол TCP. С другой стороны, в телефонной системе нет никаких RAS, которые позволяли бы телефонным аппаратам заявлять о своем присутствии, поэтому разработчики H.323 могли применять как UDP, так и TCP для передачи сообщений RAS, и они выбрали протокол с наименьшими накладными расходами — UDP.

Теперь, когда терминалу уже выделена пропускная способность, он может послать по TCP-соединению сообщение *SETUP* (стандарт Q.931). В нем указывается номер вызываемого абонента (или IP-адрес и порт, если вызывается удаленный компьютер). Вратарь отвечает Q.931-сообщением *CALL PROCEDING*, подтверждая тем самым факт корректного приема запроса. Затем вратарь пересылает сообщение *SETUP* на шлюз.

Шлюз, который является с одной стороны компьютером, а с другой — телефонным коммутатором, осуществляет обычный звонок на обычный телефон. Оконечная телефонная станция вызываемого абонента выполняет свою обычную работу (у абонента звенит звонок), а кроме этого отсылает обратно Q.931-сообщение *ALERT*, извещая ПК о том, что началась серия звонков. Когда абонент поднимает трубку, оконечная телефонная станция отправляет сообщение *CONNECT*, сообщая компьютеру о том, что соединение установлено.

После установки соединения вратарь перестает принимать участие в этом процессе, хотя шлюз, конечно, продолжает работать, обеспечивая двустороннюю связь. Пакеты идут в обход вратаря и направляются напрямую по IP-адресу шлюза. Эту ситуацию можно сравнить с обычным каналом между двумя сторонами. Это действительно просто соединение физического уровня, по которому передаются биты, и все. Ни одна из сторон не в курсе того, что представляет собой противоположная сторона.

Для переговоров о предпочитаемых параметрах соединения используется протокол H.245. При этом используется специальный управляющий канал H.245, который всегда открыт. Каждая из сторон начинает с объявления своих возможностей. Например, может сообщаться о поддержке видео (H.323 может поддерживать видео), конференц-связи, используемых кодеках и т. п. После того как каждая из сторон узнает возможности противоположной стороны, организуются два однонаправленных канала, с которыми связываются определенные кодеки и которым присваиваются определенные параметры. Поскольку на каждой из сторон может быть установлено разное оборудование, вполне возможна ситуация, когда каждый из однонаправленных каналов использует свой кодек. По достижении договоренности по всем вопросам можно начинать передачу данных (по протоколу RTP). Управление производится RTCP, контролирующим перегрузку. Если передаются видеоданные, RTCP занимается синхронизацией звукового и видеоряда. На рис. 7.34 показаны различные виды логических каналов. После того, как на одной из сторон вешают трубку, по каналу Q.931 передается сигнал окончания связи.

После разрыва соединения вызывающий ПК должен снова связаться с вратарем и послать ему сообщение RAS с запросом освобождения зарезервированной пропускной способности. Впрочем, вместо этого он может осуществить новый звонок.

Мы до сих пор ничего не говорили о качестве обслуживания, а ведь на самом деле это довольно существенный аспект успешной передачи речи поверх IP. Дело в том, что QoS не входит в область рассмотрения H.323. Если сеть, по которой

передаются данные, способна обеспечить стабильное соединение без флуктуации (джиттера) между ПК (например, с использованием методов, обсуждавшихся в главе 5) и шлюзом, значит, нам повезло и качество обслуживания будет хорошим. В противном случае качество будет, увы, плохое. В телефонной части системы используется PCM-кодирование, исключающее флуктуацию (джиттер).

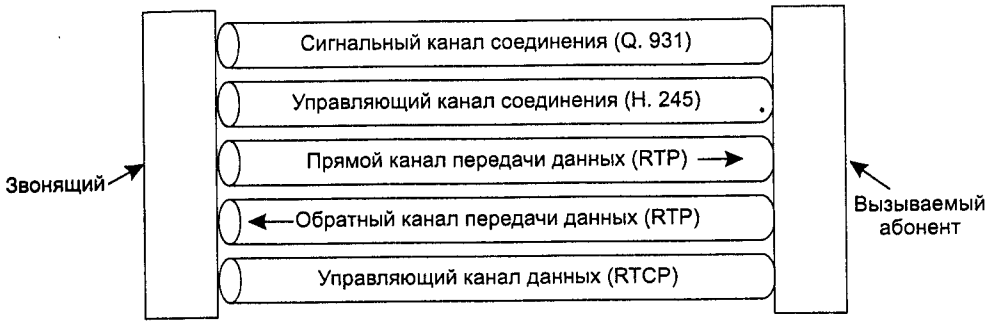


Рис. 7.34. Логические каналы между звонящим и вызываемым абонентами во время разговора

SIP — протокол запуска соединения

Стандарт H.323 был разработан ITU. В Интернет-сообществе многим он показался типичным продуктом телефонной компании: громоздким, сложным и недостаточно гибким. Было решено организовать специальный комитет IETF для создания более простой и гибкой системы передачи речи поверх IP. Основным результатом деятельности этого комитета стал протокол **SIP** (Session Initiation Protocol — протокол запуска соединения), описанный в RFC 3261. Протокол оговаривает способ установки телефонных соединений через Интернет, технологию организации систем для видеоконференций и способы создания других мультимедийных приложений. В отличие от H.323, представляющего собой целый набор протоколов, SIP — это единый модуль, способный взаимодействовать с разнообразными интернет-приложениями. Например, номера телефонов определяются в виде URL, то есть на веб-страницах можно размещать гиперссылки, щелкнув на которых, пользователь сможет установить телефонное соединение (примерно так же схема *mailto* позволяет написать электронное письмо и отправить его по указанному в ссылке адресу).

Протокол SIP позволяет устанавливать и двухсторонние соединения (то есть обычные телефонные соединения), и многосторонние (когда каждый из участников может как слушать собеседников, так и говорить), и широковещательные (когда один из участников говорит, а остальные могут только слушать). Во время сеанса связи могут передаваться аудио-, видео- или другие данные. Эта возможность используется, например, при организации сетевых игр с большим количеством участников в реальном времени. SIP занимается только установкой, управлением и разрывом соединений. Для передачи данных используются другие протоколы, например, RTP/RTCP. SIP — это протокол прикладного уровня, работающий поверх TCP или UDP.

Протокол SIP предоставляет разнообразные услуги, включая поиск вызываемого абонента (который может в данный момент быть далеко от своего домашнего компьютера), определение его возможностей, поддержку механизмов установки и разрыва телефонного соединения. В простейшем случае SIP устанавливает сеанс связи между компьютерами звонящего и вызываемого абонентов. Именно этот случай мы сейчас и рассмотрим.

Телефонные номера в SIP представляются в виде URL со схемой *sip*. Например, *sip:ilse@cs.university.edu* свяжет вас с пользователем по имени Ilse, хост которого определяется DNS-именем *cs.university.edu*. SIP URL могут содержать также адреса формата IPv4, IPv6 или реальные номера телефонов.

Протокол SIP является текстовым, он построен по модели HTTP. Одна из сторон посылает ASCII-сообщение, в котором первая строка содержит имя метода, а ниже следуют дополнительные строки, содержащие заголовки для передачи параметров. Многие заголовки взяты из стандарта MIME, что позволяет SIP взаимодействовать с существующими интернет-приложениями. Шесть методов, определяемых базовой спецификацией, перечислены в табл. 7.18.

Таблица 7.18. Методы SIP, определяемые базовой спецификацией

Метод	Описание
INVITE	Запрос запуска сеанса связи
ACK	Подтверждение запуска сеанса
BYE	Запрос окончания сеанса
OPTIONS	Опрос возможностей хоста
CANCEL	Отмена запроса
REGISTER	Информирование сервера переадресации о текущем местоположении пользователя

Для установки сеанса связи звонящий должен либо создать TCP-соединение с вызываемым абонентом и послать по нему сообщение *INVITE*, либо послать это же сообщение в UDP-пакете. В обоих случаях заголовки, содержащиеся во второй и всех последующих строках, описывают структуру тела сообщения, содержащего информацию о возможностях звонящего, типах мультимедиа и форматах. Если вызываемый абонент принимает звонок, он посылает в качестве ответа трехразрядный код результата типа HTTP (группы этих кодов перечислены в табл. 7.13, код 200 означает прием вызова). Следом за строкой с кодом результата вызываемый абонент может также сообщить данные о своих возможностях, типах мультимедиа и форматах.

Соединение устанавливается путем тройного рукопожатия, звонящий высылает *ACK* как для окончания работы протокола, так и для подтверждения приема кода 200.

Любая из сторон может послать запрос окончания сеанса связи, для этого используется метод *BYE*. Сеанс считается законченным после получения подтверждения от противоположной стороны.

Метод *OPTIONS* применяется для опроса возможностей машины. Обычно это делается перед запуском сеанса связи для того чтобы определить, поддержива-

ется ли тип сеанса, на который рассчитывает вызывающая сторона (например, передача голоса поверх IP).

Метод *REGISTER* относится к возможности протокола SIP разыскивать пользователя и соединяться с ним, даже если его нет дома. Сообщение, содержащее данный метод, отправляется на поисковый сервер SIP, хранящий данные о том, кто где находится в данный момент. Впоследствии с помощью этого сервера можно попробовать найти абонента. Операция переадресации, используемая при этом, показана на рис. 7.35. На этом рисунке мы видим, что звонящий отправляет сообщение *INVITE* на прокси-сервер. Это делает возможную переадресацию незаметной. Прокси пытается разыскать абонента и посылает *INVITE* по найденному адресу. Дальнейшее общение представляет собой коммутацию последовательности сообщений при «тройном рукопожатии». Сообщения *LOOKUP* и *REPLY* не входят в протокол SIP; на этой стадии может использоваться любой подходящий протокол в зависимости от типа поискового сервера.

Рис. 7.35. Использование прокси и серверов переадресации в протоколе SIP

SIP обладает также множеством других свойств, которые мы здесь не стали описывать подробно. Среди них есть функции ожидания вызова, отображения звонка, шифрования и идентификации звонящего. Кроме того, есть возможность звонить с компьютера на обычный телефон, если есть доступ к соответствующему шлюзу между Интернетом и телефонной системой.

Сравнительный анализ H.323 и SIP

H.323 и SIP во многом схожи, однако между ними есть и некоторые различия. Оба стандарта поддерживают как двухстороннюю, так и многостороннюю связь. Оконечным оборудованием могут служить как компьютеры, так и обычные телефоны. И там, и там стороны предварительно договариваются о параметрах, возможно шифрование данных и используются протоколы RTP/RTCP. Сводная сравнительная табл. 7.19 показывает все сходства и различия.

Несмотря на схожий набор свойств и характеристик, протоколы разительно отличаются друг от друга концепцией и философией. H.323 — это типичный тяжеловесный стандарт, характерный для телефонной индустрии. Он описывает целый стек протоколов и очень точно указывает, что разрешено, а что запрещено. Такой подход приводит к хорошо определенным протоколам на каждом уровне, тем самым упрощается задача взаимодействия сетей. Однако платой за это оказывается большой, сложный и жесткий стандарт, тяжело адаптируемый к приложениям, которые появятся в будущем.

SIP, наоборот, представляет собой типичный интернет-протокол, работа которого основана на обмене короткими текстовыми строками. Это небольшой модуль, который хорошо взаимодействует с другими протоколами Интернета, однако несколько хуже согласуется с существующими сигнальными протоколами телефонной системы. Поскольку модель системы передачи данных поверх IP, предложенная IETF, использует модульный принцип, она оказывается достаточно гибкой и может легко адаптироваться к новым приложениям. Недостаток этого протокола связан с возможными проблемами межсетевого взаимодействия. Впрочем, специально для исключения этого недостатка часто проводятся встречи разработчиков и тестирование совместной работы различных сетей.

Таблица 7.19. Сравнение H.323 и SIP

Аспект	H.323	SIP
Разработчик	ITU	IETF
Совместимость с телефонной системой	Полная	В большой мере
Совместимость с Интернетом	Отсутствует	Присутствует
Архитектура	Монолитная	Модульная
Завершенность	Полный стек протоколов	SIP обеспечивает лишь установку соединения
Переговоры относительно параметров	Ведутся обеими сторонами	Ведутся обеими сторонами
Сигналы при вызове	Q.931 поверх TCP	SIP поверх TCP или UDP
Формат сообщений	Двоичный	ASCII
Передача мультимедийных данных	RTP/RTCP	RTP/RTCP
Многосторонняя связь	Есть	Есть
Мультимедийные конференции	Возможны	Невозможны
Адресация	Номер телефона или хоста	URL
Разрыв связи	Явный или разрыв TCP-соединения	Явный или по тайм-ауту
Постоянный обмен сообщениями	Нет	Есть
Шифрование данных	Есть	Есть
Объем описания стандарта	1400 страниц	250 страниц
Реализация	Громоздкая и сложная	Умеренная по объему
Статус	Широко распространен	Перспективен

Интернет-телефония — это новая перспективная область. Неудивительно поэтому, что уже выпущено несколько книг, посвященных этой теме. Среди них

стоит упомянуть (Collins, 2001; Davidson и Peters, 2000; Kumar и др., 2001; Wright, 2001). В журнале *Internet Computing* за май-июнь 2001 можно найти несколько статей, посвященных передаче речи поверх IP.

Видео

Итак, мы достаточно долго обсуждали услуги, предоставляемые компьютерными сетями человеческому уху. Пора обратиться к глазам (нет-нет, за этим разделом не следует раздел, посвященный носу). Сетчатка глаза человека обладает инерционными свойствами, то есть яркое изображение, быстро появившееся на сетчатке, остается на ней несколько миллисекунд, прежде чем угаснуть. Если последовательности одинаковых изображений появляются и исчезают с частотой около 50 Гц, глаз человека не замечает, что он смотрит на дискретные изображения. Все видео (то есть телевизионные) системы используют этот принцип для создания движущихся изображений.

Аналоговые системы

Чтобы понять, как работают видеосистемы, лучше всего начать с рассмотрения простого старомодного черно-белого телевидения. Для представления двухмерного изображения в виде одномерной зависимости электрического напряжения от времени камера быстро сканирует электронным лучом изображение, разбивая его на горизонтальные линии и запоминая по мере продвижения интенсивность света. Закончив сканирование **кадра**, луч возвращается в исходную точку. Зависимость интенсивности света от времени — это та функция, значения которой распространяются в виде телевизионного сигнала. Телевизионные приемники для восстановления изображения повторяют процесс сканирования. Путь, который проходит электронный луч в передающей камере и принимающей телевизионной трубке, показан на рис. 7.36. (Что касается камер с ПЗС-матрицами (с зарядовой связью), то их принцип работы основан на интегрировании, а не на сканировании, однако некоторые камеры и все мониторы все же занимаются именно сканированием.)

В разных странах приняты различные стандарты, описывающие точные параметры сканирования В системе, принятой в Северной и Южной Америке, а также Японии, экран разбивается на 525 горизонтальных линий развертки, соотношение горизонтального и вертикального размеров экрана составляет 4:3, кадры передаются с частотой 30 кадров в секунду. Европейская система PAL/SECAM подразумевает разбиение кадра на 625 линий, размеры экрана у нее также 4:3, а частота кадров составляет 25 кадров в секунду. В обеих системах самые верхние и самые нижние линии кадра не показываются (это связано с попыткой адаптировать прямоугольную картинку к круглой форме электронно-лучевой трубки). На экране телевизоров показываются только 483 из 525 линий развертки для системы NTSC и 576 из 625 — для системы PAL/SECAM. Во время обратного хода луч выключается, и этот интервал времени многие телевизионные станции (особенно в Европе) используют для передачи телетекста (текстовых страниц, содержащих новости, прогнозов погоды, биржевых сводок и т. п.).

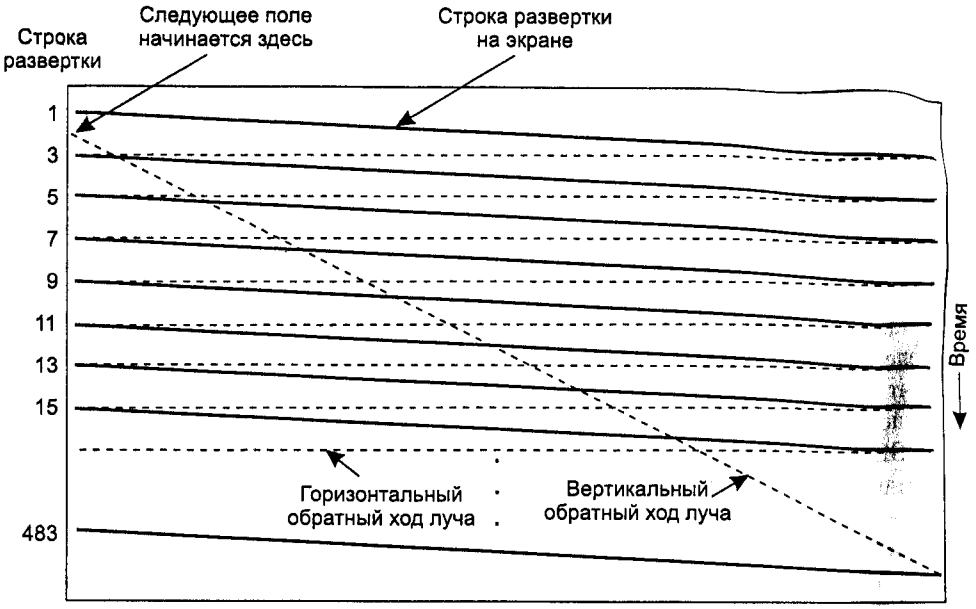


Рис. 7.36. Путь сканирующего луча в видео- и телесистемах NTSC

Хотя частоты в 25 кадров в секунду достаточно, чтобы передать плавное движение, при такой частоте кадровой развертки многие зрители (особенно пожилые) заметят мигание изображения, связанное с тем, что сетчатка глаза успеет восстановиться, прежде чем появится новый кадр. Увеличение частоты кадров потребовало бы увеличения объемов хранимой и передаваемой информации. Вместо этого был выбран другой подход. Линии развертки показываются на экране телевизора не подряд, а через одну: сначала все нечетные, а затем все четные. Каждый такой полукадр называют **полем**. Эксперименты показали, что хотя люди замечают мерцание при частоте 25 кадров в секунду, при частоте 50 полей в секунду оно уже не заметно. Этот метод называется **чересстрочной разверткой**. Телевидение или видеосистема, в которой не используется чересстрочная развертка, называется **поступательным**. Обратите внимание на то, что движущиеся изображения показываются со скоростью 24 кадра в секунду, но каждый кадр полностью виден в течение 1/24 секунды.

В цветном видео используется тот же принцип развертки, что и в монохромном (черно-белом), с той разницей, что вместо одного луча изображение представляется синхронно двигающимися тремя лучами: красным, зеленым и синим (RGB — red, green, blue). Комбинации этих трех цветов оказывается достаточно для передачи любого цвета благодаря особенностям устройства человеческого глаза. При передаче по каналу связи эти три цветовых сигнала объединяются в единый **смешанный** сигнал.

При изобретении цветного телевидения разные страны решили выбрать различные методы, что привело к созданию систем, до сих пор остающихся несо-

вместимыми. (Не следует путать различные методы кодирования цвета с различными стандартами записи видеосигнала, такими как VHS, Betamax и P2000.) Во всех странах правительство требовало от разработчиков цветного телевидения, чтобы программы, передаваемые в цвете, могли также приниматься на черно-белых телевизорах. Поэтому простейшая схема, представляющая собой просто раздельное кодирование RGB-сигналов, была неприемлема. Кроме того, такая схема не является самой эффективной.

Первая система цветного телевидения была стандартизована в США Национальным комитетом по телевизионным стандартам, **NTSC** (National Television Standards Committee). Стандарт получил название по сокращенному имени комитета. Спустя несколько лет в Европе были разработаны свои системы цветного телевидения. К этому времени технология стала уже значительно совершеннее, что привело к созданию систем с более высокой помехозащищенностью и улучшенной цветопередачей. Во Франции и Восточной Европе был принят стандарт **SECAM** (SEquence de Couleurs Avec Memoire — последовательные цвета с запоминанием), а в остальной части Европы — стандарт **PAL** (Phase Alternation Line — построчное изменение фазы). Различие в качестве передачи цвета между системами NTSC и PAL/SECAM послужило причиной появления шутки, согласно которой NTSC следует расшифровывать как «никогда не повторяющийся цвет» (Never Twice the Same Color).

Для совместимости с черно-белыми телевизионными приемниками во всех трех системах сигналы RGB линейно объединяются в сигнал **яркости** и два сигнала **цветности**. Однако в каждой из этих систем эти сигналы формируются с использованием различных коэффициентов. Экспериментально доказано, что человеческий глаз обладает гораздо более высокой чувствительностью к изменению сигнала яркости, чем к изменению сигнала цветности. В результате сигнал цветности не обязательно кодировать так же точно, как сигнал яркости. Поэтому было решено передавать сигнал яркости в том же формате, что и черно-белый сигнал, а два узкополосных сигнала цветности — отдельно на более высокой частоте. Телевизоры обычно снабжаются регуляторами яркости и насыщенности цвета. Знакомство с сигналами яркости и цветности важно для понимания принципов действия алгоритмов видеосжатия.

В последние годы появился значительный интерес к телевидению высокой четкости **HDTV** (High Definition TeleVision). Системы HDTV отличаются от обычных систем телевидения примерно двукратным увеличением числа строк развертки. США, Европа и Япония уже успели создать собственные, как всегда совершенно не совместимые между собой, системы HDTV. Базовые принципы HDTV, касающиеся сканирования, яркости и цветности, подобны уже существующим системам. Однако в этих системах в основном используется формат экрана 16:9 вместо 4:3, что лучше соответствует стандарту, принятому в кинематографе для фильмов, снятых на ленте шириной 35 мм и имеющих формат 3:2.

Цифровые видеосистемы

Простейшая форма представления цифрового видео заключается в последовательности кадров, состоящих из прямоугольной сетки элементов рисунка (**пикселов**). Если каждый пиксел представлять одним битом, то мы получим бинарное изображение, состоящее только из черного и белого цветов, без оттенков серого. Качество такого изображения можно оценить, если послать фотографию по факсу: оно будет ужасным. (Попробуйте сделать так, если у вас есть возможность. Можно, кстати, вместо этого сделать фотокопию цветной фотографии на копировальном аппарате, который не умеет осуществлять преобразование в растровый формат.)

Следующий шаг заключается в использовании 8 бит на пиксел, тогда можно получить, например, 256 различных оттенков серого цвета, что достаточно для черно-белого телевидения высокого качества. В цветном телевидении хорошие системы используют по 8 бит на каждый из трех цветов RGB несмотря на то, что почти все системы передают смешанный сигнал. Хотя числом, состоящим из 24 бит, можно обозначить около 16 миллионов цветовых оттенков, человеческий глаз не в состоянии даже различить такое огромное количество цветовых оттенков, не говоря уж о больших количествах. Цифровые цветные изображения создаются тремя сканирующими лучами, по одному на цвет. Геометрия сканирования кадра полностью совпадает с аналоговой системой, показанной на рис. 7.36, с той разницей, что непрерывные линии развертки заменяются аккуратными рядами дискретных пикселов.

Для сглаживания при передаче движения, как и в аналоговом видео, в цифровом видео необходимо отображать по меньшей мере 25 кадров в секунду. Однако, поскольку качественные мониторы обычно сами сканируют по несколько раз изображения, хранящиеся в их памяти, с частотой 75 и более кадров в секунду, проблема мерцания изображения решается на уровне монитора сама собой, и чередование строк в цифровом видео не требуется.

Другими словами, плавность движущегося изображения определяется количеством *различающихся* изображений в секунду, тогда как мерцание зависит от частоты перерисовки экрана. Не следует путать эти два параметра. Неподвижное изображение, рисуемое с частотой 20 кадров в секунду, не будет дергаться, но будет мерцать, поскольку возбуждение сетчатки глаза успеет угаснуть, прежде чем появится следующий кадр. Фильм, в котором выводится 20 различных кадров в секунду, каждый из которых повторяется по четыре раза, не будет мерцать, но будет заметно отсутствие плавности движений.

Важность этих двух параметров станет ясна, если мы попробуем оценить пропускную способность, необходимую для передачи цифрового видеосигнала по сети. Во всех современных компьютерных мониторах используется соотношение размеров экрана 4:3, поэтому в них вполне могут использоваться недорогие серийные электронно-лучевые трубки, предназначенные для телевизоров. Стандартные используемые разрешения экрана составляют 1024×768, 1280×960 и 1600×1200. Для показа цифрового видео даже на экране с самым низким из этих разрешений при 24 битах на пиксел и 25 кадрах в секунду потребуется передавать поток данных со скоростью 472 Мбит/с. С этим может справиться лишь канал SONET OC-12, однако подведение таких линий в каждый дом пока что не стоит на повестке дня. Удвоение частоты, позволяющее избавиться от мерцания, выглядит еще менее привлекательно. Оптимальное решение состоит в том, что-

бы передавать 25 кадров в секунду, и позволить компьютеру самому хранить эти кадры и отображать их с удвоенной частотой. В обычном широковещательном телевидении такая стратегия почти не используется, так как у обычных телевизоров нет памяти. Кроме того, для хранения аналоговых сигналов в ОЗУ необходимо сначала преобразовать их в цифровую форму, что потребует дополнительного оборудования. Таким образом, чередование строк применяется в обычном широковещательном телевидении, но оно не нужно в цифровом видео.

Сжатие видеоданных

Итак, теперь должно быть очевидно, что о передаче мультимедийной информации в несжатом виде не может быть и речи. Есть лишь одна надежда: на повсеместное сжатие данных. К счастью, за несколько последних десятилетий было разработано множество методов сжатия, делающих возможной передачу мультимедийной информации. В данном разделе мы рассмотрим некоторые методы сжатия движущихся изображений.

Для всех систем сжатия требуется два алгоритма: один для компрессии данных источником информации, а другой — для распаковки ее получателем. В литературе эти алгоритмы называют, соответственно, алгоритмами **кодирования** и **декодирования**. Мы также будем пользоваться здесь этой терминологией.

Эти алгоритмы обладают определенной асимметрией, которую следует хорошо понять. Во-первых, во многих приложениях мультимедийный документ, например фильм, кодируется всего один раз, при его создании и помещении на мультимедийный сервер, но декодируется тысячи раз при просмотре пользователями. Эта асимметрия означает, что алгоритм кодирования может быть довольно медленным и требовать дорогого оборудования, тогда как алгоритм декодирования должен быть быстрым и должен работать даже на дешевом оборудовании. В конце концов, оператор мультимедийного сервера может позволить себе арендовать на пару недель параллельный суперкомпьютер, чтобы закодировать всю свою видеотеку, но нельзя требовать от клиентов, чтобы они арендовали суперкомпьютер на 2 часа для просмотра видеофильма. В большинстве современных систем сжатия высокая скорость декодирования является основной задачей, достигаемой даже ценой большой сложности и медленности алгоритма кодирования.

С другой стороны, для мультимедийных данных реального времени, например видеоконференций, медленное кодирование неприемлемо. Кодирование здесь должно происходить «на лету», в режиме реального времени. Поэтому в системах мультимедиа реального времени применяются другие алгоритмы или их параметры, чем при хранении фильмов на дисках. Чаще всего используется существенно меньшее сжатие.

Еще один аспект асимметрии состоит в том, что процесс кодирования/декодирования не обязан быть обратимым. Это значит следующее. При сжатии обычного файла, его передаче и декомпрессии получатель обязан получить копию, совпадающую с оригиналом с точностью до бита. При передаче мультимедиа абсолютная точность не требуется. Вполне допустимы небольшие отклонения видеосигнала от оригинала после его кодирования и декодирования. Система, в которой декодированный сигнал не точно соответствует кодированному оригиналу, называется системой **с потерями**. Если же выходной сигнал идентичен входному, то такая система называется системой **без потерь**. Системы с потерями являются важными, так как, допуская небольшие потери информации, можно достичь очень большого коэффициента сжатия.

