**Веб-кэширование**

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

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

Отметим, что веб-кэш является и сервером, и клиентом одновременно: когда он получает запросы от браузера и отправляет ему ответы, он выступает в роли сервера, когда он обменивается сообщениями с веб-сервером, то играет роль клиента.

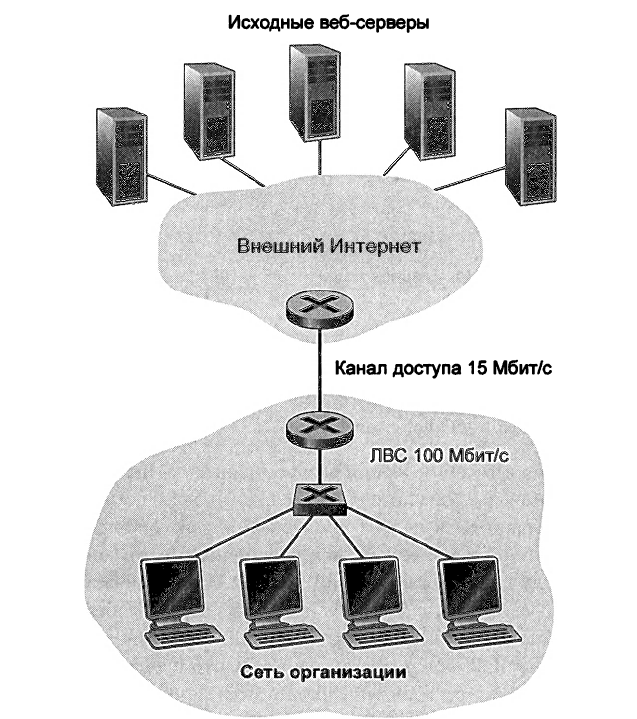


**Рис. 2.11. Запросы клиентов через прокси-сервер**

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

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

Давайте рассмотрим пример для лучшего понимания того, какую же выгоду дает использование прокси-серверов. На рис. 2.12 показаны две сети — сеть организации и публичный Интернет. Сеть организации представляет собой высокоскоростную ЛВС. Маршрутизатор в органи­зации и маршрутизатор в Интернете соединены между собой каналом связи со скоростью 15 Мбит/с. Веб-серверы, находящиеся в Интернете, размещены по всем миру. Размер среднего объекта будем считать рав­ным 1 Мбит, а среднюю скорость запросов от браузеров организации к веб-серверам — составляющей 15 запросов в секунду. Допустим, что размер HTTP-запроса очень мал и не создает трафика ни в сетях, ни в канале связи между маршрутизаторами.



**Внешний Интернет**

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

Также предположим, что среднее время, прошедшее с момента, когда маршрутизатор со стороны Интернета перенаправляет HTTP-запрос (внутри IP-дейтаграммы) до получения ответа (обычно в составе нескольких дейтаграмм), равно двум секундам. Будем называть это время «задержкой Интернета». Общее время ответа — то есть время от момента запроса объектом браузера до получения им объекта — это сумма задержки ЛВС, задержки доступа (то есть задержки сигнала между двумя маршрутизаторами) и задержки Интернета. Произведем расчет для оценки этой задержки. Интенсивность трафика в сети ЛВС равна

