Базовые протоколы TCP/IP

Кафедра ИБ БГАРФ

Зензин Александр Степанович, к.т.н.

Обзор

1. Протоколы транспортного уровня TCP и UDP
   - 1.1 Порты и сокеты
   - 1.2. Протокол UDP и UDP-дейтаграммы
   - 1.3. Протокол TCP и TCP-сегменты
   - 1.4. Логические соединения — основа надежности TCP
   - 1.5. Повторная передача и скользящее окно
   - 1.6. Реализация метода скользящего окна в протоколе TCP
   - 1.7. Управление потоком
3. Общие свойства и классификация протоколов маршрутизации
4. Протокол RIP
   - 4.1. Построение таблицы маршрутизации
   - 4.2. Адаптация маршрутизаторов RIP к изменениям состояния сети
   - 4.3. Пример зацикливания пакетов
   - 4.4. Методы борьбы с ложными маршрутами в протоколе RIP
4. Протокол OSPF
   - 4.1. Два этапа построения таблицы маршрутизации
   - 4.2. Метрики

Базовые протоколы TCP/IP

Протоколы TCP и UDP исполняют посредническую роль между приложениями и транспортной инфраструктурой сети. В то время как задачей уровня межсетевого взаимодействия, к которому относится протокол IP, является передача данных между сетевыми интерфейсами в составной сети, главная задача транспортного уровня, которую решают протоколы TCP и UDP, заключается в передаче данных между прикладными процессами, выполняющимися на компьютерах в сети.

Протоколы маршрутизации, в отличие от сетевых протоколов, таких как IP и IPX, не являются обязательными, так как таблица маршрутизации может строиться администратором сети вручную. Однако в крупных сетях со сложной топологией и большим количеством альтернативных маршрутов протоколы маршрутизации выполняют очень важную и полезную работу, автоматизируя построение таблиц маршрутизации, а также отыскивая новые маршруты при изменениях сети: отказах или появлении новых линий связи и маршрутизаторов.

Протокол ICMP является средством оповещения отправителя о причинах недоставки его пакетов адресату. Помимо диагностики ICMP используется для мониторинга сети. Так, в основе популярных утилит мониторинга IP-сетей ping и traceroute лежат ICMP-сообщения.

Протоколы транспортного уровня TCP и UDP
Порты и сокеты

К транспортному уровню стека TCP/IP относятся:

- протокол управления передачей (Transmission Control Protocol, TCP), описанный в стандарте RFC 793;
- протокол пользовательских дейтаграмм (User Datagram Protocol, UDP), описанный в стандарте RFC 768.

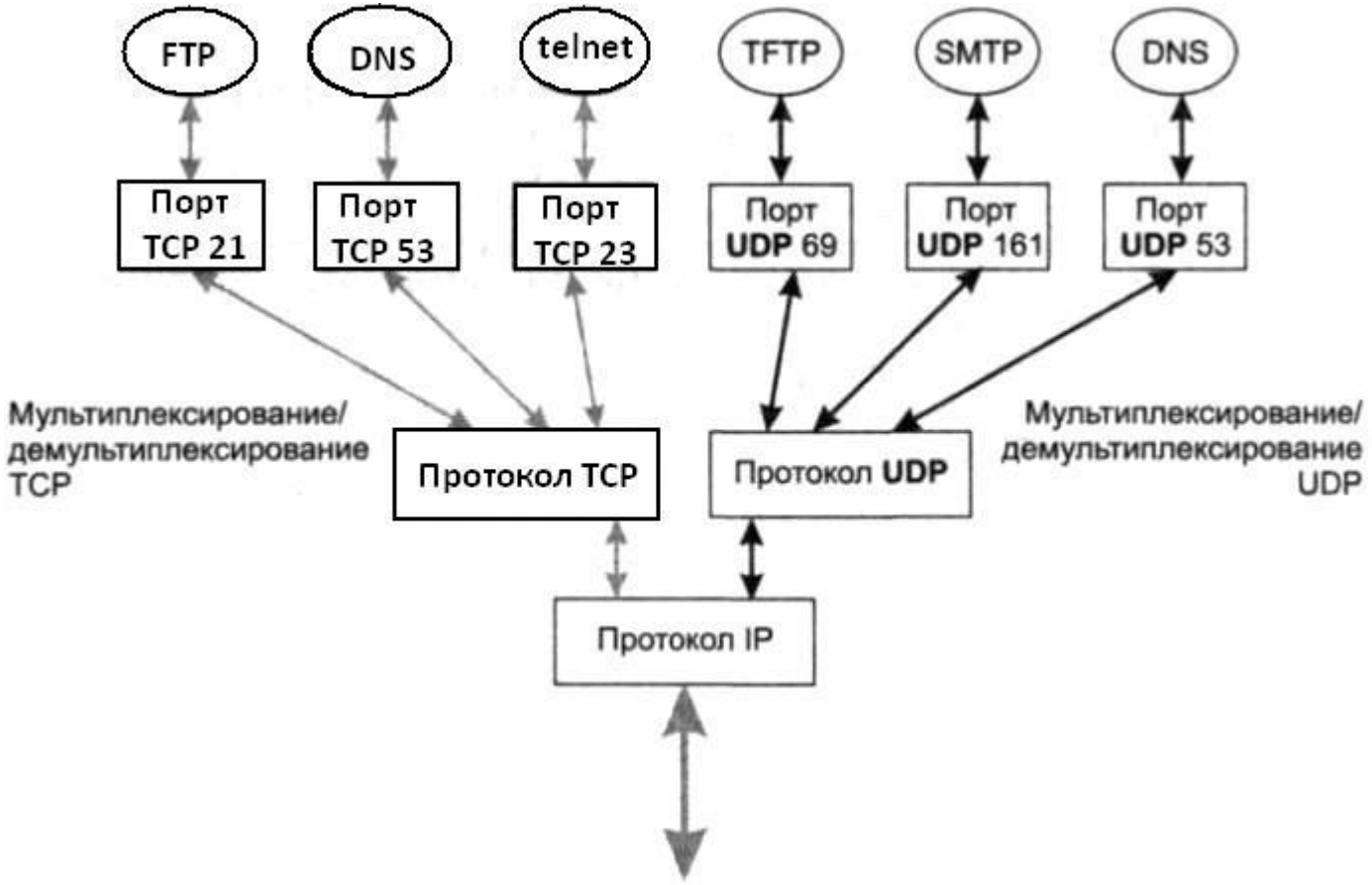
Протоколы TCP и UDP, как и протоколы прикладного уровня, устанавливаются на конечных узлах.

В то время как задачей уровня сетевого взаимодействия, к которому относится протокол IP, является передача данных между сетевыми интерфейсами в составной сети, главная задача протоколов транспортного уровня TCP и UDP заключается *в передаче данных между прикладными процессами*, выполняющимися на компьютерах в сети.

Каждый компьютер может выполнять несколько процессов, более того, даже отдельный прикладной процесс может иметь несколько точек входа, выступающих в качестве адресов назначения для пакетов данных. Поэтому доставка данных на сетевой интерфейс компьютера-получателя — это еще не конец пути, так как данные необходимо переправить конкретному процессу-получателю. Процедура распределения протоколами TCP и UDP поступающих от сетевого уровня пакетов между прикладными процессами называется **демультиплексированием** (см. рис ниже).

Протоколы транспортного уровня TCP и UDP
Порты и сокеты

Мультиплексирование и демультиплексирование на транспортном уровне.



Существует и обратная задача: данные, генерируемые разными приложениями, работающими на одном конечном узле, должны быть переданы общему для всех них протокольному модулю IP для последующей отправки в сеть. Эту работу, называемую **мультиплексированием**, тоже выполняют протоколы TCP и UDP.

Протоколы транспортного уровня TCP и UDP
Порты и сокеты

Протоколы TCP и UDP ведут для каждого приложения две системные очереди: очередь данных, поступающих к приложению из сети, и очередь данных, отправляемых этим приложением в сеть. Такие системные очереди называются **портами**, причем входная и выходная очереди одного приложения рассматриваются как один порт. Для идентификации портов им присваивают номера.

Если процессы представляют собой популярные системные службы, такие как FTP, telnet, HTTP, TFTP, DNS и т. п., то за ними закрепляются **стандартные назначенные номера**, называемые также **хорошо известными** (well-known) номерами портов. Эти номера закрепляются и публикуются в стандартах Интернета (RFC 1700, RFC 3232). Так, номер 21 закреплен за серверной частью службы удаленного доступа к файлам FTP, а 23 — за серверной частью службы удаленного управления telnet. Назначенные номера из диапазона от 0 до 1023 являются *уникальными в пределах Интернета* и закрепляются за приложениями *централизованно*.

Для тех приложений, которые еще не стали столь распространенными, номера портов назначаются локально разработчиками этих приложений или операционной системой в ответ на поступление запроса от приложения. На каждом компьютере операционная система ведет список занятых и свободных номеров портов. При поступлении запроса от приложения, выполняемого на данном компьютере, операционная система выделяет ему первый свободный номер. Такие номера называют **динамическими**. В дальнейшем все сетевые приложения должны адресоваться к данному приложению с указанием назначенного ему динамического номера порта. После того как приложение завершит работу, его номер возвращается в список свободных и может быть назначен другому приложению.

Протоколы транспортного уровня TCP и UDP
Порты и сокеты

Динамические номера являются уникальными в пределах каждого компьютера, но при этом обычной является ситуация совпадения номеров портов приложений, выполняемых на разных компьютерах. Как правило, клиентские части известных приложений (DNS, WWW, FTP, telnet и др.) получают динамические номера портов от ОС.

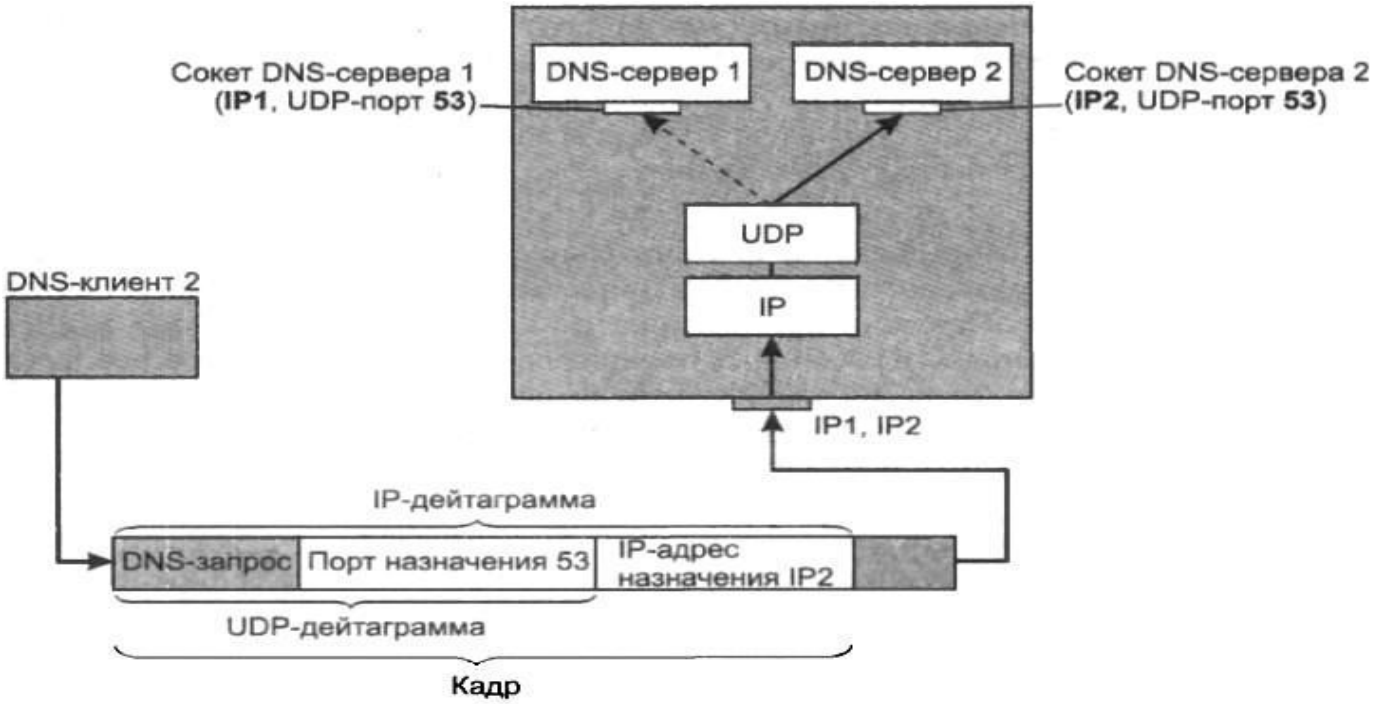
Все, что было сказано о портах, в равной степени относится к обоим протоколам транспортного уровня (TCP и UDP). В принципе, нет никакой зависимости между назначением номеров портов для приложений, использующих протокол TCP, и приложений, работающих с протоколом UDP. Приложения, которые передают данные на уровень IP по протоколу UDP, получают номера, называемые UDP-портами. Аналогично, приложениям, обращающимся к протоколу TCP, выделяются TCP-порты.

В том и другом случаях это могут быть как назначенные, так и динамические номера. Диапазоны чисел, из которых выделяются номера TCP- и UDP-портов, совпадают: от 0 до 1023 для назначенных и от 1024 до 65 535 для динамических. Однако никакой связи между назначенными номерами TCP- и UDP-портов нет. Даже если номера TCP- и UDP-портов совпадают, они идентифицируют разные приложения. Например, одному приложению может быть назначен TCP-порт 1750, а другому — UDP-порт 1750. В некоторых случаях, когда приложение может обращаться по выбору к протоколу TCP или UDP (например, таким приложением является DNS), ему, исходя из удобства запоминания, назначаются совпадающие номера TCP- и UDP-портов (в данном примере — это хорошо известный номер 53).