Стандарт JPEG

Видеосигнал представляет собой обычную последовательность изображений (сопровождаемую звуком). Если мы сможем найти хороший алгоритм сжатия неподвижного изображения, нам с не меньшим успехом удастся сжать видеоданные. Уже существуют достаточно хорошие алгоритмы сжатия изображений, поэтому мы начнем изучение принципов сжатия видеоданных именно с этого. Стандарт **JPEG** для сжатия неподвижных изображений с непрерывно меняющимся цветом (например, фотографий) был разработан группой экспертов в области фотографии JPEG (Joint Photography Experts Group — Объединенная группа экспертов по машинной обработке фотоизображений). Эта группа работала под совместным покровительством Международного союза телекоммуникаций ITU, Международной организации по стандартизации ISO и еще одной организации, занимающейся стандартизацией, — IEC (International Electrotechnical Commission — Международная электротехническая комиссия). Стандарт JPEG является очень важным для мультимедиа, так как в первом приближении мультимедийный стандарт для движущихся изображений, MPEG, представляет собой просто кодирование каждого кадра отдельно алгоритмом JPEG плюс некоторые дополнительные процедуры межкадрового сжатия и обнаружения движения. Стандарт JPEG определен как международный стандарт 10918.

У метода сжатия JPEG есть четыре режима и множество параметров. Он больше напоминает номенклатуру товаров в небольшом магазине, чем просто один алгоритм. Для нас сейчас важен только последовательный режим с потерями, показанный на рис. 7.37. Кроме того, мы сконцентрируемся лишь на использовании JPEG для кодирования 24-битового RGB-видеоизображения и опустим некоторые незначительные детали.

Рис. 7.37. Работа алгоритма JPEG в последовательном режиме с потерями

Первый этап кодирования изображения алгоритмом JPEG представляет собой подготовку блока. Рассмотрим частный случай кодирования 24-битового RGB-видеоизображения размером 640×480 пикселов, показанного на рис. 7.38, *а*. Поскольку разделение на яркость и цветность позволяет сильнее сжать изображение,

мы сосчитаем сначала яркость Y и два значения цветности I и Q (по стандарту NTSC) в соответствии со следующими формулами:

$$Y = 0{,}30\ R + 0{,}59\ G + 0{,}11\ B;$$

$$I = 0{,}60\ R - 0{,}28\ G - 0{,}32\ B;$$

$$Q = 0{,}21\ R - 0{,}52\ G + 0{,}31\ B.$$

В стандарте PAL значения цветности называются U и V, и коэффициенты используются другие, но идея та же самая. Стандарт SECAM отличается и от NTSC, и от PAL.

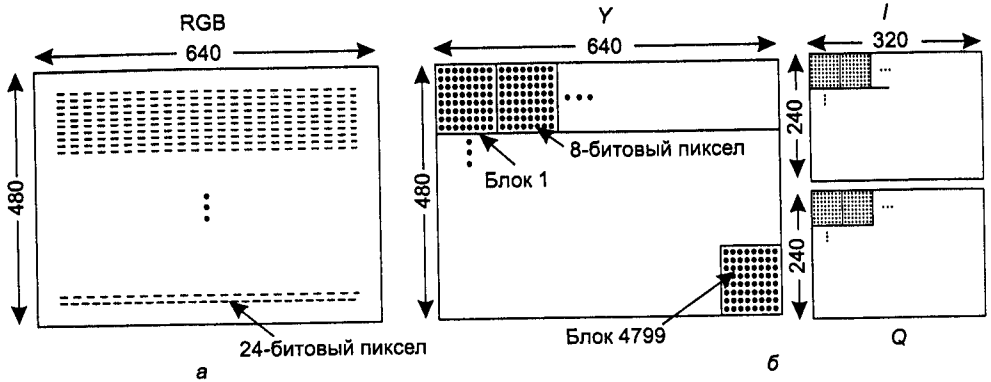


Рис. 7.38. Исходные данные в формате RGB (*а*); после подготовки блока (*б*)

Для значений Y, I и Q строятся отдельные матрицы с элементами, значения которых лежат в диапазоне от 0 до 255. Затем значения цветности I и Q усредняются по квадратным блокам из четырех пикселов, что уменьшает размеры матриц цветности в 4 раза до размера 320×240. Это сжатие является преобразованием с потерями, но человеческий глаз его почти не замечает, так как чувствительность глаза к яркости значительно выше, чем к цветности. Тем не менее, уже на этом этапе общий объем данных уменьшается вдвое. Затем из каждого элемента вычитается число 128, чтобы переместить число 0 в середину диапазона. Наконец, каждая матрица разбивается на квадраты по 8×8 пикселов. Таким образом, матрица Y состоит из 4800 квадратных блоков, а матрицы I и Q содержат по 1200 блоков каждая, как показано на рис. 7.38, *б*.

На втором этапе кодирования изображения алгоритмом JPEG к каждому из 7200 квадратных блоков отдельно применяется **дискретное косинусное преобразование**. На выходе получается 7200 матриц 8×8 коэффициентов дискретного косинусного преобразования (ДКП). Элемент (0, 0) такой матрицы представляет собой среднее значение блока. Остальные элементы содержат информацию о спектральной интенсивности каждой пространственной частоты. В теории дискретное косинусное преобразование является преобразованием без потерь (обратимым), но на практике использование чисел с плавающей точкой и трансцендентных функций приводит к ошибкам округления, в результате чего часть информации теряется. Обычно элементы матрицы ДКП-коэффициентов быстро убывают с расстоянием от элемента (0, 0), как показано на рис. 7.39.

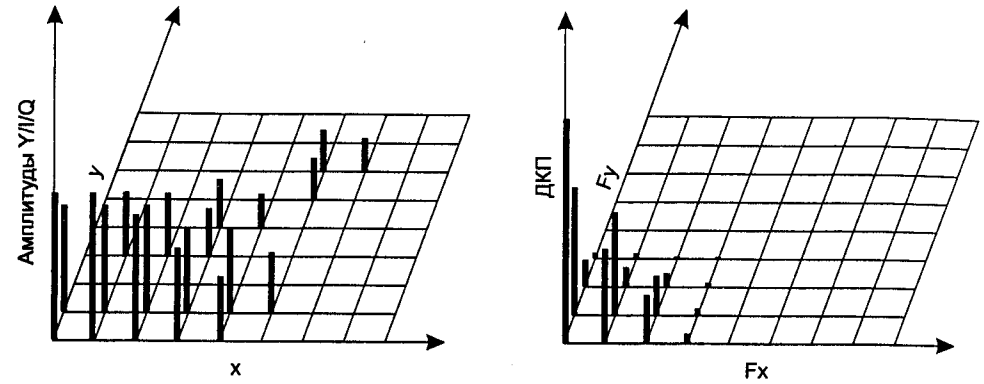


Рис. 7.39. Один блок матрицы Y (*а*); ДКП-коэффициенты (*б*)

По завершении дискретного косинусного преобразования алгоритм JPEG переходит к этапу 3, называемому **квантованием**, в котором наименее важные ДКП-коэффициенты удаляются. Это преобразование (с потерями) выполняется делением всех коэффициентов ДКП-матрицы на табличные весовые коэффициенты. Если все весовые коэффициенты равны 1, то это преобразование ничего не меняет. Однако при резком росте весовых коэффициентов по мере удаления от элемента матрицы (0, 0) более высокие пространственные частоты быстро теряются.

ДКП-коэффициенты

150	80	40	14	4	2	1	0
92	75	36	10	6	1	0	0
52	38	26	8	7	4	0	0
12	8	6	4	2	1	0	0
4	3	2	0	0	0	0	0
2	2	1	1	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Квантованные коэффициенты

150	80	20	4	1	0	0	0
92	75	18	3	1	0	0	0
26	19	13	2	1	0	0	0
3	2	2	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Таблица квантования

1	1	2	4	8	16	32	64
1	1	2	4	8	16	32	64
2	2	2	4	8	16	32	64
4	4	4	4	8	16	32	64
8	8	8	8	8	16	32	64
16	16	16	16	16	16	32	64
32	32	32	32	32	32	32	64
64	64	64	64	64	64	64	64

Рис. 7.40. Квантование ДКП-коэффициентов

Пример этого этапа работы алгоритма показан на рис. 7.40. Здесь мы видим исходную ДКП-матрицу, таблицу квантования и результат деления каждого

ДКП-элемента на соответствующий элемент таблицы квантования. Значения в таблице квантования не являются частью стандарта JPEG. Каждое приложение должно содержать свою таблицу весовых коэффициентов, что позволяет ему контролировать соотношение потерь и коэффициента сжатия.

На четвертом этапе значение, содержащееся в элементе (0, 0) каждого блока (в левом верхнем углу квадрата), заменяется его отклонением относительно значения в предыдущем блоке. Так как эти значения представляют собой усредненные величины своих блоков, они должны меняться медленно, следовательно, полученные в результате разности должны быть невелики. Для остальных значений разности не вычисляются. Значения элементов (0, 0) называются DC-компонентами, а остальные элементы — AC-компонентами.

На пятом этапе 64 элемента блока выстраиваются в ряд, к которому применяется кодирование длин серий. Чтобы сконцентрировать нули в конце ряда, сканирование блока выполняется зигзагом, как показано на рис. 7.41. В нашем примере в конце блока группируется 38 нулей. Эта строка нулей заменяется просто числом 38. В этой замене и состоит результат работы метода, называемого **кодированием длин серий**.

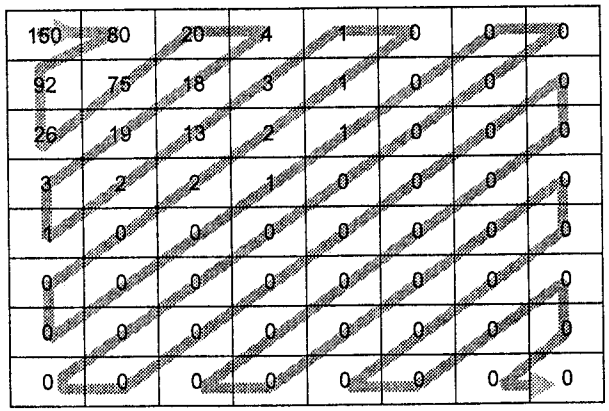


Рис. 7.41. Порядок передачи квантованных значений

Теперь у нас есть список чисел, представляющий изображение (в трансформированном виде). На шестом этапе применяется код Хаффмана (Huffman), присваивающий часто встречающимся последовательностям короткие коды, а редко встречающимся — более длинные коды.

Алгоритм JPEG может показаться сложным, но это только потому, что он и в самом деле является таковым. Тем не менее, он широко применяется, так как позволяет сжимать фотографии в 20 и более раз. Для декодирования сжатого изображения нужно выполнить все те же операции в обратном порядке. Алгоритм JPEG почти симметричен: декодирование занимает столько же времени, сколько и кодирование. Это справедливо далеко не для всех алгоритмов, как мы увидим позже.

Стандарт MPEG

Наконец мы подходим к ключевому вопросу мультимедиа — стандартам **MPEG** (Motion Pictures Experts Group — экспертная группа по вопросам движущегося изображения). Эти стандарты, ставшие международными в 1993 году, описывают основные алгоритмы, используемые для сжатия видеофильмов. Поскольку фильмы содержат как изображение, так и звук, алгоритмы MPEG занимаются сжатием и того, и другого. Принципы сжатия аудиоданных и неподвижных изображений мы рассмотрели ранее, поэтому этот раздел мы посвятим обсуждению алгоритмов сжатия видео.

Первым законченным стандартом стал стандарт MPEG-1 (Международный Стандарт 11172). Его целью было создание выходного потока данных качества бытового видеомагнитофона (352×240 для NTSC) на скорости 1,2 Мбит/с. Мы уже видели ранее, что несжатый поток видеоинформации может составлять 472 Мбит/с (для экрана размером 1024×768). Для экрана 352×240 потребуется в 9,3 раза меньший поток данных, то есть 50,7 Мбит/с. Тем не менее и это значение намного превышает 1,2 Мбит/с, так что данную задачу никак нельзя назвать тривиальной. Видеофильм в формате MPEG-1 можно передавать по витой паре на умеренные расстояния. Кроме того, фильмы в этом формате можно хранить на компакт-дисках.

Следом появился стандарт MPEG-2 (Международный стандарт 13818), разрабатывавшийся для сжатия видеофильмов качества широковещания до скорости потока от 4 до 6 Мбит/с, что позволяло передавать этот фильм в цифровом виде по стандартному телевизионному каналу NTSC или PAL. Позднее стандарт MPEG-2 был расширен для поддержки видеосигнала более высокого разрешения, включая HDTV. Сейчас он очень популярен, поскольку составляет основу DVD и цифрового спутникового телевидения.

Основные принципы стандартов MPEG-1 и MPEG-2 одинаковы. В первом приближении стандарт MPEG-2 является расширением стандарта MPEG-1 с дополнительными возможностями, форматами кадров и вариантами кодирования. Сначала мы рассмотрим MPEG-1, а затем MPEG-2.

Алгоритм MPEG-1 состоит из трех частей: аудио- и видеошифраторов и системы, объединяющей эти два потока данных, как показано на рис. 7.42. Системы кодирования аудио и видео работают независимо, в результате чего возникает проблема синхронизации этих потоков в приемнике. Для решения этой проблемы используются системные часы, работающие на частоте 90 кГц. Показания этих часов хранятся в 33-битных счетчиках, что позволяет указывать время в интервале 24 часов. Эти отметки времени вносятся в закодированный результат и передаются получателю, который может использовать их для синхронизации изображения и звука.

Перейдем к рассмотрению сжатия видеоинформации с помощью алгоритма MPEG-1. В видеофильмах имеется избыточность двух типов: пространственная и временная. Алгоритм сжатия MPEG-1 использует оба типа. Чтобы использовать пространственную избыточность, можно просто кодировать каждый кадр отдельно алгоритмом JPEG. Иногда такой подход применяется, особенно если нужен прямой доступ к каждому кадру, например, при редактировании видеоизображе-

ния. Этот режим сжатия позволяет снизить поток данных до скорости от 8 до 10 Мбит/с.

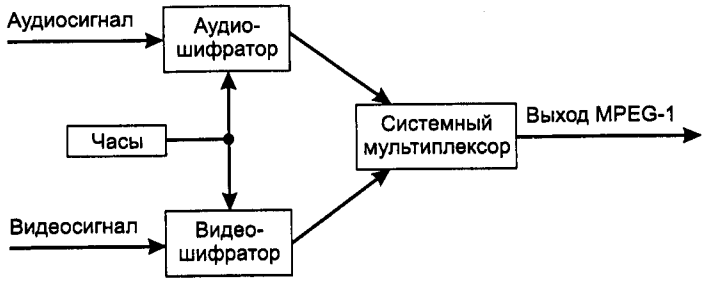


Рис. 7.42. Синхронизация аудио- и видеопотоков в MPEG-1

Дополнительного сжатия можно достичь, используя преимущество того факта, что идущие подряд кадры часто бывают почти идентичны. Может показаться, что этот эффект не столь уж велик, так как в большинстве фильмов сцены часто меняются — в среднем каждые 3–4 с.. Тем не менее даже последовательность в 75 очень близких кадров позволяет применить значительно большее сжатие, нежели с помощью компрессии отдельных кадров алгоритмом JPEG.

В сценах, в которых камера и задний план неподвижны и один или два актера медленно двигаются, почти все пикселы в соседних кадрах будут идентичны. В этом случае простое вычитание каждого кадра из предыдущего и обработка разности алгоритмом JPEG даст достаточно хороший результат. Однако этот метод плохо подходит к сценам, в которых камера поворачивается или наезжает на снимаемый объект. Для таких сцен необходим какой-то способ компенсировать это движение камеры. В этом и состоит основное отличие MPEG от JPEG.

Выходной поток MPEG-1 состоит из кадров четырех следующих типов:

1. I (Intracoded — автономные) — независимые неподвижные изображения, кодированные алгоритмом JPEG.
2. P (Predictive — предсказывающие) — содержат разностную информацию относительно предыдущего кадра.
3. B (Bidirectional — двунаправленные) — содержат изменения относительно предыдущего и последующего кадров.
4. D (DC-coded — усредненные) — содержат усредненное по блоку значение, используемые для быстрой перемотки вперед.

I-кадры представляют собой обычные неподвижные изображения, кодированные алгоритмом JPEG с использованием полного разрешения яркости и половинного разрешения для обоих сигналов цветности. I-кадры должны периодически появляться в выходном потоке по трем причинам. Во-первых, должна быть возможность просмотра фильма не с самого начала. Зритель, пропустивший первый кадр, не сможет декодировать все последующие кадры, если все кадры будут зависеть от предыдущих и I-кадры не будут время от времени включаться в поток. Во-вторых, дальнейшее декодирование фильма станет невозможным в случае ошибки при передаче какого-либо кадра. В-третьих, наличие таких кадров существенно упростит индикацию во время быстрой перемотки вперед или назад. По этим причинам I-кадры включаются в выходной поток примерно один-два раза в секунду.

P-кадры, напротив, представляют собой разность между соседними кадрами. Они основаны на идее **макроблоков**, покрывающих 16×16 пикселов в пространстве яркости и 8×8 пикселов в пространстве цветности. При кодировании макроблока в предыдущем кадре ищется наиболее близкий к нему макроблок.

Пример использования макроблоков показан на рис. 7.43. Здесь мы видим три последовательных кадра с одинаковым задним планом, различающихся только положением человека. Макроблоки, содержащие задний план, будут полностью совпадать друг с другом, но макроблоки, содержащие человека, будут смещаться на некоторую величину. Именно для этих макроблоков алгоритм должен найти максимально похожие макроблоки из предыдущего кадра.

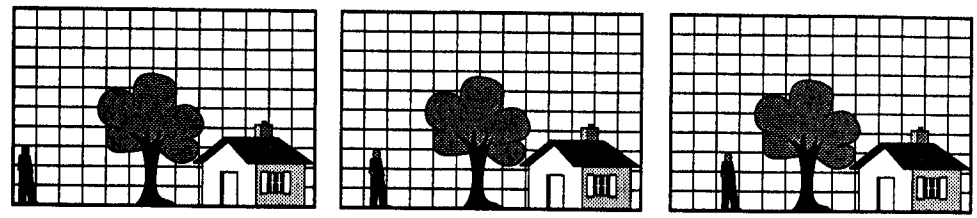


Рис. 7.43. Три последовательных кадра

Стандарт MPEG-1 не указывает, как искать, насколько далеко искать или насколько хорошо должен подходить похожий макроблок. Все эти вопросы оставлены на усмотрение разработчика программы, реализующей стандарт MPEG-1. Например, макроблок можно искать в предыдущем кадре в текущей позиции с заданными смещениями по горизонтали и вертикали. Для каждой позиции может вычисляться количество совпадений матрицы яркости. Позиция с наибольшим значением совпадений будет объявляться победительницей при условии, что это значение превосходит некое пороговое значение. В противном случае макроблок будет считаться не найденным. Возможны, конечно, и значительно более сложные алгоритмы.

Макроблок, для которого найден похожий на него макроблок в предыдущем кадре, кодируется в виде разности значений яркости и цветности. Затем матрицы разностей подвергаются дискретному косинусному преобразованию, квантованию, кодированию длин серий и кодированию Хаффмана так же, как это делает алгоритм JPEG. В выходном потоке макроблок представляется в виде вектора сдвига (насколько далеко сдвинулся макроблок по горизонтали и вертикали от положения в предыдущем кадре), за которым следует список чисел, кодированных по Хаффману. Если же в предыдущем кадре не нашлось подходящего макроблока, текущее значение кодируется алгоритмом JPEG, как в I-кадре.

Очевидно, что этот алгоритм обладает сильной асимметричностью. В принципе, программа может искать соответствие макроблоку в любой позиции предыдущего кадра. Такой подход позволит минимизировать выходной поток MPEG-1 за счет очень медленного кодирования. Следовательно, такой метод применим для одноразового кодирования при создании фильмотеки, но не годится для видеоконференций реального времени.

Аналогично разработчики каждой реализации могут самостоятельно определять алгоритм сравнения макроблоков. Эта свобода позволяет разработчикам соревноваться в качестве и скорости своих алгоритмов при полной совместимости создаваемых алгоритмами потоков MPEG-1. Независимо от алгоритма поиска конечный выходной поток будет состоять либо из текущего блока, закодированного по стандарту JPEG, либо из закодированной в соответствии с тем же стандартом JPEG разности текущего и одного из предыдущего кадров, расстояние до которого должно быть указано.

Декодирование потока MPEG-1 выполняется довольно просто. Декодирование I-кадров аналогично декодированию JPEG-изображений. Чтобы декодировать P-кадр, декодер должен сохранить в буфере предыдущий кадр, а затем во втором буфере построить новый кадр из макроблоков как в абсолютной, так и в относительной кодировке. Кадры собираются макроблок за макроблоком.

B-кадры аналогичны P-кадрам — с той лишь разницей, что позволяют привязывать макроблок либо к предыдущему, либо к следующему кадру. Такая дополнительная свобода позволяет достичь хорошей компенсации движения. Для декодирования B-кадров необходимо удерживать в памяти сразу три кадра: предыдущий, текущий и следующий. Хотя B-кадры позволяют добиться наибольшего сжатия, они поддерживаются не всеми реализациями.

D-кадры используются только для индикации изображения с низким разрешением при быстрой перемотке фильма. Выполнение обычного декодирования видеоформата MPEG-1 уже само по себе требует достаточной мощности процессора. Выполнять же это декодирование в десять раз быстрее при поиске практически нереально. Каждый D-кадр представляет собой просто среднее значение блока, без дальнейшего кодирования, что упрощает его отображение в режиме реального времени.

Перейдем теперь к стандарту MPEG-2. В основном, этот стандарт схож со стандартом MPEG-1, но он не поддерживает D-кадры. Кроме того, в нем применяется дискретное косинусное преобразование матрицы размером не 8×8, а 10×10, что увеличивает число ДКП-коэффициентов в полтора раза, результатом чего является и более высокое качество. Поскольку стандарт MPEG-2 предназначается как для широковещательного телевидения, так и для DVD, в нем имеется поддержка и поступательного, и чересстрочного отображения (в отличие от MPEG-1, поддерживающего только поступательное отображение). У этих двух стандартов имеются также и другие различия, правда, не столь значительные.

В отличие от стандарта MPEG-1, поддерживающего только одно разрешение, стандарт MPEG-2 поддерживает четыре: низкое (352×240), основное (720×480), высокое-1440 (1440×1152) и высокое (1920×1080). Низкое разрешение предназначено для бытовых видеомагнитофонов и совместимости со стандартом MPEG-1.

Основное является нормой для широковещания в режиме NTSC. Остальные два режима предназначены для HDTV. Для обеспечения высокого качества изображения MPEG-2 обычно работает со скоростью 4–8 Мбит/с.

Видео по заказу

Видео по заказу иногда сравнивают с электронным видеопрокатом. Пользователь (клиент) выбирает из большого списка доступных видеофильмов один и берет его, чтобы просмотреть дома. В случае видео по заказу выбор производится не выходя из дома, с помощью пульта дистанционного управления телевизора, а заказанный фильм начинается немедленно. В пункт проката идти не нужно. Надо ли говорить, что реализация видео по заказу несколько сложнее его описания. В данном разделе мы познакомимся с основными идеями и их реализацией.

Действительно ли видео по заказу подобно видеопрокату или оно больше напоминает выбор фильма в системе кабельного телевидения, состоящей из 500 или 5000 каналов? От ответа на этот вопрос зависят важные технические решения. В частности, пользователи видеопроката привыкли к тому, что можно остановить просмотр, совершить прогулку на кухню или в ванную комнату, а затем возобновить просмотр с того места, на котором они остановили фильм. У телезрителей же кнопки ПАУЗА нет.

Если видео по заказу собирается успешно конкурировать с пунктами проката видеокассет, возможно, нужно позволить пользователям останавливать, запускать и перематывать видеофильмы. Чтобы обеспечить пользователю такую возможность, каждому зрителю нужно передавать отдельную копию фильма.

С другой стороны, если рассматривать видео по заказу скорее как обычное телевидение с заранее составленным расписанием, тогда возможен другой подход, при котором популярный фильм передается сразу по нескольким каналам с интервалом в 10 минут. Пользователь, желающий посмотреть этот фильм, должен подождать начала фильма несколько минут. Хотя при такой реализации приостановка и возобновление просмотра невозможны, пользователь, вернувшийся к телевизору после короткого перерыва, может переключиться на один из соседних каналов и найти тот же фильм, но идущий с отставанием на 10 или 20 минут. Такую схему называют **«почти видео по заказу»**. Ее реализация обходится значительно дешевле, так как хотя для передачи одного фильма и используется полтора десятка параллельных каналов, но предполагается, что этот фильм одновременно смотрят тысячи зрителей. Различие между видео по заказу и этой схемой примерно такое же, как между личным и общественным транспортом.

Просмотр видео по заказу представляет собой всего лишь одну из большого набора новых услуг, которые станут возможными, как только сети с большой пропускной способностью получат широкое распространение. Общая модель показана на рис. 7.44. В центре системы мы видим глобальную сетевую магистраль (национальную или интернациональную) с высокой пропускной способностью. С ней соединены тысячи локальных распределительных сетей, таких как сети кабельного телевидения или телефонные сети. Локальные распределительные сети доходят до домов пользователей, в которых они заканчиваются телевизион-

ными приставками (английское название: set-top box — коробочка, которую кладут на телевизор), являющимися, по сути, мощными специализированными персональными компьютерами.

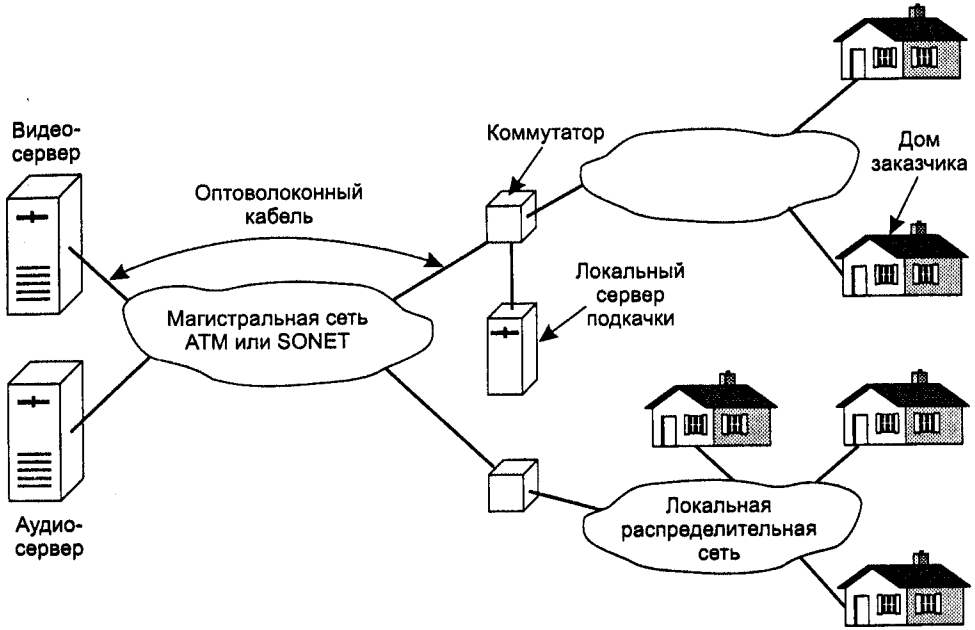


Рис. 7.44. Общая модель системы видео по заказу

С магистралью с помощью высокоскоростных оптоволоконных кабелей соединены тысячи поставщиков информации. Некоторые из них будут предоставлять видеофильмы и аудиокомпакт-диски за плату, другие будут специализироваться на таких услугах, как интернет-магазины (с возможностью прокрутить тарелку с супом и изменить масштаб списка ингредиентов или с демонстрацией видеоролика, объясняющего, как пользоваться газонокосилкой с бензиновым мотором). Без сомнения, быстро станут доступными бесчисленные возможности, такие как спорт, новости, сериалы, доступ к Всемирной паутине и т. д.

