8. Протокол TCP

Протокол управления передачей данных TCP (Transmission Control Protocol) описан в документе RFC 793. Протокол TCP — это основной транспортный протокол стека протоколов TCP/IP. Он обеспечивает надежную передачу данных, базируясь на услугах протокола IP. На протокол TCP, в частности, возложена задача управления потоками и перегрузками, что подразумевает согласование скорости передачи данных отправителя с техническими возможностями получателя и промежуточных устройств. Поступающие к получателю данные буферизуются средствами протокола TCP. Перед отправкой данные также буферизуются.

Для достижения требуемой для конкретного логического соединения пропускной способности важно правильно настроить параметры протокола TCP, влияющие на производительность. В конечном счете, пропускная способность логических соединений определяет, насколько быстро приложения обмениваются данными по сети. Такие протоколы прикладного уровня, как HTTP и FTP, передают протоколу TCP свои данные для транспортировки. Поэтому скоростные характеристики TCP оказывают непосредственное влияние на производительность приложений.

8.1 Формат заголовка

Этот протокол использует только один тип протокольного блока данных (PDU — Protocol Data Unit), который называется сегментом TCP. Сегмент состоит из заголовка и поля данных (полезной нагрузки). На рис. 8.1 показан формат заголовка сегмента TCP.

<table>
<tr><td colspan="3">Порт отправителя (16 бит)</td><td>Порт получателя (16 бит)</td></tr>
<tr><td colspan="4">Номер в последовательности (данных) (32 бита)</td></tr>
<tr><td colspan="4">Номер подтверждения (32 бита)</td></tr>
<tr><td>Смещение данных (4 бита)</td><td>Резерв (6 бит)</td><td>Контрольные биты (6 бит)</td><td>Окно (16 бит)</td></tr>
<tr><td colspan="3">Контрольная сумма (16 бит)</td><td>Указатель срочности (16 бит)</td></tr>
<tr><td colspan="3">Опции (длина переменная)</td><td>Выравнивание (до 32 бит)</td></tr>
</table>

Рис. 8.1. Формат заголовка протокола TCP

Минимальная длина составляет 20 байт. Такая большая величина обусловлена тем, что один и тот же заголовок используется протоколом для самых различных целей. Для определения назначения большинства полей предназначены контрольные биты. Формат и значения поля «Контрольные биты» описаны в табл. 8.1.

Таблица 8.1. Формат и значения поля «Контрольные биты»

Бит	1	2	3	4	5	6
Сокращение	URG	ACK	PSH	RST	SYN	FIN
Назначение	Поле «Указатель срочности» задействовано	Поле «Номер подтверждения» задействовано	Включена функция проталкивания	Перезагрузка данного соединения	Синхронизация номеров в очереди	Данных для передачи нет

Из рис. 8.1 видно, что протокол TCP использует следующие поля:

- Поле «Номер в последовательности»- (Sequence Number) определяет номер первого байта в очереди (последовательности) байтов в текущем сегменте. Исключение составляют случаи, когда установлен бит (флаг) синхронизации SYN. Тогда это поле обозначает начальный номер в последовательности (Initial Sequence Number, ISN), и первый байт данных имеет номер в очереди ISN+1.
- Поле «Номер подтверждения» (Acknowledgment Number) содержит следующий номер в последовательности получаемых подтверждений, который ожидает отправитель в ответ на отосланный сегмент. Иными словами, на отосланный сегмент с данными отправитель ожидает сегмент с подтверждением его успешного приема. В заголовке в поле «Номер в последовательности» занесен указанный номер подтверждения. При этом должен быть установлен контрольный бит подтверждения ACK. Подтверждения (или, как часто говорят, ACK) посылаются постоянно, как только соединение будет установлено. «Номер в последовательности» и «Номер подтверждения» указываются по числу байтов в поле данных, а не по длине всего сегмента. Например, если сегмент содержит в поле «Номер в

последовательности» значение 1000 и несет в поле данных 600 байт данных, то поле «Номер в последовательности» указывает на номер первого байта в поле данных. Следующий (в логическом порядке) сегмент будет иметь в поле «Номер в последовательности» значение 1601. Мы видим, что протокол TCP изначально логически ориентирован на работу с потоком данных. Он принимает байты от пользовательского приложения, группирует их, а затем распределяет по сегментам и формирует поток сегментов с нумерацией каждого байта.