Стандартные назначенные номера портов уникально идентифицируют тип приложения (FTP, или HTTP, или DNS и т. д.), однако они не могут использоваться для однозначной идентификации прикладных процессов, связанных с каждым из этих типов приложений.

Протоколы транспортного уровня TCP и UDP
Порты и сокеты

Пусть, например, на одном хосте запущены две копии DNS-сервера — DNS-сервер 1, DNS-сервер 2. Демультиплексирование протокола UDP на основе сокетов представлено на рисунке.

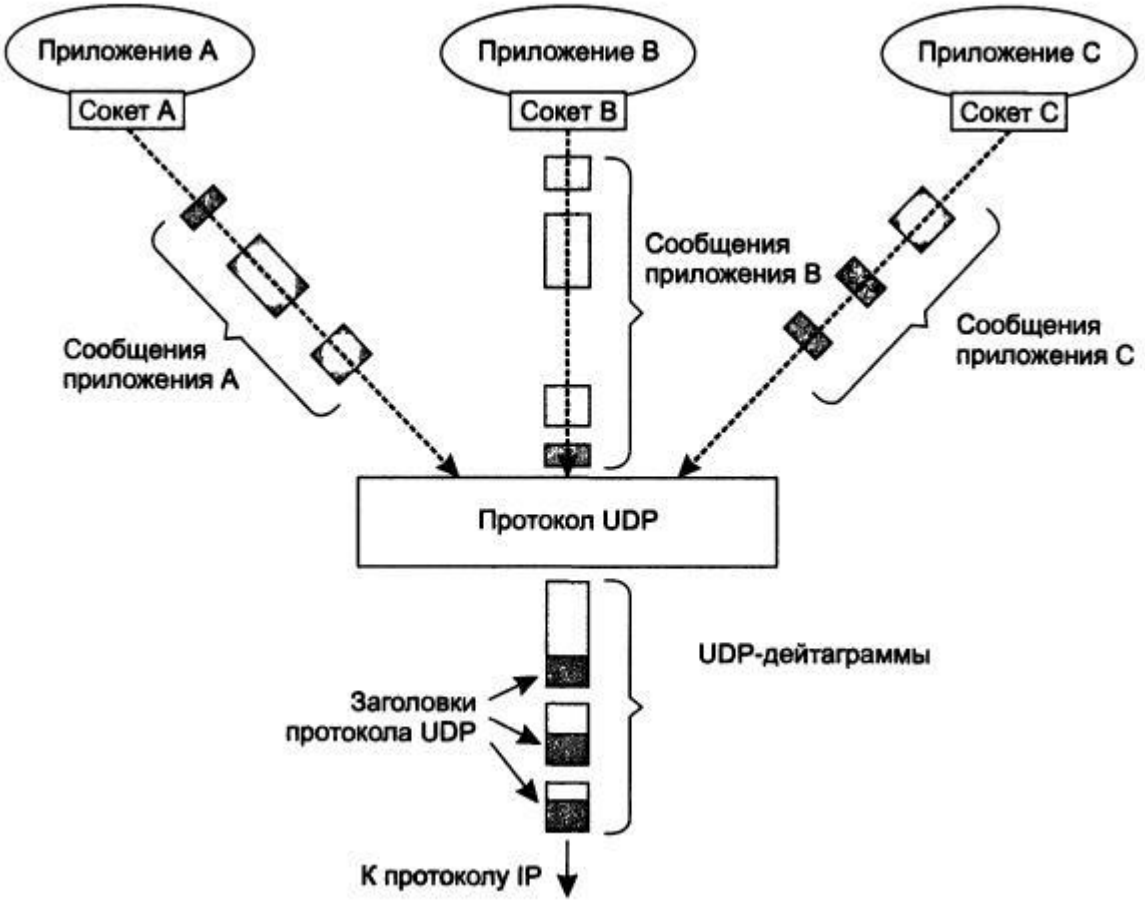


Каждый из этих DNS-серверов имеет хорошо известный UDP-порт 53. Какому из этих серверов нужно было бы направить запрос клиента, если бы в DNS-запросе в качестве идентификатора сервера был указан только номером порта?

Чтобы снять неоднозначность в идентификации приложений, разные копии связываются с разными IP-адресами. Для этого сетевой интерфейс компьютера, на котором выполняется несколько копий приложения, должен иметь соответствующее число IP-адресов - на рисунке это IP1 и IP2. Во всех IP-пакетах, направляемых DNS-серверу 1, в качестве IP-адреса указывается IP1, а DNS-серверу 2 — адрес IP2. (UDP, TCP - сокеты).

Протокол UDP и UDP - дейтаргаммы

При работе на хосте-отправителе данные от приложений поступают протоколу UDP через порт в виде сообщений (см. рисунок ниже). Протокол UDP добавляет к каждому отдельному сообщению свой 8-байтный заголовок, формируя из этих сообщений собственные протокольные единицы, называемые **UDP-дейтаграммами**, и передает их нижележащему протоколу IP. В этом и заключаются его функции по *мультиплексированию* данных.



Протокол UDP и UDP - дейтаргаммы

Каждая дейтаграмма переносит отдельное пользовательское сообщение. Сообщения могут иметь различную длину, не превышающую однако длину поля данных протокола IP, которое, в свою очередь, ограничено размером кадра технологии нижнего уровня. Поэтому если буфер UDP переполняется, то сообщение приложения отбрасывается.

Заголовок UDP состоит из четырех 2-байтных полей:

- номер UDP-порта отправителя;
- номер UDP-порта получателя;
- контрольная сумма;
- длина дейтаграммы.

Далее приведен пример заголовка UDP с заполненными полями:

Source Port - 0x0035
Destination Port - 0x0411
Total length - 132 (0x84) bytes
Checksum = 0x5333

В этой UDP-дейтаграмме в поле данных, длина которого, как следует из заголовка, равна (132 - 8) байт, помещено сообщение DNS-сервера, что можно видеть по номеру порта источника (Source Port = 0x0035). В шестнадцатеричном формате это значение равно стандартному номеру порта DNS-сервера — 53.

Судя по простоте заголовка, протокол UDP не сложен. Действительно, его функции сводятся к простой передаче данных между прикладным и сетевым уровнями, а также примитивному контролю искажений в передаваемых данных. При контроле искажений протокол UDP только диагностирует, но не исправляет ошибку. Если контрольная сумма показывает, что в поле данных UDP-дейтаграммы произошла ошибка, протокол UDP просто отбрасывает поврежденную дейтаграмму.

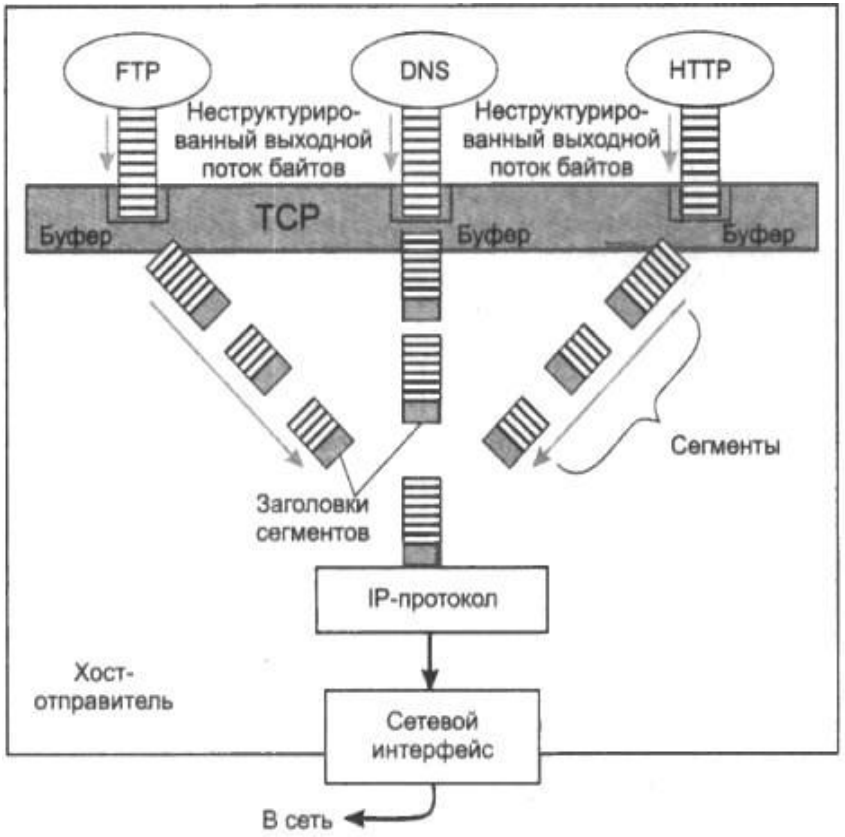
Протокол UDP и UDP – дейтаргаммы
Протокол TCP и TCP-сегменты

Работая на хосте-получателе, протокол UDP принимает от протокола IP извлеченные из пакетов UDP-дейтаграммы. Полученные из IP-заголовка IP-адрес назначения и из UDP-заголовка номер порта используются для формирования UDP-сокета, однозначно идентифицирующего приложение, которому направлены данные. Протокол UDP освобождает дейтаграмму от UDP-заголовка. Полученное в результате сообщение он передает приложению на соответствующий UDP-сокет. Таким образом, протокол UDP выполняет демультиплексирование на основе сокетов.

Протокол TCP и TCP-сегменты

При работе на хосте-отправителе протокол TCP рассматривает информацию, поступающую к нему от прикладных процессов, как неструктурированный поток байтов. Поступающие данные буферизуются средствами TCP. Для передачи на сетевой уровень из буфера «вырезается» некоторая непрерывная часть данных, которая называется **сегментом** и снабжается заголовком.

В отличии от UDP, который создает свои дейтаграммы на основе логически обособленных единиц данных – сообщений, генерируемых приложениями, протокол TCP делит поток данных на сегменты без учета их смысла или внутренней структуры.



Протокол TCP и TCP - сегменты

Заголовок TCP-сегмента содержит значительно больше полей, чем заголовок UDP, что отражает более развитые возможности протокола TCP. Формат заголовка представлен на рисунке.

Протокол TCP и TCP - сегменты

Краткие описания большинства полей помещены на рисунке, а более подробно мы их рассмотрим, когда будем изучать функции протокола TCP.

Коротко поясним значение однобитных полей, называемых флагами, или кодовыми битами (code bits). Они расположены сразу за резервным полем и содержат служебную информацию о типе данного сегмента. Положительное значение сигнализируется установкой этих битов в единицу:

- URG — срочное сообщение;
- ACK — квитанция на принятый сегмент;
- PSH — запрос на отправку сообщения без ожидания заполнения буфера;
- RST — запрос на восстановление соединения;
- SYN — сообщение, используемое для синхронизации счетчиков переданных данных при установлении соединения;
- FIN — признак достижения передающей стороной последнего байта в потоке передаваемых данных.

Логические соединения — основа надежности TCP

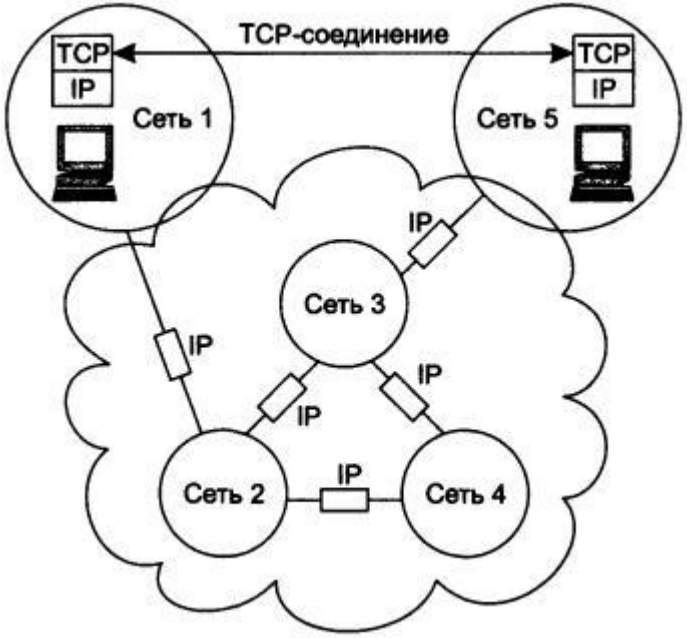
Основным отличием TCP от UDP является то, что на протокол TCP возложена дополнительная задача — обеспечить надежную доставку сообщений, используя в качестве основы ненадежный дейтаграммный протокол IP.

Для решения этой задачи протокол TCP использует метод продвижения данных с установлением логического соединения. Как было сказано ранее, логическое соединение дает возможность участникам обмена следить за тем, чтобы данные не были потеряны, искажены или продублированы, а также чтобы они пришли к получателю в том порядке, в котором были отправлены.

Протокол TCP устанавливает логические соединения между *прикладными процессами*, причем в каждом соединении участвуют только два процесса. TCP-соединение является *дуплексным*, то есть каждый из участников этого соединения может одновременно получать и отправлять данные.

На рисунке показаны сети, соединенные маршрутизаторами, на которых установлен протокол IP. Установленные на конечных узлах протокольные модули TCP решают задачу обеспечения надежного обмена данными путем установления между собой **логических соединений**.

При установлении логического соединения модули TCP договариваются между собой о параметрах процедуры обмена данными.