В систему также входят локальные серверы подкачки, позволяющие снижать нагрузку на главные магистрали в часы пик. Вопросы стыковки отдельных узлов этой системы и вопросы собственности на них в настоящее время являются предметами жарких споров. Далее мы рассмотрим устройство основных частей системы: видеосерверов и распределительных сетей.

Видеосерверы

Для предоставления видео по заказу необходимы специализированные **видеосерверы**, способные хранить и передавать одновременно большое количество фильмов. Общее число когда-либо снятых фильмов было оценено в 65 000 (Minoli, 1995). В сжатом с помощью алгоритма MPEG-2 виде обычный фильм занимает

около 4 Гбайт, таким образом, для хранения 65 000 фильмов потребуется около 260 Тбайт. Добавьте к этому все когда-либо сделанные старые телевизионные программы, спортивные передачи, новости, рекламу и т. д., и станет ясно, что мы имеем дело с проблемой хранения данных в промышленных масштабах.

Дешевле всего хранить большие объемы информации на магнитной ленте. Так было раньше, возможно, так будет и в дальнейшем. На кассету типа Ultrium можно записать 200 Гбайт (50 фильмов) по цене около $1–2 за фильм. Уже сейчас можно приобрести большие механические кассетные серверы, хранящие тысячи кассет, снабженные автоматическими манипуляторами, способными менять кассеты в магнитофоне. Основными проблемами таких систем остаются время доступа (особенно к 50-му фильму на кассете), скорость передачи и ограниченное количество магнитофонов (для одновременного показа n фильмов нужно n магнитофонов).

К счастью, опыт работы с пунктами видеопроката, библиотеками и другими подобными организациями показывает, что не все фильмы (книги) одинаково популярны. Экспериментально доказано, что если в пункте проката есть N фильмов, то доля заявок на конкретный фильм, стоящий на k-м месте в списке популярности, примерно равна C/k. Здесь C — это число, дополняющее сумму долей до 1, а именно:

$$C = 1/(1 + 1/2 + 1/3 + 1/4 + 1/5 + ... + 1/N).$$

Таким образом, например, самый популярный фильм берут примерно в семь раз чаще, чем седьмой в списке популярности. Такая зависимость называется законом Ципфа (Zipf, 1949).

Тот факт, что одни фильмы значительно популярнее других, наводит на идею иерархической организации хранилища фильмов, показанной на рис. 7.45. В изображенной пирамиде производительность возрастает при движении вверх.

Рис. 7.45. Иерархия хранилища видеосервера

Фильмы можно хранить и на компакт-дисках. Современные диски DVD позволяют хранить 4,7 Гбайт данных, чего вполне достаточно для записи одного фильма, однако следующее поколение дисков будет иметь примерно удвоенную емкость. Хотя по сравнению с жесткими магнитными дисками время поиска на оптическом диске на порядок выше (50 мс вместо 5 мс), их низкая цена и высокая надежность делает автомат с тысячами сменных DVD хорошей альтернативой магнитной ленте для более часто используемых фильмов.

На следующем уровне в этой иерархии находятся жесткие магнитные диски. Они обладают малым временем доступа (5 мс), высокой скоростью передачи

(320 Мбайт/с для SCSI 320) и значительной емкостью (свыше 100 Гбайт), что делает их вполне подходящими для хранения фильмов, которые действительно смотрят зрители (в отличие от тех, что хранятся на тот случай, если кто-либо захочет их посмотреть). Основной недостаток накопителей на жестких дисках заключается в их слишком высокой цене, чтобы хранить на них редко просматриваемые фильмы.

На вершине пирамиды, показанной на рис. 7.45, находится ОЗУ. Оперативная память является самым быстрым запоминающим устройством, но в то же время и самым дорогим. Оперативная память лучше всего подходит для хранения фильмов, посылаемых в данный момент различным получателям одновременно (например, настоящее видео по заказу 100 пользователям, начавшим просмотр в различное время). Когда цены на микросхемы памяти упадут до 50 долларов за гигабайт, 4 Гбайт ОЗУ, необходимые для хранения одного фильма, будут стоить 200 долларов, а 100 фильмов, находящихся в памяти, потребуют ОЗУ на 20 000 долларов (400 Гбайт). То есть видеосервер, имеющий в наличии 100 фильмов, скоро сможет позволить себе хранить их все в ОЗУ. А если у видеосервера всего 100 клиентов, которые день за днем смотрят одни и те же 20 фильмов, это уже не только реальное, но и выгодное решение.

Поскольку видеосервер представляет собой в действительности просто огромное устройство ввода-вывода, работающее в режиме реального времени, его аппаратная и программная архитектура должна отличаться от используемой в PC или рабочей станции UNIX. Аппаратная архитектура типичного видеосервера показана на рис. 7.46. Сервер состоит из одного или нескольких высокопроизводительных процессоров, у каждого из которых есть своя локальная память, память общего доступа, большой кэш ОЗУ для хранения популярных фильмов, множества различных накопителей для хранения фильмов, а также сетевое оборудование, например, оптического интерфейса с магистралью SONET или ATM с каналом OC-12 или выше. Все эти подсистемы соединены сверхскоростной шиной (по меньшей мере 1 Гбайт/с).

Познакомимся теперь с программным обеспечением видеосервера. Процессоры принимают запросы пользователей, находят нужные фильмы, перемещают данные между устройствами, выписывают счета клиентам, а также выполняют множество других операций. Для некоторых из этих функций фактор времени не является критичным, но для большинства он критичен, поэтому на некоторых (если не на всех) центральных процессорах должна работать операционная система реального времени, такая как микроядро реального времени. Такие системы обычно разбивают задание на отдельные небольшие задачи, для каждой из которых известен срок ее окончания. Затем программа, следящая за расписанием выполнения задач, может запустить, например, алгоритм выполнения задач в порядке сроков выполнения или алгоритм монотонной скорости (Liu и Layland, 1973).

Программное обеспечение, работающее в центральных процессорах, также определяет тип интерфейса, предоставляемого сервером своим клиентам (серверам подкачки и телевизионным приставкам). Наиболее популярными являются два интерфейса. Один из них представляет собой традиционную файловую систему, в которой клиент может открывать, читать, писать и закрывать файлы. Помимо

сложностей, связанных с иерархией хранения и с соображениями реального времени, в таком интерфейсе клиенту придется еще заниматься и вопросами файловой системы, например, UNIX.

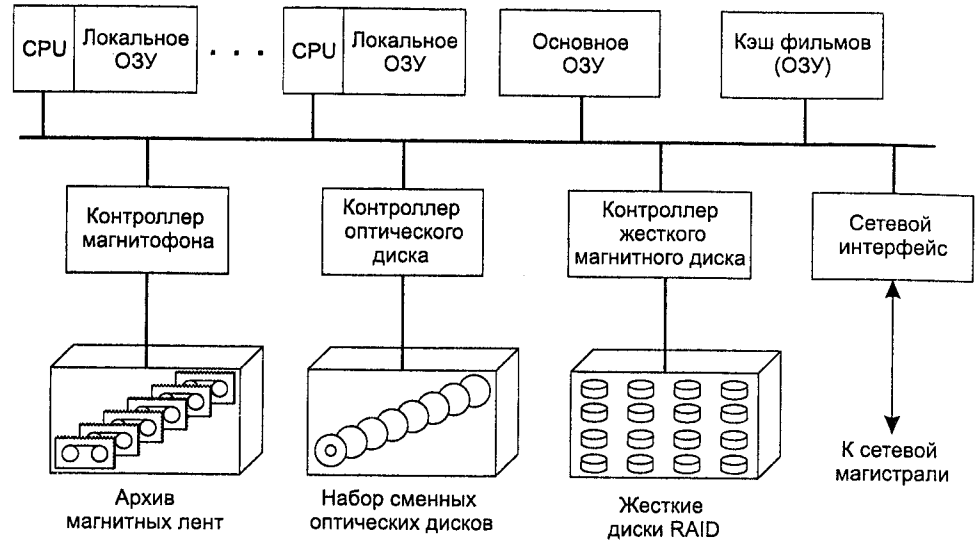


Рис. 7.46. Аппаратная архитектура типичного видеосервера

В основе второго типа интерфейса лежит модель видеомагнитофона. Сервер получает команды открытия, начала или приостановки воспроизведения, а также команды быстрой перемотки в обе стороны. Основное отличие этого интерфейса от модели UNIX состоит в том, что, получив команду PLAY (воспроизведение), сервер начинает выдавать данные с постоянной скоростью без необходимости передачи новых команд.

Центральной частью программного обеспечения видеосервера является программа управления дисками. Она выполняет две основные задачи: копирование фильмов на жесткий диск с оптического диска и обработка дисковых запросов для нескольких выходных потоков. Вопрос размещения фильма очень важен, так как он сильно влияет на производительность.

Возможны два способа организации дискового хранения: дисковая ферма и дисковый массив. **Дисковая ферма** подразумевает хранение на каждом диске нескольких фильмов целиком. Каждый фильм должен дублироваться на двух или даже более дисках, чтобы обеспечить высокую надежность и производительность. **Дисковый массив**, или набор **RAID** (Redundant Array of Independent Disks — матрица независимых дисковых накопителей с избыточностью), является формой организации хранения, в которой фильм хранится сразу на нескольких дисках, например, блок 0 на диске 0, блок 1 на диске 1 и так далее циклически. Такая организация хранения называется **разбиением на полосы**.

Разбитый на полосы массив диска обладает рядом преимуществ перед дисковой фермой. Во-первых, все n дисков могут работать параллельно, увеличивая

производительность системы в n раз. Во-вторых, такой набор можно сделать отказоустойчивым, если к каждой группе дисков добавить еще один диск, на котором хранить поблочную сумму по модулю два всех данных набора. Наконец, такой способ автоматически решает вопрос балансировки нагрузки. (Все популярные фильмы не смогут оказаться на одном диске). С другой стороны, организация дисковых массивов является более сложной, чем дисковая ферма, и чрезвычайно чувствительной к множественным отказам. Она также плохо подходит для таких операций видеомагнитофона как быстрая перемотка назад или вперед.

Другая задача дискового программного обеспечения состоит в обслуживании выходных потоков реального времени и удовлетворении соответствующих временных требований. Еще несколько лет назад это было связано с необходимостью реализации сложных алгоритмов планирования, однако по мере снижения цен на модули памяти стал возможен более простой подход. Для передачи каждого видеопотока необходимо хранить в буфере ОЗУ отрезок фильма длиной, скажем, 10 с (5 Мбайт). Буфер заполняется путем выполнения дисковой операции и освобождается сетевым процессом. Имея 500 Мбайт оперативной памяти, мы сможем одновременно буферизировать 100 потоков. Конечно, дисковая подсистема должна обладать устойчивой скоростью передачи данных 50 Мбайт/с для нормального заполнения буферов, однако матрица RAID, составленная из высокопроизводительных дисков типа SCSI, запросто может справиться с этой задачей.

Распределительная сеть

Распределительная сеть представляет собой набор коммутаторов и линий связи между источником данных и их получателями. Как было показано на рис. 7.44, она состоит из магистрали, соединенной с локальными распределительными сетями. Обычно магистраль является коммутируемой, а локальная распределительная сеть — нет.

Основным требованием, предъявляемым к магистрали, является высокая пропускная способность. Раньше вторым требованием был низкий уровень флуктуации (джиттера), то есть неравномерности трафика, однако теперь, когда даже самые дешевые ПК получили возможность буферизировать 10 с высококачественного видео в формате MPEG-2, это требование перестало быть актуальным.

Локальное распределение представляет собой чрезвычайно неупорядоченное, хаотическое явление — это связано с тем, что различные компании пытаются предоставлять услуги в различных регионах по различным сетям. Телефонные компании, компании кабельного телевидения, а также совершенно новые организации пытаются первыми застолбить участок. В результате наблюдается быстрое распространение новых технологий. В Японии некоторые компании, занимающиеся канализационными трубами, утверждают, что именно они обладают преимуществом, так как принадлежащие им трубы большого диаметра подведены в каждый дом (через них действительно прокладывается оптоволоконный кабель, однако нужно очень аккуратно следить за местами его вывода из труб). В настоящее время уже используется множество схем локального распределения видео по заказу, из которых четырьмя основными являются ADSL, FTTC, FTTH и HFC. Мы рассмотрим их все по очереди.

Система ADSL (Asymmetric Digital Subscriber Line — асимметричная цифровая абонентская линия) была первой попыткой телефонных компаний поучаствовать в деле локального распределения. Мы изучали ADSL в главе 2 и сейчас не будем повторять этот материал. Идея состояла в том, что практически каждый дом в США, Европе и Японии уже оснащен медными витыми парами (для аналоговой телефонной связи). Если бы эти провода можно было использовать для видео по заказу, то телефонные компании могли бы сорвать большой куш.

Проблема заключалась в том, что по этим проводам невозможно передать на их обычную длину в 10 км даже поток данных MPEG-1, не говоря уже о MPEG-2. Для передачи полноцветного видео с плавными движениями и высоким разрешением необходима скорость 4–8 Мбит/с в зависимости от требуемого качества. Таких скоростей с помощью ADSL достичь невозможно (вернее, возможно, но только при передаче на очень малые расстояния).

Второй вариант распределительной сети был также разработан телефонными компаниями. Он получил название **FTTC** (Fiber To The Curb — оптоволоконный кабель к распределительной коробке). В данной модели телефонная компания прокладывает оптический кабель от каждой конечной телефонной станции до устройства, расположенного по соседству с абонентами и называющегося блоком **ONU** (Optical Network Unit — блок оптической сети). К блоку ONU может быть подключено около 16 медных витых пар. Поскольку эти витые пары короткие, по ним можно передавать дуплексные потоки T1 или T2 и, соответственно, фильмы в форматах MPEG-1 или MPEG-2. Более того, благодаря симметричности схемы, FTTC может поддерживать видеоконференции.

Третье решение, предложенное телефонными компаниями, состоит в прокладке оптоволоконного кабеля в каждый дом. Эта схема называется **FTTH** (Fiber To The Home — оптоволоконный кабель к дому). При этом каждый абонент получит возможность пользоваться каналом связи OC-1, OC-3 или даже выше. Такое решение очень дорого и вряд ли получит широкое распространение в ближайшие годы, но очевидно, что возможности этого варианта практически безграничны. На рис. 7.31 мы видели схему, используя которую, каждый желающий может содержать собственную радиостанцию. А как вам идея о том, чтобы каждый член семьи был главой собственной телестудии? Все перечисленные системы— ADSL, FTTC и FTTH— представляют собой локальные распределительные сети с топологией «точка—точка», что неудивительно, учитывая то, как создавалась современная телефонная система.

Полностью противоположный подход реализуется в настоящее время поставщиками кабельного телевидения. Он называется **HFC** (Hybrid Fiber Coax — гибрид оптоволоконного и коаксиального кабелей) и показан на рис. 2.41, *а*. Суть подхода в следующем. Современные коаксиальные кабели с полосой пропускания от 300 до 450 МГц будут заменены кабелями с полосой в 750 МГц, что повысит их пропускную способность с 50–75 каналов шириной в 6 МГц до 125 таких каналов. 75 из 125 каналов по-прежнему будут использоваться для передачи аналогового телевизионного сигнала.

В 50 новых каналах будет использоваться квадратурная амплитудная модуляция QAM-256, в результате чего по каждому каналу можно будет передавать

данные со скоростью около 40 Мбит/с, что в сумме составит 2 Гбит/с. Каждым кабелем предполагается охватить около 500 домов, Таким образом, каждому дому может быть предоставлен выделенный канал с пропускной способностью 4 Мбит/с, который можно будет использовать для передачи MPEG-2.

При всех достоинствах данного подхода, для его реализации потребуется замена всех имеющихся кабелей на новые, установка новых концевых адаптеров и удаление всех односторонних усилителей — в общем, замена всей системы кабельного телевидения. Таким образом, общий объем новой инфраструктуры здесь сопоставим с тем, что требуется телефонным компаниям для организации FTTC. В обоих случаях поставщик локальных сетевых услуг должен прокладывать волоконно-оптический кабель довольно близко к домам заказчиков. Кроме того, в обоих случаях оптоволоконный кабель заканчивается оптоэлектрическим преобразователем. Разница этих двух систем в том, что в FTTC на последнем участке используется выделенная витая пара, соединяющая абонента с распределительной коробкой соединением «точка—точка», тогда как в HFC последний сегмент распределительной сети представляет собой общий коаксиальный кабель. Технически эти две системы различаются не столь сильно, как это часто утверждают их сторонники.

Тем не менее, на одно различие следует обратить внимание. Система HFC использует общий носитель без коммутации и маршрутизации. Любая информация, проходящая по кабелю, может быть прочитана любым абонентом, подключенным к этому кабелю, без каких-либо хлопот. Система FTTC, являясь полностью коммутируемой, не обладает этим свойством. В результате операторы системы HFC требуют от видеосерверов способности шифровать потоки, чтобы клиенты, не заплатившие за просмотр фильма, не могли смотреть его. Операторам системы FTTC шифрование не нужно, так как оно увеличивает сложность системы, снижает ее производительность и не повышает защищенность системы. Нужно ли шифрование с точки зрения компании-владельца видеосервера? Телефонная компания, управляющая сервером, может решить намеренно отказаться от шифрования видеофильмов, якобы по соображениям эффективности, но на самом деле, чтобы экономически подорвать своих конкурентов, предоставляющих аналогичные услуги с помощью системы HFC.

Во всех подобных распределительных сетях в каждой локальной области обслуживания обычно используется один или несколько серверов подкачки. Они представляют собой уменьшенные версии видеосерверов, о которых рассказывалось ранее. Большим достоинством этих локальных серверов является то, что они берут на себя некоторую часть магистральной нагрузки.

На локальные серверы можно заранее копировать фильмы. Если пользователи отошлют заявки на фильмы, которые они хотели бы просмотреть, заранее, такие фильмы могут копироваться на локальные серверы в периоды минимальной загрузки линий. Можно ввести гибкие тарифы на заблаговременные заказы фильмов по аналогии с бронированием авиабилетов. Так, например, можно давать 27-процентную скидку при заблаговременном заказе фильмов (от 24 до 72 часов) для просмотра во вторник или в четверг до 18:00 или после 23:00. Фильмы, заказанные в первое воскресенье месяца до 8:00 для просмотра в среду после полудня в день, дата которого является простым числом, получают 43-процентную скидку, и т. д.

Система MBone

В то время как столько различных промышленных предприятий составляют и публикуют свои грандиозные планы на будущее, Интернет-сообщество спокойно занимается реализацией своей собственной мультимедийной системы, называющейся **MBone** (Multicast Backbone — широковещательная магистраль). В данном разделе мы познакомимся с этой системой и с принципом ее работы.

Система MBone представляет собой телевещание по Интернету. В отличие от видео по заказу, в котором упор делается на заказе и просмотре заранее сжатых и хранящихся на сервере видеофильмов, система MBone используется для широковещательного распространения по Интернету передач, выходящих в прямом эфире, в цифровой форме. Эта система существует с 1992 года. Она использовалась для организации многочисленных научных телеконференций, особенно встреч IETF (Internet Engineering Task Force — проблемная группа проектирования Интернета), а также для освещения различных значимых событий науки, например, запусков космических челноков. По системе MBone были проведены трансляции концерта Rolling Stones и некоторых эпизодов Каннского кинофестиваля. Следует ли это событие считать значимым научным событием — вопрос спорный.

Технически система MBone представляет собой виртуальную сеть поверх сети Интернет. Она состоит из соединенных туннелями островов, на которых возможна многоадресная рассылка, как показано на рис. 7.47. На рисунке сеть MBone состоит из шести островов, от *A* до *F*, соединенных семью туннелями. Каждый остров (обычно это локальная сеть или группа соединенных друг с другом локальных сетей) поддерживает многоадресную рассылку на аппаратном уровне. По туннелям, соединяющим острова, рассылаются MBone-пакеты. Когда-нибудь в будущем, когда все маршрутизаторы будут напрямую поддерживать многоадресную рассылку, такая оверлейная структура станет ненужной, но пока что именно она обеспечивает работу системы.

На каждом острове находится один или несколько специальных **многоадресных маршрутизаторов**. Некоторые из них являются обычными маршрутизаторами, но большая часть представляет собой просто рабочие станции UNIX, на которых на пользовательском уровне работает специальное программное обеспечение. Многоадресные маршрутизаторы логически соединены туннелями. MBone-пакеты инкапсулируются в IP-пакеты и рассылаются как обычные одноадресные пакеты по IP-адресу многоадресного маршрутизатора.

Туннели настраиваются вручную. Обычно туннель проходит по пути, для которого существует физическое соединение, хотя это не требуется. Если вдруг физический путь, по которому проходит туннель, прервется, многоадресные маршрутизаторы, использующие этот туннель, даже не заметят этого, так как Интернет автоматически изменит маршрут всего IP-трафика между ними.

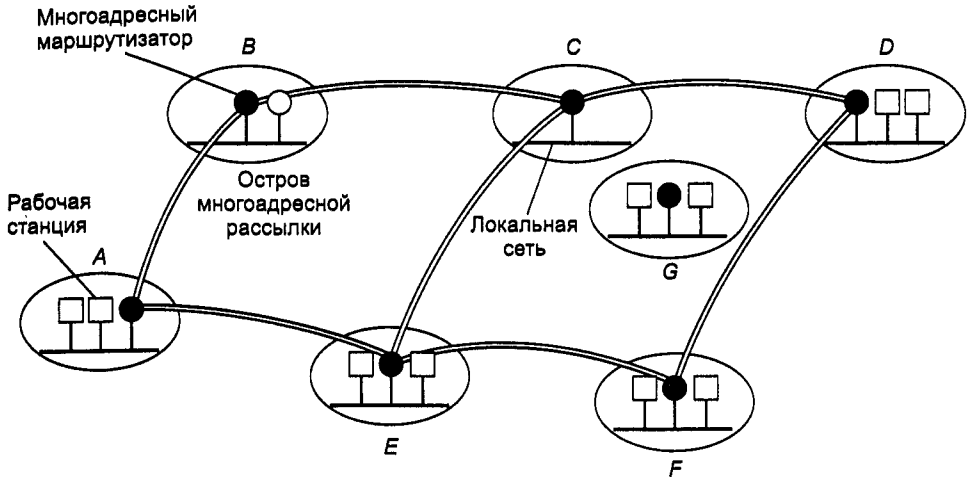


Рис. 7.47. Структура системы MBone

При появлении нового острова, желающего присоединиться к системе MBone, как, например, остров *G* на рис. 7.47, его администратор посылает по списку рассылки системы MBone сообщение, в котором объявляет о создании нового острова. После этого администраторы соседних сайтов связываются с ним для установки туннелей. Иногда при этом заново прокладываются уже существующие туннели, так как появление нового острова позволяет оптимизировать топологию системы. Физически туннели не существуют. Они определяются таблицами многоадресных маршрутизаторов и могут добавляться, удаляться или перемещаться при помощи простого изменения содержимого таблиц. В системе MBone обычно у каждой страны есть магистраль, с которой соединены региональные острова. Пара туннелей системы пересекает Атлантический и Тихий океаны, что делает ее глобальной системой.

Таким образом, в любой момент времени система MBone состоит из островов и туннелей, независимо от числа используемых в данный момент групповых адресов. Эта ситуация очень напоминает нормальную (физическую) подсеть, поэтому к ней применимы нормальные алгоритмы маршрутизации. Соответственно, в системе MBone изначально использовался алгоритм маршрутизации **DVMRP** (Distance Vector Multicast Routing Protocol — протокол дистанционной маршрутизации сообщений с использованием векторной многоканальной трансляции), основанный на дистанционно-векторном алгоритме Беллмана—Форда (Bellman—Ford). Например, остров *C* может быть связан с островом *A* либо через остров *B*, либо через остров *E* (или, возможно, через остров *D*). Алгоритм делает свой выбор на основании сообщаемых ему узлами значений своей удаленности от острова *A* и своей удаленности от других островов. Подобным образом каждый остров может определить лучший маршрут до всех остальных островов. Однако, как мы скоро увидим, эти маршруты необязательно используются именно таким образом.

Познакомимся теперь с тем, как осуществляется многоадресная рассылка. Для этого источник должен сначала получить групповой адрес класса D, действующий подобно частоте радиостанции или номеру канала. Адреса класса D зарезервированы для программ, ищущих в базе данных свободные групповые адреса. Одновременно может производиться несколько операций многоадресной рассылки, и хост может «настроиться» на интересующую его передачу, прослушивая соответствующие групповые адреса.

Периодически каждый многоадресный маршрутизатор посылает широковещательный IGMP-пакет, сфера распространения которого ограничена пределами его острова. В этом пакете содержится предложение сообщить, кого какой канал интересует. Хосты, заинтересованные в получении (или в продолжении получения) какого-либо одного или нескольких каналов, посылают в ответ свои IGMP-пакеты. Чтобы избежать перегрузки локальной сети, эти ответы посылаются не сразу, а располагаются во времени особым образом. Чтобы не тратить понапрасну пропускную способность локальных сетей, каждый многоадресный маршрутизатор хранит таблицу каналов, которые он должен направлять в свою локальную сеть.

Когда источник аудио- или видеопотока создает новый пакет, он распространяет его с помощью аппаратно реализованной многоадресной рассылки внутри своего острова. Затем этот пакет подбирается локальным многоадресным маршрутизатором, который копирует его по всем туннелям, с которыми он соединен.

Получив такой пакет по туннелю, каждый многоадресный маршрутизатор проверяет, прибыл ли этот пакет по оптимальному маршруту, то есть по маршруту, который, как указано в таблице маршрутизатора, следует использовать для данного источника (как если бы это был пункт назначения). Если пакет прибыл по наилучшему маршруту, он копируется маршрутизатором во все его туннели. В противном случае пакет игнорируется. Если, например, в таблицах маршрутизатора *C* (рис. 7.47) говорится, что маршрут к острову *A* должен пролегать через остров *B*, — тогда если многоадресный пакет от острова *A* прибудет на остров *C* через остров *B*, то он будет скопирован на острова *D* и *E*. Если же такой пакет от острова *A* прибудет на остров *C* через остров *E*, то он будет проигнорирован. Этот алгоритм представляет собой просто алгоритм пересылки в обратном направлении, рассматривавшийся в главе 5. Хотя он и не совершенен, он достаточно хорош и прост в реализации.

Помимо использования алгоритма пересылки в обратном направлении, позволяющего избежать перегрузки Интернета, для ограничения сферы распространения многоадресных пакетов также используется IP-поле *Time to live* (время жизни). Каждый пакет начинает свой путь с определенным значением этого поля (определяемым источником). Каждому туннелю присваивается весовой коэффициент. Пакет пропускается сквозь туннель, только если он обладает достаточным весом. В противном случае пакет отвергается. Например, туннелям, пересекающим океаны, обычно назначаются весовой коэффициент, равный 128, поэтому область распространения всех пакетов, время жизни которых не превы-

шает 127, ограничивается одним континентом. После прохождения пакета по туннелю значение его поля *Time to live* уменьшается на вес туннеля.

Много внимания было уделено улучшению алгоритма маршрутизации. Одно из предложений основывалось на идее дистанционно-векторной маршрутизации. В этом случае MBone-сайты группируются в области, а алгоритм становится иерархическим (Thyagarajan и Deering, 1995).