(15 запросов/с) \* (1 Мбит/запрос)/(100 Мбит/с) = 0,15

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

(15 запросов/с) \* (1 Мбит/запрос)/(15 Мбит/с) = 1

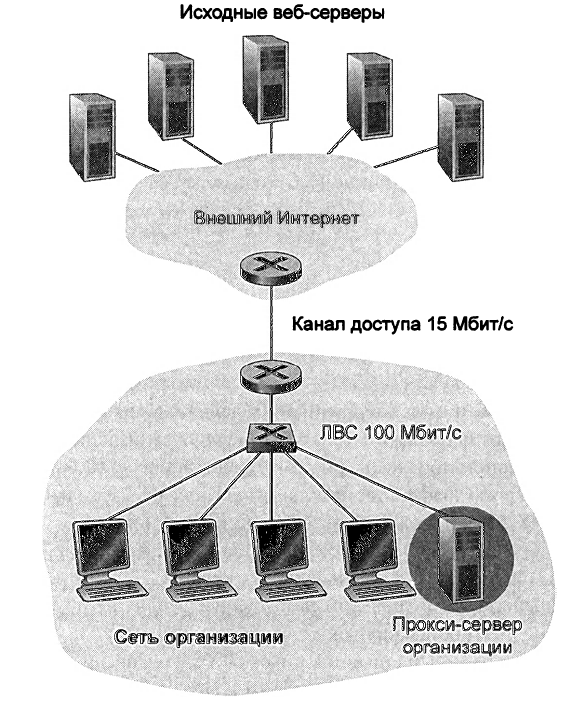
Величина интенсивности трафика в ЛВС, равная 0,15, обычно при­водит к задержкам порядка 10 мс; следовательно, задержками ЛВС можно пренебречь. Однако, как обсуждалось ранее, по мере того, как интенсивность трафика достигает единицы (как в случае с сетью доступа на рис. 2.12), задержка становится очень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

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

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

0,4 \* (0,01с) + 0.6 \* (2,01с)

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

**Рис. 2.13 Добавление прокси-сервера в сеть организации**

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

**Метод get с условием**

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

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

GET /fruit/kiwi.gif HTTP/1.1 Host: [www.exotiquecuisine.com](http://www.exotiquecuisine.com)

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

НТТР/1.1 200 ОК

Date: Sat, 8 Oct 2011 15:39:29

Server: Apache/1.3.0 (Unix)

Last-Modified: Wed, 7 Sep 2011 09:23:24

Content-Type: image/gif

(данные данные данные данные данные...)

Прокси-сервер перенаправляет объект браузеру, но, кроме того, со­храняет у себя в кэше его локальную копию. При этом важно отметить, что вместе с объектом сохраняется дата его последнего изменения. Да­лее, допустим, спустя неделю, браузер запрашивает опять тот же самый объект через прокси-сервер. Так как объект на веб-сервере мог за про­шедшую неделю измениться, прокси-сервер производит проверку ак­туальности его версии, вставляя в запрос условный оператор GET. Это выглядит следующим образом:

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

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

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

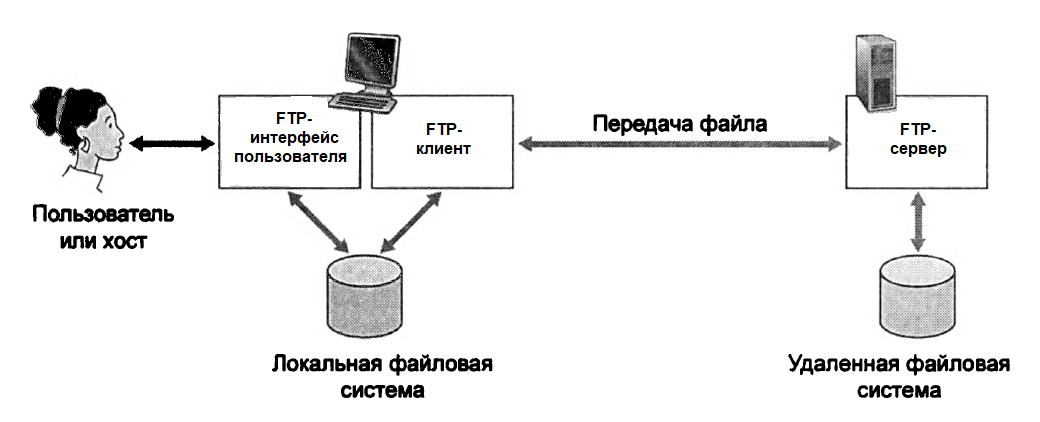
НТТР/1.1 304 Not Modified Date: Sat, 15 Oct 2011 15:39:29 Server: Apache/1.3.0 (Unix) (пустое тело сообщения)