Логические соединения — основа надежности TCP

В протоколе TCP каждая сторона соединения посылает противоположной стороне следующие параметры:

- максимальный размер сегмента, который она готова принимать;
- максимальный объем данных (возможно несколько сегментов), которые она разрешает другой стороне передавать в свою сторону, даже если та еще не получила квитанцию на предыдущую порцию данных (размер окна);
- начальный порядковый номер байта, с которого она начинает отсчет потока данных в рамках данного соединения.

В результате переговорного процесса модулей TCP с двух сторон соединения определяются параметры соединения. Одни из них остаются постоянными в течение всего сеанса связи, а другие адаптивно изменяются. В частности, в зависимости от загрузки буфера принимающей стороны, а также надежности работы сети динамически изменяется размер окна отправителя.

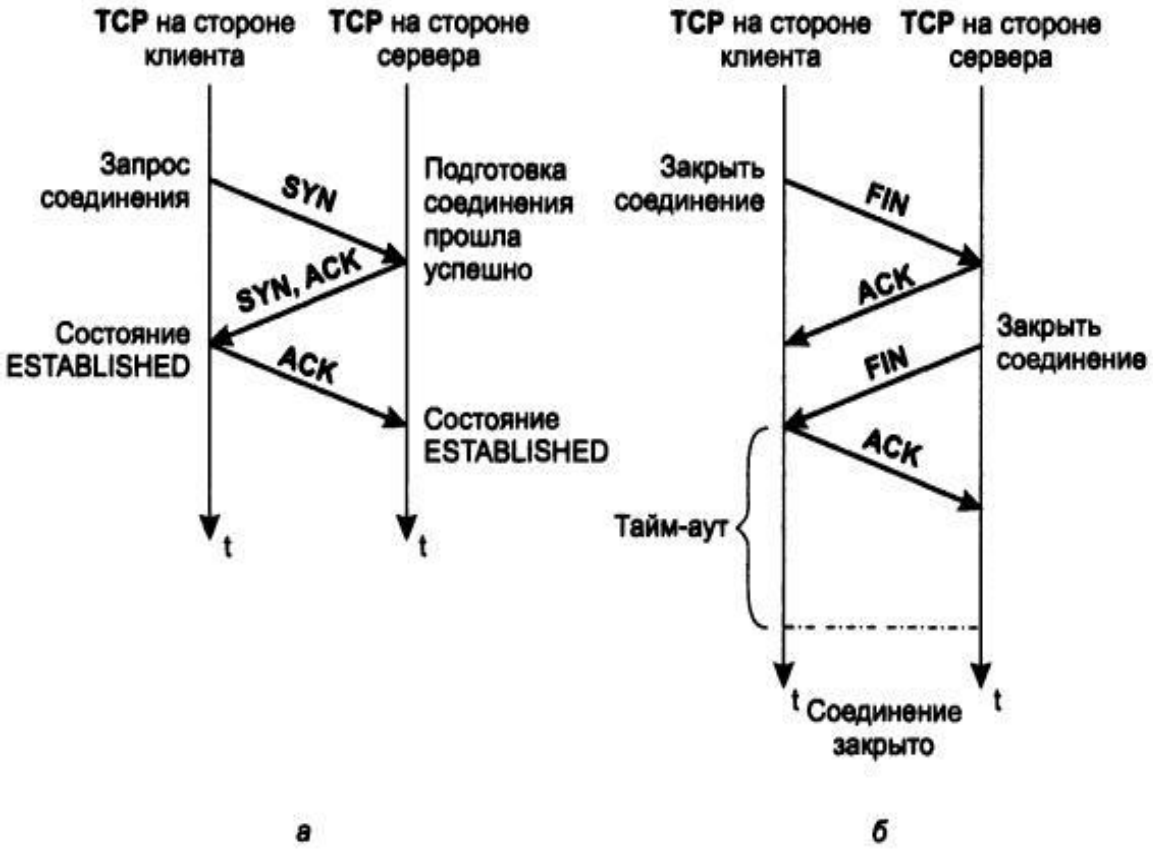
Рассмотрим процедуру установления и разрыва логического соединения при нормальном течении процесса (см следующий рисунок).

Соединение устанавливается по инициативе клиентской части приложения. При необходимости выполнить обмен данными с серверной частью приложение-клиент обращается к нижележащему протоколу TCP, который в ответ на это обращение посылает сегмент-запрос на установление соединения протоколу TCP, работающему на стороне сервера (рис. 2, а). В числе прочего в запросе содержится флаг SYN, установленный в 1.

Логические соединения — основа надежности TCP

Получив запрос, модуль TCP на стороне сервера пытается создать «инфраструктуру» для обслуживания нового клиента. Он обращается к операционной системе с просьбой о выделении определенных системных ресурсов для организации буферов, таймеров, счетчиков. Эти ресурсы закрепляются за соединением с момента создания и до момента разрыва. Если на стороне сервера все необходимые ресурсы были получены и все необходимые действия выполнены, то модуль TCP посылает клиенту сегмент с флагами ACK и SYN.

В ответ клиент посылает сегмент с флагом ACK и переходит в состояние установленного логического соединения (состояние ESTABLISHED). Когда сервер получает флаг ACK, он также переходит в состояние ESTABLISHED. На этом процедура установления соединения заканчивается, и стороны могут переходить к обмену данными.



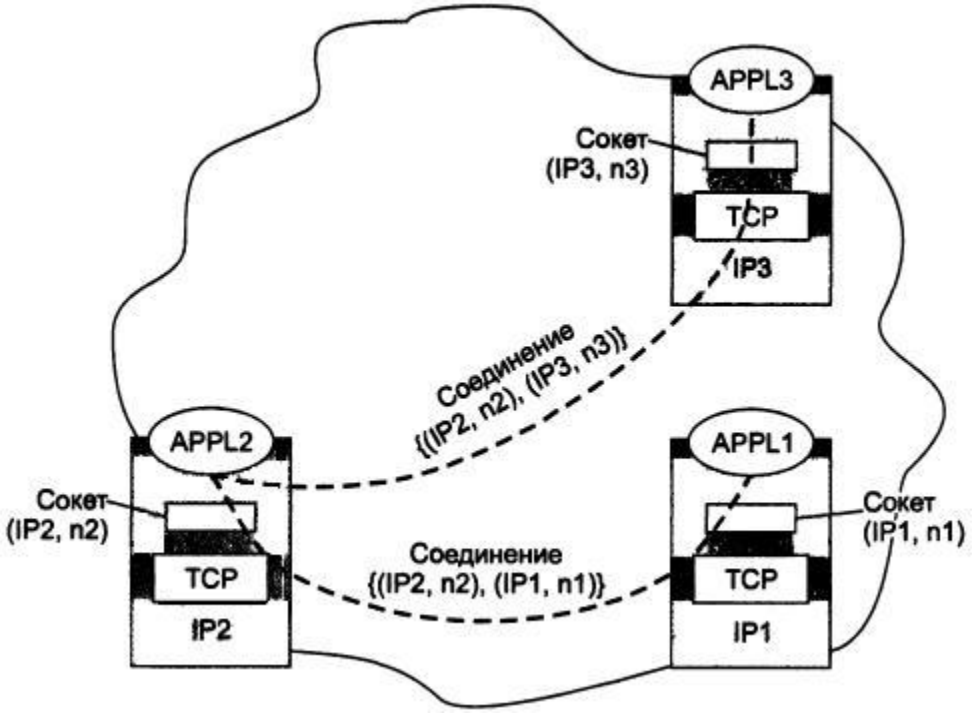
Логические соединения — основа надежности TCP

Соединение может быть разорвано в любой момент по инициативе любой стороны. Для этого клиент и сервер должны обменяться сегментами FIN и АСК, в последовательности, показанной на рис. 1, б (здесь инициатором является клиент). Соединение считается закрытым по прошествии некоторого времени, в течение которого сторона-инициатор убеждается, что ее завершающий сигнал АСК дошел нормально и не вызвал никаких «аварийных» сообщений со стороны сервера. (Это очень упрощенно, но логическое TCP – соединение однозначно идентифицируется парой сокетов, определенных для этого соединения двумя действующими процессами).

Сокет одновременно может участвовать *в нескольких соединениях*. На рисунке показаны три компьютера с адресами IPI, IP2, IP3.

На каждом компьютере выполняется по одному приложению — APPL1, APPL2 и APPL3, сокеты которых — соответственно (IP1, nl), (IP2, n2), (IP3, п3), а номера TCP-портов приложений — n1 , п2, п3.

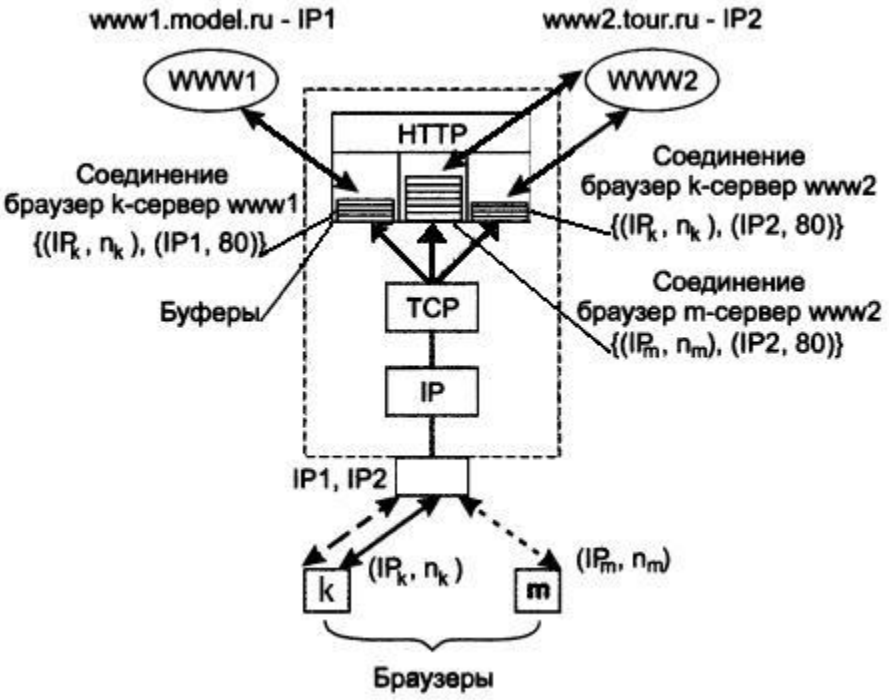
На рисунке показаны два логических соединения, которое установило приложение 2 с приложением 1 и приложением 3. Логические соединения идентифицируются как {(IP2, n2), (IP1, nl)} и {(IP2, n2), (IP3, п3)} соответственно. Мы видим, что в обоих соединениях участвует *один и тот же сокет* — (IP2, п2).



Логические соединения — основа надежности TCP

А теперь рассмотрим на примере, как протокол TCP выполняет демультиплексирование. Пусть некий поставщик услуг оказывает услугу по веб-хостингу, то есть на его компьютере клиенты могут разворачивать свои веб-серверы. Веб-сервер основан на протоколе прикладного уровня HTTP, который передает свои сообщения в TCP-сегментах. Модуль TCP ожидает запросы от веб-клиентов (браузеров), «прослушивая» хорошо известный порт 80.

На рисунке показан вариант хостинга с двумя веб-серверами — сервером www1.model.ru, имеющим IP-адрес IP1, и сервером www2.tour.ru с адресом IP2. К каждому из них может обращаться множество клиентов, причем клиенты могут одновременно работать как с сервером www1, так и с сервером www2. Для каждой пары клиент-сервер протоколом TCP создается отдельное логическое соединение.



На рисунке показаны два браузера, имеющие соответственно сокеты (IP_k, n_k) и (IP_m, n_m).Пользователь браузера k обращается одновременно к серверам WWW1 и WWW2.

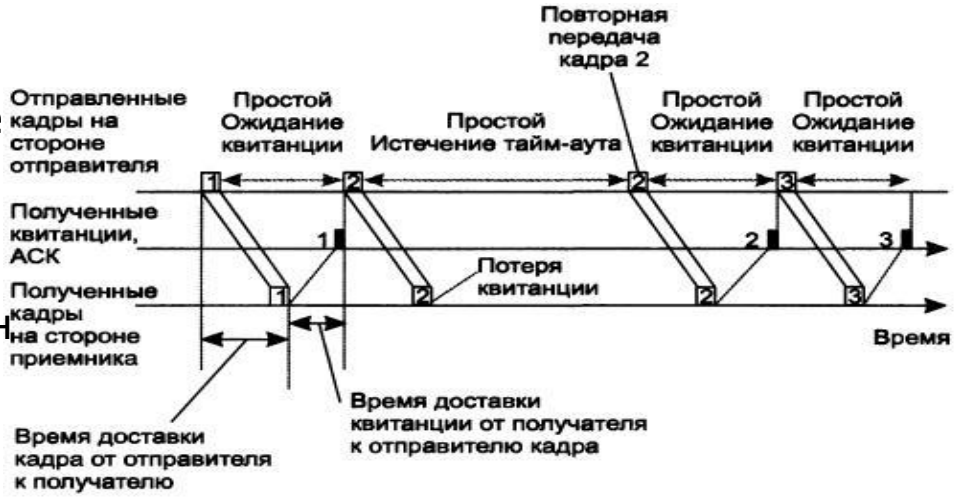
Наличие отдельных соединений для работы с каждым из этих серверов обеспечивает не только надежную доставку, но и разделение информационных потоков — у пользователя никогда не возникает вопроса, каким сервером ему была послана та или иная страница. Одновременно с пользователем браузера k с сервером WWW2 работает пользователь браузера m (ЛС изолируют их потоки). Количество буферов = числу клиентов.