Другое предложение состоит в использовании вместо дистанционно-векторной маршрутизации модифицированной формы маршрутизации с учетом состояния линий. В частности рабочая группа Интернета IETF занимается изменением алгоритма маршрутизации OSPF (Open Shortest Path First — первоочередное открытие кратчайших маршрутов), чтобы сделать его более подходящим для многоадресной рассылки в пределах автономной системы. Новый алгоритм получил название **MOSPF** (многоадресный алгоритм OSPF) (Moy, 1994). В дополнение к обычной информации, используемой для выбора маршрута, в этом алгоритме также строится полная карта всех островов многоадресной рассылки и всех туннелей. Зная полную топологию системы, несложно рассчитать лучший маршрут между любыми двумя островами по имеющимся туннелям. Например, можно воспользоваться алгоритмом Дейкстры (Dijkstra).

Вторая область исследований охватывает маршрутизацию внутри автономной системы. Этот алгоритм называется **PIM** (Protocol Independent Multicast — не зависящая от протокола многоадресная рассылка). Он разрабатывается другой рабочей группой IETF. Создано две версии алгоритма PIM, применяющихся в зависимости от плотности островов (почти все желают смотреть видео или, наоборот, почти никто ничего не хочет смотреть). Вместо того чтобы создавать оверлейную топологию, как это делается в алгоритмах DVMPR и MOSPF, в обеих версиях алгоритма PIM используются стандартные таблицы одноадресной маршрутизации.

В плотном варианте алгоритма PIM (PIM-DM) идея состоит в отсечении бесполезных путей. Когда многоадресный пакет прибывает по «неправильному» туннелю, обратно посылается специальный отсекающий пакет, предлагающий отправителю прекратить отправлять по этому туннелю пакеты данному адресату. Если же пакет прибывает по «правильному» туннелю, он копируется во все остальные еще не отсеченные туннели. Если все остальные туннели маршрутизатора отсечены, а в его области этот канал никто смотреть не желает, маршрутизатор сам посылает отсекающий пакет по «правильному» туннелю. Таким образом, многоадресная рассылка автоматически адаптируется к спросу на видеоданные.

Вариант алгоритма PIM для редко расположенных островов (PIM-SM), описанный в RFC 2363, действует по-другому. Идея этого варианта заключается в том, чтобы не забивать Интернет излишней многоадресной рассылкой из-за, скажем, трех человек из университета Беркли, желающих устроить небольшую видеоконференцию с помощью IP-адресов класса D. В этом случае алгоритм PIM создает так называемые точки встречи. Каждый источник посылает в эти точки свои пакеты. Любой сайт, желающий присоединиться к видеоконференции, просит установить туннель с точкой встречи. Таким образом, в этом варианте алгоритма PIM трафик переносится при помощи обычной одноадресной рассылки. Популярность варианта PIM-SM возрастает, и система MBone все чаще прибегает к его использованию. Соответственно, алгоритм MOSPF встречается все реже и реже. С другой стороны, сама система MBone переживает период стагнации и, возможно, никогда не достигнет процветания.

Несмотря на этот грустный вывод относительно MBone, мультимедиа в целом представляет собой захватывающую и быстро меняющуюся область. Ежедневно появляется информация о создании новых технологий и приложений. Многоадресная рассылка и качество обслуживания становятся все более популярны, их обсуждению посвящена книга (Striegel и Manimaran, 2002). Еще одной интересной темой является многоадресная рассылка по беспроводным сетям (Gossain и др., 2002). Вообще же область знаний, касающаяся многоадресной рассылки и всего, что с этим связано, вероятно, будет волновать умы человечества еще долгие годы.

Резюме

Именование доменов в Интернете реализуется при помощи иерархической схемы, называемой службой имен доменов (DNS). В системе DNS на верхнем уровне находятся популярные родовые домены, включая *com*, *edu* и около двухсот национальных доменов. DNS реализована в виде распределенной базы данных, серверы которой расположены по всему миру. В ней хранятся записи с IP-адресами, адресами почтовых обменников и прочей информацией. Обратившись к DNS-серверу, процесс может преобразовать имя домена Интернета в IP-адрес, требующийся для общения с доменом.

Электронная почта — это одно из самых популярных приложений Интернета. Ею пользуются все, начиная от детей младшего школьного возраста и заканчивая стариками преклонных годов. Большинство систем электронной почты соответствуют стандартам, описанным в RFC 2821 и 2822. Сообщения, пересылаемые по e-mail, содержат ASCII-заголовки, определяющие свойства самого сообщения. Можно пересылать данные разных типов, указывая эти типы в MIME-заголовках. Передача писем осуществляется по протоколу SMTP, устанавливающему TCP-соединение между хостом-источником и хостом-приемником. Почта передается напрямую по этому TCP-соединению.

Еще одним безумно популярным приложением Интернета является Всемирная паутина (WWW). Она представляет собой систему связанных между собой гипертекстовых документов. Изначально каждый документ был страницей, написанной на HTML и содержащей гиперссылки на другие страницы. Сегодня при написании страниц все чаще используется обобщенный язык XML. Кроме того, немалая часть содержимого документов генерируется динамически при помощи скриптов, работающих как на стороне сервера (PHP, JSP и ASP), так и на стороне клиента (JavaScript). Браузер выводит документы на экран, устанавливая TCP-соединение с сервером, запрашивая у него страницу и разрывая после этого соединение. В сообщениях с такими запросами содержится множество заголовков, позволяющих сообщить дополнительную информацию. Для повыше-

ния производительности Всемирной паутины применяются кэширование, репликация и сети доставки содержимого.

На горизонте Интернета начинают появляться беспроводные веб-системы. Первыми такими системами являются WAP и i-mode. Для них обеих характерны наличие маленького экрана мобильного телефона и низкая пропускная способность, однако следующее поколение этих систем, наверное, будет более мощным.

Мультимедиа — это еще одна звезда, восходящая на сетевом небосклоне. Эта область включает приложения, занимающиеся оцифровкой звука и видеоизображений и их передачей по сетям. Для передачи звука требуется относительно низкая пропускная способность, благодаря этому данный вид мультимедиа более распространен в сетях. Потоковое аудио, интернет-радио, передача речи поверх IP — все эти приложения сегодня реально работают. Кроме того, постоянно появляются новые приложения. Видео по заказу — это перспективная область, к которой сейчас проявляется большой интерес. Наконец, MBone представляет собой экспериментальную систему всемирного цифрового телевещания в Интернете.

Вопросы

1. Многие коммерческие компьютеры имеют три разных и в то же время абсолютно уникальных идентификатора. Как они выглядят?
2. Основываясь на информации, приведенной в листинге 7.1, определите, к какому классу сети принадлежит хост *little-sister.cs.vu.nl*, A, B или C?
3. В листинге 7.1 после слова *rowboat* не поставлена точка. Почему?
4. Попробуйте угадать, что означает смайлик `:-X` (иногда изображаемый как `:-#`).
5. DNS использует UDP вместо TCP. Если DNS-пакет теряется, он автоматически не восстанавливается. Приводит ли это к возникновению проблем, и если да, то как они решаются?
6. Кроме того, что UDP теряет пакеты, на них еще и накладывается ограничение по длине, причем оно может быть довольно строгим: 576 байт. Что произойдет, если длина искомого имени DNS превысит это число? Можно ли будет послать его в двух пакетах?
7. Может ли компьютер иметь одно имя DNS и несколько IP-адресов? Как такое может быть?
8. Может ли компьютер иметь два имени DNS в разных доменах верхнего уровня? Если да, приведите правдоподобный пример. Если нет, объясните, почему это невозможно.
9. Число компаний, имеющих собственный веб-сайт, в последнее время сильно возросло. В результате в домене *com* существуют сайты тысяч фирм, что приводит к сильной нагрузке на сервер, обслуживающий этот домен верхнего уровня. Предложите способ решения этой проблемы без изменения схемы именования (то есть без изобретения нового домена верхнего уровня). Возможно, ваше решение потребует внесения изменений в клиентские программы.

10. Некоторые системы электронной почты поддерживают поле *Content-Return:*. В нем указывается, нужно ли возвращать содержимое письма в том случае, если оно не будет доставлено получателю. Это поле входит в состав конверта или заголовка письма?
11. Системы электронной почты хранят адресные книги e-mail, с помощью которых пользователь может найти нужный адрес. Для поиска по таким книгам имена адресатов должны быть разбиты на стандартные компоненты (например, имя, фамилия). Обсудите некоторые проблемы, которые следует решить, чтобы можно было разработать соответствующий международный стандарт.
12. Адрес e-mail состоит из имени пользователя, знака @ и имени домена DNS с записью MX. В качестве имени пользователя может указываться реальное имя человека, фамилия, инициалы или любые другие идентификаторы. Допустим, причиной потери многих писем, приходящих в адрес большой компании, является то, что авторы писем не знают точные имена пользователей. Существует ли возможность решения этой проблемы без изменения DNS? Если да, предложите свой вариант и объясните принцип его работы. Если нет, объясните, почему.
13. Имеется двоичный файл длиной 3072 байта. Каков будет его размер после кодирования с помощью системы base64? Пара символов CR+LF вставляется через каждые 80 байт, а также в конце сообщения.
14. Рассмотрите схему кодирования MIME quoted-printable. Укажите проблемы, которые мы не затронули в тексте, и предложите способ их решения.
15. Назовите пять типов MIME, не указанных в тексте. Информацию можно взять из настроек браузера или из Интернета.
16. Предположим, вы хотите переслать другу MP3-файл, однако провайдер, услугами которого пользуется ваш друг, ограничивает максимальный размер входящей почты до 1 Мбайт, а файл занимает 4 Мбайт. Можно ли решить поставленную задачу, используя RFC 822 и MIME?
17. Предположим, некто устанавливает каникулярного демона, после чего посылает сообщение и сразу же выходит из системы. К сожалению, получатель сообщения уже с неделю находится в отпуске и на его машине также установлен каникулярный демон. Что произойдет? Будут ли каникулярные демоны переписываться без конца, пока кто-нибудь не вернется из отпуска и не прервет их диалог?
18. В любом стандарте, таком как RFC 822, должно быть описание точной грамматики — это требуется для межсетевого взаимодействия. Даже самые простые элементы должны быть четко определены. Например, в заголовках SMTP допустимы пробелы между символами. Приведите *два* правдоподобных альтернативных определения этих пробелов.
19. Каникулярный демон является частью пользовательского агента или агента передачи сообщений? Понятно, что он настраивается с помощью пользовательского агента, но какая часть системы занимается реальной отправкой автоматических ответов? Поясните свой ответ.

20. Протокол POP3 позволяет пользователям запрашивать и загружать почту из удаленного почтового ящика. Означает ли это, что внутренний формат почтовых ящиков должен быть стандартизован, чтобы любые клиентские программы, использующие POP3, могли обратиться к почтовому ящику на любом сервере? Аргументируйте свой ответ.
21. С точки зрения провайдера POP3 и IMAP отличаются друг от друга довольно сильно. Пользователи POP3 обычно опустошают почтовые ящики ежедневно. Пользователи IMAP хранят свою почту на сервере неопределенно долго. Представьте, что провайдер хочет посоветоваться с вами, решая, какие протоколы ему поддерживать. Какие соображения вы выскажете в ответ?
22. Какие протоколы использует Webmail: POP3, IMAP или ни тот, ни другой? Если какой-то из этих двух, то почему выбран именно он? Если ни тот, ни другой, то к какому из них ближе по духу реально используемый протокол?
23. При пересылке веб-страницы предваряются заголовками MIME. Зачем?
24. Когда бывают нужны внешние программы просмотра? Как браузер узнает, какую из этих программ использовать?
25. Возможна ли ситуация, при которой щелчок пользователя на одной и той же ссылке с одним и тем же MIME-типом в Internet Explorer и в Netscape приводит к запуску совершенно разных вспомогательных приложений? Ответ поясните.
26. Многопотоковый веб-сервер организован так, как показано на рис. 7.9. На прием запроса и поиск в кэше уходит 500 мкс. В половине случаев файл обнаруживается в кэше и немедленно возвращается. В другой половине случаев модуль блокируется на 9 мс, в течение которых ставится в очередь и обрабатывается дисковый запрос. Сколько модулей должен поддерживать сервер, чтобы процессор постоянно находился в работе (предполагается, что диск не является узким местом системы)?
27. Стандартный URL со схемой *http* подразумевает, что веб-сервер прослушивает порт 80. Тем не менее, веб-сервер может прослушивать и другой порт. Предложите синтаксис URL, который позволил бы обращаться к серверу, прослушивающему нестандартный порт.
28. Хотя об этом и не было сказано в тексте, существует альтернативный вариант записи URL, использующий вместо имени DNS IP-адрес. Пример такого URL может выглядеть так: http://192.31.231.66/index.html. Как браузер узнает, что следует вслед за схемой: имя DNS или IP-адрес?
29. Представьте, что сотрудник факультета компьютерных наук Стэнфордского университета написал новую программу, которую он хочет распространять по FTP. Он помещает программу newprog.c в каталог ftp/pub/freebies. Как будет выглядеть URL этой программы?
30. В соответствии с табл. 7.9 www.aportal.com хранит список предпочтений клиента в виде cookie. Недостаток такого решения: размер cookie ограничен 4 Кбайт, и если нужно сохранить много данных о пользователе (например, о том, что он хочет видеть на странице множество биржевых сводок, новостей спортивных команд, типов новых историй, погоду сразу во многих городах, специальные предложения по разным категориям товаров и т. д.), то вскоре может быть достигнут 4-килобайтный порог этих описаний. Предложите альтернативный способ хранения данных о пользователе, в котором эта проблема не возникала бы.

31. Некий «Банк для лентяев» хочет организовать специальную онлайновую банковскую систему для своих ленивых клиентов. После регистрации в системе и идентификации с помощью пароля пользователь получает cookie-файл, содержащий идентификационный номер клиента. Таким образом, ему не приходится всякий раз при входе в систему повторять ввод своих идентификационных данных. Как вам такая идея? Будет ли она работать? Насколько вообще хороша такая идея?
32. В листинге на рис 7.12 в теге `<IMG>` устанавливается значение параметра *ALT*. При каких условиях браузер использует его и как?
33. Как в языке HTML с изображением ассоциируется гиперссылка? Покажите на примере.
34. Напишите тег `<A>`, необходимый для того, чтобы связать строку «ACM» с гиперссылкой на адрес http://www/acm.org.
35. Создайте форму на языке HTML для новой компании Interburger, принимающей заказы на гамбургеры по Интернету. Бланк заказа должен содержать имя заказчика, его адрес, размер гамбургера (гигантский или огромный) и наличие сыра. Оплата гамбургеров производится наличными после доставки, поэтому информация о кредитной карте не требуется.
36. Создайте форму, предлагающую пользователю ввести два числа. После нажатия кнопки «Подтверждение» сервер должен вернуть сумму введенных чисел. Напишите PHP-скрипт для серверной части.
37. Для каждого из перечисленных случаев укажите: 1) возможно ли и 2) лучше ли использовать PHP-скрипт или JavaScript и почему:

    1) календарь на любой месяц, начиная с сентября 1752 года;

    2) расписание рейсов из Амстердама в Нью-Йорк;

    3) вывод полинома с коэффициентами, введенными пользователем.
38. Напишите программу на JavaScript, принимающую на входе целочисленные значения, превышающие 2, и сообщающую, является ли введенное число простым. В JavaScript синтаксис выражений `if` и `while` совпадает с аналогичными выражениями в C и Java. Оператор выполнения арифметических действий по произвольному модулю: `%`. Если понадобится найти квадратный корень числа x, воспользуйтесь функцией `Math.sqrt(x)`.
39. HTML-страница состоит из следующего кода:

```
<html><body>
<a href="www.info-source.com/welcome.html">Информация</a>
</body></html>
```

    Когда пользователь щелкает на гиперссылке, открывается TCP-соединение и на сервер отправляется некоторая последовательность строк. Перечислите эти строки.

40. Заголовок *If-Modified-Since* может использоваться для проверки актуальности страницы, хранящейся в кэше. Соответствующие запросы могут посылаться на страницы, содержащие изображения, звуки, видео и т. д., а также на обычные страницы на HTML. Как вы думаете, эффективность этого метода будет выше для изображений JPEG или для страниц HTML? Хорошенько подумайте над значением слова «эффективность» и после этого объясните свой ответ.
41. В день очень важного спортивного события (скажем, финала международного чемпионата по популярному виду спорта) огромное количество посетителей стремятся попасть на официальный веб-сайт мероприятия. Схожа ли эта ситуация внезапного роста трафика с выборами во Флориде в 2000 году и почему?
42. Имеет ли смысл отдельному провайдеру функционировать в качестве сети доставки содержимого? Если да, то как должна работать система? Если нет, то чем плоха такая идея?
43. При каких обстоятельствах мысль об использовании сети доставки содержимого является неудачной?
44. Беспроводные веб-терминалы обладают низкой пропускной способностью, что приводит к необходимости эффективного кодирования. Предложите схему эффективной передачи текста на английском языке по беспроводным линиям связи на WAP-устройство. Можно предположить, что терминал обладает постоянной памятью объемом несколько мегабайт и процессором средней мощности. *Подсказка:* вспомните, как передается текст на японском языке, где каждый символ представляет собой слово.
45. Компакт-диск вмещает 650 Мбайт данных. Используется ли сжатие для аудиокомпакт-дисков? Аргументируйте свой ответ.
46. На рис. 7.26, *в* показано, что шум квантования возникает из-за использования 4-битных отсчетов для представления 9 уровней сигнала. Первый отсчет (в нуле) является точным, а остальные — нет. Чему равна относительная погрешность для отсчетов, взятых в моменты времени, равные 1/32, 2/32 и 3/32 периода?
47. Можно ли использовать психоакустическую модель для уменьшения требуемой пропускной способности для систем интернет-телефонии? Если да, то каковы условия, при которых эта модель будет работать (если они вообще есть)? Если нет, то почему?
48. Сервер аудиопотока расположен на удалении от проигрывателя, дающем задержку 50 мс в одном направлении. Он выдает данные со скоростью 1 Мбит/с. Если проигрыватель содержит буфер объемом 1 Мбайт, то что можно сказать о расположении нижнего и верхнего пределов заполнения этого буфера?
49. Алгоритм чередования, показанный на рис. 7.29, хорош тем, что потеря пакета не приводит к возникновению паузы в звучании. Тем не менее, при использовании этого алгоритма в интернет-телефонии выявляется некий недостаток. Какой?

50. Возникают ли при передаче речи поверх IP те же проблемы с брандмауэрами, что и при передаче потокового аудио? Ответ обсудите.
51. Какая скорость требуется для передачи несжатого цветного изображения с размером 800×600 с 8 битами на пиксел при 40 кадрах в секунду?
52. Может ли ошибка в одном бите в кадре MPEG повредить более одного кадра? Аргументируйте свой ответ.
53. Рассмотрим пример видеосервера, обслуживающего 100 000 клиентов. Каждый клиент смотрит два фильма в месяц. Предположим, что половину всех фильмов начинают просматривать в 20:00. Сколько фильмов одновременно должен передавать сервер в этот период? Если для передачи каждого фильма требуется 4 Мбит/с, сколько соединений типа OC-12 нужно для соединения сервера с сетью?
54. Предположим, что закон Ципфа соблюдается для доступа к видеосерверу, на котором хранится 10 000 фильмов. Допустим, сервер хранит 1000 самых популярных фильмов на магнитных дисках, а остальные 9000 фильмов — на оптических дисках. Какая часть запросов будет адресована магнитным дискам? Напишите небольшую программу, вычисляющую данное значение численно.
55. Некие нехорошие люди зарегистрировали имена доменов, которые незначительно отличаются от всемирно известных, таких как www.microsoft.com и которые пользователь может посетить, просто случайно опечатавшись при наборе адреса. Приведите пример по крайней мере пяти таких доменов.
56. Существует множество сайтов, зарегистрированных под именами www.слово.com, где *слово* — это обычное слово английского языка. Для каждой из приведенных категорий составьте список из пяти веб-сайтов и вкратце опишите их суть (например, www.cosmos.com — это сайт, посвященный проблемам космоса). Вот список категорий: животные, продукты питания, предметы быта, части тела. Что касается последней категории, просьба указывать объекты, расположенные выше талии.
57. Разработайте несколько собственных значков emoji, используя битовую карту размером 12×12. Попытайтесь изобразить «ее парня», «его подружку», профессора и политика.
58. Напишите POP3-сервер, работающий со следующими командами: USER, PASS, LIST, RETR, DELE и QUIT.
59. Перепишите листинг 6.1, превратив его в реальный веб-сервер, использующий команду GET для работы с протоколом HTTP 1.1. Он должен реагировать на сообщение *Host*. Сервер должен кэшировать файлы, которые были недавно запрошены с диска, и обслуживать запросы, по возможности выдавая файлы из кэша.

Глава 8
Безопасность в сетях

- Криптография
- Алгоритмы с симметричным криптографическим ключом
- Алгоритмы с открытым ключом
- Цифровые подписи
- Управление открытыми ключами
- Защита соединений
- Протоколы аутентификации
- Конфиденциальность электронной переписки
- Защита информации во Всемирной паутине
- Социальный аспект
- Резюме
- Вопросы

В первые десятилетия своего существования компьютерные сети использовались, в первую очередь, университетскими исследователями для обмена электронной почтой и сотрудниками корпораций для совместного использования принтеров. В таких условиях вопросы безопасности не привлекали большого внимания. Однако теперь, когда миллионы обычных граждан пользуются сетями для управления своими банковскими счетами, заполнения налоговых деклараций, приобретают товары в интернет-магазинах, проблема сетевой безопасности становится все более актуальной. В этой главе мы рассмотрим вопросы безопасности сетей с различных точек зрения, укажем на подводные камни и обсудим различные алгоритмы и протоколы, позволяющие повысить безопасность сетей.

Тема безопасности включает в себя множество вопросов, связанных с различными человеческими грехами. В простейшем виде службы безопасности гарантируют,

что любопытные личности не смогут читать, или, что еще хуже, изменять сообщения, предназначающиеся другим получателям. Службы безопасности пресекают попытки получения доступа к удаленным службам теми пользователями, которые не имеют на это прав. Кроме того, система безопасности позволяет определить, написано ли сообщение «Оплатите счета до пятницы» тем отправителем, чье имя в нем указано, или же это фальсификация. Кроме того, системы безопасности решают проблемы, связанные с перехватом и повторным воспроизведением сообщений и с людьми, пытающимися отрицать, что они посылали данные сообщения.

Большинство проблем безопасности возникает из-за злоумышленников, пытающихся извлечь какую-либо пользу для себя или причинить вред другим. Несколько наиболее распространенных типов нарушителей перечислено в табл. 8.1. Из этого списка должно быть ясно, что задача обеспечения безопасности сетей включает в себя значительно больше, нежели просто устранение программных ошибок. Часто стоит задача перехитрить умного, убежденного и иногда хорошо финансированного противника. Также очевидно, что меры, способные остановить случайного нарушителя, мало повлияют на серьезного преступника. Статистика, собираемая полицией, говорит о том, что большинство атак предпринимается не извне (людьми, прослушивающими линии связи), а изнутри — завистливыми или недовольными чем-либо людьми. Следовательно, системы безопасности должны учитывать и этот факт.

Таблица 8.1. Типы нарушителей и цели их действий

Нарушитель	Цель
Студент	Прочитать из любопытства чужие письма
Хакер	Проверить на прочность чужую систему безопасности; украсть данные
Торговый агент	Притвориться представителем всей Европы, а не только Андорры
Бизнесмен	Разведать стратегические маркетинговые планы конкурента
Уволенный сотрудник	Отомстить фирме за увольнение
Бухгалтер	Украсть деньги компании
Биржевой брокер	Не выполнить обещание, данное клиенту по электронной почте
Аферист	Украсть номера кредитных карт для продажи
Шпион	Узнать военные или производственные секреты противника
Террорист	Украсть секреты производства бактериологического оружия

В первом приближении проблемы безопасности сетей могут быть разделены на четыре пересекающиеся области: секретность, аутентификация, обеспечение строгого выполнения обязательств и обеспечение целостности. Секретность (конфиденциальность) означает предотвращение попадания информации в руки неавторизованных пользователей. Именно это обычно приходит в голову при упоминании безопасности сетей. Аутентификация позволяет определить, с кем вы разговариваете, прежде чем предоставить собеседнику доступ к секретной информации или вступить с ним в деловые отношения. Проблема обеспечения строгого

выполнения обязательств имеет дело с подписями. Как доказать, что ваш клиент действительно прислал электронной почтой заказ на десять миллионов винтиков с левосторонней резьбой по 89 центов за штуку, если впоследствии он утверждает, что цена была 69 центов? Наконец, как можно быть уверенным, что принятое вами сообщение не подделано и не модифицировано по пути злоумышленником?

Все эти аспекты (секретность, аутентификация, обеспечение строгого выполнения обязательств и обеспечение целостности) встречаются и в традиционных системах, но с некоторыми существенными отличиями. Секретность и целостность достигаются с помощью заказных писем и хранения документов в несгораемых сейфах. Сегодня ограбить почтовый поезд значительно сложнее, чем во времена Джесси Джеймса.

Кроме того, обычно несложно отличить на глаз оригинальный бумажный документ от фотокопии. В качестве проверки попробуйте сделать фотокопию настоящего банковского чека. В понедельник попытайтесь обналичить настоящий чек в вашем банке. А во вторник попробуйте сделать то же самое с фотокопией. Проследите за разницей в поведении банковского служащего. Увы, но оригинал и копия электронных чеков неотличимы друг от друга. Понадобится некоторое время, прежде чем банки привыкнут к этому.

Люди опознают друг друга по лицам, голосам и почеркам. Доказательства подлинности бумажных документов обеспечиваются подписями на печатных бланках, печатями, рельефными знаками и т. д. Подделка обычно может быть обнаружена специалистами по почерку, бумаге и чернилам. При работе с электронными документами все это недоступно. Очевидно, требуются другие решения.

Прежде чем перейти к обсуждению сути самих решений, имеет смысл потратить несколько минут и попытаться определить, к какому уровню стека протоколов относится система сетевой безопасности. Вероятно, какое-то одно место для нее найти сложно. Каждый уровень должен внести свой вклад. На физическом уровне с подслушиванием можно бороться за счет помещения передающих кабелей в герметичные трубы, наполненные аргоном под высоким давлением. Любая попытка просверлить трубу приведет к утечке части газа из трубы, в результате давление снизится, и это послужит сигналом тревоги. Подобная техника применяется в некоторых военных системах.

На уровне передачи данных пакеты, передаваемые по двухточечной линии, могут кодироваться при передаче в линию и декодироваться при приеме. Все детали этого могут быть известны только уровню передачи данных, причем более высокие уровни могут даже не догадываться о том, что там происходит. Однако такое решение перестает работать в том случае, если пакету нужно преодолеть несколько маршрутизаторов, поскольку при этом пакет придется расшифровывать на каждом маршрутизаторе, что сделает его беззащитным перед атаками внутри маршрутизатора. Кроме того, такой метод не позволит защищать отдельные сеансы, требующие защиты (например, осуществление покупок в интернет-магазинах), и при этом не защищать остальные. Тем не менее, этот метод, называемый **шифрованием в канале связи**, легко может быть добавлен к любой сети и часто бывает полезен.

На сетевом уровне могут быть установлены брандмауэры, позволяющие отвергать подозрительные пакеты, приходящие извне. К этому же уровню относится IP-защита.

На транспортном уровне можно зашифровать соединения целиком, от одного конца до другого. Максимальную защиту может обеспечить только такое сквозное шифрование.

Наконец, проблемы аутентификации и обеспечения строго выполнения обязательств могут решаться только на прикладном уровне.

Итак, очевидно, что безопасность в сетях — это вопрос, охватывающий все уровни протоколов, именно поэтому ему посвящена отдельная глава.