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

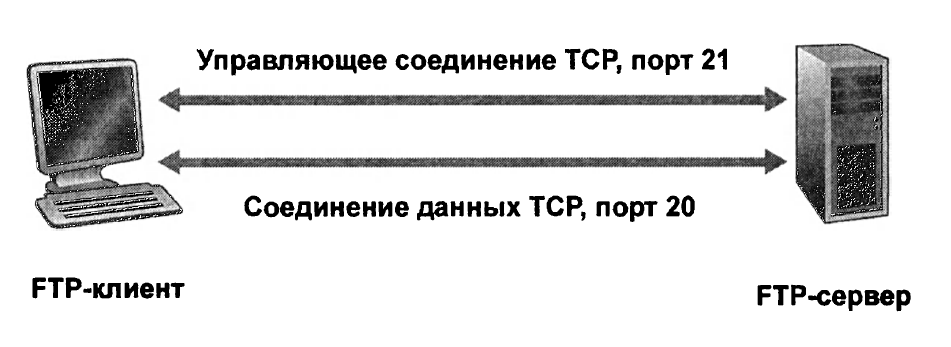
На этом мы завершаем обсуждение протокола HTTP (протокола прикладного уровня), первого из протоколов Интернета, рассмотрен­ного нами подробно. Мы познакомились с форматом HTTP-сообщений и с действиями, предпринимаемыми клиентом и веб-сервером при от­правке и получении сообщений. Мы также коснулись инфраструктуры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

**Передача файлов по протоколу FTP**

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

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

HTTP и FTP являются протоколами передачи данных и имеют мно­го общего, например, они оба работают поверх протокола TCP Однако эти два протокола прикладного уровня имеют и очень существенные различия: протокол FTP использует для передачи файла два параллель­ных TCP-соединения: **управляющее соединение** и **соединение данных.** Управляющее соединение используется для отправки контрольной ин­формации между двумя хостами — такой, как идентификатор пользова­теля, пароль, команды для смены каталога, а также команды передачи и получения файлов. Соединение данных используется для передачи самого файла. Говорят, что FTP отправляет управляющую информацию **вне полосы** (out-of-band). Протокол HTTP, как вы помните, отсылает строки заголовков запроса и ответа в одном и том же ТСР-соединении вместе с передаваемым файлом. Поэтому говорят, что протокол HTTP отправляет управляющую информацию **внутри полосы** (in-band). Соединение данных и управляющее соединение отображены на рис. 2.15.

****

**Рис. 2.15** Управляющее соединение и соединение данных

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

В течение пользовательского сеанса FTP-сервер должен хранить со­стояние этого сеанса, а именно серверу нужно как-то связывать управляющее соединение с конкретной пользовательской учетной записью, сервер должен отслеживать текущий каталог пользователя по мере того, как он пробегает по дереву каталогов. Такое отслеживание состояния каждого пользовательского сеанса значительно ограничивает общее число одновременно обслуживаемых сеансов FTP. Напомним, что HTTP, наоборот, является протоколом, не сохраняющим состояние сеанса пользователя.

**Команды и ответы протокола FTP**

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

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

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

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

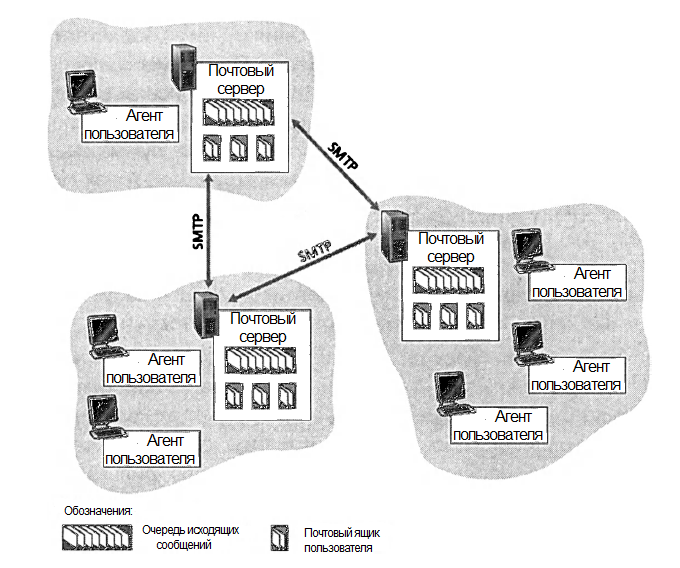
* 331 Username OK, password required
* 125 Data connection already open; transfer starting
* 425 Can't open data connection
* 452 Error writing file