Повторная передача и скользящее окно

Один из наиболее естественных приемов, используемых для организации надёжной передачи — это квитирование. Отправитель отсылает данные и ждет, пока к нему не придет квитанция, подтверждающая, что его данные благополучно дошли до адресата. В протоколе TCP используется частный случай квитирования — алгоритм скользящего окна. Прежде чем перейти к подробному рассмотрению особенностей реализации этого алгоритма в протоколе TCP, очень полезно обсудить его с общих позиций.

Итак, существует два метода организации процесса обмена квитанциями: метод простоя источника и метод скользящего окна.

Метод **простоя** источника требует, чтобы источник, пославший кадр (в данном случае не имеет значения, какое название используется для единицы передаваемых данных), дожидался от приемника квитанции, извещающей о том, что исходный кадр получен и данные в нем корректны, и только после этого посылал следующий кадр (или повторял искаженный).



Если же квитанция в течение тайм-аута не пришла, то кадр (или квитанция) считается утерянным и его передача повторяется. На рисунке показано, что второй кадр отсылается только после того, как пришла квитанция, подтверждающая доставку первого кадра. Однако затем произошла длительная пауза в отправке следующего третьего кадра. В течение этой паузы источник был вынужден повторить передачу кадра 2, так как квитанция на первую его копию была потеряна. Понятно, что при таком алгоритме работы источника принимающая сторона должна уметь распознавать дублирующиеся кадры и избавляться от них.

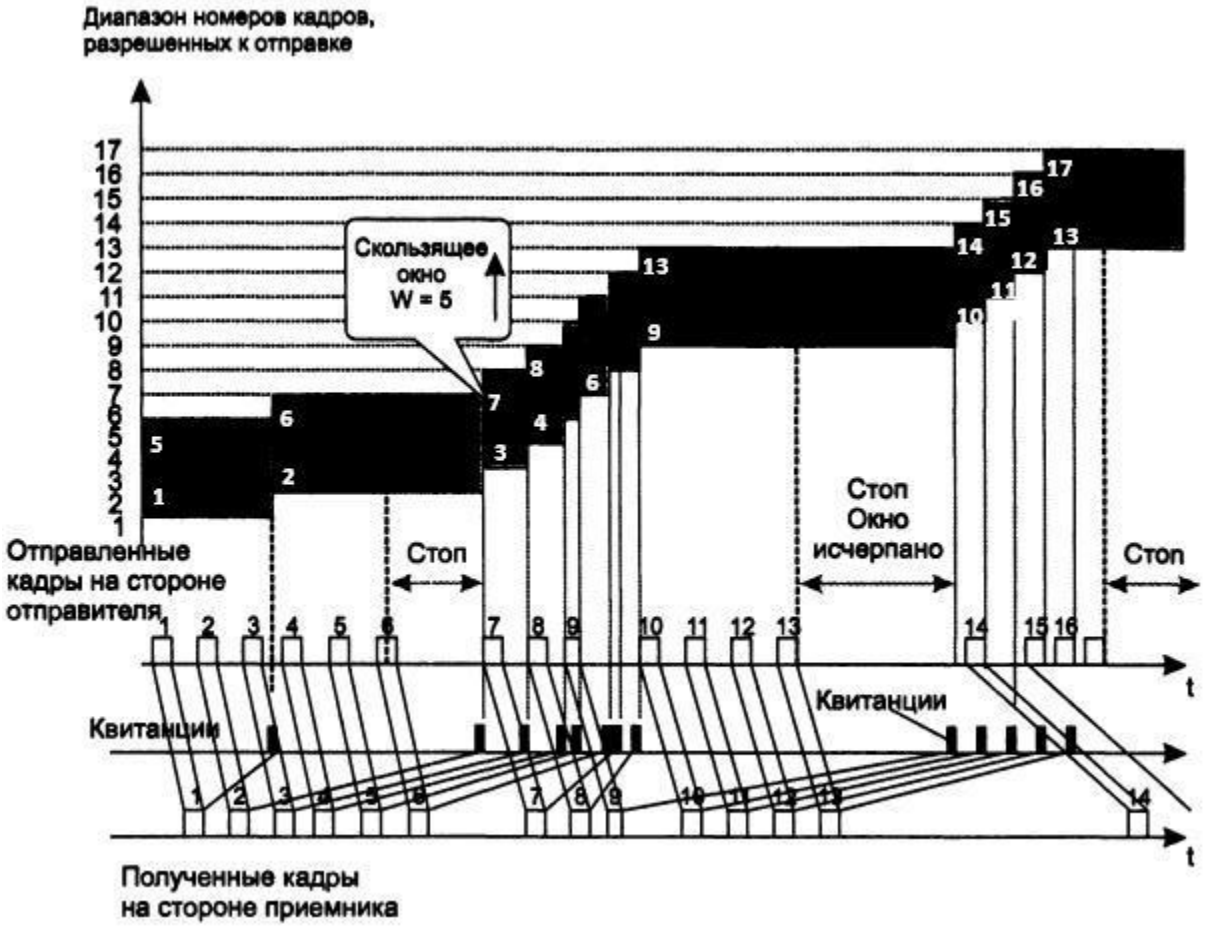
Повторная передача и скользящее окно

Достаточно очевидно, что при использовании данного метода производительность обмена данными ниже потенциально возможной — передатчик мог бы посылать следующий кадр сразу же после отправки предыдущего, но он обязан ждать прихода квитанции.

Второй метод называется методом **скользящего окна** (sliding window).

В этом методе для повышения скорости передачи данных источнику разрешается передать некоторое количество кадров в непрерывном режиме, то есть в максимально возможном для источника темпе еще до получения на эти кадры квитанций. Количество кадров, которые разрешается передавать таким образом, называется **размером окна**.

Рисунок иллюстрирует применение данного метода для окна размером 5 кадров.



Повторная передача и скользящее окно

В начальный момент, когда еще не послано ни одного кадра, окно определяет диапазон номеров кадров от 1 до 5 включительно. Источник начинает передавать кадры и через какое-то время получать в ответ квитанции. Для простоты предположим, что квитанции поступают в той же последовательности (но не обязательно в том же темпе), что и кадры, которым они соответствуют. В момент получения отправителем квитанции 1 окно сдвигается на одну позицию вверх, определяя новый диапазон разрешенных к отправке кадров (от 2 до 6).

Процессы отправки пакетов и получения квитанций идут достаточно независимо друг от друга. В нашем примере отправитель продолжает передавать кадры, но некоторое время не получает на них квитанции. После передачи кадра 6 окно исчерпывается, и источник приостанавливает передачу.

После получения квитанции 2 (на кадр 2) окно сдвигается вверх на единицу, определяя диапазон разрешенных к передаче кадров от 3 до 7. Аналогичное «скольжение» окна вверх происходит после получения каждой квитанции: окно сдвигается вверх на 1, но его размер при этом не меняется и остается равным 5. После прихода квитанции 8 окно оказывается в диапазоне от 9 до 13 и остается таковым достаточно долго, так как по каким-то причинам источник перестает получать подтверждения о доставке кадров. Отправив последний разрешенный кадр 13, передатчик снова прекращает передачу с тем, чтобы возобновить ее после прихода квитанции 9.

При отправке кадра в источнике устанавливается тайм-аут. Если за установленное время квитанция на отправленный кадр не придет, то кадр (или квитанция на него) считается утерянным, и кадр передается снова. Если же поток квитанций поступает регулярно в пределах допуска в 5 кадров, то скорость обмена достигает максимально возможной величины для данного канала и принятого протокола.

Реализация метода скользящего окна в протоколе TCP

Алгоритм скользящего окна в протоколе TCP имеет некоторые существенные особенности. В частности, в рассмотренном обобщенном алгоритме скользящего окна единицей передаваемых данных является кадр, и размер окна также определяется в кадрах, в то время как в протоколе TCP дело обстоит совсем по-другому.

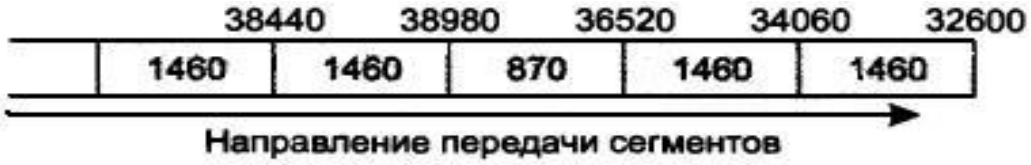
Хотя единицей передаваемых данных протокола TCP является сегмент (аналог кадра), окно определено на множестве нумерованных байтов неструктурированного потока данных, передаваемого приложением протоколу TCP.

В ходе переговорного процесса модули TCP обоих участвующих в обмене сторон договариваются между собой о параметрах процедуры обмена данными. Одни из них остаются постоянными в течение всего сеанса связи, другие в зависимости, например, от интенсивности трафика и/или размеров буферов адаптивно изменяются. Одним из таких параметров является начальный номер байта, с которого будет вестись отсчет в течение всего функционирования данного соединения. У каждой стороны свой начальный номер. Нумерация байтов в пределах сегмента осуществляется, начиная от заголовка.

Когда отправитель посылает TCP-сегмент, он помещает в поле последовательного номера номер первого байта данного сегмента, который служит идентификатором сегмента.

Реализация метода скользящего окна в протоколе TCP

На рисунке ниже показаны четыре сегмента размером 1460 байт и один — 870 байт. Идентификаторами этих сегментов являются номера 32600, 34060, 35520 и т. д. На основании этих номеров получатель TCP-сегмента не только отличает данный сегмент от других, но и позиционирует полученный фрагмент относительно общего потока байтов. Кроме того, он может сделать вывод, например, что полученный сегмент является дубликатом или что между двумя полученными сегментами пропущены данные и т. д.



В качестве квитанции получатель сегмента отсылает ответное сообщение (сегмент), в поле подтвержденного номера которого он помещает число, на единицу превышающее максимальный номер байта в полученном сегменте. Так, для первого отправленного сегмента, изображенного на рисунке, квитанцией о получении (подтвержденным номером) будет число 34060, для второго — 35520 и т. д. Подтвержденный номер часто интерпретируют не только как оповещение о благополучной доставке, но и как номер следующего ожидаемого байта данных.

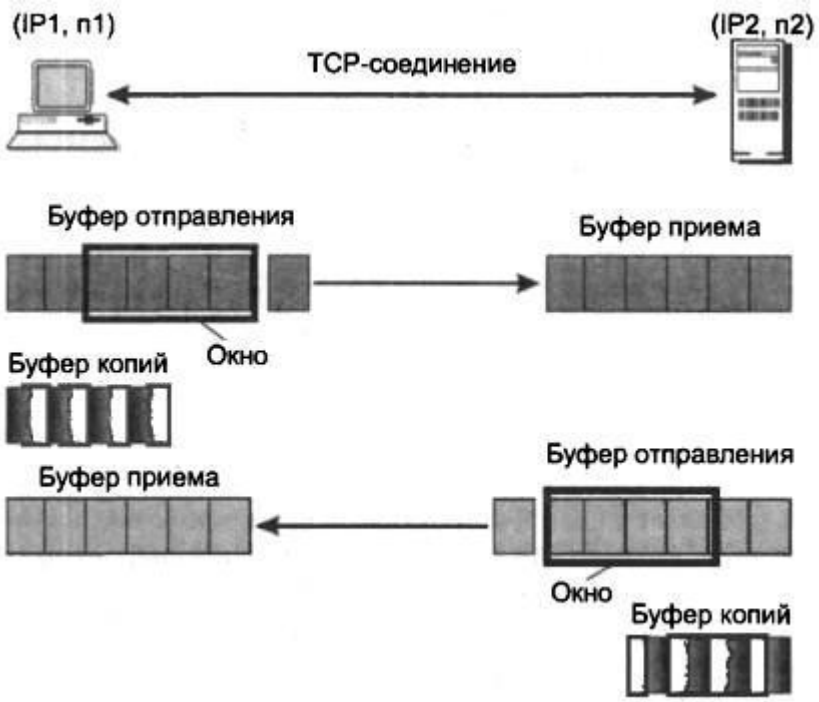
Квитанция в протоколе TCP посылается только в случае правильного приема данных. Таким образом, отсутствие квитанции означает либо потерю сегмента, либо потерю квитанции, либо прием искаженного сегмента.

В соответствии с определенным форматом один и тот же TCP-сегмент может нести в себе как пользовательские данные (в поле данных), так и квитанцию (в заголовке), которой подтверждается получение данных от другой стороны.

Реализация метода скользящего окна в протоколе TCP

Поскольку протокол TCP является дуплексным, каждая сторона одновременно выступает и как отправитель, и как получатель. У каждой стороны есть пара буферов: один — для хранения принятых сегментов, другой — для сегментов, которые только еще предстоит отправить. Кроме того, имеется буфер для хранения копий сегментов, которые были отправлены, но квитанции о получении которых еще не поступили.

И при установлении соединения, и в ходе передачи обе стороны, выступая в роли получателя, посылают друг другу так называемые окна приема. Каждая из сторон, получив окно приема, «узнает», сколько байтов ей разрешается отправить с момента получения последней квитанции. Другими словами, посылая окна приема, обе стороны пытаются регулировать поток байтов в свою сторону, сообщая своему «визави», какое количество байтов (начиная с номера байта, о котором уже была выслана квитанция) они готовы в настоящий момент принять.