Несмотря на то, что эта глава большая, важная и содержит множество технических описаний, в настоящий момент вопрос безопасности в сетях может показаться несколько неуместным. Ведь хорошо известно, что большинство «дыр» в системах безопасности возникают из-за неумелых действий персонала, невнимательного отношения к процедурам защиты информации или недобросовестности самих сотрудников защищаемого объекта. Доля проблем, возникающих из-за преступников-интеллектуалов, прослушивающих линии связи и расшифровывающих полученные данные, сравнительно мала. Посудите сами: человек может прийти в совершенно произвольное отделение банка с найденной на улице магнитной кредитной карточкой, посетовать на то, что он забыл свой PIN-код, а бумажку, на которой он написан, съел, и ему тотчас «напомнят» секретный шифр (чего только не сделаешь ради добрых отношений с клиентами). Ни одна криптографическая система в мире здесь не поможет. В этом смысле книга Росса Андерсона (Anderson, 2001) действительно ошеломляет, так как в ней приводятся сотни примеров того, как системы безопасности в самых различных производственных областях не справлялись со своей задачей. И причинами этих неудач были, мягко говоря, неряшливость в ведении дел или простое пренебрежение элементарными правилами безопасности. Тем не менее, мы оптимистично надеемся на то, что электронная коммерция со временем станет более популярной, компании станут более тщательно производить операции, связанные с защитой информации, и техническая сторона вопроса все же будет превалировать над фактором человеческой невнимательности.

На всех уровнях, за исключением физического, защита информации базируется на принципах криптографии. Поэтому мы начнем изучение систем безопасности с детального рассмотрения основ криптографии. В разделе «Криптография» мы изучим базовые принципы. Далее до раздела «Управление открытыми ключами» будут рассмотрены некоторые классические алгоритмы и структуры данных, применяемые в криптографии. После этого мы свяжем теорию с практикой и посмотрим, как все эти концепции применяются в компьютерных сетях. В конце этой главы будут приведены некоторые мысли, касающиеся технологии и социальных вопросов.

Прежде чем приступить к изложению, позвольте назвать вопросы, о которых *не* пойдет речь в этой главе. Подбирая материал, мы старались сконцентрировать внимание на вопросах, связанных именно с компьютерными сетями, а не с опе-

рационными системами или приложениями (хотя провести четкую грань зачастую бывает довольно трудно). Например, ничего не говорится об авторизации пользователя с использованием биометрии, защите паролей, атак, связанных с переполнением буферов, вирусах типа «троянский конь», получении доступа путем обмана, логических бомбах, вирусах, червях и т. п. Обо всем этом очень много говорится в главе 9 книги *«Современные операционные системы»* (Таненбаум, 2001). Все желающие узнать о вопросах системной защиты могут обратиться к этой книге.

Итак, в путь.

Криптография

Слово **криптография** происходит от греческих слов, означающих «скрытное письмо». У криптографии долгая и красочная история, насчитывающая несколько тысяч лет. В данном разделе мы всего лишь кратко упомянем некоторые отдельные моменты в качестве введения к последующей информации. Желающим ознакомиться с полной историей криптографии рекомендуется (Kahn, 1995). Для получения всестороннего представления о текущем положении дел см. (Kaufman и др., 2002). С математическими аспектами криптографии можно познакомиться, прочитав книгу (Stinson, 2002). Менее формальным (с математической точки зрения) языком ведется изложение в (Burnett и Paine, 2001).

С профессиональной точки зрения понятия «шифр» и «код» отличаются друг от друга. **Шифр** представляет собой посимвольное или побитовое преобразование, не зависящее от лингвистической структуры сообщения. **Код**, напротив, заменяет целое слово другим словом или символом. Коды в настоящее время не используются, хотя история у них, конечно, славная. Наилучшим считается код, использовавшийся американскими войсками в Тихом океане во время второй Мировой войны. Просто-напросто для ведения секретных переговоров использовались носители языка индейцев навахо, словами из которого обозначались военные термины. Например, слово *чаи-дагахи-найл-цайди* (буквально — убийца черепах) означало противотанковое оружие. Язык навахо тоновый (для различения смысла используется повышение или понижение тона), весьма сложный, не имеет письменной формы. Но самое большое его достоинство заключалось в том, что ни один японец не имел о нем ни малейшего представления.

В сентябре 1945 года *Юния Сан-Диего* так описывала этот код: «В течение трех лет, где бы ни высаживались военные моряки, уши японцев различали лишь странный булькающий шум, перемежающийся с другими звуками. Все это напоминало клич тибетского монаха или звук опустошаемой бутылки с горячей водой». Японцы так и не смогли взломать этот код, и многие носители языка индейцев навахо были удостоены высоких воинских наград за отличную службу и смелость. Тот факт, что США смогли расшифровать японский код, а японцы так и не узнали язык навахо, сыграл важную роль в американской победе в Тихом океане.

Основы криптографии

Исторически использовали и развивали искусство криптографии представители четырех профессий: военные, дипломатический корпус, люди, ведущие дневники, и любовники. Из них наиболее важную роль в развитии этой области сыграли военные. В военных организациях секретные сообщения традиционно отдавались для зашифровки и передачи плохо оплачиваемым шифровальщикам. Сам объем сообщений не позволял выполнить эту работу небольшим количеством элитных специалистов.

До появления компьютеров одним из основных сдерживающих факторов в криптографии была неспособность шифровальщика выполнить необходимые преобразования — зачастую на поле боя, с помощью несложного оборудования. Кроме того, довольно сложной задачей было быстрое переключение с одного криптографического метода на другой, так как для этого требовалось переобучение большого количества людей. Тем не менее опасность того, что шифровальщик может быть захвачен противником, заставила развить способность к постоянной смене криптографических методов. Модель процесса шифрования—дешифрации показана на рис. 8.1.

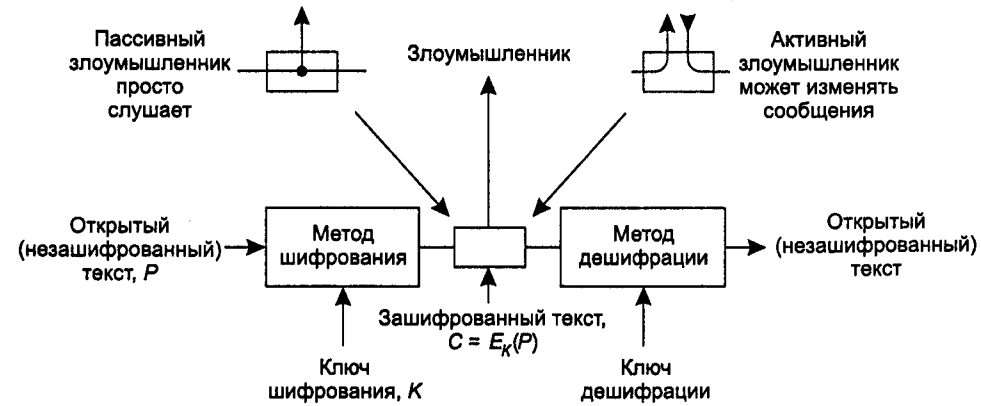


Рис. 8.1. Модель процесса шифрования—дешифрации

Сообщения, подлежащие зашифровке, называемые **открытым текстом**, преобразуются с помощью функции, вторым входным параметром которой является **ключ**. Результат процесса шифрования, называемый **зашифрованным текстом**, передается обычно по радио или через связного. Предполагается, что противник или **злоумышленник** слышит и аккуратно копирует весь зашифрованный текст. Однако в отличие от получателя, которому предназначается данное сообщение, злоумышленник не знает ключа дешифрации, и поэтому расшифровка сообщения представляет для него большие трудности, а порой она просто невозможна. Иногда злоумышленник может не только прослушивать канал связи (пассивный злоумышленник), но способен также записывать сообщения и воспроизводить их позднее, вставлять свои сообщения или модифицировать оригинальные со-

общения, прежде чем они достигнут получателя (активный злоумышленник). Искусство взлома шифров называется **криптоанализом**. Искусства изобретать шифры (криптография) и взламывать их (криптоанализ) называются вместе **криптологией**.

Обычно для обозначения открытого текста, зашифрованного текста и ключей полезно использовать специальную нотацию. Мы будем использовать формулу $C = E_K(P)$, обозначающую, что при зашифровке открытого текста P с помощью ключа K получается зашифрованный текст C. Аналогично формула $P = D_K(C)$ означает расшифровку зашифрованного текста C для восстановления открытого текста. Из этих двух формул следует, что

$$D_K(E_K(P)) = P.$$

Такая нотация предполагает, что E и D являются просто математическими функциями. Они в действительности таковыми и являются. Единственная хитрость состоит в том, что обе эти функции имеют по два параметра, один из которых (ключ) мы написали не в виде аргумента, а в виде нижнего индекса, чтобы отличать его от сообщения.

Основное правило криптографии состоит в предположении, что криптоаналитику (взломщику кода) известен используемый метод шифрования. Другими словами, злоумышленник точно знает, как работает метод шифрования E на рис. 8.1. Из-за огромных усилий, необходимых для разработки, тестирования и установки нового метода, каждый раз, когда старый метод оказывался или считался скомпрометированным, хранить секрет шифрования просто непрактично. А предположение, что метод остается секретным, когда это уже не так, могло бы причинить еще больший вред.

Здесь на помощь приходит ключ шифрования. Ключ состоит из относительно короткой строки, определяющей один из огромного количества вариантов результата шифрования. В отличие от самого метода шифрования, который может изменяться только раз в несколько лет, ключ можно менять так часто, как это нужно. Таким образом, наша базовая модель представляет собой постоянный и известный общий метод, в котором в качестве параметра используется секретный и легко изменяемый ключ. Идея, заключающаяся в предположении о том, что криптоаналитику известен метод, и краеугольным камнем секретности является эксклюзивный ключ, называется **принципом Керкгофа**. Его в 1883 году впервые высказал фламандский военный шифровальщик Аугуст Керкгоф (Auguste Kerckhoff, 1883). Таким образом, принцип Керкгофа гласит:

Алгоритмы шифрования общедоступны; секретны только ключи.

Секретности алгоритма не стоит придавать большого значения. Попытка сохранить алгоритм в тайне, называемая в торговле **безопасностью за счет неясности** (security by obscurity), обречена на провал. К тому же, опубликовав свой алгоритм, разработчик получает бесплатную консультацию от большого количества ученых-криптоаналитиков, горящих желанием взломать новую систему и тем самым продемонстрировать свои ум и ученость. Если никто не смог взломать алгоритм за пять лет с момента его опубликования, то, по-видимому, этот алгоритм достаточно прочен.

Поскольку реально в тайне хранится только ключ, основной вопрос заключается в его длине. Рассмотрим простой кодовый замок. Его основной принцип состоит в том, что вы последовательно вводите десятичные цифры. Все это знают, но ключ хранится в секрете. Ключ длиной в две цифры образует 100 вариантов. Ключ длиной в три цифры означает 1000 вариантов, а при длине ключа в шесть цифр число комбинаций достигает миллиона. Чем длиннее ключ, тем выше **показатель трудозатрат** взломщика кода. При увеличении длины ключа показатель трудозатрат для взлома системы путем простого перебора значений ключа растет экспоненциально. Секретность передаваемого сообщения обеспечивается мощным (но все же открытым) алгоритмом и длинным ключом. Чтобы не дать прочитать свою электронную почту младшему брату, достаточно 64-разрядного ключа. В коммерческих системах имеет смысл использовать 128-битный код. Чтобы защитить ваши тексты от правительств развитых государств, потребуются ключи длиной по меньшей мере в 256 бит.

С точки зрения криптоаналитика задача криптоанализа имеет три принципиальных варианта. Во-первых, у криптоаналитика может быть некоторое количество зашифрованного текста без соответствующего открытого текста. Задачки, в которых в качестве исходных данных имеется в наличии **только зашифрованный текст**, часто печатаются в различных газетах в разделе ребусов. Во-вторых, у криптоаналитика может оказаться некоторое количество зашифрованного текста и соответствующего ему открытого текста. В этом случае мы имеем дело с **проблемой известного открытого текста**. Наконец, когда у криптоаналитика есть возможность зашифровать любой кусок открытого текста по своему выбору, мы получаем третий вариант проблемы дешифрации, то есть **проблему произвольного открытого текста**. Если бы криптоаналитикам было позволено задавать вопросы типа: «Как будет выглядеть зашифрованное ABCDEFGHJKL?», задачки из газет решались бы очень легко.

Новички в деле криптографии часто полагают, что шифр является достаточно надежным, если он может выдержать атаку первого типа (только зашифрованный текст). Такое предположение весьма наивно. Во многих случаях криптоаналитик может угадать часть зашифрованного текста. Например, первое, что говорят многие компьютеры при входе в систему, это LOGIN:. После того как криптоаналитик получит несколько соответствующих друг другу пар кусков зашифрованного и открытого текста, его работа становится значительно легче. Для обеспечения секретности нужна предусмотрительность криптографа, который должен гарантировать, что система не будет взломана, даже если его оппонент сможет закодировать несколько участков открытого текста по выбору.

Исторически методы шифрования разделились на две категории: метод подстановки и метод перестановки. Мы кратко рассмотрим их в качестве введения в современную криптографию.

Метод подстановки

В шифрах, основанных на **методе подстановки**, каждый символ или группа символов заменяется другим символом или группой символов. Одним из древней-

ших шифров является приписываемый Юлию Цезарю (Julius Caesar) **шифр Цезаря**. Этот шифр заменяет все буквы алфавита на другие с помощью циклического сдвига на три позиции. Так, буква *a* становится буквой *D*, *b* становится *E*, *c* превращается в *F*, ..., а *z* — в *C*. Например, слово *attack* превращается в *DWWDFN*. В наших примерах открытый текст будет обозначаться строчными символами, а зашифрованный текст — прописными.

Некоторое обобщение шифра Цезаря представляет собой сдвиг алфавита не на три символа, а на произвольное число *k* символов. В этом случае *k* становится ключом к общему методу циклически сдвигаемых алфавитов. Шифр Цезаря, возможно, и сумел обмануть жителей Помпеи, но с тех пор ему более уже никого не удалось ввести в заблуждение.

Следующее усовершенствование состоит в установлении соответствия каждому встречающемуся в открытом тексте символу другого символа. Например,

открытый текст: a b c d e f g h i j k l m n o p q r s t u w x y z
зашифрованный текст: Q W E R T Y U I O P A S D F G H J K L Z X C V B N M

Такая система называется **моноалфавитной подстановкой**, ключом к которой является 26-символьная строка, соответствующая полному алфавиту. В нашем примере слово *attack* будет выглядеть, как *QZZQEA*.

На первый взгляд такая система может показаться надежной, так как даже если криптоаналитику известна общая система, он не знает, какой из $26! \approx 4 \cdot 10^{26}$ вариантов ключа применить. В отличие от шифра Цезаря, применение метода простого перебора в данном случае весьма сомнительно. Даже при затратах 1 нс на проверку одного варианта ключа, чтобы перепробовать все ключи, компьютеру понадобится около 10^{10} лет.

Тем не менее, подобный шифр легко взламывается даже при наличии довольно небольших кусков зашифрованного текста. Для атаки шифра может быть использовано преимущество статистических характеристик естественных языков. Например, в английском языке, буква *e* встречается в тексте чаще всего. Следом за ней по частоте использования идут буквы *t*, *o*, *a*, *n*, *i* и т. д. Наиболее часто встречающимися комбинациями из двух символов (**биграммами**) являются *th*, *in*, *er*, *re* и *an*. Наиболее часто встречающимися комбинациями из трех символов, или **триграммами**, являются *the*, *ing*, *and* и *ion*.

Криптоаналитик, пытающийся взломать моноалфавитный шифр, начнет с того, что сосчитает относительные частоты всех символов алфавита. Затем он может попытаться заменить наиболее часто встречающийся символ буквой *e*, а следующий по частоте — буквой *t*. Затем он посмотрит на триграммы и попытается найти что-либо похожее на *tXe*, после чего он сможет предположить, что *X* — это *h*. Аналогично, если последовательность *thYt* встречается достаточно часто, то, вероятно, *Y* обозначает символ *a*. Обладая этой информацией, криптоаналитик может поискать часто встречающуюся триграмму вида *aZW*, что, скорее всего, означает *and*. Продолжая в том же духе, угадывая буквы, биграммы, триграммы и зная, какие последовательности символов являются наиболее вероятными, криптоаналитик побуквенно восстанавливает исходный текст.

Другой метод заключается в угадывании сразу целого слова или фразы. Например, рассмотрим следующий зашифрованный текст, полученный от бухгалтерской фирмы (разбитый на блоки по пять символов):

CTBMN BYCTC BTJDS QXBNS GSTJC BTSWX CTQTZ CQVUJ QJSGS TJQZZ MNQJS VLNSX VSZJU JDSTS JQUUS JUBXJ DSKSU JSNTK BGAQJ ZBGYQ TLCTZ BNYBN QJSW

В сообщении бухгалтерской фирмы, скорее всего, должно встречаться слово *financial* (финансовый). Используя тот факт, что в этом слове буква *i* встречается дважды, разделенная четырьмя другими буквами, мы будем искать в зашифрованном тексте повторяющиеся символы, отстоящие друг от друга на это расстояние. В результате мы найдем 12 таких мест в тексте в позициях 6, 15, 27, 31, 42, 48, 56, 66, 70, 71, 76 и 82. Однако только в двух случаях, в позициях 31 и 42, следующий символ (соответствующий букве *n* в открытом тексте) повторяется в соответствующем месте. Из этих двух вариантов символ *a* будет иметь правильное расположение только для позиции 31. Таким образом, теперь нам известно, что слово *financial* начинается в позиции 30. Далее можно продолжать, применяя лингвистическую статистику английского языка и угадывая целые слова.

Метод перестановки

Шифры, основанные на методе подстановки, сохраняют порядок символов, но подменяют их. Шифры, использующие **метод перестановки**, меняют порядок следования символов, но не изменяют сами символы. На рис. 8.2 показан простой перестановочный шифр с колоночной перестановкой. Ключом к шифру служит слово или фраза, не содержащая повторяющихся букв. В данном примере в качестве ключа используется слово MEGABUCK. Цель ключа — пронумеровать колонки. Первой колонкой становится колонка под буквой, расположенной ближе всего к началу алфавита, и т. д. Открытый текст записывается горизонтально в строках. Шифрованный текст читается по колонкам, начиная с колонки с младшей ключевой буквой.

M	E	G	A	B	U	C	K
7	4	5	1	2	8	3	6
p	l	e	a	s	e	t	r
a	n	s	f	e	r	o	n
e	m	i	l	l	i	o	n
d	o	l	l	a	r	s	t
o	m	y	s	w	i	s	s
b	a	n	k	a	c	c	o
u	n	t	s	i	x	t	w
o	t	w	o	a	b	c	d

Открытый текст

pleasetransferonemilliondollarsto
myswissbankaccountsixtwotwo

Зашифрованный текст

AFLLSKSOSELAWAIATOOSSCTCLNMOMANT
ESILYNTWRNNTSOWDPAEDOBUOERIRICXB

Рис. 8.2. Перестановочный шифр

Чтобы взломать перестановочный шифр, криптоаналитик должен вначале понять, что он имеет дело именно с перестановочным шифром. Если взглянуть на частоту символов *E*, *T*, *A*, *O*, *I*, *N* и т. д., легко заметить, что их частоты соответствуют нормальным частотам открытого текста. В таком случае очевидно, что этот шифр является перестановочным, так как каждая буква в таком шифре представляет сама себя.

Затем нужно угадать число колонок. Во многих случаях по контексту сообщения можно угадать слово или фразу. Например, предположим, что криптоаналитик подозревает, что где-то в сообщении должно встретиться словосочетание *milliondollars*. Обратите внимание, что в результате того, что эти слова присутствуют в исходном тексте, в шифрованном тексте встречаются биграммы *MO*, *IL*, *LL*, *LA*, *IR* и *OS*. Символ *O* следует за символом *M* (то есть они стоят рядом по вертикали в колонке 4), так как они разделены в предполагаемой фразе дистанцией, равной длине ключа. Если бы использовался ключ длиной семь, тогда вместо перечисленных выше биграмм встречались бы следующие: *MD*, *IO*, *LL*, *LL*, *IA*, *OR* и *NS*. Таким образом, для каждой длины ключа в шифрованном тексте образуется различный набор биграмм. Перебрав различные варианты, криптоаналитик часто довольно легко может определить длину ключа.

Остается узнать только порядок колонок. Если число колонок k невелико, можно перебрать все $k(k-1)$ возможных комбинаций пар соседних колонок, сравнивая частоты образующихся биграмм со статистическими характеристиками английского языка. Пара с лучшим соответствием считается правильно позиционированной. Затем все оставшиеся колонки по очереди проверяются в сочетании с уже найденной парой. Колонка, в которой биграммы и триграммы дают максимальное совпадение со статистикой, предполагается правильной. Весь процесс повторяется, пока не будет восстановлен порядок всех колонок. Есть шанс, что на данном этапе текст уже будет распознаваемым (например, если вместо слова *million* мы увидим *milloin*, то сразу станет ясно, где сделана ошибка).

Некоторые перестановочные шифры принимают блок фиксированной длины на входе и выдают блок фиксированной длины на выходе. Такие шифры полностью определяются списком, сообщающим порядок, в котором символы попадают в выходной блок. Например, шифр на рис. 8.2 можно рассматривать в виде шифра с 64-символьным блоком. Его выход описывается последовательностью чисел 4, 12, 20, 28, 36, 44, 52, 60, 5, 13, ..., 62. Другими словами, четвертая входная буква, *a*, первой появится на выходе, за ней последует двенадцатая, *f*, и т. д.

Одноразовые блокноты

Разработать шифр, который невозможно взломать, на самом деле весьма просто. Методика для этого известна уже несколько десятилетий. В качестве ключа выбирается произвольная битовая строка, длина которой совпадает с длиной исходного текста. Открытый текст также преобразуется в последовательность двоичных разрядов, например, с помощью стандартной кодировки ASCII. Наконец, эти две строки поразрядно складываются по модулю 2 (операция «исключающее ИЛИ», XOR). Полученный в результате зашифрованный текст сломать невозможно, поскольку в достаточно большом отрывке любая буква, биграмма или триграмма будет равновероятной. Этот метод, известный как **одноразовый блокнот**, теоретически является панацеей от любых атак, независимо от вычислительных мощностей, которыми обладает или будет когда-либо в будущем обладать взломщик. Объясняется этот невероятный факт с помощью теории информации: дело в том, что в зашифрованном сообщении не содержится никакой информации для взломщика, поскольку любой открытый текст является равновероятным кандидатом.

Пример практического использования одноразового блокнота показан на рис. 8.3. Для начала фраза «I love you.» («Я люблю тебя.») преобразуется в 7-битный ASCII-код. Затем выбирается одноразовая последовательность, *Последовательность 1*, которая складывается по модулю 2 с сообщением. В результате получается некий шифр. Для того чтобы его разгадать, криптоаналитику придется перебрать все возможные одноразовые последовательности, всякий раз проверяя, каким получается открытый текст. Например, если попробовать расшифровать послание с помощью *Последовательности 2* (рис. 8.3), получится текст «Elvis lives» («Элвис жив»). Обсуждение правдоподобности этого утверждения выходит за рамки данной книги, однако, как мы видим, исходное сообщение разгадать не удалось, но при этом получилась вполне нормальная с точки зрения английской грамматики фраза. На самом деле, для любой последовательности из 11 символов в кодировке ASCII найдется одноразовый блокнот для ее генерации. Именно это мы имеем в виду, говоря, что в зашифрованном тексте не содержится никакой информации: из него можно извлечь любое сообщение подходящей длины.

```
Сообщение 1:
1001001 0100001 1101100 1101111 1110110 1100101 0100000 1111001 1101111 1110101 0101110
Последовательность 1:
1010010 1001011 1110010 1010101 1010010 1100011 0001011 0101010 1010111 1100110 0101011
Зашифрованный текст:
0011011 1101011 0011110 0111010 0100100 0000110 0101011 1010011 0111000 0010011 0000101

Последовательность 2:
1011110 0000111 1101000 1010011 1010111 0100110 1000111 0111010 1001110 1110110 1110110
Открытое сообщение 2
1000101 1101100 1110110 1101001 1110011 0100000 1101100 1101001 1110110 1100101 1110011
```

Рис. 8.3. Использование одноразового блокнота для шифрования сообщений и возможность получения произвольного открытого сообщения из зашифрованного путем подстановки другой ключевой последовательности

Одноразовые ключи теоретически являются очень мощным инструментом, однако у них есть ряд практических недостатков. Во-первых, такой длинный ключ невозможно запомнить, поэтому и отправитель, и получатель должны носить с собой письменную копию ключа. Если есть опасность того, что одна из этих копий может быть захвачена неприятелем, хранение письменных копий оказывается весьма нежелательным. Кроме того, полный объем данных, которые могут быть переданы, ограничен размером доступного ключа. Если шпиону повезет и он добудет большое количество информации, может выясниться, что он не сможет пе-

редать все эти сведения в центр, так как ему не хватит длины ключа. Еще одна проблема заключается в чувствительности данного метода к потерянным или вставленным символам. Если отправитель или получатель потеряют синхронизацию, все данные, начиная с этого места, будут испорчены.

С появлением компьютеров метод одноразового блокнота может получить практическое применение. Ключ можно хранить на специальном диске DVD, содержащем несколько гигабит информации. Он даже не вызовет особых подозрений, если его перевозить в коробке от видеокомпакт-диска и в начале даже записать несколько минут фильма. Конечно, в сетях со скоростями передачи данных, исчисляющимися гигабитами, необходимость менять компакт-диск через каждые 30 секунд быстро утомит. Получается, что DVD с одноразовым ключом нужно вручить лично в руки будущему получателю секретного сообщения, а такой подход резко снижает вероятность практического использования метода одноразового блокнота.

Квантовая криптография

Интересно, что решение проблемы передачи по сети одноразовой последовательности пришло из довольно далекой науки — квантовой механики. Эта область остается экспериментальной, однако многообещающей. Если удастся усовершенствовать этот метод, все задачи криптографии можно будет решать с помощью одноразовых последовательностей, ведь это самый надежный способ защиты информации. Далее мы вкратце опишем суть технологии, называемой **квантовой криптографией**. В частности, мы рассмотрим протокол **BB84**, названный так в честь его создателей и года, в котором его описание было впервые опубликовано (Bennet и Brassard, 1984).

Допустим, пользователь по имени Алиса хочет передать одноразовую последовательность другому пользователю, Бобу. Алиса и Боб называются **принципалами**, это главные герои в нашей истории. Например, Боб может быть банкиром, а Алиса — лицом, желающим вступить в деловые отношения с ним. В последнее время практически во всех материалах, касающихся криптографии, имена «Алиса» и «Боб» традиционно используются для обозначения принципалов. Шифровальщики вообще обожают разного рода традиции. Если бы мы стали описывать тут взаимоотношения Андрея и Беллы вместо Алисы и Боба, ни одному слову в этой главе никто бы не поверил (стало бы понятно, что автор, на самом деле, далек от криптографии). А так, может быть, поверят. Поэтому пусть Алиса и Боб будут героями нашей книги.

Итак, если Алисе и Бобу удастся установить единую одноразовую последовательность, их переговоры будут полностью конфиденциальными. Задача стоит следующим образом: как им обменяться секретным ключом, не передавая друг другу DVD? Мы можем предположить, что оба пользователя находятся на разных концах одного оптоволоконного кабеля, по которому они могут передавать и принимать световые импульсы. Тем не менее, нашлась бесстрашная шпионка Труди, которой удалось установить на пути этого кабеля активное подслушивающее устройство. Она может считывать сигналы, идущие в обоих направлениях. Кроме того, она может посылать как в одну, так и в другую сторону фальшивые сообщения. Ситуация для Алисы и Боба, казалось бы, безнадежная. Но тут на помощь приходит квантовая криптография, и мы сейчас увидим, что у наших героев появляется лучик надежды.