- Поле «Смещение данных» (Data Offset) определяет количество 32-битных слов в заголовке TCP. Тем самым указывается начало поля данных. Заголовок протокола TCP всегда заканчивается на 32-битной границе, даже если он содержит опции.
- Поле «Резерв» (Reserved) должно быть заполнено нулями и предназначено для будущего расширения протокола.
- Поле «Окно» (Window) содержит объявляемый размер окна в байтах.
- Поле «Контрольная сумма» (Checksum) рассчитывается по сегменту, при этом определяется 16-битное дополнение суммы всех 16-битных слов в заголовке и в поле данных. Если сегмент содержит нечетное количество байтов, то он будет дополнен нулями справа до образования 16-битного слова. Этот выравнивающий байт не передается с сегментом по сети, так как может быть «восстановлен» получателем. Контрольная сумма учитывает также 96-битный псевдозаголовок, который ставится перед заголовком протокола TCP. Псевдозаголовок включает следующие поля из заголовка протокола IP: IP-адреса отправителя и получателя, протокол и длину сегмента. С помощью добавления псевдозаголовка протокол TCP защищает самого себя от ошибочной доставки протоколом IP. Так, если протокол IP доставляет сегмент, не предназначенный данному работающему приложению, то модуль протокола TCP на принимающей стороне обнаружит некорректность доставки.
- Поле «Указатель срочности» (Urgent Pointer) сообщает текущее значение указателя срочности. Эта положительная величина определяет смещение относительно номера в очереди данного сегмента. Этот указатель сообщает номер байта, следующего за срочными данными, то есть, начиная с этого байта, данные имеют обычный статус срочности, т.е. можно использовать буферизацию. Поле используется совместно с контрольным битом URG.
- Поле «Опции» (Options) имеет переменную длину и может вообще отсутствовать. Опции располагаются в конце заголовка протокола TCP и их длина кратна 8 битам. Протокол TCP должен быть готов обрабатывать все виды опций. Опции используются для решения вспомогательных задач, например, для выбора максимального размера сегмента.

Документ RFC 793 определил опцию — максимальный размер сегмента. Поле этой опции с размером 16 бит может быть использовано только в сегменте, который посылается при запросе на создание логического соединения. Опция определяет максимальный размер сегмента в байтах, который будет использоваться будущим соединением.

Широкое распространение получили две другие опции: «Параметр масштабирования окна» (Window scale factor) и «Временной штамп» (Timestamp).

Обычно, поле «Окно» в заголовке протокола TCP определяет некий лимит или кредит, выданный отправителю на размещение его данных в отсылаемых сегментах и измеряемый в байтах. Максимальный размер обычного окна 2^{16} байт. Когда используется опция «Параметр масштабирования окна», размер окна может быть увеличен по формуле 2^{16+F}, где F — это параметр масштабирования окна. Максимальное значение F, которое может быть использовано протоколом TCP, — 14. Полное описание этого метода приведено в разделе 2.2 документа RFC 1323.

В этом же документе описан алгоритм «Временной штамп». Этот алгоритм используется для соединений, работающих с окнами большого размера. Опция «Временной штамп» используется для того, чтобы проводить точное измерение времени прохождения сигнала до получателя и обратно, то есть времени обращения (Round Trip Time, RTT). Такие замеры необходимы для корректировки времени повторной передачи при отсутствии подтверждения приема определенного сегмента. «Временной штамп» реально может быть использован в любом сегменте с данными и определяет два дополнительных поля: «Timestamp Value» (значение временного штампа) и «Timestamp Echo Reply» (ответ на временной штамп получен).

Протокол TCP может включить поле «Timestamp Value» в любой исходящий сегмент. Когда этот сегмент подтверждается принимающей стороной, то отвечающий модуль протокола TCP включает в ответный сегмент поле «Timestamp Echo Reply» с тем же самым значением, что было в поле «Timestamp Value» в сегменте, который был подтвержден. Это позволяет протоколу TCP выполнять постоянный мониторинг времени обращения.

- Поле «Выравнивание» (Padding) может иметь переменную длину и представляет собой фиктивное поле, используемое для доведения размера заголовка до целого числа 32-битных слов.

Флаг PSH (Push) реализует службу протокола TCP: продвижение (проталкивание) потока данных. Обычно протокол TCP передает данные, когда количество предназначенных для передачи байтов равно длине поля данных очередного сегмента. Приложение может потребовать, чтобы протокол передал все оставшиеся данные и пометил их флагом PSH. На принимающей стороне модуль протокола TCP так же доставит эти данные приложению сразу (то есть данные не будут ждать своей очереди в буфере). Приложение может прибегнуть к проталкиванию, если достигнут логический конец данных.