**Электронная почта в Интернете**

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

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

На рис. 2.16 представлена почтовая система Интернета. Мы видим из этой диаграммы, что она включает три основных компонента: **поль­зовательский агент, почтовые серверы** и **простой протокол передачи почты** (Simple Mail Transfer Protocol, **SMTP).** Сейчас мы опишем каж­дый из этих компонентов с точки зрения отправителя, Алисы, которая направляет сообщение электронной почты получателю — Бобу. Поль­зовательские агенты позволяют создавать и прочитывать сообщения, отвечать на них, перенаправлять и сохранять. Примерами таких агентов электронной почты являются Microsoft Outlook и Apple Mail. Когда Алиса закончила составлять письмо, ее пользовательский агент отправляет это сообщение на ее почтовый сервер, где оно помещается в очередь исходящих сообщений. Когда Боб собирается прочитать письмо, его пользовательский агент извлекает сообщение из его почтового ящика, находящегося на почтовом сервере Боба.



**Рис.** 2.16 Система электронной почты Интернета

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

**ИСТОРИЯ**

**Веб-почта**

В декабре 1995 года, спустя несколько лет после того, как люди узнали о Всемирной паутине, Джек Смит (Jack Smith) и Сабир Батиа (Sabeer Bhatia) обратились в известную венчурную компанию Draper Fisher Jurvetson с предложением разработать бесплатную систему электронной почты с доступом через веб-интерфейс. Их идея со­стояла в том, чтобы предоставить любому желающему бесплатную учетную запись электронной почты и сделать ее доступной через веб-интерфейс. В обмен на 15% капитала будущей компании Draper Fisher Jurvetson профинансировали Смита и Батиа, которые основали компанию Hotmail. Имея трех штатных сотрудников и 14 человек, работающих на временной основе, компания сумела разработать и запустить почтовую службу к июлю 1996 года. Спустя месяц после запуска служба Hotmail уже насчитывала 100 000 подписчиков, а в декабре 1997 года их число увеличилось до 12 000 000. Вскоре после этого ее приобрела компания Microsoft. Как сообщается, сумма сделки была приблизительно равна 400 миллионам долларов. Успех Hotmail часто связывают с тем, что она имела преимущество «первопроходца» и разработала свой собственный «вирусный маркетинг» (возможно, некоторые из читающих эту книгу студентов окажутся в числе новых предпринимателей, которым предстоит в будущем запустить иннова­ционные Интернет-службы с помощью этой методики).