Квантовая криптография базируется на том факте, что свет передается маленькими порциями, называемыми **фотонами** и обладающими некоторыми необычными свойствами. Кроме того, пропуская свет через поляризационный фильтр, можно добиться его поляризации. Этот факт известен, прежде всего, тем, кто носит солнцезащитные очки, а также фотографам. Световой луч (то есть поток фотонов), проходя через такой фильтр, поляризуется в направлении оси фильтра (например, вертикально). Если после этого пропустить луч через второй фильтр, интенсивность света на выходе будет пропорциональна квадрату косинуса угла между осями фильтров. Если оси расположить перпендикулярно, фотоны через фильтры проникнуть не смогут. Абсолютная ориентация осей в пространстве значения не имеет — важно только их взаимное расположение.

Чтобы сгенерировать одноразовый блокнот, Алисе понадобятся два набора поляризационных фильтров. Первый набор состоит из вертикального и горизонтального фильтров. Это называется **прямолинейным базисом**. Базис — это просто система координат. Второй набор фильтров отличается от первого только тем, что он повернут на 45°, то есть один фильтр можно представить в виде линии, идущей из нижнего левого угла в верхний правый, а другой — из верхнего левого в нижний правый угол. Это называется **диагональным базисом**. Итак, у Алисы есть два набора фильтров, и она может поставить любой из них на пути светового луча. Но четыре стеклышка-поляризатора — это нечто в стиле сказки про Алису в стране чудес. В реальности имеется кристалл, одна из четырех возможных поляризаций которого изменяется электронным способом с огромной скоростью. У Боба есть такое же устройство, как у Алисы. Тот факт, что у Алисы и Боба есть по два базиса, играет важную роль в квантовой криптографии.

В каждом базисе Алиса обозначает одно из направлений нулем, а другое, соответственно, — единицей. В примере, показанном далее, мы предполагаем, что она выбрала вертикальное направление в качестве нулевого, а горизонтальное — в качестве единичного. Независимо от этого направлению «нижний левый — верхний правый» присваивается значение 0, а направлению «верхний левый — нижний правый» — 1. Эта информация передается Бобу в виде открытого текста.

Теперь Алиса берет одноразовый блокнот, построенный, например, генератором случайных чисел (этот генератор сам по себе, кстати, представляет собой довольно сложный предмет), и передает его Бобу. Передача производится поразрядно, для каждого бита один из двух базисов выбирается случайным образом. Чтобы передать бит, фотонная пушка испускает один фотон, поляризованный таким образом, чтобы он мог пройти через базис, выбранный для этого бита. Например, базисы могут выбираться в такой последовательности: диагональный, прямолинейный, прямолинейный, диагональный, прямолинейный и т. д. Чтобы передать одноразовый блокнот, состоящий из последовательности 1001110010100110, с помощью этих базисов, посылаются фотоны, показанные на рис. 8.4, *а*. Для данного одноразового блокнота и соответствующей ему последовательности базисов единственным образом определяется поляризация, применяемая для каждого

бита. Биты, посылаемые одним фотоном за единицу времени, называются **квантобитами**.

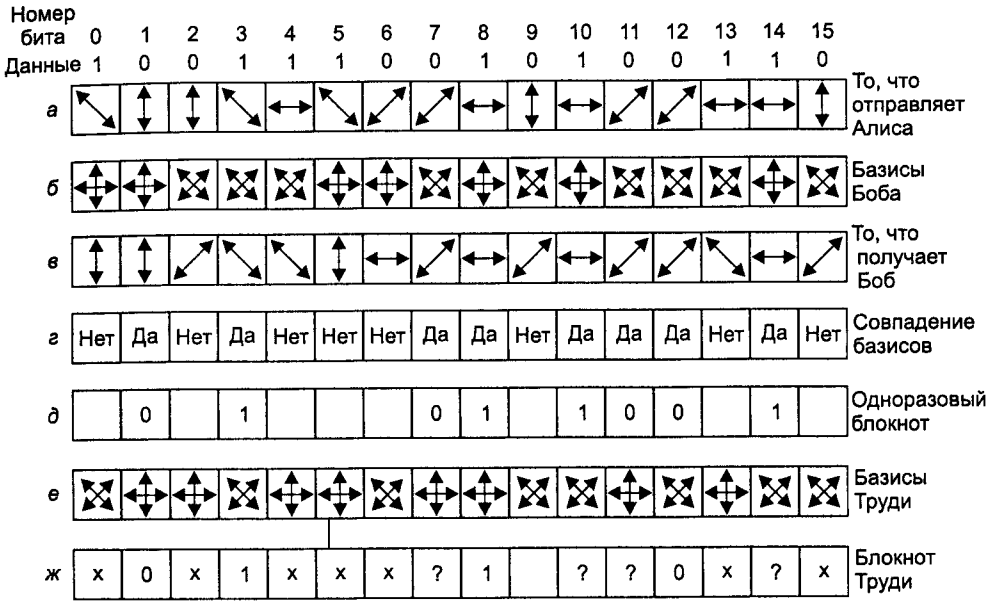


Рис. 8.4. Пример квантовой криптографии

Боб не знает, какой базис надо использовать, поэтому он использует базисы в случайном порядке для каждого прибывающего фотона, как показано на рис. 8.4, *б*. Если для данного фотона базис выбран правильно, Боб получит правильный бит. В противном случае значение бита будет случайным, так как фотон, проходя через поляризатор, повернутый на 45° относительно его собственной поляризации, с равными вероятностями попадет на направление, соответствующее единице или нулю. Это свойство фотонов является одним из основных во всей квантовой механике. Таким образом, некоторые биты будут получены правильно, некоторые — нет, но Боб не может распознать, какие из них являются правильными. Получаемая Бобом информация показана на рис. 8.4, *в*.

Так как же Боб узнает, какие базисы были подставлены правильно, а какие — нет? Для этого он открытым текстом сообщает Алисе, какие базисы он использовал при приеме каждого бита, а она в ответ сообщает (также открытым текстом), какие из базисов были подобраны Бобом корректно. Это показано на рис. 8.4, *г*. Владея этой информацией, Алиса и Боб могут составить битовую строку корректных предположений, что показано на рис. 8.4, *д*. В среднем длина этой строки будет равна половине полной длины исходной строки, однако, так как обеим сторонам это известно, они могут использовать строку корректных предположений в качестве одноразового блокнота. Все, что Алисе надо сделать, это передать битовую строку, длина которой немного превышает удвоенную длину одноразового блокнота. Проблема решена.

Но погодите, мы же забыли про Труди! Предполагается, что ей очень хочется узнать, о чем говорит Алиса, поэтому она внедряет в линию передачи свой детектор и передатчик. К сожалению (для Труди), она тоже не знает, через какой базис пропускать каждый фотон. Лучшее, что она может сделать, это выбирать базисы случайным образом, как Боб. Пример того, как она это делает, показан на рис. 8.4, *е*. Когда Боб открытым текстом сообщает Алисе, какие базисы он использовал, а та отвечает ему, какие из них были правильные, Труди, как и Боб, узнает, какие биты она угадала, а какие — нет. Как видно из рисунка, базисы Труди совпали с базисами Алисы в позициях 0, 1, 2, 3, 4, 6, 8, 12 и 13. Однако из ответа Алисы (рис. 8.4, *г*) ей становится известно, что в одноразовый блокнот входят только биты 1, 3, 7, 8, 10, 11, 12 и 14. Четыре из этих битов (1, 3, 8 и 12) были угаданы правильно, Труди их запоминает. Остальные биты (7, 10, 11 и 14) не были угаданы, и их значения остаются для Труди неизвестными. Таким образом, Бобу известен одноразовый блокнот, начинающийся с последовательности 01011001 (рис. 8.4, *д*), а все, что досталось Труди, — это обрывок 01?1??0? (рис. 8.4, *ж*).

Конечно, Алиса и Боб осознают, что Труди пытается захватить часть их одноразового блокнота, поэтому они стараются уменьшить количество той информации, которая ей может достаться. Для этого они могут произвести некоторые действия над последовательностью. Например, одноразовый блокнот можно поделить на блоки по 1024 бита и возвести каждый из блоков в квадрат, получая, таким образом, числа длиной 2048 бит. Затем можно использовать конкатенацию 2048-битных чисел в качестве одноразового блокнота. Имея лишь часть битовой строки, Труди никогда не сможет проделать эти преобразования. Действия над исходным одноразовым блокнотом, уменьшающие долю информации, получаемой Труди, называются **усилением секретности**. На практике вместо поблочного возведения в квадрат применяются сложные преобразования, в которых каждый выходной бит зависит от каждого входного.

Бедная Труди. У нее не только нет идей по поводу того, как выглядит одноразовый блокнот, но она к тому же понимает, что ее присутствие не является ни для кого секретом. Как-никак, ей приходится передавать каждый принятый бит Бобу, чтобы он думал, будто разговаривает с Алисой. Беда в том, что лучшее, что может сделать Труди, это передавать каждый принятый квантобит с использованием поляризации, с помощью которой он был принят ею. При этом примерно в половине случаев поляризация будет неправильной, что приведет к появлению множества ошибок в одноразовом блокноте Боба.

Когда, наконец, Алиса начинает отправлять данные, она шифрует их, используя сложный код с упреждающей коррекцией ошибок. С точки зрения Боба, ошибка в одном бите одноразовой последовательности — это то же самое, что ошибка передачи данных, произошедшая в одном бите. В любом случае, результат состоит в получении неправильного значения бита. С помощью упреждающей коррекции ошибок, возможно, удастся восстановить потерянную информацию, но можно, кроме того, сосчитать количество ошибок. Если оно намного превышает ожидания (связанные с вероятностью возникновения ошибок оборудования), становится очевидно, что линию прослушивает Труди, и Боб может

принять необходимые меры (например, сказать Алисе, чтобы она переключилась на радиоканал, вызвать полицию и т. д.). Если бы Труди могла скопировать фотон и обрабатывать свою копию, а исходный фотон пересылать Бобу в неизменном виде, у нее был бы шанс остаться незамеченной, однако на сегодняшний день отсутствуют методы клонирования фотонов. Но даже если Труди удастся получить копию фотона, значения квантовой криптографии это ни в коей мере не умалит.

Квантовая криптография может работать при наличии оптоволоконных каналов длиной 60 км, однако требуется сложное и дорогое оборудование. Несмотря на это сама идея весьма многообещающая. Более подробно о квантовой криптографии можно узнать из книги (Mullins, 2002).

Два фундаментальных принципа криптографии

Хотя на следующих страницах мы рассмотрим много различных криптографических систем, в основе их всех лежат два принципа, очень важных для понимания.

Избыточность

Первый принцип гласит, что все зашифрованные сообщения должны содержать определенную избыточность, то есть информацию, не требующуюся для понимания сообщения. Поясним это на примере. Представим себе компанию «Домосед», торгующую по почтовым заказам продуктами 60 000 наименований. Радуясь, что им удалось так экономно распорядиться ресурсами, программисты компании «Домосед» решили, что весь бланк заказа будет состоять из 16 байт имени клиента, за которым последует 3-байтовое поле товара (1 байт для обозначения количества и 2 байта для идентификатора товара). Последние три байта было решено закодировать с помощью очень длинного ключа, известного только клиенту и компании «Домосед».

На первый взгляд, такая схема может показаться надежной, в частности, потому, что пассивные злоумышленники не смогут расшифровать сообщения. К сожалению, в этой схеме имеется критический недостаток, полностью ее обесценивающий. Предположим, какой-нибудь недавно уволенный сотрудник хочет отомстить компании «Домосед» за увольнение. Перед самым уходом ему удается забрать с собой часть списка клиентов. За ночь он составляет программу, посылающую фиктивные заказы с настоящими именами клиентов. Поскольку списка ключей у него нет, он просто помещает в последние три байта случайные числа и посылает сотни заказов компании «Домосед».

Когда эти сообщения прибывают, компьютер компании «Домосед» по имени клиента находит ключ для дешифрации сообщения. К несчастью для компании «Домосед», почти все 3-байтовые сообщения могут восприниматься как достоверные, поэтому компьютер начинает печатать заявки на доставку товаров. Хотя может показаться странным, если клиент заказывает 837 сидений для детских качелей или 540 песочниц, однако вполне возможно, что клиент собирается заняться строительством детских игровых площадок. Таким образом, активный злоумышленник (уволенный сотрудник) способен доставить очень много неприятностей, даже не вникая в смысл сообщений, посылаемых его компьютером.

Эта проблема может быть решена при помощи добавления избыточной информации к каждому сообщению. Например, если добавить к трем шифруемым байтам еще девять, например, нулевых, уволенный сотрудник уже не сможет сформировать серьезный поток достоверно выглядящих сообщений. Мораль этой истории в том, что все сообщения должны содержать достаточное количество избыточной информации, чтобы активный злоумышленник не смог выдать случайный мусор за настоящие сообщения.

Однако добавление избыточной информации облегчает работу криптоаналитика по взлому шифра. Предположим, что конкуренция в бизнесе почтовых заказов чрезвычайно высока и что главному конкуренту компании «Домосед», фирме «Лежебока», очень хочется узнать, сколько песочниц в месяц продает компания «Домосед». Для этого «лежебоки» подключились к телефонной линии «домоседов». В исходной схеме с 3-байтовыми номерами криптоанализ был почти невозможен, поскольку, предположив значение ключа, криптоаналитик не мог проверить правильность своей догадки. Как-никак, почти все сообщения были технически корректны. С введением новой 12-байтовой схемы криптоаналитик легко сможет отличить допустимое сообщение от недопустимого. Итак:

Криптографический принцип номер 1: Сообщения должны содержать избыточные данные.

Другими словами, при расшифровке сообщения получатель должен иметь возможность проверить его подлинность путем анализа и, возможно, выполнения несложных вычислений. Избыточность требуется для того, чтобы можно было противостоять попыткам активных злоумышленников обмануть получателя фальшивыми сообщениями, содержащими мусор. Вместе с тем, добавление избыточной информации облегчает пассивным злоумышленникам задачу взлома системы, так что здесь есть определенное противоречие. Кроме того, избыточные данные никогда не должны принимать форму последовательности нулей в начале или конце сообщения, так как при зашифровке подобных сообщений некоторые алгоритмы дают более предсказуемые результаты, что облегчает работу криптоаналитиков. Многочлен циклического избыточного кода (CRC) значительно лучше подойдет для этих целей, чем просто ряд нулей, поскольку получатель сможет проверить его корректность, и это несколько усложняет работу криптоаналитика. Гораздо удобнее использовать криптографическую хэш-функцию, речь о которой пойдет ниже.

Возвращаясь к квантовой криптографии, хочется сказать о том, какую роль в этой технологии играет избыточность. Из-за того, что Труди перехватывает фотоны, некоторые биты одноразового блокнота, получаемого Бобом, будут иметь неправильные значения. Бобу требуется некоторая избыточность информации, содержащейся во входящих сообщениях, для определения наличия ошибок. Одной из грубых форм избыточности можно считать повторение передачи одного и того же сообщения. Если две пришедшие копии оказываются не идентичными, Боб узнает, что либо канал сильно зашумлен, либо кто-то перехватывает данные.

Конечно, отправлять все по два раза — это слишком. Гораздо лучше использовать код Хэмминга или Рида—Соломона для определения и коррекции ошибок. Однако должно быть понятно, что для того чтобы отличать настоящие сообщения от поддельных, необходима некоторая избыточность. Это особенно важно, если присутствует активный злоумышленник.

Ограниченный срок годности

Второй принцип криптографии состоит в том, что необходимо иметь методы, позволяющие удостовериться в том, что пришедшее сообщение свежее, то есть было послано совсем недавно. Эта мера направлена на борьбу с активными злоумышленниками, воспроизводящими перехваченные ими старые сообщения. Если не принять подобных мер, наш уволенный может подсоединиться к телефонной линии компании «Домосед» и просто повторять посланные ранее настоящие сообщения. Итак, сформулируем утверждение:

Криптографический принцип номер 2: Необходим способ борьбы с повторной отправкой посланных ранее сообщений.

Одной из подобных мер является включение в каждое сообщение временного штампа, действительного, скажем, только в течение 10 с. Получатель может просто хранить принятые сообщения в течение 10 с, отсеивая дубликаты. Сообщения возрастом более 10 с просто игнорируются как устаревшие. Другие меры защиты от дубликатов будут обсуждаться позже.

Алгоритмы с симметричным криптографическим ключом

В современной криптографии применяются те же основные идеи, что и в традиционной криптографии, то есть перестановка и подстановка, но акценты расставляются иначе. В традиционной криптографии применялись простые алгоритмы. Сегодня верно обратное: целью является создание настолько сложного и запутанного алгоритма шифрования, что даже если криптоаналитику попадут в руки целые горы зашифрованного текста, он не сможет извлечь из этого никакой пользы.

Первым классом алгоритмов шифрования, который мы изучим, будет класс **алгоритмов с симметричным ключом**. Он получил такое название благодаря тому, что для шифрации и дешифрации сообщений применяется один и тот же ключ. На рис. 8.1 показан пример использования алгоритма с симметричным ключом. В частности, мы подробно рассмотрим **блочные шифры**, которые принимают на входе n-битные блоки открытого текста и преобразуют их с использованием ключа в n-битный шифр.

Криптографические алгоритмы могут быть реализованы как аппаратно (что повышает скорость их работы), так и программно (для повышения гибкости). Несмотря на то, что большая часть наших размышлений касается алгоритмов и протоколов, не зависящих от конкретной реализации, нам кажется полезным рассмотрение принципов построения шифровальной аппаратуры. Подстановки

и перестановки могут быть реализованы при помощи простых электрических цепей. На рис. 8.5 показано устройство, называющееся **P-блоком** (литера P означает permutation — перестановка) и используемое для перестановки восьми входных разрядов. Если пронумеровать входные биты сверху вниз (01234567), выход этого конкретного P-блока будет выглядеть как 36071245. При помощи соответствующего внутреннего устройства P-блока (распайки проводов) можно заставить его выполнять любую операцию перестановки практически со скоростью света, так как никакие вычисления в нем не производятся, а просто-напросто передается сигнал со входа на выход. Такое решение соответствует принципу Керкгофа: взломщик знает, что используется метод перестановки битов. Однако он не знает ключа, заключающегося в порядке перестановок.

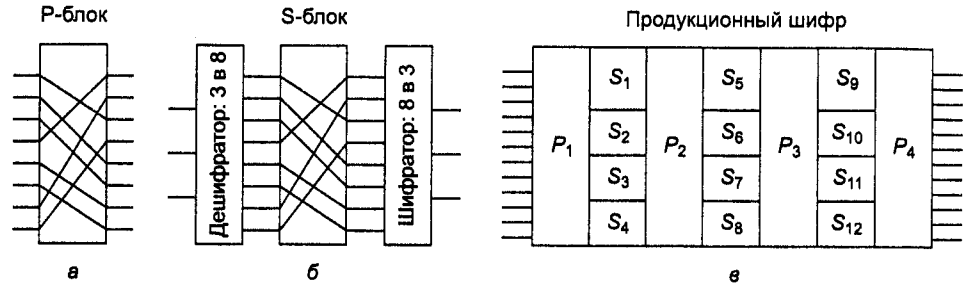


Рис. 8.5. Основные элементы продукционных шифров: P-блок (*а*); S-блок (*б*); продукционный шифр (*в*)

Подстановки (то есть замещения) выполняются **S-блоками** (S означает substitution — подстановка, замена), как показано на рис. 8.5, *б*. В данном примере на вход подается 3-битный открытый текст, а на выходе появляется 3-битный зашифрованный текст. Для каждого входного сигнала выбирается одна из восьми выходных линий декодера путем установки ее в 1. Все остальные линии устанавливаются в 0. Затем эти восемь линий проходят через P-блок, представляющий собой вторую ступень S-блока. Третья ступень производит обратное кодирование одной из восьми линий в 3-битовое двоичное число. Такое устройство заменяет восьмеричные числа 01234567 на 24506713 соответственно. То есть 0 заменяется числом 2, 1 — числом 4 и т. д. Опять же, при соответствующей распайке проводов P-блока внутри S-блока можно реализовать любой вариант подстановки. К тому же такое устройство может быть встроено в аппаратуру и работать на огромных скоростях, поскольку шифраторы и дешифраторы вносят лишь одну или две вентильных задержки (менее 1 нс), а время распространения сигнала внутри P-блока может быть менее 1 пс.

Настоящая сила этих элементов становится очевидна, если сформировать каскад из этих устройств, как показано на рис. 8.5, *в*. Получившееся в результате устройство называется **продукционным шифром**. В данном примере на первом этапе (P_1) 12 входных линий меняются местами. Теоретически, вторая ступень могла бы быть S-блоком, отображающим одно 12-разрядное число в другое 12-разрядное число. Однако такое устройство должно содержать в средней ста-

дии $2^{12} = 4096$ перекрещенных проводов. Вместо этого вход разбивается на четыре группы по 3 разряда, с каждой из которых операция замены выполняется независимо. Хотя такой метод представляет собой лишь частный случай общего решения, его мощь достаточно высока. Выход продукционного шифра можно сделать сложной функцией входа, используя достаточно большое количество дополнительных ступеней.

Продукционные шифры, работающие с k-битными входами и производящие k-битные последовательности, широко распространены. Обычно значение k колеблется от 64 до 256.

Стандарт шифрования данных DES

В январе 1977 году правительство Соединенных Штатов приняло продукционный шифр, разработанный фирмой IBM, в качестве официального стандарта для несекретных сведений. Этот шифр, получивший название **DES** (Data Encryption Standard — стандарт шифрования данных), получил широкое распространение в промышленности для защиты информации. В своем исходном виде он уже больше не является надежным, но в модифицированном виде все еще полезен. Сейчас мы объясним принципы его работы.

Схема DES-шифра показана на рис. 8.6, *а*. Открытый текст шифруется блоками по 64 бита, в результате чего на выходе получаются 64-битные блоки зашифрованного текста. Алгоритм, использующий 56-разрядный ключ, состоит из 19 отдельных этапов. На первом этапе выполняется независимая перестановка 64 разрядов открытого текста. Последний представляет собой обратную перестановку. Предпоследний этап меняет местами левые и правые 32 разряда. Остальные 16 этапов функционально идентичны, но управляются разными функциями входного ключа. Алгоритм был разработан так, чтобы дешифрация выполнялась тем же ключом, что и шифрование. Это обеспечивает соответствие алгоритма принципу симметричных ключей. Этапы при расшифровке просто выполняются в обратном порядке.

Операция, выполняемая на одном из промежуточных этапов, показана на рис. 8.6, *б*. На каждом этапе из двух порций по 32 разряда на входе формируются две порции по 32 разряда на выходе. Правая половина входа просто копируется в левые разряды выхода. Правые 32 выходных разряда представляют собой сумму по модулю 2 левой части входа и функции правой части входа и ключа данного этапа K_i. Вся сложность шифра заключается в этой функции.

Функция состоит из четырех последовательно выполняемых шагов. Сначала из 32 разрядов правой части R_{i-1} с помощью фиксированной перестановки и дублирования формируется 48-разрядное число E. На втором шаге число E и ключ K_i складываются по модулю 2. Затем выход разделяется на восемь групп по шесть разрядов, каждая из которых преобразуется независимым S-блоком в 4-разрядные группы. Наконец, эти $8 \cdot 4$ разряда пропускаются через P-блок.

На каждом из 16 этапов используются различные функции исходного ключа. Перед началом работы алгоритма к ключу применяется 56-разрядная перестановка. Перед каждым этапом ключ разделяется на две группы по 28 разрядов, каждая из которых вращается влево на число разрядов, зависящее от номера этапа. Ключ K_i получается из результата этой операции при помощи еще одной перестановки 56 разрядов. На каждом этапе из 56 разрядов ключа выбираются 48 разрядов, которые также переставляются местами.

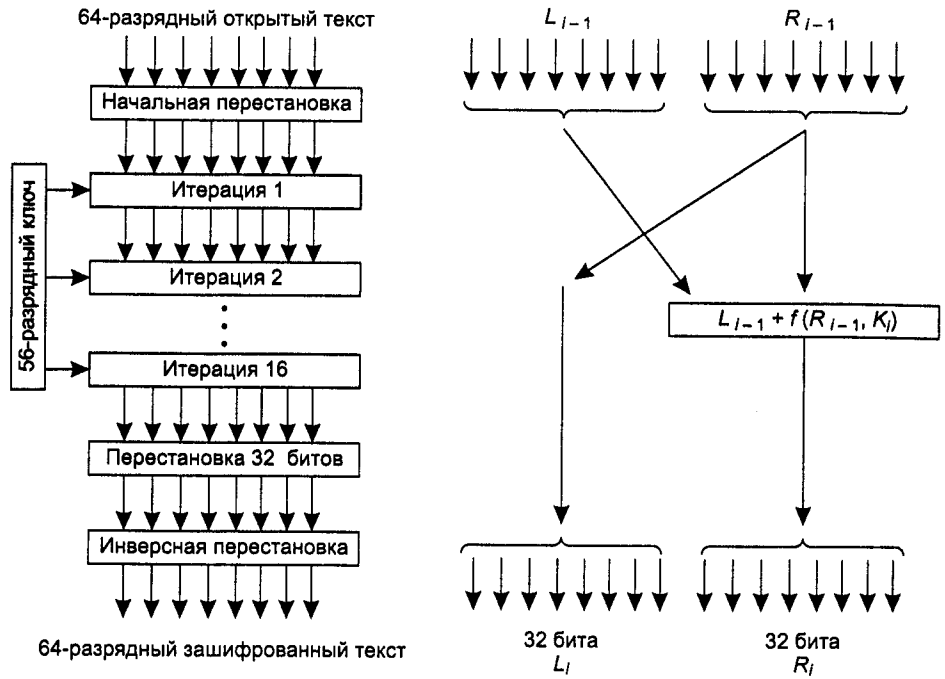


Рис. 8.6. Стандарт шифрования данных DES: общий вид (*а*); детализация одного из этапов (*б*)

Иногда для повышения надежности DES используется метод, называемый **побелкой**. Он заключается в том, что перед передачей шифра каждый блок открытого текста складывается по модулю 2 с произвольным 64-битным ключом, затем отправляется в устройство DES, после чего получившийся шифр складывается по модулю 2 со вторым 64-битным ключом. На приемном конце побелка легко устраняется путем выполнения обратных операций (это возможно, если у получателя есть два побелочных ключа). Поскольку применение этого метода увеличивает длину ключа, полный перебор в пространстве значений ключа становится еще более длительным. Обратите внимание: для побелки каждого блока применяется один и тот же ключ (то есть для каждого сообщения имеется только один побелочный ключ).

Стандарт шифрования данных DES был полон противоречий с самого момента его создания. Он основывался на шифре Люцифер (Lucifer), разработанном и запатентованном корпорацией IBM, с той разницей, что IBM использовала 128-разрядный, а не 56-разрядный ключ. Когда федеральное правительство

Соединенных Штатов пожелало стандартизировать какой-то шифр для несекретного применения, оно «пригласило» IBM на «обсуждение» этого вопроса с Агентством национальной безопасности, NSA (National Security Agency), являющимся самым крупным в мире работодателем в области математики и криптоанализа. Агентство национальной безопасности США настолько секретно, что существует даже такая популярная шутка:

Вопрос: Что означает аббревиатура NSA?
Ответ: No Such Agency — такого агентства нет.

После этих обсуждений корпорация IBM уменьшила длину ключа со 128 до 56 бит и решила держать в секрете процедуру разработки стандарта DES. Многие полагали, что длина ключа была уменьшена, чтобы гарантировать, что NSA сможет взломать DES, но организациям с более низким финансированием это будет не по силам. Вероятно, цель засекречивания проекта состояла в сокрытии потайного хода, позволяющего Агентству национальной безопасности еще легче взламывать шифр DES. Когда сотрудник этого управления предложил Институту инженеров по электротехнике и электронике (IEEE) отменить планирующуюся конференцию по криптографии, ощущения комфорта это не прибавило. Агентство национальной безопасности всегда препятствовало всему.