8.2 Состояние системы

Соединения протокола TCP переходят из одного состояния в другое в ответ на определенные события — запросы клиента, приход сегментов с флагами SYN, ACK, RST, FIN — или по истечении заданного времени. Соединение может находиться в одном из следующих состояний:

- LISTEN — ожидание запроса на соединение со стороны внешних (чужих) портов и внешних (чужих) программ TCP.
- SYN-SENT — ожидание парного запроса на установление соединения (со стороны отправителя запрос уже сделан).
- SYN-RECEIVED — ожидание подтверждения после того, как запрос на установление соединения уже принят и отправлен.
- ESTABLISHED — соединение установлено. Принимаемые от приложения данные можно передать пользователю.
- FIN-WAIT-1 — ожидание подтверждения получения от чужой программы TCP отправленного ранее запроса на закрытие соединения.
- FIN-WAIT-2 — ожидание запроса на закрытие соединения со стороны чужой программы TCP.
- CLOSE-WAIT — ожидание запроса на закрытие соединения со стороны своего клиента.
- CLOSING — ожидание подтверждения запроса о закрытии соединения со стороны чужой программы TCP.
- LAST-ACK — ожидание ответного запроса на закрытие соединения на посланный запрос о закрытии соединения, который был ранее отправлен чужой программе TCP.
- TIME-WAIT — соединение находится в этом состоянии на протяжении времени, достаточного для того, чтобы быть уверенным, что чужая программа TCP получила подтверждение своего запроса на закрытие соединения.
- CLOSED — соединение закрыто.

Основное состояние соединения — ESTABLISHED (соединение установлено). В этом состоянии происходит обмен данными между абонентами.

Для одновременного использования протокола TCP несколькими прикладными программами на одном компьютере он представлен набором адресов и портов. Поскольку идентификаторы портов выбираются каждой программой протокола TCP независимо, то они не будут уникальны. Уникальна совокупность идентификатора порта и IP-адреса. Эта совокупность называется сокет. Соединение между отправителем и получателем полностью определяются двумя сокетами на его концах. Это соединение можно использовать для передачи данных в обоих направлениях, то есть оно поддерживает дуплексный режим передачи.

8.3 Установление и закрытие соединений

При открытии и закрытии соединений в зависимости от поведения сторон могут возникать самые различные ситуации (но так или иначе соединение всегда будет находиться в одном из состояний, перечисленных выше). Упрощенно же процесс открытия соединения можно представить следующей последовательностью действий:

- Инициатор соединения посылает запрос к протоколу TCP на открытие порта для передачи. Протокол TCP на принимающей стороне заблаговременно до этого момента открывает порт для приема данных.
- После открытия порта для передачи протокол TCP на стороне приложения-инициатора посылает запрос приложению, с которым требуется установить соединение (принимающей стороне) и открывает на своей стороне порт для приема.
- Принимающая сторона открывает порт для передачи, отсылает квитанцию, подтверждающую прием запроса и также передает запрос к противоположной стороне;
- Приложение-инициатор возвращает квитанцию.

С этого момента соединение считается установленным. После этого начинается обмен информацией по данному соединению. При передаче данных по соединению каждый байт информации нумеруется. Нумерация ведется и в очереди отправления и в очереди получения. Первоначальный номер байта в очереди отправки указывается модулем TCP посылающей стороны, а первоначальный номер байта в очереди приема выясняется во время установления соединения. В это время оба модуля протокола TCP должны синхронизировать друг с другом первоначальные номера байтов в очередях. Синхронизация производится путем обмена сегментами, которые используются при установке соединения. Эти сегменты несут флаг синхронизации SYN и исходные номера для обеих очередей. Синхронизация требует, чтобы каждая сторона послала свой собственный первоначальный номер в очереди и получила подтверждение о принятии этого номера. Нумеруются и сами сегменты: номером сегмента считается номер первого байта в поле полезной нагрузки этого сегмента.

Процедуру открытия соединения с подтверждением сообщений можно показать с фиксацией состояний соединения и переходов между ними (рис. 8.2). Каждая строка на рис. 8.2 показывает состояние соединения. Стрелки «→» означают отправление сегмента от модуля TCP станции А в модуль TCP станции Б. Стрелки «←» показывают отправку сегментов в противоположном направлении. Промежуточное состояние соединения соответствует моменту посылки или получения сегмента. На рис. 8.2 показано не все содержание сегментов, — приведены только номера в очереди, флаги управления и АСК.

Модуль TCP на станции А	Модуль TCP на станции Б
CLOSED	LISTEN
SYN-SENT → <SEQ=100><CTL=SYN>	0 SYN-RECEIVED
ESTABLISHED ← <SEQ=300><ACK=101><CTL=SYN, ACK>	M SYN-RECEIVED
ESTABLISHED → <SEQ=101><ACK=301><CTL=ACK>	0 ESTABLISHED

Рис. 8.2. Процедура подтверждения трех сообщений для синхронизации соединения

Станция А указывает, что она будет использовать номер в очереди 100. В ответ станция Б посылает свой номер в очереди 300 и говорит, что ожидает получения номера 101. На рис. 8.3 показана нормальная, штатная процедура закрытия соединения.