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

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

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

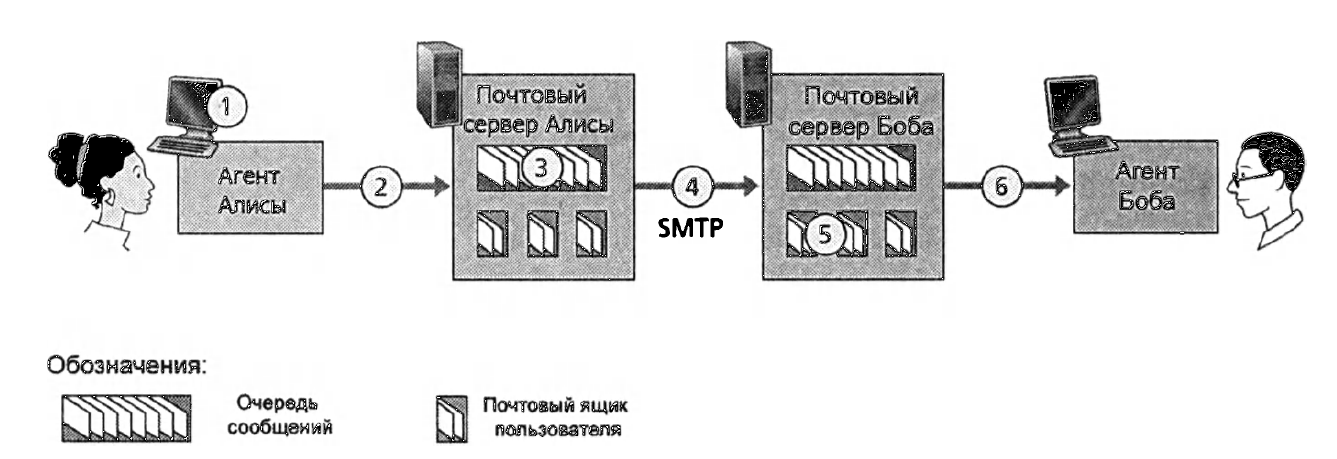
**Протокол SMTP**

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

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

1. Алиса запускает свой пользовательский агент электронной по­чты, указывает ему электронный адрес Боба (например, bob@[someschool.edu](http://someschool.edu)), создает сообщение и дает команду пользовательскому агенту его отправить.
2. Пользовательский агент Алисы отправляет сообщение на ее почто­вый сервер, где оно помещается в очередь.
3. Клиентская часть протокола SMTP, запущенная на почтовом сервере Алисы, видит сообщение в очереди и открывает ТСР-соединение с SMTP сервером, запущенном на почтовом сервере Боба.
4. После первоначального SMTP-рукопожатия клиент SMTP отправ­ляет сообщение Алисы в установленное ТСР-соединение.
5. На почтовом сервере Боба серверная часть SMTP принимает сооб­щение, которое затем помещается в почтовый ящик Боба.
6. Боб запускает свой пользовательский почтовый агент, чтобы прочи­тать сообщение в удобное ему время.

Данная схема проиллюстрирована на рис. 2.17.

**Рис. 2.17. Отправка сообщения от Алисы к Бобу**

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

Теперь давайте подробнее рассмотрим, как SMTP передает сообще­ние от отправляющего сервера к принимающему. Мы увидим, что про­токол SMTP имеет много общего с протоколами, используемыми при не­посредственном взаимодействии друг с другом двух человек. Во-первых, клиент SMTP (запущенный на почтовом сервере отправителя) устанав­ливает соединение с сервером SMTP (запущенном на почтовом сервере получателя) на порту 25. Если сервер недоступен, клиент пытается соединиться еще раз. Как только соединение устанавливается, клиент и сервер обмениваются рукопожатиями прикладного уровня — так же, как люди обычно приветствуют друг друга перед обменом информацией, SMTP-клиенты и серверы представляются друг другу перед передачей данных. В течение этой фазы рукопожатия клиент SMTP указывает по­чтовый адрес отправителя (того, кто генерирует сообщение) и почтовый адрес получателя. Как только клиент и сервер SMTP представились друг другу, клиент отправляет сообщение. SMTP может полагаться на службу TCP, которая надежно и без ошибок доставит данные на сервер. Затем клиент повторяет этот процесс в рамках того же самого ТСР-соединения, если у него есть еще одно сообщение для отправки серверу; в противном случае он дает команду протоколу TCP закрыть соединение.

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

С: 220 [hamburger.edu](http://hamburger.edu)

К: HELO [crepes.fr](http://crepes.fr)

С: 250 Hello [crepes.fr](http://crepes.fr), pleased to meet you

К: MAIL FROM: <[alice@crepes.fr](mailto:alice@crepes.fr)>

C: 250 [alice@crepes.fr](mailto:alice@crepes.fr) ... Sender ok

K: RCPT TO: <[bob@hamburger.edu](mailto:bob@hamburger.edu)>

C: 250 [bob@hamburger.edu](mailto:bob@hamburger.edu) ... Recipient ok

K: DATA

C: 354 Enter mail, end with "." on a line by itself

К: Вы любите кетчуп?

К: Как насчет анчоусов?

К: .

С: 250 Message accepted for delivery

К: QUIT

С: 221 [hamburger.edu](http://hamburger.edu) closing connection

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

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

telnet serverName 25

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

**SMTP-команд: HELO, MAIL FROM, RCPT TO, DATA,**

CRLF.CRLF и QUIT.