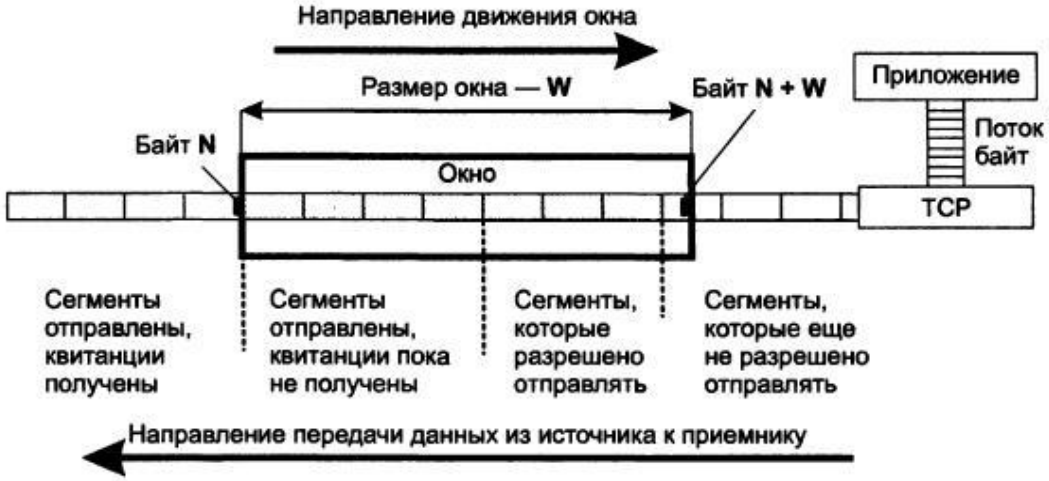
Система буферов TCP-соединения

Реализация метода скользящего окна в протоколе TCP

На рисунке ниже показан поток байтов, поступающий от приложения в выходной буфер модуля TCP. Из потока байтов модуль TCP «нарезает» последовательность сегментов и поочередно отправляет их приложению-получателю. Для определенности на рисунке принято направление перемещения данных справа налево. В этом потоке можно указать несколько логических границ:

- Первая граница отделяет сегменты, которые уже были отправлены и на которые уже пришли квитанции. Последняя квитанция пришла на байт с номером N.
- По другую сторону этой границы располагается окно размером IF байт. Часть байтов, входящих в окно, составляют сегменты, которые также уже отправлены, но квитанции на которые пока не получены.
- Оставшаяся часть окна — это сегменты, которые пока не отправлены, но могут быть отправлены, так как входят в пределы окна.
- И наконец, последняя граница указывает на начало последовательности сегментов, ни один из которых не может быть отправлен до тех пор, пока не придет очередная квитанция и окно не будет сдвинуто вправо.

Если размер окна равен W, а последняя по времени квитанция содержала значение N, то отправитель может посылать новые сегменты до тех пор, пока в очередной сегмент не попадет байт с номером N + W. Этот сегмент выходит за рамки окна, и передачу в таком случае необходимо приостановить до прихода следующей квитанции.



Реализация метода скользящего окна в протоколе TCP

Получатель может послать квитанцию, подтверждающую получение сразу нескольких сегментов, если они образуют непрерывный поток байтов. Например, (см. рис., а), если в буфер, плотно без пропусков заполненный потоком байтов до 2354 включительно, поочередно поступили сегменты (2355-3816), (3817-5275) и (5276-8400), где цифры в скобках означают номера первых и последних байтов каждого сегмента, то получателю достаточно отправить только одну квитанцию на все три сегмента, указав в ней в качестве номера квитанции значение 8401. Таким образом, процесс квитирования в TCP является *накопительным*.

Вполне возможны ситуации, когда сегменты приходят к получателю не в том порядке, в котором были посланы, то есть в приемном буфере может образоваться «прогалина» (рис., б). Пусть, к примеру, после указанных ранее трех сегментов вместо следующего по порядку сегмента (8401-10566) пришел сегмент (10567-12430). Очевидно, что послать в качестве номера квитанции значение 12431 нельзя, потому что это бы означало, что получены все байты вплоть до 12430. Поскольку в потоке байтов образовался разрыв, получатель может только еще раз повторить квитанцию 8401, говоря тем самым, что все еще ожидает поступления потока байтов, начиная с 8401, то есть подтверждает получение не отдельных блоков данных, а непрерывной последовательности байтов.

Накопительный принцип квитирования: а — плотное заполнение буфера (в момент t4 передается квитанция на байт 8401), б — неплотное заполнение буфера (в момент t5 снова передается квитанция на байт 8401)

Реализация метода скользящего окна в протоколе TCP

Когда протокол TCP передает в сеть сегмент, он «на всякий случай» помещает его копию в буфер, называемый также очередью повторной передачи, и запускает таймер. Когда приходит квитанция на этот сегмент, соответствующая копия удаляется из очереди. Если же квитанция не приходит до истечения срока, то сегмент, вернее его копия, посылается повторно. Может случиться так, что копия сегмента придет тогда, когда исходный сегмент уже окажется на месте, тогда дубликат попросту отбрасывается.

Управление потоком

Какой размер окна должен назначить источник приемнику, и наоборот? Точнее, каким на каждой из сторон должно быть выбрано время ожидания (тайм-аут) очередной квитанции? От ответа на этот вопрос зависит производительность протокола TCP.

При выборе величины *тайм-аута* должны учитываться скорость и надежность линий связи, их протяженность и многие другие факторы. Тайм-аут не должен быть слишком коротким, чтобы по возможности исключить избыточные повторные передачи, снижающие полезную пропускную способность системы, но он не должен быть и слишком длинным, чтобы избежать длительных простоев, связанных с ожиданием несуществующей или «заблудившейся» квитанции.

В протоколе TCP тайм-аут определяется с помощью достаточно сложного *адаптивного алгоритма*, идея которого состоит в следующем. При каждой передаче засекается время от момента отправки сегмента до прихода квитанции о его приеме (время оборота). Получаемые значения времени оборота усредняются с весовыми коэффициентами, возрастающими от предыдущего замера к последующему. Это делается с тем, чтобы усилить влияние последних замеров. В качестве тайм-аута выбирается среднее время оборота, умноженное на некоторый коэффициент. Практика показывает, что значение этого коэффициента должно превышать 2. В сетях с большим разбросом времени оборота при выборе тайм-аута учитывается и дисперсия этой величины.

Размер окна приема связан с наличием в данный момент места в буфере данных у принимающей стороны. Поэтому в общем случае окна приема на разных концах соединения имеют разный размер. Например, можно ожидать, что сервер, вероятно обладающий большим буфером, пошлет клиентской станции окно приема большее, чем клиент серверу. В зависимости от состояния сети то одна, то другая стороны могут объявлять новые значения окон приема, динамически уменьшая и увеличивая их.

Управление потоком

Варьируя величину окна, можно влиять на загрузку сети. Чем больше окно, тем большая порция неподтвержденных данных может быть послана в сеть. Но если пришло большее количество данных, чем может быть принято модулем TCP, данные отбрасываются. Это ведет к излишним пересылкам информации и ненужному росту нагрузки на сеть и модуль TCP.

В то же время окно малого размера может ограничить передачу данных скоростью, которая определяется временем путешествия по сети каждого посылаемого сегмента. Чтобы избежать применения малых окон, в некоторых реализациях TCP предлагается получателю данных откладывать реальное изменение размеров окна до тех пор, пока свободное место не составит 20-40 % от максимально возможного объема памяти для этого соединения. Но и отправителю не стоит спешить с посылкой данных, пока окно принимающей стороны не станет достаточно большим. Учитывая эти соображения, разработчики протокола TCP предложили схему, согласно которой при установлении соединения заявляется большое окно, но впоследствии его размер существенно уменьшается. Существуют и другие прямо противоположные алгоритмы настройки окна, когда вначале выбирается минимальное окно, а затем, если сеть справляется с предложенной нагрузкой, его размер резко увеличивается.

Управлять размером окна приема может не только та сторона, которая посылает это окно, чтобы регулировать поток данных в свою сторону, но и вторая сторона — потенциальный отправитель данных. Если вторая сторона фиксирует ненадежную работу линии связи (регулярно запаздывают квитанции, часто требуется повторная передача), то она может по собственной инициативе уменьшить окно. В таких случаях действует правило: в качестве действующего размера окна выбирается минимальное из двух значений: значения, диктуемого приемной стороной, и значения, определяемого «на месте» отправителем.

Управление потоком

Признаком перегрузки TCP-соединения является возникновение очередей на промежуточных узлах (маршрутизаторах) и на конечных узлах (компьютерах). При переполнении приемного буфера конечного узла «перегруженный» модуль TCP, отправляя квитанцию, помещает в нее новый уменьшенный размер окна. Если он совсем отказывается от приема, то в квитанции указывается окно нулевого размера. Однако даже после этого приложение может послать сообщение на отказавшийся от приема порт. Для этого сообщение должно сопровождаться указателем срочности. В такой ситуации порт обязан принять сегмент, даже если для этого придется вытеснить из буфера уже находящиеся там данные. После приема квитанции с нулевым значением окна протокол-отправитель время от времени делает контрольные попытки продолжить обмен данными. Если протокол-приемник уже готов принимать информацию, то в ответ на контрольный запрос он посылает квитанцию с указанием ненулевого размера окна.

Как видно из нашего далеко не полного описания двух протоколов транспортного уровня стека TCP/IP, на один из них — TCP — возложена сложная и очень важная задача: **обеспечение надежной передачи данных через ненадежную сеть**.

В то же время функциональная простота протокола UDP обуславливает простоту алгоритма его работы, компактность и высокое быстродействие. Поэтому те приложения, в которых реализован собственный, достаточно надежный механизм обмена сообщениями, основанный на установлении соединения, предпочитают для непосредственной передачи данных по сети использовать менее надежные, но более быстрые средства транспортировки, в качестве которых по отношению к протоколу TCP и выступает протокол UDP. Протокол UDP может применяться и тогда, когда хорошее качество линий связи обеспечивает достаточный уровень надежности и без применения дополнительных приемов наподобие установления логического соединения и квитирования передаваемых пакетов. Заметим также, что протокол TCP не годится для широковещательной и групповой рассылки.

Общие свойства и классификация протоколов маршрутизации

Протоколы маршрутизации обеспечивают поиск и фиксацию маршрутов продвижения данных через составную сеть TCP/IP Давайте остановимся на некоторых общих свойствах протоколов данного класса.

Начнем с того, что существуют такие способы продвижения пакетов в составных сетях, которые вообще не требуют наличия таблиц маршрутизации на маршрутизаторах.

Наиболее простым способом передачи пакетов по сети является так называемая *лавинная маршрутизация*, когда каждый маршрутизатор передает пакет всем своим непосредственным соседям, исключая тот, от которого его получил. Понятно, что это — не самый рациональный способ, так как пропускная способность сети используется крайне расточительно, тем не менее такой подход работоспособен (именно так мосты и коммутаторы локальных сетей поступают с кадрами, имеющими неизвестные адреса).

Еще одним видом маршрутизации, не требующим наличия таблиц маршрутизации, является маршрутизация от источника (source routing). В этом случае отправитель помещает в пакет информацию о том, какие промежуточные маршрутизаторы должны участвовать в передаче пакета к сети назначения. На основе этой информации каждый маршрутизатор считывает адрес следующего маршрутизатора, и если он действительно является адресом его непосредственного соседа, передает ему пакет для дальнейшей обработки. Вопрос о том, как отправитель узнает точный маршрут следования пакета через сеть, остается открытым. Маршрут может задавать либо вручную администратор, либо автоматически узел-отправитель, но в этом случае ему нужно поддерживать какой-либо протокол маршрутизации, который сообщит ему о топологии и состоянии сети. Маршрутизация от источника была опробована на этапе зарождения Интернета и сохранилась как практически неиспользуемая возможность протокола IPv4. В, IPv6 маршрутизация от источника является одним из стандартных режимов продвижения пакетов, существует даже специальный заголовок для реализации этого режима.

Общие свойства и классификация протоколов маршрутизации

Тем не менее большинство протоколов маршрутизации нацелено *на создание таблиц маршрутизации*.

Выбор рационального маршрута может осуществляться на основании различных критериев. Сегодня в IP-сетях применяются протоколы маршрутизации, в которых маршрут выбирается по **критерию кратчайшего расстояния**. При этом расстояние измеряется в различных *метриках.* Чаще всего используется простейшая метрика — количество **хопов**, то есть количество маршрутизаторов, которые нужно преодолеть пакету до сети назначения. В качестве метрик применяются также пропускная способность и надежность каналов, вносимые ими задержки и любые комбинации этих метрик.

Различные протоколы маршрутизации обладают *разным временем конвергенции*.

Протокол маршрутизации должен обеспечить создание на маршрутизаторах согласованных друг с другом таблиц маршрутизации, то есть таких таблиц, которые обеспечат доставку пакета от исходной сети в сеть назначения за конечное число шагов. Современные протоколы маршрутизации поддерживают согласованность таблиц, однако это их свойство не абсолютно — при изменениях в сети, например при отказе каналов передачи данных или самих маршрутизаторов, возникают периоды нестабильной работы сети, вызванной временной несогласованностью таблиц разных маршрутизаторов. Протоколу маршрутизации обычно нужно некоторое время, которое называется **временем конвергенции**, чтобы после нескольких итераций обмена служебной информацией все маршрутизаторы сети внесли изменения в свои таблицы и в результате таблицы снова стали согласованными.

Общие свойства и классификация протоколов маршрутизации

Различают протоколы, выполняющие статическую и адаптивную (динамическую) маршрутизацию.

При **статической маршрутизации** все записи в таблице имеют неизменяемый, статический статус, что подразумевает бесконечный срок их жизни. Записи о маршрутах составляются и вводятся в память каждого маршрутизатора вручную администратором сети. При изменении состояния сети администратору необходимо срочно отразить эти изменения в соответствующих таблицах маршрутизации, иначе может произойти их рассогласование, и сеть будет работать некорректно.