Модуль TCP на станции А	Модуль TCP на станции Б
ESTABLISHED	ESTABLISHED
FIN-WAIT-1 → <SEQ=100><ACK=300><CTL=FIN, ACK>	0 CLOSE-WAIT
FIN-WAIT-2 ← <SEQ=300><ACK=101><CTL=ACK>	M CLOSE-WAIT
TIME-WAIT ← <SEQ=300><ACK=101><CTL=FIN, ACK>	M LAST-ACK
TIME-WAIT → <SEQ=101><ACK=301><CTL=ACK>	0 CLOSED
CLOSED	CLOSED

Рис. 8.3. Нормальная процедура закрытия соединения

Как видно, при закрытии соединения происходит обмен сегментами, несущими управляющий флаг FIN, указывающий на отсутствие данных для передачи.

8.4 Плавающее окно

Как и большинство протоколов, осуществляющих управление потоком данных (например, HDLC и X.25), протокол TCP использует механизм плавающих окон. Протоколы HDLC и X.25 используют этот механизм в классическом виде — на каждый отправленный блок данных должно быть получено подтверждение. Протокол TCP несколько отходит от классической схемы.

Основной недостаток классической схемы заключается в том, что только один сегмент может быть передан за один сеанс. В данном случае под сеансом понимается посылка сегмента и получение подтверждения на его успешный прием. Такая схема не позволяет работать с максимальной производительностью. Эффективность может быть значительно повышена, если разрешить передачу множества сегментов за один сеанс с группировкой всех подтверждений для них в одно.

Рассмотрим схему работы этого метода на примере двух станций — А и Б, которым необходимо обмениваться данными. Станция Б выделяет буферное пространство для приема n сегментов. Поэтому станция Б может принять n сегментов, а станция А может послать те же n сегментов без необходимости ожидания подтверждений об их приеме. Для отслеживания подтверждений о принятых сегментах каждый из них помечается номером в последовательности. Станция Б подтверждает получение сегмента посылкой подтверждения на него, которое содержит номер в последовательности следующего ожидаемого сегмента. Это подтверждение также косвенным образом извещает станцию А о том, что станция Б готова получить следующие n сегментов, начиная с указанного номера. Такая схема работы годится для подтверждения получения множества сегментов. Например, станция Б получает сегменты 2, 3 и 4, но воздерживалась от отправки подтверждения на сегменты 2 и 3 до получения сегмента 4. Посылая подтверждение с номером в последовательности 5, станция Б подтверждает получение сегментов 2, 3 и 4 за один раз. Таким образом, можно сказать, что станция А ведет список номеров в последовательности сегментов, которые ей разрешено посылать, а станция Б поддерживает список номеров в последовательности сегментов, которые она готова принять. Эти списки называют окнами сегментов, а. такую схему передачи сообщений часто называют управлением потоком с использованием плавающего окна.

Так как номер в последовательности занимает одно поле в сегменте, то очевидно, что номер не может быть слишком большим. Например, если это поле занимает 3 бита, то номер в последовательности сегментов может иметь значения от 0 до 7. Соответственно, сегменты нумеруются по модулю 8, то есть за номером в последовательности 7 следует номер в последовательности 0. Таким образом, для поля номера в последовательности, состоящего из k бит, границы изменения номера равны 0 и 2^k-l, а сегменты нумеруются по модулю 2^k. С учетом приведенных рассуждений на рис. 8.4 показаны значения передаваемых и принимаемых сегментов на принимающей и передающей сторонах с фиксацией границ плавающего окна. В этом примере для простоты размер поля «Номер в последовательности» принят равным 3 битам. Серые прямоугольники указывают сегменты, которые могут быть посланы. Отправитель может послать 8 сегментов, начиная с сегмента с номером 0. Каждый раз, когда сегмент посылается, ширина окна (серого прямоугольника) уменьшается. При получении подтверждения ширина окна увеличивается. Сегменты, находящиеся между черной вертикальной чертой и серым прямоугольником (окном) уже были посланы, но еще не были подтверждены. Отправитель должен хранить копии этих сегментов в своем буфере на случай, если потребуется их повторная передача.