**Сравнение с протоколом HTTP**

Давайте произведем краткое сравнение протоколов SMTP и HTTP. Оба протокола используются для передачи файлов с одного хоста на другой: HTTP передает файлы (или объекты) от веб-сервера веб-клиенту (обычно с помощью браузера); SMTP передает файлы (то есть сообщения электронной почты) от одного почтового сервера к другому. При передаче файлов оба протокола используют постоянное соединение. В этом они похожи. Однако, существует и значительная разница между ними: во-первых, протокол HTTP является **протоколом получения (pull-протокол)** — пользователи применяют его, чтобы получить информацию с сервера, которую туда кто-то загрузил, в любой удобный им момент. В частности, TCP-соединение инициируется компьютером, который хочет получить файл. С другой стороны, протокол SMTP является **протоколом отправки (push-протокол)** — сервер-источник отправляет файл почтовому серверу-приемнику. В этом случае ТСР-соединение инициируется компьютером, который хочет отправить файл.

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

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

**Форматы почтового сообщения**

Когда Алиса отправляет обычную почтовую открытку Бобу, она пи­шет на конверте адрес Боба, свой собственный адрес, дату и так далее — сюда она может включить любую сопроводительную информацию, которую посчитает нужным добавить. Аналогично при отправлении электронной почты от одного адресата к другому такая же сопроводи­тельная информация содержится в заголовке, предваряющем тело са­мого сообщения. Эта информация обычно содержится в наборе строк заголовка, которая определяется документом RFC 5322. Заголовочные строки и тело сообщения разделяются пустой строкой (то есть возвра­том каретки и переводом строки). В документе RFC 5322 указан точный формат для строк заголовка электронной почты, а также их смысловое значение. Как и в случае с HTTP, каждая строка заголовка содержит обычный читаемый текст, состоящий из ключевого слова, после которого следует двоеточие и значение. Некоторые ключевые слова обязательные, а некоторые — нет. Каждый заголовок должен иметь строку From: и строку То:, может включать строку Subject: и другие необязательные строки. Важно отметить, что эти строки заголовка *отличаются* от SMTP-команд, изученных ранее (даже несмотря на то, что они содержат такие же слова, как From и То). Команды, которые рассматривались в том разделе, составляли часть рукопожатия протокола SMTP; строки заголовка в этом случае являются частью самого почтового сообщения.

Типичный заголовок сообщения выглядит следующим образом:

From: alicegcrepes.fr То: [bob@hamburger.edu](mailto:bob@hamburger.edu) Subject: Поиск смысла жизни.

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

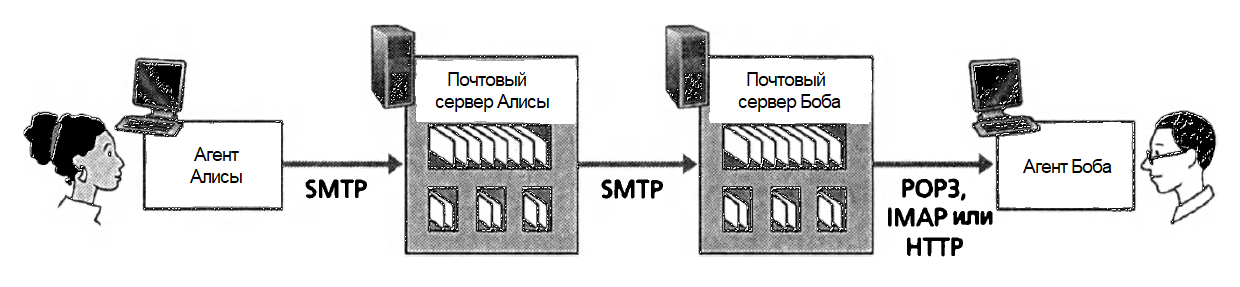
**Протоколы доступа к электронной почте**

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

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

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

Осталось одно недостающее звено в этой головоломке! Каким об­разом получатель, такой как Боб, имеющий пользовательский почтовый агент на своем локальном компьютере, будет получать свои сообщения, которые размещены на почтовом сервере его Интернет-провайдера? Заметим, что пользовательский агент Боба не может использовать SMTP, чтобы получить эти сообщения, так как SMTP — это push-протокол (протокол отправки, а не получения). Ключом к этой задаче будет специальный протокол доступа к почте, который передает сообщения с почтового сервера Боба на его локальное устройство. Существует несколько популярных протоколов доступа к почте, включая **РОРЗ (Post Office Protocol, протокол почтового отделения, версия 3), IMAP (Internet Mail Access Protocol, протокол доступа к почте через Интернет)** и HTTP.