При **адаптивной маршрутизации** все изменения конфигурации сети автоматически отражаются в таблицах маршрутизации благодаря *протоколам маршрутизации*. Эти протоколы собирают информацию о топологии связей в сети, что позволяет им оперативно отрабатывать все текущие изменения. В таблицах маршрутизации при адаптивной маршрутизации обычно имеется информация об интервале времени, в течение которого данный маршрут будет оставаться действительным. Это время называют временем жизни (TTL) маршрута. Если по истечении времени жизни существование маршрута не подтверждается протоколом маршрутизации, то он считается нерабочим, пакеты по нему больше не посылаются.

Общие свойства и классификация протоколов маршрутизации

Протоколы адаптивной маршрутизации бывают *распределенными* и *централизованными.*

При *распределенном подходе* все маршрутизаторы сети находятся в равных условиях, они находят маршруты и строят собственные таблицы маршрутизации, работая в тесной кооперации друг с другом, постоянно обмениваясь информацией о конфигурации сети. При *централизованном подходе* в сети существует один выделенный маршрутизатор, который собирает всю информацию о топологии и состоянии сети от других маршрутизаторов. На основании этих данных выделенный маршрутизатор (который иногда называют *сервером маршрутов*) строит таблицы маршрутизации для всех остальных маршрутизаторов сети, а затем распространяет их по сети, чтобы каждый маршрутизатор получил собственную таблицу и в дальнейшем самостоятельно принимал решение о продвижении каждого пакета.

Применяемые сегодня в IP-сетях протоколы маршрутизации относятся к адаптивным распределенным протоколам, которые, в свою очередь, делятся на две группы:

- дистанционно-векторные алгоритмы (Distance Vector Algorithm, DVA);
- алгоритмы состояния связей (Link State Algorithm, ISA).

В **дистанционно-векторных алгоритмах** (DVA) каждый маршрутизатор периодически и широковещательно рассылает по сети вектор, компонентами которого являются расстояния (измеренные в той или иной метрике) от данного маршрутизатора до всех известных ему сетей. Пакеты протоколов маршрутизации обычно называют **объявлениями о расстояниях**, так как с их помощью маршрутизатор объявляет остальным маршрутизаторам известные ему сведения о конфигурации сети.

Общие свойства и классификация протоколов маршрутизации

Получив от некоторого соседа вектор расстояний (дистанций) до известных тому сетей, маршрутизатор наращивает компоненты вектора на величину расстояния от себя до данного соседа. Кроме того, он дополняет вектор информацией об известных ему самому других сетях, о которых он узнал непосредственно (если они подключены к его портам) или из аналогичных объявлений других маршрутизаторов. Обновленное значение вектора маршрутизатор рассылает своим соседям. В конце концов, каждый маршрутизатор узнает через соседние маршрутизаторы информацию обо всех имеющихся в составной сети сетях и о расстояниях до них.

Затем он выбирает из нескольких альтернативных маршрутов к каждой сети тот маршрут, который обладает наименьшим значением метрики. Маршрутизатор, передавший информацию о данном маршруте, отмечается в таблице маршрутизации как следующий (next hop). Пример - протокол RIP.

Алгоритмы состояния связей (LSA) обеспечивают каждый маршрутизатор информацией, достаточной для построения точного графа связей сети. Все маршрутизаторы работают на основании одного и того же графа, что делает процесс маршрутизации более устойчивым к изменениям конфигурации. Каждый маршрутизатор использует граф сети для нахождения оптимальных *по некоторому критерию* маршрутов до каждой из сетей, входящих в составную сеть.

Чтобы понять, в каком состоянии находятся линии связи, подключенные к его портам, маршрутизатор периодически обменивается короткими пакетами HELLO со своими ближайшими соседями. В отличие от протоколов DVA, которые регулярно передают вектор расстояний, протоколы LSA ограничиваются короткими сообщениями, а передача более объемных сообщений происходит только в тех случаях, когда с помощью сообщений HELLO был установлен факт изменения состояния какой-либо связи. Пример – протокол IS-IS, OSPF.

Протокол RIP
Построение таблицы маршрутизации

Протокол RIP (Routing Information Protocol — протокол маршрутной информации) является внутренним протоколом маршрутизации дистанционно-векторного типа.

Будучи простым в реализации, этот протокол чаще всего используется в небольших сетях. Для IP имеются две версии RIP — RIPvl и RIPv2. Протокол RIPvl не поддерживает масок. Протокол RIPv2 передает информацию о масках сетей, поэтому он в большей степени соответствует требованиям сегодняшнего дня. Так как построение таблиц маршрутизации в обеих версиях протокола принципиально не отличается, в дальнейшем для упрощения записей будет описываться работа версии 1.

Для измерения расстояния до сети стандарты протокола RIP допускают различные виды метрик: хопы, значения пропускной способности, вносимые задержки, надежность сетей (то есть соответствующие признакам D, T и R в поле качества сервиса IP-пакета), а также любые комбинации этих метрик. Метрика должна обладать свойством аддитивности — метрика составного пути должна быть равна сумме метрик составляющих этого пути. В большинстве реализаций RIP используется простейшая метрика — количество хопов, то есть количество промежуточных маршрутизаторов, которые нужно преодолеть пакету до сети назначения.

Рассмотрим процесс построения таблицы маршрутизации с помощью протокола RIP на примере составной сети, изображенной на следующем рисунке. Мы разделим этот процесс на 5 этапов.

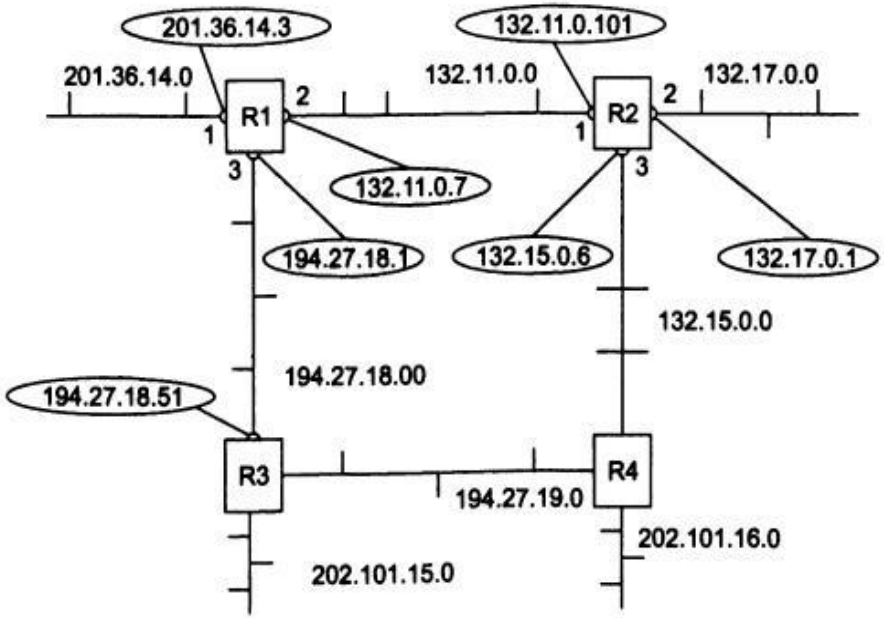
Протокол RIP
Построение таблицы маршрутизации

Этап 1 - создание минимальной таблицы. Данная составная сеть включает восемь IP-сетей, связанных четырьмя маршрутизаторами с идентификаторами: Rl, R2, R3 и R4. Маршрутизаторы, работающие по протоколу RIP, могут иметь идентификаторы, однако для протокола они не являются необходимыми. В RIP-сообщениях эти идентификаторы не передаются.

В исходном состоянии на каждом маршрутизаторе программным обеспечением стека TCP/ IP автоматически создается минимальная таблица маршрутизации, в которой учитываются только непосредственно подсоединенные сети. На рисунке адреса портов маршрутизаторов в отличие от адресов сетей помещены в овалы.

Таблица 1 позволяет оценить примерный вид минимальной таблицы маршрутизации маршрутизатора R1.

Номер сети	Адрес следующего маршрутизатора	Порт	Расстояние
201.36.14.0	201.36.14.3	1	1
132.11.0.0	132.11.0.7	2	1
194.27.18.0	194.27.18.1	3	1



Минимальные таблицы маршрутизации в других маршрутизаторах будут выглядеть соответственно, например, таблица маршрутизатора R2:

Номер сети	Адрес следующего маршрутизатора	Порт	Расстояние
132.11.0.0	132.11.0.101	1	1
132.17.0.0	132.17.0.1	2	1
132.15.0.0	132.15.0.6	3	1

Протокол RIP
Построение таблицы маршрутизации

Этап 2 — рассылка минимальной таблицы соседям. После инициализации каждый маршрутизатор начинает посылать своим соседям сообщения протокола RIP, в которых содержится его минимальная таблица. RIP-сообщения передаются в дейтаграммах протокола UDP и включают два параметра для каждой сети: ее IP-адрес и расстояние до нее от передающего сообщение маршрутизатора.

По отношению к любому маршрутизатору соседями являются те маршрутизаторы, которым данный маршрутизатор может передать IP-пакет по какой-либо своей сети, не пользуясь услугами промежуточных маршрутизаторов. Например, для маршрутизатора R1 соседями являются маршрутизаторы R2 и R3, а для маршрутизатора R4 — маршрутизаторы R2 и R3.

Таким образом, маршрутизатор R1 передает маршрутизаторам R2 и R3 следующие сообщения:

сеть 201.36.14.0, расстояние 1;
сеть 132.11.0.0, расстояние 1;
сеть 194.27.18.0, расстояние 1.

Протокол RIP
Построение таблицы маршрутизации

Этап 3 — получение RIP-сообщений от соседей и обработка полученной информации. После получения аналогичных сообщений от маршрутизаторов R2 и R3 маршрутизатор R1 наращивает каждое полученное поле метрики на единицу и запоминает, через какой порт и от какого маршрутизатора получена новая информация (адрес этого маршрутизатора станет адресом следующего маршрутизатора, если эта запись будет внесена в таблицу маршрутизации). Затем маршрутизатор начинает сравнивать новую информацию с той, которая хранится в его таблице маршрутизации.

Номер сети	Адрес следующего маршрутизатора	Порт	Расстояние
201.36.14.0	201.36.14.3	1	1
132.11.0.0	132.11.0.7	2	1
194.27.18.0	194.27.18.1	3	1
132.17.0.0	132.11.0.101	2	2
132.15.0.0	132.11.0.101	2	2
194.27.19.0	194.27.18.51	3	2
202.101.15.0	194.27.18.51	3	2
~~132.11.0.0~~	~~132.11.0.101~~	~~2~~	~~2~~
~~194.27.18.0~~	~~194.27.18.51~~	~~3~~	~~2~~

Номер сети	Адрес следующего маршрутизатора	Порт	Расстояние
201.36.14.0	201.36.14.3	1	1
132.11.0.0	132.11.0.7	2	1
194.27.18.0	194.27.18.1	3	1

Записи с четвертой по девятую получены от соседних маршрутизаторов, и они претендуют на помещение в таблицу. Однако только записи с четвертой по седьмую попадают в таблицу, а записи восьмая и девятая — нет. Это происходит потому, что они содержат данные об уже имеющихся в таблице маршрутизатора R1 сетях, а расстояние до них больше, чем в существующих записях. В результате в таблице маршрутизации о каждой сети остается только одна запись.

Аналогичные операции с новой информацией выполняют и остальные маршрутизаторы сети.

Протокол RIP
Построение таблицы маршрутизации

Этап 4 — рассылка новой таблицы соседям. Каждый маршрутизатор отсылает новое RIP-сообщение всем своим соседям. В этом сообщении он помещает данные обо всех известных ему сетях: как непосредственно подключенных, так и удаленных, о которых маршрутизатор узнал из RIP-сообщений.

Этап 5 — получение RIP-сообщений от соседей и обработка полученной информации. Этап 5 повторяет этап 3 — маршрутизаторы принимают RIP-сообщения, обрабатывают содержащуюся в них информацию и на ее основании корректируют свои таблицы маршрутизации.

Посмотрим, как это делает маршрутизатор R1:

На этом этапе маршрутизатор R1 получает от маршрутизатора R3 информацию о сети 132.15.0.0, которую тот, в свою очередь, на предыдущем цикле работы получил от маршрутизатора R4. Маршрутизатор уже знает о сети 132.15.0.0, причем старая информация имеет лучшую метрику, чем новая, поэтому новая информация об этой сети отбрасывается.

О сети 202.101.16.0 маршрутизатор R1 узнает на этом этапе впервые, причем данные о ней приходят от двух соседей — от R3 и R4. Поскольку метрики в этих сообщениях указаны одинаковые, то в таблицу попадают данные, пришедшие первыми. В нашем примере считается, что маршрутизатор R2 опередил маршрутизатор R3 и первым переслал свое RIP-сообщение маршрутизатору R1.