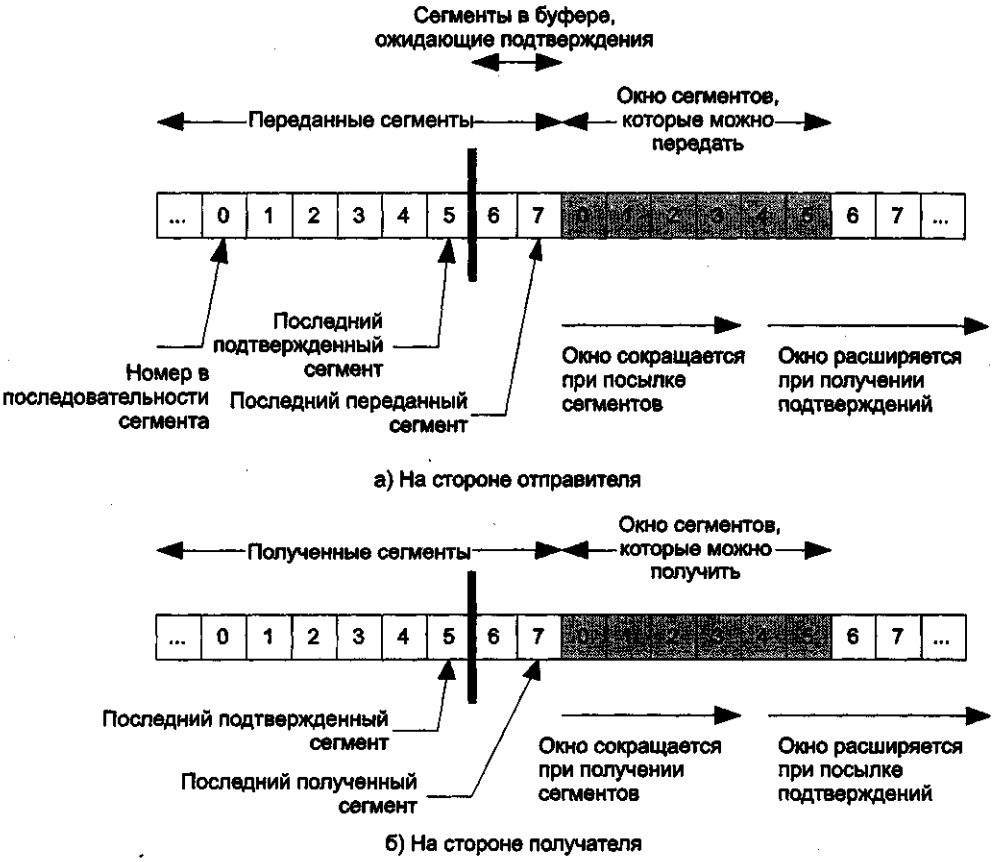

Рисунок 8.4 Сегменты в потоке данных отправителя и получателя.

8.5 Пропускная способность

Рассмотрим методы определения максимально возможной пропускной способности соединения протокола TCP. Пропускная способность зависит от размера окна передачи, задержки и скорости пересылки данных. Используем следующие обозначения:

W — размер окна передачи в байтах;

R — скорость передачи данных (бит/с) по определенному соединению, доступная на стороне отправителя;

D — задержка (в секундах) при передаче данных между отправителем и получателем через определенное соединение.

Для простоты рассуждений проигнорируем влияние служебных битов в сегменте TCP. Предположим, что отправитель начинает передавать последовательность байтов получателю через установленное соединение. Для того чтобы первый байт достиг получателя, потребуется время, равное D. Такое же время — D секунд — потребуется для получения подтверждения. В течение этого времени отправитель может передать 2RD бит, или RD/4 байт. На самом деле, отправитель ограничен размером окна в W байт и не может сдвигать окно, пока не получит подтверждение. Только при W≥ RD/4 на этом соединении достигается максимально возможная пропускная способность. Если W<RD/4, то близость пропускной способности к максимальной определяется отношением W к RD/4. Следовательно, нормированная пропускная способность S может быть выражена как:

$$S=\begin{cases} 1, & W \geq RD/4 \\ \dfrac{4W}{RD}, & W < RD/4 \end{cases}.$$

На рис. 8.5 показан пример определения максимальной пропускной способности в зависимости от произведения RD.