**Рис. 2.18. Протоколы электронной почты и их взаимодействующие компоненты**

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

**Протокол РОРЗ**

Протокол доступа к электронной почте РОР3, описанный в RFC 1939, является достаточно простым. Его простота определяет отно­сительно скромную функциональность. РОР3 начинает работать, когда агент пользователя (клиент) открывает TCP-соединение с почтовым сервером по порту 110. После установления соединения работа прото­кола РОР3 проходит три состояния: авторизация, транзакция и обнов­ление. Во время первой фазы работы протокола (авторизации) агент пользователя отправляет учетные данные (имя и пароль) для аутенти­фикации его на сервере. Во второй фазе (транзакции) пользовательский агент получает сообщение, а также может помечать сообщения для уда­ления, удалять эти пометки, запрашивать почтовую статистику. Фаза обновления наступает после того, как клиент дал команду quit, завершая сеанс РОР3. В этот момент почтовый сервер удаляет помеченные сообщения и закрывает соединение.

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

На этапе авторизации существует две основных команды: user <username> и pass <password>. Для демонстрации этих команд за­пустите Telnet, чтобы соединиться напрямую с РОР3-сервером, исполь­зуя порт 110, и наберите эти команды. Предположим, что mailServer — это имя вашего почтового сервера. У вас выйдет что-то похожее на следующее:

telnet mailServer 110

+ОК РОРЗ server ready

user bob

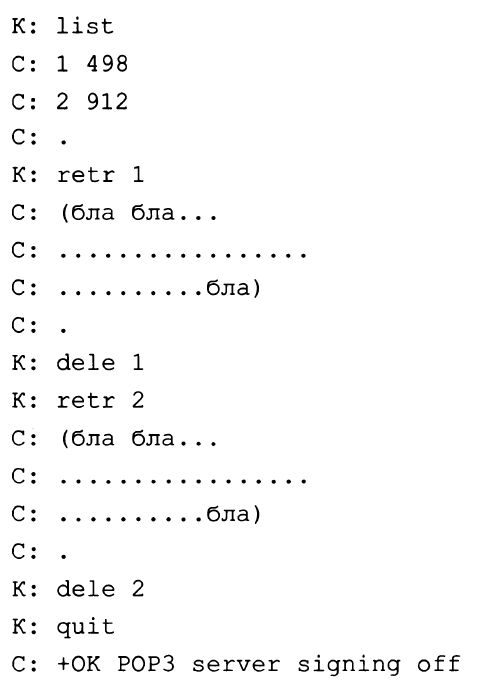
+OK

pass hungry

+OK user successfully logged on

Если вы ошибетесь в наборе команды, сервер РОР3 выдаст ответ с сообщением — ERR.

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



Сначала пользовательский агент запрашивает почтовый сервер вывести список с размером всех хранящихся в ящике сообщений, затем агент читает и удаляет каждое сообщение с сервера. Отметим, что после фазы авторизации пользовательский агент использует только 4 команды: list, retr, dele и quit, синтаксис которых определен документом RFC 1939. После обработки команды quit РОР3-сервер переходит в фазу обновления и удаляет сообщения 1 и 2 из почтового ящика пользователя.

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

Во время РОР3-сеанса между пользовательским агентом и почтовым сервером РОР3-сервер обрабатывает некоторую информацию: в частности, он отслеживает, какие сообщения пользователь пометил для удаления. Однако состояние РОР3-сеанса сервер не сохраняет, и это значительно упрощает реализацию РОР3-серверов.