В 1977 году ученые Стэнфордского университета, занимающиеся исследованиями в области криптографии, Диффи (Diffie) и Хеллман (Hellman), разработали машину для взлома кода DES и оценили стоимость ее создания в 20 млн долларов. По небольшому участку открытого текста и соответствующего ему зашифрованному тексту эта машина путем полного перебора 2^{56} вариантов за один день могла найти 56-разрядный ключ. На сегодняшний день такая машина могла бы стоить около 1 млн долларов.

Тройное шифрование с помощью DES

Уже в 1979 году корпорация IBM поняла, что ключ стандарта DES слишком короток, и разработала метод, позволяющий существенно увеличить его надежность с помощью тройного шифрования (Tuchman, 1979). Выбранный метод, ставший с тех пор Международным стандартом 8732, показан на рис. 8.7. Здесь используются два ключа и три этапа. На первом этапе открытый текст зашифровывается (блок Encryption на рисунке) обычным DES ключом K_1. На втором этапе DES работает в режиме дешифрации (блок Decryption), используя ключ K_2. Наконец, выполняется еще одна операция шифрования с ключом K_1.

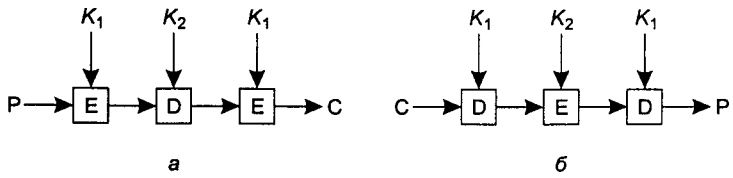


Рис. 8.7. Тройное шифрование с помощью DES (*а*); дешифрация (*б*)

Сразу возникают два вопроса. Во-первых, почему используются только два ключа, а не три? Во-вторых, почему используется последовательность операций EDE (Шифрация Дешифрация Шифрация), а не EEE (Шифрация Шифрация Шифрация)? Причина использования всего двух ключей в том, что даже самые параноидальные шифровальщики в мире считают, что в настоящее время ключа длиной 112 бит вполне достаточно для коммерческих приложений (хотя в мире криптографии паранойя считается достоинством, а не болезнью). Переход на 168 разрядов повлечет лишь дополнительные расходы по хранению и транспортировке дополнительного ключа, а реальной пользы принесет мало.

Использование последовательности шифрования, дешифрации и снова шифрования объясняется обратной совместимостью с существующими DES-системами с одним ключом. Обе функции, шифрования и дешифрации, устанавливают соответствия между наборами 64-разрядных чисел. С точки зрения криптографии обе функции одинаково надежны. Однако использование EDE вместо EEE позволяет компьютеру, применяющему тройное шифрование, общаться с компьютером, применяющим обычное одиночное шифрование, просто установив $K_1 = K_2$. Таким образом, тройное шифрование можно легко включать в виде дополнительного режима работы, что не представляет интереса для ученых криптографов, но довольно важно для корпорации IBM и ее клиентов.

Улучшенный стандарт шифрования AES

В какой-то момент стало понятно, что ресурс DES (даже с тройным шифрованием) уже приближается к концу. Тогда Национальный институт стандартов и технологий (NIST) — агентство Министерства торговли, занимающееся разработкой стандартов для Федерального правительства США, — решило, что правительству нужен новый криптографический стандарт для несекретных данных. NIST ясно осознавал все противоречия, связанные с DES, и прекрасно понимал, что как только будет объявлено о создании нового стандарта, все, кто хоть что-то смыслит в криптографической политике, по умолчанию будут предполагать, что и здесь имеется лазейка, с помощью которой Агентство национальной безопасности с легкостью сможет расшифровывать любую информацию. На таких условиях вряд ли кто-то согласится применять у себя новую технологию, и она, скорее всего, так и умрет в безвестности.

Исходя из этих предпосылок, институт стандартов и технологий применил неожиданный для правительственного бюрократического аппарата подход: он решил просто спонсировать криптографический конкурс. В январе 1997 года ученые со всего мира были приглашены для представления своих разработок, касающихся нового стандарта, который назвали **AES** (Advanced Encryption Standard — улучшенный стандарт шифрования). Требования, предъявляемые к разработкам, были таковы:

1. Алгоритм должен использовать симметричный блочный шифр.
2. Все детали разработки должны быть общедоступны.
3. Должны поддерживаться длины ключей 128, 192 и 256 бит.

4. Должна быть возможна как программная, так и аппаратная реализация.
5. Алгоритм должен быть общедоступным или базирующимся на не дискредитировавших себя понятиях.

Было рассмотрено 15 серьезных предложений. На общедоступных конференциях разработчики представляли свои проекты, а оппоненты должны были приложить максимум усилий для поиска возможных недостатков в каждом из проектов. В августе 1998 года Институтом стандартов и технологий были выбраны пятеро финалистов. Выбор основывался в основном на таких аспектах, как обеспечиваемая безопасность, эффективность, простота, гибкость, а также требования к памяти (это важно для встроенных систем). Был проведен еще ряд конференций, на которых было высказано множество критических замечаний. На последней конференции было проведено независимое голосование. Его результаты выглядели следующим образом:

1. Rijndael (Джон Домен (John Daemen) и Винсент Раймен (Vincent Rijmen), 86 голосов).
2. Serpent (Росс Андерсон (Ross Anderson), Эли Бихам (Eli Biham) и Ларс Кнудсен (Lars Knudsen), 59 голосов).
3. Twofish (команда, возглавляемая Брюсом Шнайером (Bruce Schneier), 31 голос).
4. RC6 (компания RSA Laboratories, 23 голоса).
5. MARS (корпорация IBM, 13 голосов).

В октябре 2000 года NIST объявил о том, что он также голосует за Rijndael, и уже в ноябре 2001 года Rijndael становится стандартом правительства США, опубликованным как Федеральный стандарт обработки информации, FIPS 197. Благодаря полной открытости конкурса, а также благодаря техническим возможностям Rijndael и тому факту, что выигравшая конкурс команда состояла из двух молодых бельгийских шифровальщиков (которые вряд ли стали бы сотрудничать с NSA, предоставляя какие-то лазейки), ожидается, что Rijndael станет доминирующим мировым криптографическим стандартом, по крайней мере, на ближайшее десятилетие. Название Rijndael (произносится примерно как Райндол) представляет собой сокращение фамилий авторов: Раймен + Домен.

Rhindael поддерживает длины ключей и размеры блоков от 128 до 256 бит с шагом в 32 бита. Длины ключей и блоков могут выбираться независимо друг от друга. Тем не менее, стандарт AES говорит о том, что размер блока должен быть равен 128 битам, а длина ключа — 128, 192 или 256 бит. Однако вряд ли кто-то будет использовать 192-битные ключи, поэтому фактически AES применяется в двух вариантах: со 128-битными блоками и 128-битными ключами, а также со 128-битными блоками и 256-битными ключами.

Далее приводится описание алгоритма, и там мы рассматриваем только один случай — 128/128, поскольку именно это, скорее всего, станет нормой для коммерческих приложений. 128-битный ключ означает, что размер пространства его значений равен $2^{128} \approx 3 \cdot 10^{38}$. Даже если Агентству национальной безопасности удастся собрать машину на миллионе параллельных процессоров, каждый из которых будет способен вычислять один ключ в пикосекунду, на перебор всех значений потребуется около 10^{10} лет. К тому времени Солнце уже давно потухнет, и нашим далеким потомкам придется читать распечатку со значением ключа при свете свечи.

Rijndael

С математической точки зрения, метод Rijndael основывается на теории полей Галуа, благодаря чему можно строго доказать некоторые его свойства, касающиеся секретности. Тем не менее, можно рассматривать его и с точки зрения кода программы на языке C, не вдаваясь в математические подробности.

Как и в DES, в Rijndael применяются замены и перестановки. И там, и там используются несколько итераций, их число зависит от размера ключа и блока и равно 10 для 128-разрядного ключа и 128-битных блоков; для максимального размера ключа и блоков число итераций равно 14. Однако, в отличие от DES, все операции выполняются над целыми байтами, что позволяет создавать эффективные реализации как в аппаратном, так и в программном исполнении. Схематичный алгоритм метода Rijndael приведен в листинге 8.1.

Листинг 8.1. Схематичный алгоритм метода Rijndael

```
#define LENGTH 16/* Число байтов в блоке данных или ключе */
#define NROWS 4/* Число строк в массиве state */
#define NCOLS 4/* Число столбцов в массиве state */
#define ROUNDS 10/* Число итераций */
typedef unsigned char byte/8-разрядное целое без знака */

rijndael(byte plaintext[LENGTH], byte ciphertext[LENGTH], byte key[LENGTH])
{
int r;/* Счетчик цикла */
byte state[NROWS][NCOLS];/* Текущее состояние */
struct{byte k[NROWS][NCOLS];} rk[ROUNDS + 1];/* Ключи итерации */

expand_key(key,rk);/* Сформировать ключи итерации */
copy_plaintext_to_text(state, plaintext);/* Инициализация текущего состояния */
xor_roundkey_into_state(state, rk[0]);/* Сложить по модулю 2 ключ с текущим состоянием
*/

for(r=1; r<=ROUNDS; r++) {
substitute(state);/* Пропустить каждый байт через S-блок */
rotate_rows(state);/* Повернуть строку i на i байт */
if(r < ROUNDS) mix_columns(state);/* Смешивающая функция */
xor_roundkey_into_state(state, rk[r]);/* Сложить по модулю 2 ключ с текущим состоянием
*/
}
copy_state_to_ciphertext(ciphertext, state);/* Вернуть результат */
}
```

У функции `rijndael` три аргумента: `plaintext` — массив размером 16 байт, содержащий входные данные, `ciphertext` — массив размером 16 байт, в который будет возвращен шифр, а также `key` — 16-разрядный ключ. В процессе вычислений текущее состояние данных сохраняется в байтовом массиве `state`, размер которо-

го равен NROWS × NCOLS. Для 128-битных блоков данных размер этого массива равен 4×4 байта. В 16 байтах целиком уместится один блок.

Массив state изначально содержит открытый текст и модифицируется на каждом этапе вычислений. На некоторых этапах выполняется побайтовая подстановка. На других этапах — перестановка байтов внутри массива. Могут выполняться и другие преобразования. В конечном итоге содержимое state представляет собой зашифрованный текст, который и возвращается в качестве результата функции.

Алгоритм начинается с распространения ключа по 11 массивам одинакового размера, представляющим состояние (state). Эти массивы хранятся в rk — массиве структур, содержащих массивы состояний. Одна из этих структур будет использована в начале вычислений, а остальные 10 — во время 10 итераций (по одной на итерацию). Вычисление ключа для каждой итерации производится довольно сложным образом, и мы не будем рассматривать детали этого процесса. Достаточно сказать, что для этого осуществляются циклические повороты и суммирования по модулю 2 различных групп разрядов ключа. Подробности можно узнать в (Daemen и Rijmen, 2002).

Следующий шаг состоит в копировании открытого текста в массив state для того, чтобы его можно было обрабатывать во время последующих итераций. Копируется текст в колонки по 4 байта: первые 4 байта попадают в колонку 0, вторые — в колонку 1 и т. д. И колонки, и строки нумеруются с нуля, а итерации — с единицы. Процесс создания 12 байтовых массивов размером 4×4 показан на рис. 8.8.

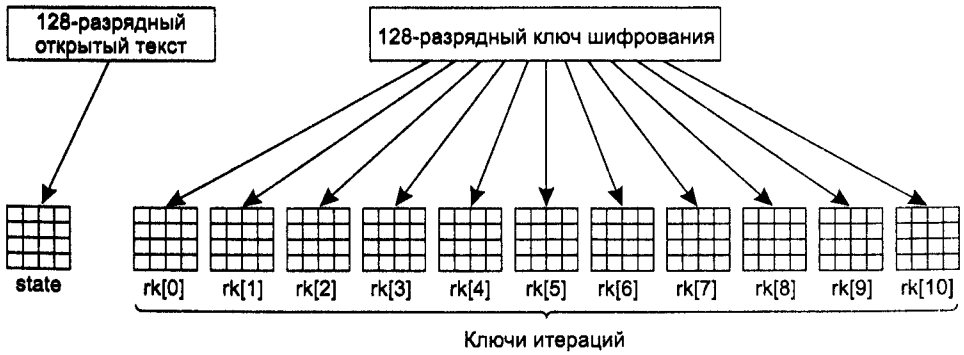


Рис. 8.8. Создание массивов state и rk

Перед началом основного цикла вычислений производится еще одно действие: rk[0] поразрядно складывается по модулю 2 с массивом state. Другими словами, каждый из 16 байт в массиве state заменяется суммой по модулю 2 от него самого и соответствующего байта в rk[0].

Только после этого начинается главное развлечение. В цикле проводятся 10 итераций, в каждой из которых массив state подвергается преобразованию. Каждый раунд (итерация) состоит из четырех шагов. На шаге 1 в state производится посимвольная подстановка. Каждый байт по очереди используется в качестве индекса для S-блока, заменяющего его значение на соответствующую запись S-блока. На этом шаге получается прямой моноалфавитный подстановочный шифр. В отличие от DES, где используются несколько S-блоков, в Rijndael S-блок всего один.

На шаге 2 каждая из четырех строк поворачивается влево. Строка 0 поворачивается на 0 байт (то есть не изменяется), строка 1 — на 1 байт, строка 2 — на 2 байта, а строка 3 — на 3 байта. Смысл заключается в разбрасывании данных вокруг блока. Это аналогично перестановкам, показанным на рис. 8.5.

На шаге 3 происходит независимое перемешивание всех колонок. Делается это с помощью операции матричного умножения, в результате которого каждая новая колонка оказывается равной произведению старой колонки на постоянную матрицу. При этом умножение выполняется с использованием конечного поля Галуа, $GF(2^8)$. Несмотря на то, что все это кажется довольно сложным, алгоритм устроен так, что каждый элемент матрицы вычисляется посредством всего лишь двух обращений к таблице и трех суммирований по модулю 2 (Daemen и Rijmen, 2002, приложение E).

Наконец, на шаге 4 ключ данной итерации складывается по модулю 2 с массивом state.

Благодаря обратимости всех действий расшифровка может быть выполнена с помощью такого же алгоритма, но с обратным порядком следования всех шагов. Однако есть одна хитрость, которая позволяет заниматься расшифровкой, используя алгоритм шифрования с измененными таблицами.

Данный алгоритм обладает не только очень высокой защищенностью, но и очень высокой скоростью. Хорошая программная реализация на машине с частотой 2 ГГц может шифровать данные со скоростью 700 Мбит/с. Такой скорости достаточно для шифрации видео в формате MPEG-2 в реальном масштабе времени. Аппаратные реализации работают еще быстрее.

Режимы шифрования

Несмотря на всю свою сложность, AES (или DES, или любой другой блочный код) представляет собой, по сути дела, моноалфавитный подстановочный шифр с большими длинами символов (в AES используются 128-битные символы, в DES — 64-битные). Для любого отрывка открытого текста шифр при прогоне через один и тот же шифрующий блок будет получаться всегда одинаковым. Скажем, если вы будете 100 раз пытаться зашифровать текст *abcdefgh*, задавая всякий раз один и тот же ключ для алгоритма DES, вы получите 100 одинаковых копий шифра. Взломщик может попытаться использовать этот недостаток при попытке расшифровки текста.

Режим электронного шифроблокнота

Чтобы понять, каким образом это свойство моноалфавитного подстановочного шифра может быть использовано для частичного взлома шифра, мы рассмотрим (тройное) кодирование по стандарту DES только лишь потому, что изображать 64-разрядные блоки проще, чем 128-разрядные. Надо иметь при этом в виду, что AES сталкивается с теми же самыми проблемами. Самый очевидный способ ко-

дирования длинного сообщения заключается в разбиении его на отдельные блоки по 8 байт (64 бита) с последующим кодированием этих блоков по очереди одним и тем же ключом. Последний блок при необходимости можно дополнить до 64 бит. Такой метод называется **режимом электронного шифроблокнота**, по аналогии со старомодными шифроблокнотами, содержавшими слова и соответствующие им шифры (обычно это были пятизначные десятичные числа).

На рис. 8.9 показано начало компьютерного файла, в котором перечислены поощрительные премии сотрудникам компании. Этот файл состоит из последовательных 32-разрядных записей, по одной на сотрудника, следующего формата: 16 байт на имя, 8 байт на должность и 8 байт на премию. Каждый из шестнадцати 8-байтовых блоков (пронумерованных от 0 до 15) кодируется шифром DES.

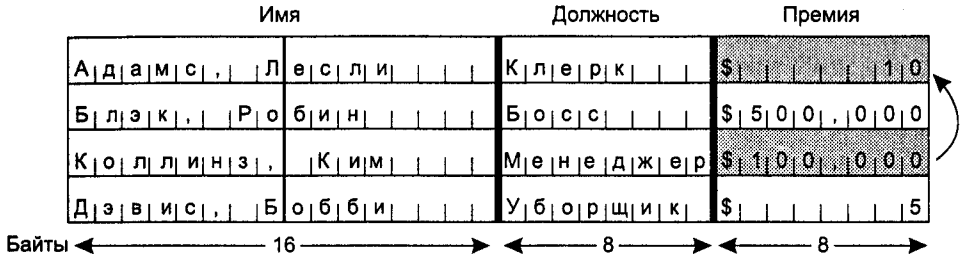


Рис. 8.9. Открытый текст файла, зашифрованного в виде 16 DES-блоков

Лесли только что поругалась с боссом и не рассчитывает на большую премию. Ким, напротив, — любимица босса, что всем известно. Лесли может получить доступ к файлу после того как он будет зашифрован, но до того как он будет отослан в банк. Может ли Лесли исправить ситуацию, имея доступ только к зашифрованному файлу?

В данном случае это очень просто. Все, что нужно сделать Лесли, — это скопировать зашифрованный блок 12 (содержащий премию Ким) и заменить им блок 4 (содержащий премию Лесли). Даже не зная содержимого блока 12, Лесли может рассчитывать на значительно более веселое Рождество. (Скопировать зашифрованный блок 8 тоже можно, но вероятность, что это будет обнаружено, выше; кроме того, Лесли, в общем-то, не жадная).

Режим сцепления блоков шифра

Чтобы противостоять атакам подобного типа, все блочные шифры можно модернизировать таким образом, чтобы замена одного блока вызывала повреждение других блоков открытого текста после их расшифровки, превращая эти блоки (начиная с модифицированного места) в мусор. Один из таких способов — **сцепление блоков шифра**. При этом методе, показанном на рис. 8.10, каждый блок открытого текста перед зашифровкой складывается по модулю 2 с предыдущим уже зашифрованным блоком. При этом одинаковым блокам открытого текста уже не соответствуют одинаковые блоки зашифрованного текста. Таким образом, шифр перестает быть большим моноалфавитным подстановочным шифром. Первый блок складывается по модулю 2 со случайным **вектором инициализации**, IV (Initialization Vector), передаваемым вместе с зашифрованным текстом.

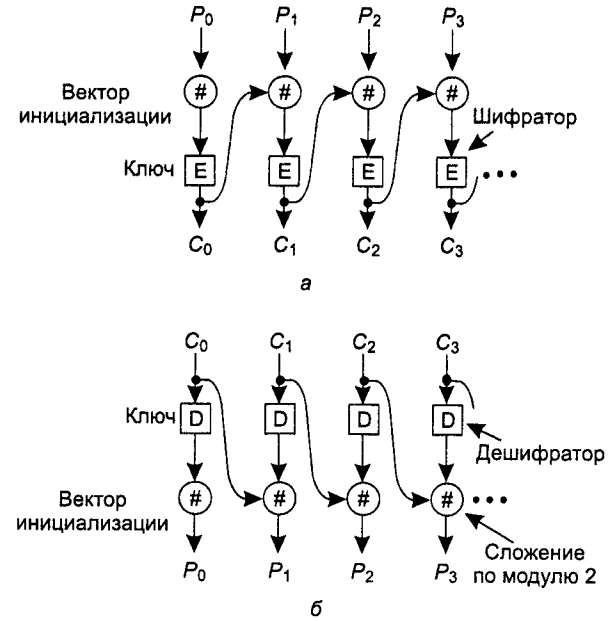


Рис. 8.10. Сцепление зашифрованных блоков: шифрование (*а*); дешифрация (*б*)

Рассмотрим работу сцепления блоков шифра на примере рис. 8.10. Начнем с вычисления $C_0 = E(P_0 \text{ XOR } IV)$. Затем мы вычислим $C_1 = E(P_1 \text{ XOR } C_0)$ и т. д. Расшифровка производится по формуле $P_0 = IV \text{ XOR } D(C_0)$, и т. д. Обратите внимание на то, что блок i является функцией всех блоков открытого текста с 0 по $i - 1$, поэтому один и тот же исходный блок текста преобразуется в разные зашифрованные блоки в зависимости от их расположения. При использовании такого способа шифрования преобразование, произведенное Лесли, приведет к появлению двух блоков чепухи, начиная с поля премии Лесли. Для сообразительного офицера службы безопасности эта странность может послужить подсказкой в последующем расследовании.

Сцепление блоков шифра также обладает тем достоинством, что усложняет криптоанализ, так как одни и те же блоки открытого текста преобразуются в разные зашифрованные блоки. Именно по этой причине и применяется описанный метод.

Режим шифрованной обратной связи

Однако у метода сцепления блоков шифра есть и недостаток, заключающийся в том, что прежде чем может начаться шифрование или дешифрация, должен появиться целый 64-битовый блок данных. Для пользователей интерактивных терминалов, набирающих строки короче восьми символов и ждущих ответа, такой

метод не подходит. Для побайтового шифрования может применяться **режим шифрованной обратной связи** с использованием (тройного) DES, как показано на рис. 8.11. Для стандарта AES идея остается той же самой, только используется 128-разрядный сдвиговый регистр. На рисунке мы видим состояние шифрующей машины после того, как байты с 0 по 9 уже зашифрованы и посланы. Когда прибывает десятый байт открытого текста, как показано на рис. 8.11, *а*, алгоритм DES обрабатывает 64-разрядный сдвиговый регистр, чтобы произвести 64-разрядный зашифрованный блок. Самый левый байт этого зашифрованного текста извлекается и складывается по модулю 2 с P_{10}. Этот байт передается по линии. Затем сдвиговый регистр сдвигается влево на 8 разрядов. При этом байт C_2 извлекается с левого конца регистра, а байт C_{10} вставляется в него на освободившееся место справа от C_9. Обратите внимание на то, что содержимое сдвигового регистра зависит от всей предыстории открытого текста, так что повторяющиеся фрагменты исходного текста будут кодироваться каждый раз по-разному. Как и для метода сцепленных блоков шифра, для начала шифрования этим методом требуется вектор инициализации.

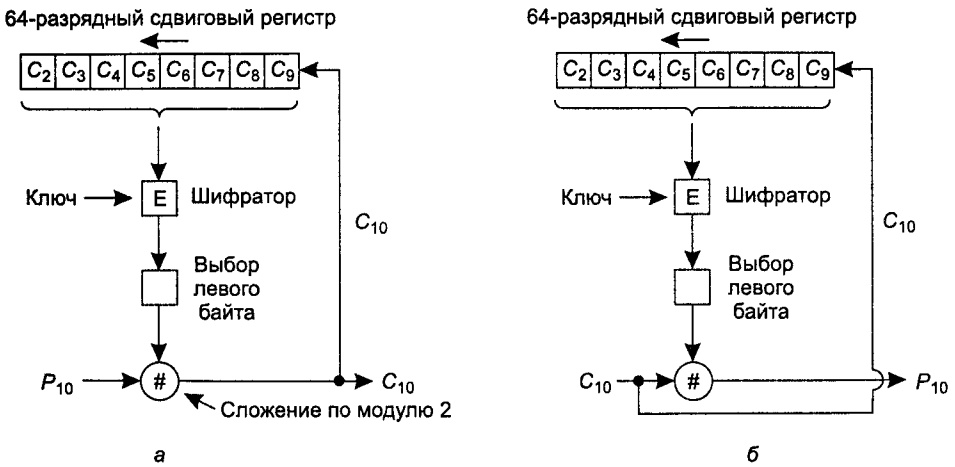


Рис. 8.11. Режим шифрованной обратной связи (*а*); обратная связь по выходу (*б*)

При использовании режима шифрованной обратной связи дешифрация аналогична шифрованию. В частности, содержимое сдвигового регистра *шифруется*, а не *дешифруется*, поэтому байт, который складывается по модулю 2 с C_{10} для получения P_{10}, равен тому байту, который складывается по модулю 2 с P_{10} для получения C_{10}. Пока содержимое двух сдвиговых регистров идентично, дешифрация выполняется корректно. Это показано на рис. 8.11, *б*.

Проблемы с режимом шифрованной обратной связи начинаются, когда при передаче шифрованного таким способом текста один бит случайно инвертируется. При этом испорченными окажутся 8 байтов, расшифровываемые в то время, когда поврежденный байт находится в сдвиговом регистре. Когда поврежденный байт покинет сдвиговый регистр, на выходе опять станет расшифровываться правильный открытый текст. Таким образом, результат инверсии одного бита оказывается относительно локализованным и не портит всего остатка сообщения.

Режим группового шифра

Тем не менее, существуют приложения, в которых один испорченный при передаче бит приводит к порче 64 бит открытого текста, а это многовато. Для таких приложений существует четвертый вариант, называемый **режимом группового** (потокового) **шифра**. Суть его заключается в том, что выходной блок получается шифрацией вектора инициализации с использованием ключа. Затем этот выходной блок снова шифруется с использованием ключа, в результате чего получается второй выходной блок. Для получения третьего блока шифруется второй блок, и т. д. Последовательность (произвольной длины) выходных блоков, называемая **ключевым потоком**, воспринимается как одноразовый блокнот и складывается по модулю 2 с открытым текстом. В результате получается шифрованный текст, как показано на рис. 8.12, *а*. Обратите внимание: вектор инициализации используется только на первом шаге. После этого шифруются выходные блоки. Кроме того, ключевой поток не зависит от данных, поэтому он в случае необходимости может быть вычислен заранее и совершенно не чувствителен к ошибкам передачи. Процесс дешифрации показан на рис. 8.12, *б*.

Дешифрация осуществляется путем генерации точно такого же ключевого потока на принимающей стороне. Поскольку он зависит только от вектора инициализации и ключа, ошибки передачи шифрованного текста на него не влияют. Таким образом, ошибка в одном бите передаваемого шифра приводит к ошибке только одного бита расшифрованного текста.

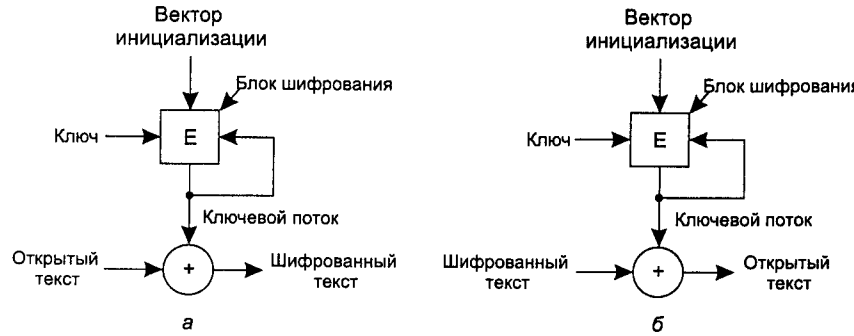


Рис. 8.12. Групповой шифр: шифрование (*а*); дешифрация (*б*)

Важно никогда не использовать одну и ту же пару ключ — вектор инициализации в одном и том же групповом шифре, поскольку при этом всякий раз будет получаться одинаковый ключевой поток. Повторное использование ключевого потока может привести к неприятному эффекту **взлома шифра при помощи многократного использования ключевого потока**. Допустим, блок открытого текста P_0 шифруется с помощью ключевого потока, в результате чего получается сумма

по модулю 2 P_0 и K_0. Затем берется второй блок открытого текста, Q_0, и шифруется тем же ключевым потоком (получаем Q_0 XOR K_0). Криптоаналитик, перехвативший оба блока шифрованного текста, может просто сложить их вместе по модулю 2 и получить в результате P_0 XOR Q_0, убирая тем самым ключ. Теперь у него есть сумма по модулю 2 двух блоков открытого текста. Если один из них известен (или его можно угадать), найти второй — не проблема. В любом случае, взломать сумму по модулю 2 двух блоков открытого текста можно, используя статистические свойства сообщения. Скажем, если передается английский текст, то наиболее часто встречающейся буквой в потоке будет «е», и т. д. Короче говоря, имея сумму по модулю 2 двух частей открытого текста, взломщик с высокой вероятностью сможет вычислить обе части.