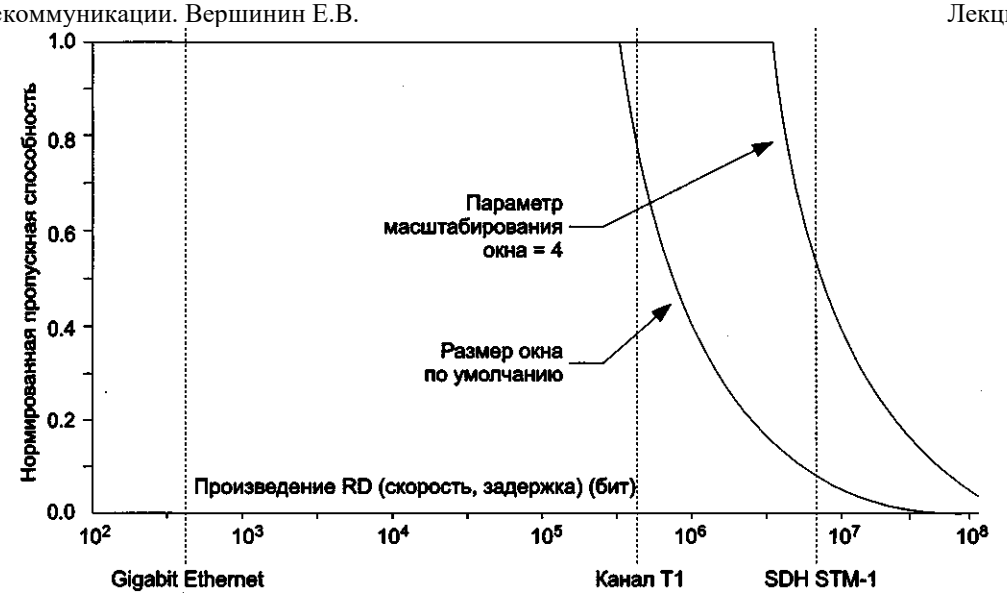

Рисунок 8.5 Влияние параметра масштабирования окна на эффективность передачи.

Максимальный размер окна может составлять 2^{16}-1 = 65535 байт. Такой размер окна должен быть достаточен для большинства приложений. В качестве примера давайте разберем три различные технологии, применяемые для передачи сегментов TCP и использующие такой размер окна. Для технологии Gigabit Ethernet с длиной магистрали, равной 100 м, произведение RD будет меньше, чем 10^3 бит. На больших расстояниях, пусть даже при более низких скоростях, например, в случае использования канала T1 (1.544 Мбит/с), произведение RD становится большим, следовательно, эффективность падает. Тем не менее, она остается, как видно из рис. 8.5, приемлемой — около 0.8. На значительных расстояниях при дальнейшем увеличении скорости (допустим, речь идет о передаче информации с использованием технологии SDH (канал 155 Мбит/с) между двумя городами) рассматриваемая технология становится крайне неэффективной — как видим, нормализованная пропускная способность падает до 0.1. Очевидно, что в данном случае размер окна слишком мал. Необходимо использовать другой параметр масштабирования окна, который позволил бы эффективно задействовать пропускную способность канала. Достаточно увеличить параметр масштабирования окна до 4 — это приведет к значительному увеличению размера окна до 2^{20}-1 ≈ 1 Мбайт.

Как видим, перечисленные выше параметры оказывают основное влияние на эффективность передачи протокола TCP. Однако существует множество усложняющих факторов, которые также следует принять во внимание. Во-первых, в большинстве случаев соединения TCP мультиплексируются в один канал, так что каждое соединение получает часть его доступной пропускной способности. Это приводит к снижению скорости передачи R и, следовательно, к снижению эффективности работы протокола. Во-вторых, большинство соединений TCP проходят через маршрутизаторы. В этом случае время D будет равно сумме задержек в каждой сети и задержек на каждом маршрутизаторе в пути. При этом суммарная задержка на маршрутизаторах часто вносит основной вклад во время задержки D, особенно при возникновении перегрузок. В-третьих, значение скорости R, используемое в приведенной выше формуле, определяет скорость передачи данных, доступную для соединения, только на стороне отправителя. Если на одном из переходов в пути от отправителя до получателя скорость передачи меньше этой скорости, то попытка передачи на максимальной скорости приведет к образованию узкого места, что неизбежно повысит время D. И наконец, в-четвертых, если сегмент теряется, он должен быть передан вновь, что приводит к снижению пропускной способности. Степень влияния потерь сегментов на эффективность передачи зависит от политики повторных передач. В современных распределенных сетях несколько сегментов протокола TCP могут быть потеряны из-за ошибок на линиях. Большинство же сегментов теряются при использовании механизмов сброса пакетов на маршрутизаторах или коммутаторах (например, Frame Relay) в моменты сетевых перегрузок.

8.6 Контроль за перегрузками

Контроль за перегрузками в сетях IP достаточно сложно реализовать по целому ряду причин. К ним можно отнести следующие:

- Протокол IP не ориентирован на установление соединения. Он не обеспечивает обнаружение перегрузки и по этой причине не может быть использован для контроля за перезагрузками.
- Протокол TCP осуществляет контроль потока из конца в конец соединения и может лишь по косвенным признакам определить перегрузку в сети. Более того, так как задержки в распределенных сетях постоянно изменяются, то информация, полученная на основании косвенных признаков (например, размер окна), не является достоверной.
- Не существует распределенного алгоритма для связывания различных протоколов TCP. То есть, протоколы на разных компьютерах не могут взаимодействовать друг с другом для поддержания определенного уровня общего потока. Более того, на самом деле они ведут себя очень «эгоистично» по отношению к свободным ресурсам канала.

Сообщение «Подавление источника» (Source Quench) протокола ICMP можно рассматривать в качестве грубого инструмента, предназначенного для сдерживания потока трафика от отправителя, но его нельзя назвать эффективным методом контроля за перегрузками.

Задачу контроля за перегрузками можно возложить на протокол RSVP.