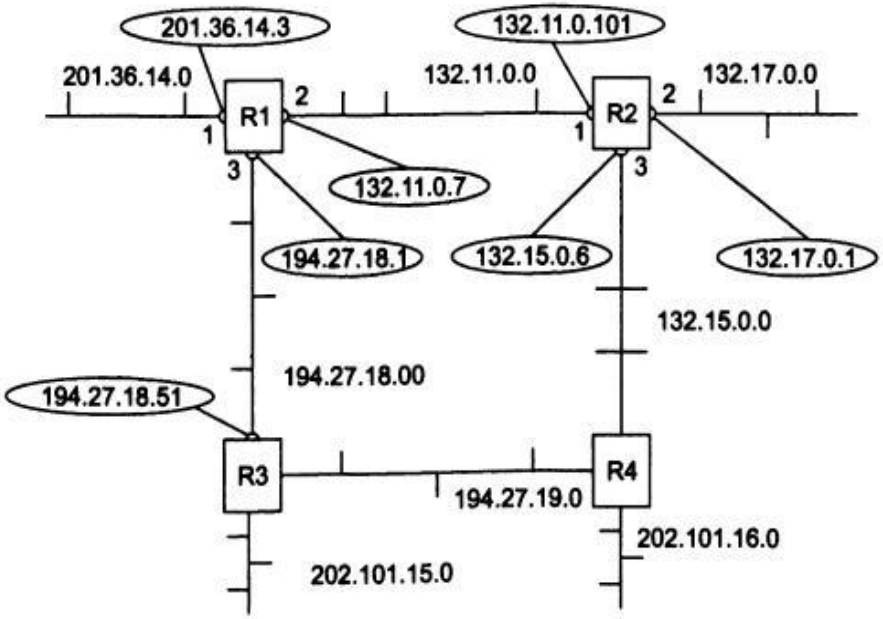
Номер сети	Адрес следующего маршрутизатора	Порт	Расстояние
201.36.14.0	201.36.14.3	1	1
132.11.0.0	132.11.0.7	2	1
194.27.18.0	194.27.18.1	3	1
132.17.0.0	132.11.0.101	2	2
132.15.0.0	132.11.0.101	2	2
132.15.0.0	194.27.18.51	3	3
194.27.19.0	194.27.18.51	3	2
104.27.10.0	132.11.0.101	2	3
202.101.15.0	194.27.18.51	3	2
202.101.16.0	132.11.0.101	2	3
202.101.16.0	104.27.18.51	3	3

Протокол RIP
Построение таблицы маршрутизации

Если маршрутизаторы периодически повторяют этапы рассылки и обработки RIP-сообщений, то за конечное время в сети установится корректный режим маршрутизации. Под корректным режимом маршрутизации здесь понимается такое состояние таблиц маршрутизации, когда все сети достижимы из любой сети с помощью некоторого рационального маршрута. Пакеты будут доходить до адресатов и не зацикливаться в петлях, подобных той, которая образуется маршрутизаторами Rl, R2, R3 и R4 (см. ниже).

Очевидно, если в сети все маршрутизаторы, их интерфейсы и соединяющие их линии связи остаются работоспособными, то объявления по протоколу RIP можно делать достаточно редко, например один раз в день. Однако в сетях постоянно происходят изменения — меняется работоспособность маршрутизаторов и линий связи, кроме того, маршрутизаторы и линии связи могут добавляться в существующую сеть или же выводиться из ее состава.

Для адаптации к изменениям в сети протокол RIP использует ряд механизмов.



Адаптация маршрутизаторов RIP к изменениям состояния сети

К новым маршрутам маршрутизаторы RIP приспосабливаются просто — они передают новую информацию в очередном сообщении своим соседям и постепенно эта информация становится известна всем маршрутизаторам сети. А вот к изменениям, связанным с потерей какого-либо маршрута, маршрутизаторы RIP адаптируются сложнее. Это связано с тем, что в формате сообщений протокола RIP нет поля, которое бы указывало на то, что путь к данной сети больше не существует.

Для уведомления о том, что некоторый маршрут недействителен, используются два механизма:

- истечение времени жизни маршрута;
- указание специального (бесконечного) расстояния до сети, ставшей недоступной.

Механизм истечения времени жизни маршрута основан на том, что каждая запись таблицы маршрутизации (как и записи таблицы продвижения моста/коммутатора), полученная по протоколу RIP, имеет время жизни (TTL). При поступлении очередного RIP-сообщения, которое подтверждает справедливость данной записи, таймер времени жизни устанавливается в исходное состояние, а затем из него каждую секунду вычитается единица. Если за время тайм-аута не придет новое сообщение об этом маршруте, он помечается как недействительный.

Время тайм-аута связано с периодом рассылки векторов по сети. В протоколе RIP период рассылки выбран равным 30 секундам, а в качестве тайм-аута выбрано шестикратное значение периода рассылки, то есть 180 секунд. Шестикратный запас времени нужен для уверенности в том, что сеть действительно стала недоступной, а не просто произошли потери RIP-сообщений (а это возможно, так как протокол RIP использует транспортный протокол UDP, который не обеспечивает надежной доставки сообщений).

Адаптация маршрутизаторов RIP к изменениям состояния сети

Если какой-либо маршрутизатор отказывает, переставая слать своим соседям сообщения о сетях, которые можно достичь через него, то через 180 секунд все записи, порожденные этим маршрутизатором, у его ближайших соседей станут недействительными. После этого процесс повторится уже для ближайших соседей — они вычеркнут подобные записи уже через 360 секунд.

Как видно, сведения о сетях, пути к которым не могут теперь проходить через отказавший маршрутизатор, распространяются по сети не очень быстро. В этом заключается одна из причин выбора в качестве периода рассылки небольшой величины в 30 секунд. Механизм тайм-аута работает в тех случаях, когда маршрутизатор не может послать соседям сообщение об отказавшем маршруте, так как либо он сам неработоспособен, либо неработоспособна линия связи, по которой можно было бы передать сообщение.

Когда же сообщение послать можно, маршрутизаторы RIP используют прием, заключающийся в указании бесконечного расстояния до сети у ставшей недоступной. В протоколе RIP бесконечным условно считается расстояние в 16 хопов. Получив сообщение, в котором расстояние до некоторой сети равно 16 (или 15, что приводит к тому же результату, так как маршрутизатор наращивает полученное значение на 1), маршрутизатор должен проверить, исходит ли эта «плохая» информация о сети от того же маршрутизатора, сообщение которого послужило в свое время основанием для записи о данной сети в таблице маршрутизации. Если это тот же маршрутизатор, то информация считается достоверной и маршрут помечается как недоступный.

Причиной выбора в качестве «бесконечного» расстояния столь небольшого числа является то, что в некоторых случаях отказы связей в сети вызывают длительные периоды некорректной работы маршрутизаторов RIP, выражающейся в зацикливании пакетов в петлях сети. И чем меньше расстояние, используемое в качестве «бесконечного», тем такие периоды короче.

Пример зацикливания пакетов

Рассмотрим случай зацикливания пакетов на примере сети, изображенной на рисунке. Пусть маршрутизатор R1 обнаружил, что его связь с непосредственно подключенной сетью 201.36.14.0 потеряна (например, по причине отказа интерфейса 201.36.14.3). Маршрутизатор R1 отмечает в своей таблице маршрутизации, что сеть 201.36.14.0 недоступна. В худшем случае он обнаружит это сразу же после отправки очередных RIP-сообщений, так что до начала нового цикла его объявлений, в котором он должен сообщить соседям, что расстояние до сети 201.36.14.0 стало равным 16, остается почти 30 секунд. Каждый маршрутизатор работает на основании своего внутреннего таймера, не синхронизируя работу по рассылке объявлений с другими маршрутизаторами. Поэтому весьма вероятно, маршрутизатор R2 опередит маршрутизатор R1 и передаст ему свое сообщение раньше, чем R1 успеет передать новость о недостижимости сети 201.36.14.0. А в этом сообщении имеются данные, порожденные записью в таблице маршрутизации R2 (табл. 1).

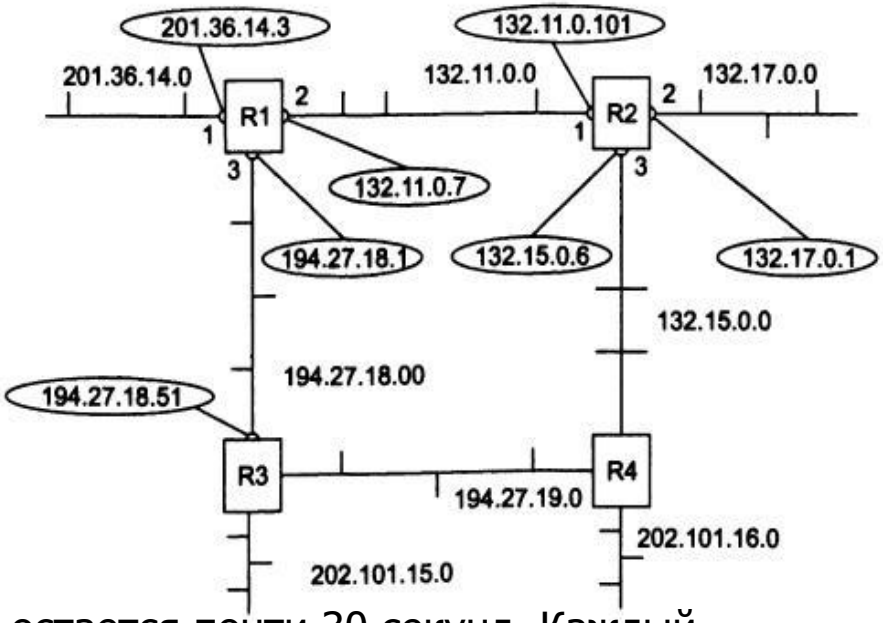


Таблица 1. Таблица маршрутизации маршрутизатора R2

Номер сети	Адрес следующего маршрутизатора	Порт	Расстояние
201.36.14.0	132.11.0.7	1	2

Эта запись, полученная маршрутизатором R1, была корректна до отказа интерфейса 201.36.14.3; теперь она устарела, но маршрутизатор R2 об этом не знает.

Пример зацикливания пакетов

Далее маршрутизатор R1 получает новую информацию о сети 201.36.14.0 — эта сеть достижима через маршрутизатор R2 с метрикой 2. Раньше R1 также получал эту информацию от R2, но игнорировал ее, так как его собственная метрика для 201.36.14.0 была лучше. Теперь R1 должен принять данные о сети 201.36.14.0, полученные от R2, и заменить запись в таблице маршрутизации о недостижимости этой сети (табл. 2).

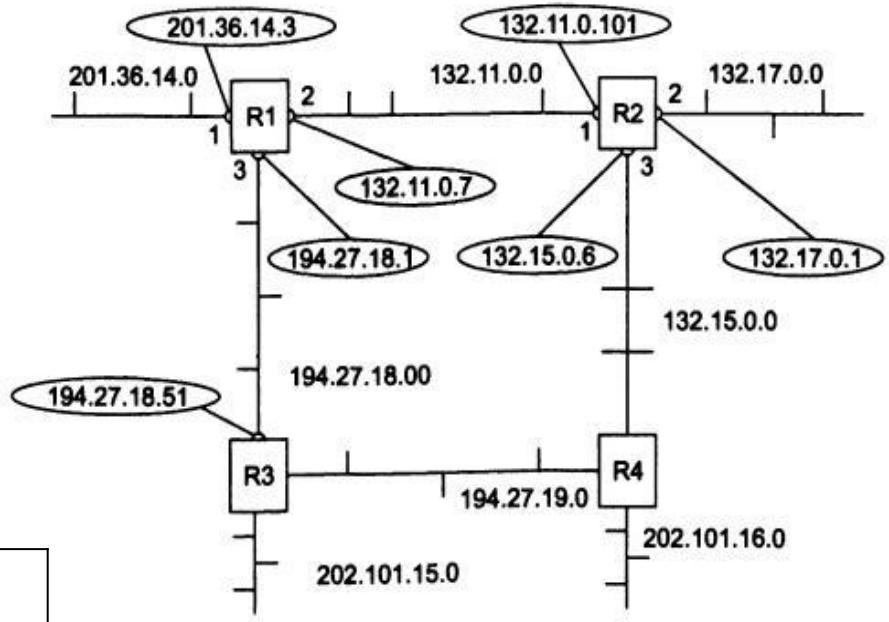


Таблица 2. Таблица маршрутизации для R1

Номер сети	Адрес следующего маршрутизатора	Порт	Расстояние
201.36.14.0	132.11.0.101	2	3

В результате в сети образуется маршрутная петля: пакеты, направляемые узлам сети 201.36.14.0, станут передаваться маршрутизатором R2 маршрутизатору R1, а маршрутизатор R1 будет возвращать их маршрутизатору R2. IP-пакеты продолжат циркулировать по этой петле до тех пор, пока не истечет время жизни каждого пакета.

Рассмотрим периоды времени, кратные времени жизни записей в таблицах маршрутизаторов.

Пример зацикливания пакетов

- Время 0-180 с. После отказа интерфейса в маршрутизаторах R1 и R2 будут сохраняться некорректные записи. Маршрутизатор R2 по-прежнему снабжает маршрутизатор R1 своей записью о сети 201.36.14.0 с метрикой 2, так как ее время жизни не истекло. Пакеты зацикливаются.
- Время 180-360 с. В начале этого периода у маршрутизатора R2 истекает время жизни записи о сети 201.36.14.0 с метрикой 2, так как маршрутизатор R1 в предыдущий период посылал ему сообщения о сети 201.36.14.0 с худшей метрикой, чем у R2, и они не могли подтверждать эту запись. Теперь маршрутизатор R2 принимает от маршрутизатора R1 запись о сети 201.36.14.0 с метрикой 3 и трансформирует ее в запись с метрикой 4. Маршрутизатор R1 не получает новых сообщений от маршрутизатора R2 о сети 201.36.14.0 с метрикой 2, поэтому время жизни его записи начинает уменьшаться. Пакеты продолжают зацикливаться.
- Время 360-540 с. У маршрутизатора R1 истекает время жизни записи о сети 201.36.14.0 с метрикой 3. Маршрутизаторы R1 и R2 опять меняются ролями — R2 снабжает R1 устаревшей информацией о пути к сети 201.36.14.0, уже с метрикой 4, которую R1 преобразует в метрику 5. Пакеты продолжают зацикливаться.

Если бы в протоколе RIP не было выбрано расстояние 16 в качестве недостижимого, то описанный процесс длился бы бесконечно (вернее, пока не была бы исчерпана разрядная сетка поля расстояния, и при очередном наращивании расстояния было бы зафиксировано переполнение).