Режим счетчика

Все режимы, кроме электронного шифроблокнота, обладают одним и тем же неприятным свойством: доступ к произвольной части зашифрованных данных невозможен. Допустим, например, что файл передается по сети и затем сохраняется на диске в зашифрованном виде. Так иногда делают, если принимающий компьютер представляет собой ноутбук, который может быть украден. Хранить все важные данные в зашифрованном виде действительно полезно: риск утечки секретной информации в случае попадания аппаратуры в руки «нехорошим дядям» резко снижается.

Однако доступ к дискам зачастую бывает необходимо осуществлять в произвольном порядке. Особенно это касается файлов баз данных. Если файл зашифрован в режиме сцепления блоков, придется вначале дешифровать все блоки, предшествующие нужному. Согласитесь, такой способ работы несколько неудобен. По этой причине был введен еще один режим шифрования — **режим счетчика**. Он показан на рис. 8.13. Здесь открытый текст не шифруется напрямую. Вместо этого шифруется вектор инициализации плюс некоторая константа, а уже получающийся в результате шифр складывается по модулю 2 с открытым текстом. Сдвигаясь на 1 по вектору инициализации при шифрации каждого нового блока, можно легко получить способ дешифрации любого места файла. При этом нет необходимости расшифровывать все предшествующие блоки.

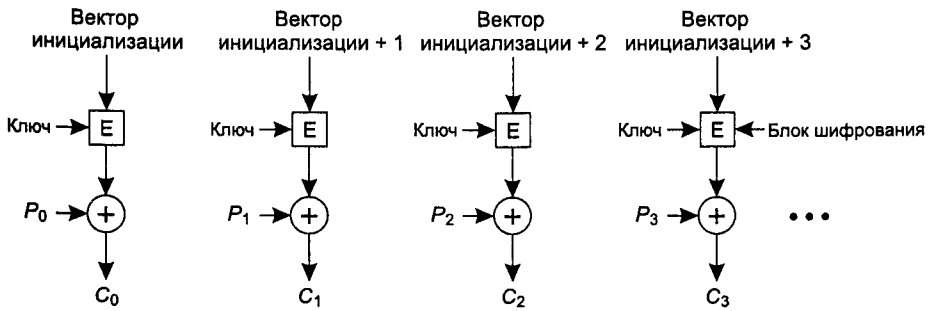


Рис. 8.13. Шифрование в режиме счетчика

Несмотря на то что режим счетчика весьма полезен, у него есть один существенный недостаток, который стоит упомянуть. Допустим, уже использовавшийся однажды ключ K будет использован повторно (с другим открытым текстом, но с тем же вектором инициализации), и взломщик захватит весь шифрованный текст, который был послан в обоих случаях. Ключевые потоки остаются неизменными, в итоге возникает риск взлома за счет повторного использования ключевого потока (тот же эффект, на который мы уже указывали, обсуждая режим группового шифра). Все, что криптоаналитику остается сделать, это сложить по модулю 2 два перехваченных сообщения. Тем самым он полностью снимет какую бы то ни было криптографическую защиту, и в его распоряжении окажется сумма по модулю 2 двух блоков открытого текста. Этот недостаток вовсе не означает, что режим счетчика в целом неудачен. Это говорит лишь о том, что как ключи, так и векторы инициализации должны выбираться независимо и случайным образом. Даже если один и тот же ключ случайно попадется дважды, от перехвата информацию может спасти отличающийся вектор инициализации.

Другие шифры

DES и Rijndael — это самые известные криптографические алгоритмы с симметричными ключами. Тем не менее, стоит отметить, что в природе существует еще много других шифров с симметричными ключами. Некоторые из них встраиваются в различные программно-аппаратные продукты. Наиболее распространенные из них перечислены в табл. 8.2.

Таблица 8.2. Некоторые распространенные криптографические алгоритмы с симметричными ключами

Название	Автор	Длина ключа	Комментарии
Blowish	Брюс Шнайер (Bruce Schneier)	1–448 бит	Старый и медленный
DES	IBM	56 бит	Слишком слабый для современных систем
IDEA	Массей (Massey) и Ксюэйа (Xuejia)	128 бит	Хороший, но запатентованный
RC4	Рональд Ривест (Ronald Rivest)	1–2048 бит	Внимание: есть слабые ключи
RC5	Рональд Ривест (Ronald Rivest)	128–256 бит	Хороший, но запатентованный
Rijndael	Домен (Daemen) и Раймен (Rijmen)	128–256 бит	Лучший
Serpent	Андерсон (Anderson), Байхэм (Biham) и Кнудсен (Knudsen)	128–256 бит	Очень сильный
Тройной DES	IBM	168 бит	На втором месте после Rijndael
Twofish	Брюс Шнайер (Bruce Schneier)	128–256 бит	Очень сильный; широко распространен

Криптоанализ

Прежде чем закончить разговор об использовании симметричных ключей в криптографии, необходимо хотя бы упомянуть о четырех направлениях развития криптоанализа. Первый подход называется **дифференциальным криптоанализом** (Biham и Shamir, 1993). Он может использоваться для взлома любых блочных шифров. Для начала анализируется пара блоков открытого текста, различающихся лишь небольшим числом бит. При этом внимательно анализируется происходящее при каждой внутренней итерации во время шифрации. Во многих случаях можно заметить, что одни битовые последовательности встречаются чаще других, и это соображение используется для взлома, основанного на теории вероятностей.

Второй подход, который следует обозначить, называется **линейным криптоанализом** (Matsui, 1994). С его помощью можно взломать DES только с 2^{43} известными открытыми текстовыми блоками. Принцип работы основан на суммировании по модулю 2 некоторых бит открытого текста и изучении результатов для шаблонных последовательностей. Если повторять эту процедуру много раз, половина бит будет иметь нулевые значения, половина — единичные. Тем не менее, довольно часто это соотношение изменяется в ту или иную сторону. Это отклонение, каким бы малым оно ни было, может использоваться для снижения показателя трудозатрат криптоаналитика. Более подробную информацию см. в материалах автора этого метода, Мацуи (Matsui).

Третье направление развития связано с анализом потребляемой электроэнергии для вычисления секретного ключа. Обычно в компьютерах напряжение 3 В соответствует логической единице, а 0 В — логическому нулю. Таким образом, обработка единицы требует большего потребления электроэнергии, чем обработка нуля. Если криптографический алгоритм состоит из цикла, в котором разряды ключа обрабатываются поочередно, взломщик, заменив главные системные n-гигагерцевые часы медленными (например, с частотой 100 Гц) и повесив «крокодилы» на ножки питания и заземления центрального процессора, может с большой точностью отслеживать мощность, потребляемую каждой машинной инструкцией. По этим данным узнать секретный ключ оказывается на удивление просто. Этот метод криптоанализа можно победить лишь аккуратным кодированием алгоритма на языке Ассемблера таким образом, чтобы энергопотребление не зависело ни от общего ключа, ни от ключей каждой итерации.

Четвертый подход основан на временном анализе. Криптографические алгоритмы содержат большое количество условных операторов (`if`), тестирующих биты итерационных ключей. Если части `then` и `else` выполняются за различное время, то, замедлив системные часы и измерив длительность всех шагов, можно вычислить ключи итераций. По этим ключам обычно можно вычислить и общий ключ. Анализ энергопотребления и временной анализ могут применяться и одновременно, что позволяет упростить задачу криптоанализа. Несмотря на то, что анализы энергозатрат и времени выполнения операций могут показаться несколько экзотическими, на самом деле они представляют собой мощные методы, способные взломать любой шифр, если только он не имеет специальной защиты.

Алгоритмы с открытым ключом

Исторически процесс передачи ключа всегда был слабым звеном почти во всех системах шифрования. Независимо от того, насколько прочна была сама криптосистема, если нарушитель мог украсть ключ, система становилась бесполезной. До 1976 года все криптологи исходили из предпосылки, что ключ дешифрации должен быть идентичен ключу шифрования (или один может легко получиться из другого). В то же время, ключи должны были быть у всех пользователей системы. Таким образом, казалось, что эта проблема неустранима: ключи должны быть защищены от кражи, и в то же время их нужно распространять среди пользователей, поэтому их нельзя просто хранить в банковском сейфе.

В 1976 году два исследователя из Стэнфордского университета, Диффи (Diffie) и Хеллман (Hellman), предложили радикально новую криптосистему, в которой ключ шифрования и ключ дешифрации были различными, кроме того, ключ дешифрации нельзя было получить из ключа шифрования. Предложенные ими алгоритм шифрования E и алгоритм дешифрации D (оба параметризованные ключом) должны были удовлетворять следующим трем требованиям:

1. $D(E(P)) = P$.
2. Крайне сложно вывести D из E.
3. E нельзя взломать при помощи произвольного открытого текста.

Первое требование состоит в том, что если применить алгоритм дешифрации D к зашифрованному сообщению $E(P)$, то мы опять получим открытый текст P. Без этого авторизованный получатель просто не сможет расшифровать сообщение. Второе требование говорит само за себя. Третье требование необходимо, потому что, как мы скоро увидим, злоумышленники могут экспериментировать с алгоритмом столько, сколько пожелают. При таких условиях нет никаких причин, по которым ключ шифрования нельзя было бы сделать общедоступным.

Этот метод работает следующим образом. Некто, например Алиса, желая получать секретные сообщения, сначала формирует два алгоритма, удовлетворяющие перечисленным выше требованиям. Затем алгоритм шифрования и его ключ открыто объявляются, отсюда название — **шифрование с открытым ключом**. Это можно сделать, разместив открытый ключ, например, на домашней страничке Алисы. Для обозначения алгоритма шифрования, параметризованного открытым ключом Алисы, мы будем использовать запись E_A. По аналогии (секретный) алгоритм дешифрации, параметризованный персональным ключом Алисы, мы будем обозначать D_A. Боб делает то же самое, открыто объявляя E_B, но храня в тайне D_B.

Теперь посмотрим, сможем ли мы решить проблему установки надежного канала между Алисой и Бобом, которые ранее никогда не встречались. Оба ключа шифрования Алисы и Боба, E_A и E_B, являются открытыми. (Вообще, все пользователи сети могут, становясь пользователями, опубликовать свои ключи шифрования.) Теперь Алиса берет свое первое сообщение P, вычисляет $E_B(P)$ и посылает его Бобу. Боб расшифровывает его с помощью своего секретного ключа D_B, то есть вычисляет $D_B(E_B(P)) = P$. Больше никто не может прочитать это зашифро-

ванное сообщение $E_B(P)$, так как предполагается, что система шифрования достаточно надежна, а получить ключ D_B на основании известного ключа E_B очень трудно. Посылая ответ, Боб передает $E_A(R)$. Таким образом, Алиса и Боб получают надежный секретный канал связи.

Обратите внимание на используемую здесь терминологию. Шифрование с открытым ключом предполагает у каждого пользователя наличие двух ключей — открытого ключа, используемого всеми для шифрования сообщений, посылаемых этому пользователю, и закрытого ключа, требующегося пользователю для дешифрации приходящих к нему сообщений. Мы будем и далее называть эти ключи *открытым* и *закрытым*, чтобы отличать их от *секретных* ключей, используемых для шифрования и дешифрации в обычной криптографии с симметричным ключом.

Алгоритм RSA

Единственная загвоздка состоит в том, чтобы найти алгоритмы, удовлетворяющие всем трем требованиям. Поскольку преимущества шифрования с открытым ключом очевидны, многие исследователи неустанно работали над созданием подобных алгоритмов, и некоторые из них уже опубликованы. Один хороший метод был разработан группой исследователей Массачусетского технологического института (Rivest и др., 1978). Он назван по начальным буквам фамилий трех разработчиков: **RSA** (Rivest, Shamir, Adleman). Этот метод вот уже четверть века выдерживает попытки взлома и считается очень прочным. На его основе построены многие практические системы безопасности. Главный недостаток RSA заключается в том, что для обеспечения достаточного уровня защищенности требуется ключ длиной, по крайней мере, 1024 бита (против 128 бит в алгоритмах с симметричными ключами). Из-за этого алгоритм работает довольно медленно.

В основе метода RSA лежат некоторые принципы теории чисел. Опишем в общих чертах, как пользоваться этим методом. Подробности см. в соответствующих источниках.

1. Выберем два больших простых числа p и q (обычно длиной 1024 бита).
2. Сосчитаем $n = pq$ и $z = (p - 1)(q - 1)$.
3. Выберем число d, являющееся взаимно простым с числом z.
4. Найдем такое число e, что остаток от деления произведения ed на число z равен 1.

Вычислив заранее эти параметры, можно начинать шифрование. Сначала разобьем весь открытый текст (рассматриваемый в качестве битовой строки) на блоки так, чтобы каждое сообщение P попадало в интервал $0 < P < n$. Это несложно сделать, если разбить открытый текст на блоки по k бит, где k — максимальное целое число, для которого $2^k < n$.

Чтобы зашифровать сообщение P, вычислим $C = P^e \pmod{n}$. Чтобы расшифровать C, сосчитаем $P = C^d \pmod{n}$. Можно доказать, что для всех значений P в указанном диапазоне функции шифрования и дешифрации являются взаимно обратными. Чтобы выполнить шифрование, нужны e и n. Для дешифрации требуются d и n. Таким образом, открытый ключ состоит из пары (e, n), а закрытый ключ — из пары (d, n).

Надежность метода обеспечивается сложностью нахождения множителей больших чисел. Если бы криптоаналитик мог разложить на множители (открытое) число n, он мог бы тогда найти значения p и q, а следовательно, и число z. После этого числа e и d можно найти при помощи алгоритма Евклида. К счастью, математики пытались решить проблему разложения на множители больших чисел по меньшей мере 300 лет, и накопленный опыт позволяет предположить, что эта проблема чрезвычайно трудна.

Ривест (Rivest) с коллегами утверждает, что для разложения на множители числа из 500 цифр необходимо 10^{25} лет, если применять грубую силу. Предполагается, что задействованы лучший известный алгоритм и компьютер, выполняющий одну инструкцию за 1 мкс. Даже при сохранении экспоненциального роста скоростей компьютеров потребуются века, чтобы найти множители числа из 500 цифр, а к этому времени наши потомки могут просто выбрать еще большие p и q.

Тривиальный учебный пример работы алгоритма RSA приведен на рис. 8.14. Для этого примера мы выбрали $p = 3$, а $q = 11$, что дает значения $n = 33$, а $z = 20$. Число d можно выбрать равным 7, так как числа 20 и 7 не имеют общих делителей. При таком выборе значение e можно найти, решив уравнение $7e = 1 \pmod{20}$, откуда следует, что $e = 3$. Зашифрованный текст C получается из открытого сообщения P по формуле $C = P^3 \pmod{33}$. Получатель расшифровывает сообщение по формуле $P = C^7 \pmod{33}$. В качестве примера на рисунке показано шифрование слова «SUZANNE».

Открытый текст (P)		Зашифрованный текст (C)			После дешифрации	
Символ	Число	P^3	$P^3 \pmod{33}$	C^7	$C^7 \pmod{33}$	Символ
S	19	6859	28	13492928512	19	S
U	21	9261	21	1801088541	21	U
Z	26	17576	20	1280000000	26	Z
A	01	1	1	1	1	A
N	14	2744	5	78125	14	N
N	14	2744	5	78125	14	N
E	05	125	26	8031810176	5	E
Вычисление отправителя			Вычисление получателя			

Рис. 8.14. Пример работы алгоритма RSA

Поскольку выбранные для данного примера простые числа так малы, P должно быть менее 33, поэтому каждый блок открытого текста может содержать лишь одну букву. В результате получается моноалфавитный подстановочный шифр, что не очень впечатляет. Если бы мы вместо этого выбрали числа p и q порядка 2^{512}, тогда число n было бы около 2^{1024}. В этом случае каждый блок мог бы содержать до 1024 бит, или 128 восьмиразрядных символов, против 8 символов шифра DES или 16 AES.

Следует отметить, что использование алгоритма RSA в описанном ранее виде аналогично использованию симметричного алгоритма в режиме ECB (Electronic Code Book — электронный шифроблокнот), в котором одинаковые блоки на входе преобразуются в одинаковые блоки на выходе. Таким образом, для шифрования данных требуется сцепление блоков в каком-либо виде. Однако на практике алгоритм RSA с открытым ключом используется только для передачи одноразового секретного ключа, после чего применяется какой-нибудь алгоритм с симметричным ключом типа AES или тройного DES. Система RSA слишком медленная, чтобы шифровать большие объемы данных, однако она широко применяется для распространения ключей.

Другие алгоритмы с открытым ключом

Хотя алгоритм RSA получил широкое распространение, он ни в коей мере не является единственным известным алгоритмом с открытым ключом. Первым алгоритмом с открытым ключом стал «алгоритм ранца» (Merkle и Hellman, 1978). Его идея состоит в том, что имеется большое количество объектов различного веса. Владелец этих объектов кодирует сообщение, выбирая подмножество объектов и помещая их в ранец. Общий вес объектов в рюкзаке известен всем, как и список всех возможных объектов. Список объектов, находящихся в рюкзаке, хранится в секрете. При определенных дополнительных ограничениях, задача определения возможного списка объектов по известному общему весу считалась неразрешимой для вычисления, то есть считалось, что решение можно найти только полным перебором различных сочетаний предметов списка. Поэтому она была положена в основу алгоритма с открытым ключом.

Изобретатель алгоритма Ральф Меркле (Ralph Merkle) был настолько уверен в надежности своего алгоритма, что предложил 100 долларов любому, кто сумеет его взломать. Ади Шамир (Adi Shamir), «S» в группе RSA, мгновенно взломал его и получил награду. Это не смутило Меркле. Он усилил алгоритм и предложил за его взлом уже 1000 долларов. Рон Ривест (Ron Rivest), «R» в RSA, тут же взломал улучшенную версию алгоритма и получил награду. Меркле не рискнул предложить 10 000 долларов за следующую версию, поэтому «A», Леонарду Эйдлману (Leonard Adleman), не повезло. Несмотря на то, что алгоритм ранца был в очередной раз исправлен, он не считается надежным и редко используется.

Другие схемы с открытым ключом основаны на сложности вычисления дискретных логарифмов. Алгоритмы, использующие этот принцип, были разработаны Эль-Гамалем (El Gamal, 1985) и Шнорром (Schnorr, 1991).

Существуют и некоторые другие методы, например, основанные на эллиптических кривых (Menezes и Vanstone, 1993). Однако две основные категории составляют алгоритмы, основанные на сложности нахождения делителей больших чисел и вычислений дискретных логарифмов. Эти задачи считаются особенно сложными, так как математики уже много лет пытаются их решить без особых успехов.

Цифровые подписи

Подлинность различных бумажных документов (юридических, финансовых и др.) определяется наличием или отсутствием авторизованной рукописной подписи. Фотокопии документами не считаются. Чтобы системы компьютерных сообщений могли заменить физическое перемещение документов, написанных чернилами на бумаге, нужно решить проблему подписи.

Проблема разработки замены рукописной подписи довольно сложна. По существу, требуется система, с помощью которой одна сторона могла бы послать другой стороне «подписанное» сообщение так, чтобы:

- получатель мог проверить объявленную личность отправителя;
- отправитель не мог позднее отрицать содержимое сообщения;
- получатель не мог позднее изменить подписанное сообщение.

Первое требование существенно, например, для финансовых систем. Когда компьютер клиента заказывает компьютеру банка купить тонну золота, банковский компьютер должен быть уверен, что компьютер, пославший заказ, действительно принадлежит компании, со счета которой следует снять денежную сумму. Другими словами, банк должен иметь возможность установить подлинность клиента (а клиент — подлинность банка).

Второе требование необходимо для защиты банка от мошенничества. Предположим, что банк покупает тонну золота, и сразу после этого цена на золото резко падает. Бесчестный клиент может подать в суд на банк, заявляя, что он никогда не посылал заказа на покупку золота. Банк может показать сообщение в суде, но клиент будет отрицать, что посылал его. Таким образом, должно быть обеспечено требование **невозможности отречения от данных ранее обязательств**. Методы составления цифровых подписей, которые мы изучим далее, предназначены в том числе и для этого.

Третье требование нужно для защиты клиента в случае, если цена на золото после его покупки банком взлетает вверх и банк пытается создать подписанное сообщение, в котором клиент просит купить не одну тонну золота, а один слиток. При таком сценарии банк, совершив мошенничество, забирает оставшуюся часть золота себе.

Подписи с симметричным ключом

Один из методов реализации цифровых подписей состоит в создании некоего центрального авторитетного органа, которому все доверяют, — назовем его, например, Большим Братом (Big Brother, BB). Затем каждый пользователь выбирает секретный ключ и лично относит его в офис Большого Брата. Таким образом, например, секретный ключ Алисы, K_A, известен только Алисе и Большому Брату.

Когда Алиса хочет послать открытым текстом своему банкиру Бобу подписанное сообщение P, она формирует сообщение, зашифрованное K_A (ключом

Алисы), $K_A(B, R_A, t, P)$, где B — идентификатор Боба, R_A — случайное число, выбранное Алисой, t — временной штамп, подтверждающий свежесть сообщения. Затем она посылает его Большому Брату, как показано на рис. 8.15. Большой Брат видит, что это сообщение от Алисы, расшифровывает его и посылает Бобу. Сообщение, посылаемое Бобу, содержит открытый текст сообщения Алисы и подпись Большого Брата $K_{BB}(A, t, P)$. Получив подписанное сообщение, Боб может выполнять заказ Алисы.

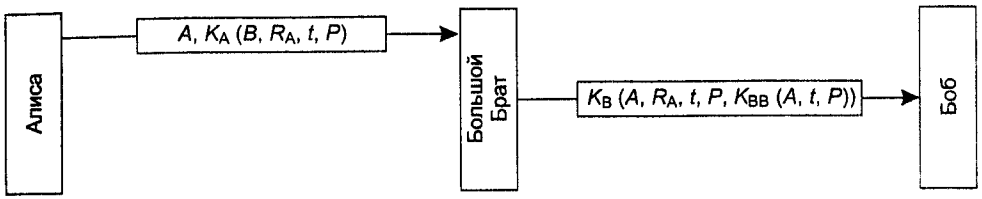


Рис. 8.15. Цифровая подпись Большого Брата

Что случится, если позднее Алиса станет отрицать отправку этого сообщения? Естественно, в случае возникновения подобных конфликтов конфликтующие стороны первым делом подают друг на друга в суд (по крайней мере, в Соединенных Штатах). Наконец, когда дело попадает в суд, Алиса энергично утверждает, что она не отправляла Бобу обсуждаемое. Судья спрашивает Боба, почему он уверен, что данное сообщение пришло именно от Алисы, а не от злоумышленницы Труди. Боб сначала заявляет, что Большой Брат не принял бы сообщения от Алисы, если бы оно не было зашифровано ее ключом K_A, — у злоумышленника просто нет возможности послать Большому Брату фальшивое сообщение от Алисы.

Затем Боб элегантным жестом демонстрирует суду сообщение A: $K_{BB}(A, t, P)$. Боб заявляет, что это сообщение подписано Большим Братом, что доказывает, что Алиса послала сообщение P Бобу. Затем судья просит Большого Брата (которому все доверяют) проверить подпись под этим сообщением. Когда Большой Брат подтверждает, что Боб говорит правду, судья решает дело в пользу Боба. Дело закрыто.

Теоретически проблема с этим протоколом цифровых подписей, который показан на рис. 8.15, может возникнуть, если злоумышленник повторно воспроизведет оба сообщения. Для минимизации вероятности этого используются временные штампы. Кроме того, Боб может просмотреть все недавние сообщения, проверяя, не встречалось ли в них такое же R_A. В этом случае сообщение считается дубликатом и просто игнорируется. Очень старые сообщения, обнаруживаемые по значению временного штампа, также игнорируются. Для защиты от мгновенной атаки повторным воспроизведением Боб просто проверяет значение случайного числа R_A, содержащегося в каждом приходящем сообщении, запоминая все такие числа, полученные за последний час. Если в течение часа такое значение получено еще не было, Боб может быть уверен, что пришедшее сообщение является новым заказом.

Подписи с открытым ключом

Главная проблема, связанная с применением шифрования с симметричным ключом для цифровых подписей, состоит в том, что все должны согласиться доверять Большому Брату. Кроме того, Большой Брат получает возможность читать все подписываемые им сообщения. Наиболее логичными кандидатами на управление сервером Большого Брата являются правительство, банки или нотариальные бюро. Однако эти организации вызывают доверие не у всех граждан. Таким образом, было бы гораздо лучше, если бы для получения подписи на электронном документе не требовалась авторитетная доверительная организация.

К счастью, здесь может помочь шифрование с открытым ключом. Предположим, что алгоритмы шифрования и дешифрации с открытым ключом, помимо обычного свойства $D(E(P)) = P$, обладают свойством $E(D(P)) = P$. Таким свойством, например, обладает алгоритм RSA, поэтому такое предположение не является голословным. В этом случае Алиса может послать Бобу подписанное открытое сообщение P, переслав ему $E_B(D_A(P))$. Обратите внимание на то, что Алиса знает свой собственный (закрытый) ключ дешифрации D_A, так же как и открытый ключ Боба E_B, так что сформировать такое сообщение ей по силам.

Получив это сообщение, Боб расшифровывает его как обычно, используя свой закрытый ключ D_B и получая в результате $D_A(P)$, как показано на рис. 8.16. Он сохраняет этот зашифрованный текст в надежном месте, после чего расшифровывает его открытым ключом шифрования Алисы E_A, получая открытый текст.

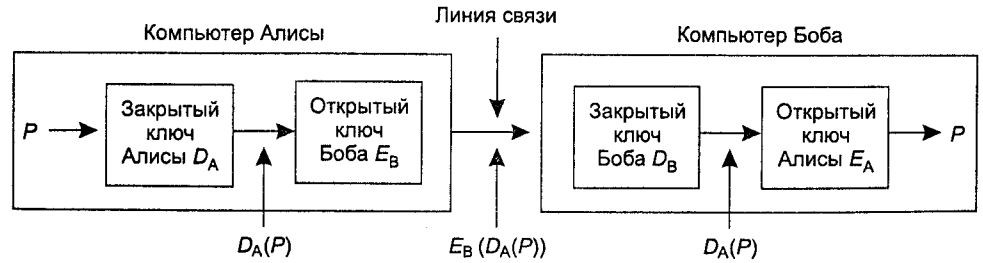


Рис. 8.16. Цифровая подпись, полученная при помощи шифрования с открытым ключом

Чтобы понять, как работает цифровая подпись в данном случае, предположим, что Алиса впоследствии отрицает, что посылала Бобу сообщение P. Когда дело доходит до суда, Боб предъявляет суду P и $D_A(P)$. Судья легко может убедиться, что у Боба есть действительное сообщение, зашифрованное ключом D_A, просто применив к нему ключ E_A. Боб не знает закрытого ключа Алисы, следовательно, получить зашифрованное этим ключом сообщение он мог только от Алисы. Сидя в тюрьме за лжесвидетельство и мошенничество, Алиса сможет заняться разработкой новых интересных алгоритмов с открытым ключом.

Хотя схема использования шифрования с открытым ключом довольно элегантна, она обладает серьезными недостатками, связанными, правда, скорее не с самим алгоритмом, а со средой, в которой ему приходится работать. Во-первых, Боб может доказать, что это сообщение было послано Алисой, только пока