Протокол TCP может влиять на загрузку сети, управляя потоком данных с помощью плавающего окна, применяя различные методы отправки/приема данных и отсылки подтверждения, следя за уровнем ошибок и используя различные методы повторной передачи. Ниже будут рассмотрены алгоритмы медленного старта и контроля за перегрузками, реализованные в протоколе TCP. Основное предназначение этих алгоритмов — предотвращение перегрузки в сети.

8.7 Управление потоком данных

Управление потоком данных использует механизм плавающего окна, но кроме этого, применяется также более гибкая схема приема/передачи данных и отсылки подтверждений на успешный прием данных. Управление потоком протокола TCP использует так называемую схему с выделением лимита на передачу данных. По этой схеме каждый передаваемый байт имеет свой собственный номер в последовательности (SN). Когда протокол TCP посылает сегмент, он выставляет в поле номера в последовательности номер первого байта в поле данных этого сегмента. На принимающей стороне пришедший сегмент подтверждается сообщением, в котором указывается (A= i, W=j). Такая запись имеет следующее значение: если величина A (АСК) равна i, это значит, что сообщение подтверждает получение всех байтов, вплоть до номера в последовательности i-1; следующие ожидаемые байты имеют номер в последовательности i. Кроме того, выдается разрешение на посылку дополнительного окна W (Window) j байтов, то есть байтов с номерами в последовательности от i до i +j - 1. На рис. 8.6 иллюстрируется работа этого механизма. Окна передачи и приема указывают количество байтов данных. Для большей наглядности покажем поток данных, идущий только в одном направлении, и предположим, что в каждом сегменте посылаются 200 байт данных. Во время установления соединения номера в последовательностях отправителя и получателя синхронизированы и станция А имеет начальный лимит на отсылку данных 1400 байт, начиная с номера байта 1001. После посылки 600 байт в трех сегментах станция А уменьшает свое окно отсылки до 800 байт (номера с 1601 до 2400). После получения этого сегмента станция В подтверждает получение всех байтов, вплоть до 1601, и формирует свое окно приема на 1000 байт. Это означает, что станция А может посылать байты, начиная с номера 1601 и заканчивая номером 2600, то есть пять сегментов. Однако к тому моменту, когда сообщение от станции В доходит до станции А, последняя уже успела выслать два сегмента, содержащие байты 1601-2000, что позволял начальный лимит. Следовательно, оставшийся лимит станции А на этот момент составляет всего 400 байт или два сегмента. Во время обмена станция А продвигает левую границу своего окна каждый раз, когда осуществляет передачу. Правая граница передвигается только тогда, когда станция получает новый лимит.