В результате маршрутизатор R2 на очередном этапе описанного процесса получает от маршрутизатора R1 метрику 15, которая после наращивания, превращаясь в метрику 16, фиксирует недостижимость сети. Таким образом, в нашем примере период нестабильной работы сети длился 36 минут!

Пример зацикливания пакетов

Ограничение в 15 хопов сужает область применения протокола RIP до сетей, в которых число промежуточных маршрутизаторов не может быть больше 15. Для более масштабных сетей нужно применять другие протоколы маршрутизации, например OSPF, или разбивать сеть на автономные области.

Приведенный пример хорошо иллюстрирует главную причину нестабильности маршрутизаторов, работающих по протоколу RIP. Эта причина коренится в самом принципе работы дистанционно-векторных протоколов — использовании информации, полученной из «вторых рук». Действительно, маршрутизатор R2 передает маршрутизатору R1 информацию о достижимости сети 201.36.14.0, за достоверность которой он сам не отвечает.

ПРИМЕЧАНИЕ

Не следует думать, что при любых отказах интерфейсов и маршрутизаторов в сетях возникают маршрутные петли. Если бы маршрутизатор R1 успел передать сообщение о недостижимости сети 201.36.14.0 раньше ложной информации маршрутизатора R2, то маршрутная петля не образовалась бы. Так что маршрутные петли даже без дополнительных методов борьбы с ними возникают в среднем не более чем в половине потенциально возможных случаев.

Методы борьбы с ложными маршрутами в протоколе RIP

Хотя протокол RIP не в состоянии полностью исключить в сети переходные состояния, когда некоторые маршрутизаторы пользуются устаревшей информацией о несуществующих маршрутах, имеется несколько методов, которые во многих случаях решают подобные проблемы.

Практически все сегодняшние маршрутизаторы, работающие по протоколу RIP, используют *технику* ***расщепления горизонта***. Если бы маршрутизатор R2 в рассмотренном ранее примере поддерживал технику расщепления горизонта, то он бы не передал маршрутизатору R1 устаревшую информацию о сети 201.36.14.0, так как получил он ее именно от маршрутизатора R1.

Однако расщепление горизонта не помогает в тех случаях, когда петли образуются не двумя, а большим числом маршрутизаторов. Рассмотрим более детально ситуацию, которая возникнет в сети, приведенной выше, в случае потери связи маршрутизатора R1 с сетью 201.36.14.0. Пусть все маршрутизаторы этой сети поддерживают технику расщепления горизонта. Маршрутизаторы R2 и R3 не будут возвращать маршрутизатору в этой ситуации данные о сети 201.36.14.0 с метрикой 2, так как они получили эту информацию от маршрутизатора R1. Однако они будут передавать маршрутизатору информацию о достижимости сети 201.36.14.0 с метрикой 4 через себя, так как получили эту информацию по сложному маршруту, а не непосредственно от маршрутизатора R1. Например, маршрутизатор R2 получает эту информацию по цепочке R4-R3-R1, поэтому маршрутизатор R1 снова может быть обманут, пока каждый из маршрутизаторов в цепочке R3-R4-R2 не вычеркнет запись о достижимости сети 201.36.14.0.

Методы борьбы с ложными маршрутами в протоколе RIP

Для предотвращения зацикливания пакетов по составным петлям при отказах связей применяются два других приема, называемые тригтерными обновлениями и замораживанием изменений.

Прием **триггерных обновлений** состоит в том, что маршрутизатор, получив данные об изменении метрики до какой-либо сети, не ждет истечения периода передачи таблицы маршрутизации, а передает данные об изменившемся маршруте немедленно. Этот прием может во многих случаях предотвратить передачу устаревших сведений об отказавшем маршруте, но он перегружает сеть служебными сообщениями, поэтому триггерные объявления также делаются с некоторой задержкой. По этой причине возможна ситуация, когда регулярное обновление в каком-либо маршрутизаторе чуть опережает по времени приход триггерного обновления от предыдущего в цепочке маршрутизатора, и данный маршрутизатор успевает передать по сети устаревшую информацию о несуществующем маршруте.

Второй прием — **замораживание изменений** — позволяет исключить подобные ситуации. Он связан с введением тайм-аута на принятие новых данных о сети, которая только что стала недоступной. Этот тайм-аут предотвращает принятие устаревших сведений о некотором маршруте от тех маршрутизаторов, которые находятся на некотором расстоянии от отказавшей связи и передают устаревшие сведения о ее работоспособности. Предполагается, что в течение тайм-аута «замораживания изменений» эти маршрутизаторы вычеркнут данный маршрут из своих таблиц, так как не получат о нем новых записей и не будут распространять устаревшие сведения по сети.

Протокол OSPF. Два этапа построения таблицы маршрутизации

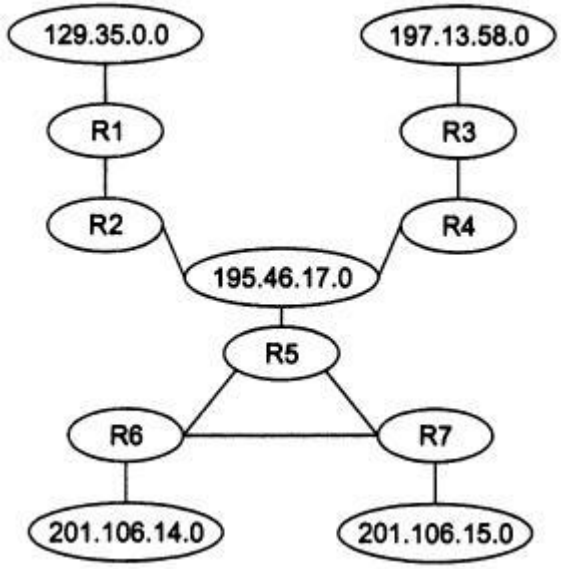
Протокол OSPF (Open Shortest Path First — выбор кратчайшего пути первым) является последним (он принят в 1991 году) протоколом, основанном на алгоритме состояния связей, и обладает многими особенностями, ориентированными на применение в больших гетерогенных сетях.

OSPF разбивает процедуру построения таблицы маршрутизации на два этапа, к первому относится построение и поддержание базы данных о состоянии связей сети, ко второму — нахождение оптимальных маршрутов и генерация таблицы маршрутизации.

Построение и поддержание базы данных о состоянии связей сети. Связи сети могут быть представлены в виде графа, в котором вершинами графа являются маршрутизаторы и подсети, а ребрами — связи между ними (рис). Каждый маршрутизатор обменивается со своими соседями той информацией о графе сети, которой он располагает к данному моменту.

Этот процесс похож на процесс распространения векторов расстояний до сетей в протоколе RIP, однако сама информация качественно иная — это информация о топологии сети. Сообщения, с помощью которых распространяется топологическая информация, называются **объявлениями о состоянии связей** (Link State Advertisement, LSA) сети.

При транзитной передаче объявлений LSA маршрутизаторы не модифицируют информацию, как это происходит в дистанционно-векторных протоколах, в частности в RIP, а передают ее в неизменном виде. В результате все маршрутизаторы сети сохраняют в своей памяти идентичные сведения о текущей конфигурации графа связей сети.



Протокол OSPF. Два этапа построения таблицы маршрутизации

Для контроля состояния связей и соседних маршрутизаторов OSPF-маршрутизаторы передают друг другу особые сообщения HELLO каждые 10 секунд. Небольшой объем этих сообщений делает возможным частое тестирование состояния соседей и связей с ними. В том случае, когда сообщения HELLO перестают поступать от какого-либо непосредственного соседа, маршрутизатор делает вывод о том, что состояние связи изменилось с работоспособного на неработоспособное и вносит соответствующие коррективы в свою топологическую базу данных. Одновременно он отсылает всем непосредственным соседям объявление LSA об этом изменении, те также вносят исправления в свои базы данных и, в свою очередь, рассылают данное объявление LSA своим непосредственным соседям.

Нахождение оптимальных маршрутов и генерация таблицы маршрутизации. Задача нахождения оптимального пути на графе является достаточно сложной и трудоемкой. В протоколе OSPF для ее решения используется итеративный **алгоритм Дийкстры**. Каждый маршрутизатор сети, действуя в соответствии с этим алгоритмом, ищет оптимальные маршруты от своих интерфейсов до всех известных ему подсетей. В каждом найденном таким образом маршруте запоминается только один шаг — до следующего маршрутизатора. Данные об этом шаге и попадают в таблицу маршрутизации.

Протокол OSPF. Два этапа построения таблицы маршрутизации

Если состояние связей в сети изменилось и произошла корректировка графа сети, каждый маршрутизатор заново ищет оптимальные маршруты и корректирует свою таблицу маршрутизации. Аналогичный процесс происходит и в том случае, когда в сети появляется новая связь или новый сосед, объявляющий о себе с помощью своих сообщений HELLO. При работе протокола OSPF конвергенция таблиц маршрутизации к новому согласованному состоянию происходит достаточно быстро, быстрее, чем в сетях, в которых работают дистанционно-векторные протоколы. Это время состоит из времени распространения по сети объявления LSA и времени работы алгоритма Дийкстры, который обладает быстрой сходимостью. Однако вычислительная сложность этого алгоритма предъявляет высокие требования к мощности процессора маршрутизатора.

Когда состояние сети не меняется, то объявления о связях не генерируются, топологические базы данных и таблицы маршрутизации не корректируются, что экономит пропускную способность сети и вычислительные ресурсы маршрутизаторов. Однако у этого правила есть исключение: каждые 30 минут OSPF-маршрутизаторы обмениваются всеми записями базы данных топологической информации, то есть синхронизируют их для более надежной работы сети. Так как этот период достаточно большой, то данное исключение незначительно сказывается на загрузке сети.

Метрики

При поиске оптимальных маршрутов протокол OSPF по умолчанию использует метрику, учитывающую пропускную способность каналов связи. Кроме того, допускается применение двух других метрик, учитывающих задержки и надежность передачи пакетов каналами связи. Для каждой из метрик протокол OSPF строит отдельную таблицу маршрутизации. Выбор нужной таблицы происходит в зависимости от значений битов TOS в заголовке пришедшего IP-пакета. Если в пакете бит D (Delay — задержка) установлен в 1, то для этого пакета маршрут должен выбираться из таблицы, в которой содержатся маршруты, минимизирующие задержку. Аналогично, пакет с установленным битом T (Throughput -пропускная способность) должен маршрутизироваться по таблице, построенной с учетом пропускной способности каналов, а установленный в единицу бит R (Reliability — надежность) указывает на то, что должна использоваться таблица, для построения которой критерием оптимизации служит надежность доставки.

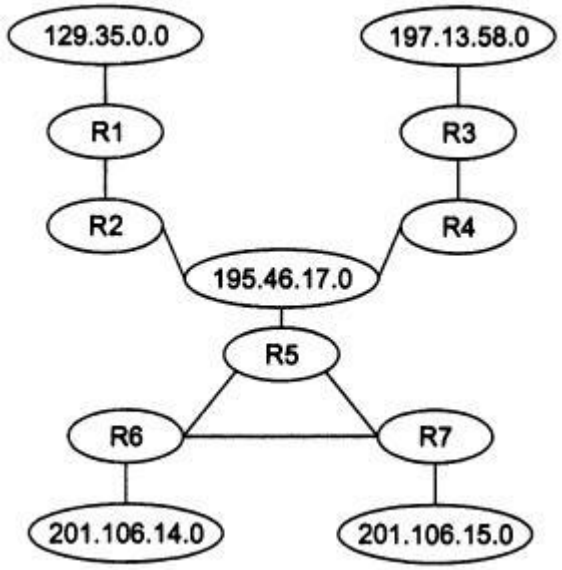
Протокол OSPF поддерживает стандартные для многих протоколов (например, для протокола покрывающего дерева) значения расстояний для метрики, отражающей пропускную способность: так, для сети Ethernet она равна 10, для Fast Ethernet — 1, для канала T-11, обладающего пропускной способностью 1,544 Мбит/с, — 65, для канала с пропускной способностью 56 Кбит/с — 1785. При наличии высокоскоростных каналов, таких как Gigabit Ethernet или STM-16/64, администратору нужно задать другую шкалу скоростей, назначив единичное расстояние наиболее скоростному каналу.

При выборе оптимального пути на графе с каждым ребром графа связывается метрика, которая добавляется к пути, если данное ребро в него входит.

Метрики

Пусть в приведенном примере маршрутизатор R5 связан с маршрутизаторами R6 и R7 каналами T-1, а маршрутизаторы R6 и R7 связаны между собой каналом 56 Кбит/с. Тогда R7 определит оптимальный маршрут до сети 201.106.14.0 как составной, проходящий сначала через R5, а затем через R6, поскольку у этого маршрута метрика будет равна 65 + 65 = 130 единиц. Непосредственный маршрут через R6 не будет оптимальным, так как его метрика равна 1785.

Протокол OSPF разрешает хранить в таблице маршрутизации несколько маршрутов к одной сети, если они обладают равными метриками. В таких случаях маршрутизатор может работать в режиме баланса загрузки маршрутов, отправляя пакеты попеременно по каждому из маршрутов.



К сожалению, вычислительная сложность протокола OSPF быстро растет с увеличением размера сети. Для преодоления этого недостатка в протоколе OSPF вводится понятие **области сети**. Маршрутизаторы, принадлежащие некоторой области, строят граф связей только для этой области, что упрощает задачу. Между областями информация о связях не передается, а пограничные для областей маршрутизаторы обмениваются только информацией об адресах сетей, имеющихся в каждой из областей, и *расстоянием от пограничного маршрутизатора до каждой сети*. При передаче пакетов между областями выбирается один из пограничных маршрутизаторов области, а именно тот, у которого расстояние до нужной сети меньше.