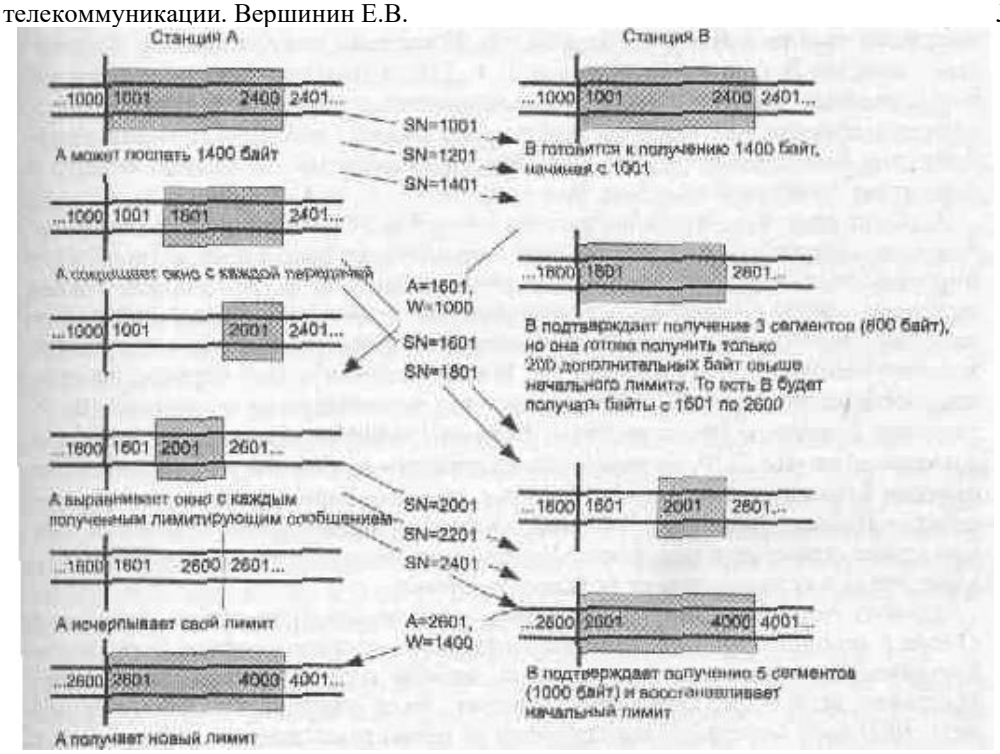

Рисунок 8.6 Схема управления потоком данных

На практике обе стороны одновременно задействуют режимы передачи и приема, так как данные могут передаваться в обоих направлениях (происходит полнодуплексная передача). Механизм выделения лимита является достаточно гибким. Например, рассмотрим ситуацию, при которой последнее сообщение, посланное станцией B, было (A=i, W=j). Последним байтом данных, полученным станцией B, был байт с номером i-1. Для увеличения лимита до значения k, при условии, что kj и дополнительные данные не поступали, станция B формирует сообщение (A= i, W = k). Для подтверждения входящего сегмента, содержащего m байт данных (m) без выделения дополнительного лимита, станция B формирует сообщение (A = i+m, W =j-m).

Следует отметить, что от получателя не требуется немедленного подтверждения приходящих сегментов. Он может ожидать некоторое время, а затем сформировать подтверждение сразу на несколько сегментов. Получатель должен проводить какую-то политику, регулирующую количество данных, которое он позволяет передавать отправителю. Можно выделить две политики получателя: консервативную и оптимистическую. Консервативная схема управления потоком основана на том, что лимит выделяется в соответствии с имеющимся доступным буферным пространством. Если это правило применить к ситуации, показанной на рис. 8.6, то первое лимитирующее сообщение говорит о том, что станция B может разместить 1000 байт в своем буфере, а второе — о том, что станция B может разместить 1400 байт. Консервативная схема управления потоком может ограничить пропускную способность логического соединения в ситуации, когда в сети возникают большие задержки.

Получатель может более эффективно использовать пропускную способность канала с помощью оптимистического выделения лимита, сообщая о свободном буферном пространстве, которого он на данный момент фактически не имеет. Например, если буфер получателя заполнен, но он ожидает, что сможет освободить 1000 байт буферного пространства за время прохождения информации из конца в конец соединения, он может послать кредит на 1000 байт. Если получатель может поддерживать скорость, заданную отправителем, то такая схема способна повысить пропускную способность и не принесет вреда. Если же отправитель работает быстрее получателя, то некоторые сегменты будут отбрасываться из-за занятого буфера, что повлечет за собой повторную передачу. В таком случае оптимистическое управление потоком может усугубить ситуацию с перегрузкой в сети.

8.8 Политики отправки и приема сегментов

При отсутствии данных, помеченных флагом PSH, протокол TCP на стороне отправителя может сам решать, когда следует осуществить передачу. Когда данные передаются модулю протокола TCP пользовательским приложением, они записываются в передающий буфер. Протокол TCP может создавать сегмент для каждой группы данных, предоставленных приложением, или он может ожидать накопления определенного количества данных и только после этого формировать и отправлять сегмент. То, какая политика отправки лучше, всецело зависит от требований к производительности. Если передачи сегментов происходят редко, но при этом передаются большие объемы данных, то сегменты можно формировать сразу при поступлении данных — накладные расходы здесь будут невелики. G другой стороны, даже если объем передаваемых данных невелик, иногда имеет смысл слать их сразу же при поступлении — при этом накладные расходы велики, но такая система обеспечивает быструю реакцию на изменения состояния сети.

При отсутствии данных, помеченных флагом PSH, протокол TCP на стороне получателя также может самостоятельно решать, когда следует доставить данные пользовательскому приложению. Так, он может доставлять данные при получении каждого сегмента по порядку или заносить данные из нескольких сегментов в буфер приема. Как и в случае с отправкой, выбор политики доставки зависит от требований к производительности. Если данные приходят редко и имеют большой объем, то приложение получит их сразу. С другой стороны, если данные поступают часто и маленькими порциями, то немедленная их передача приложению приведет к неэффективному расходованию ресурсов приложения и протокола TCP.

Если сегменты поступают в порядке их отсылки по установленному соединению, протокол TCP помещает их данные в буфер получения для доставки пользовательскому приложению. Но вполне возможна ситуация, при которой определенный сегмент поступит не в порядке отправления. В этом случае протокол TCP на принимающей стороне может либо принимать только те сегменты, которые поступают в порядке отправления (другие сегменты просто отбрасываются), либо принимать все сегменты, номера которых зафиксированы в окне приема (вне зависимости от порядка их поступления).

Первый вариант действий упрощает реализацию протокола, но при этом возникает дополнительная нагрузка на сеть, так как отправитель будет ожидать некоторое время, а затем повторно передаст отброшенные сегменты, которые хотя и были успешно получены, но затем были отброшены из-за некорректного порядка поступления. Более того, если один сегмент был потерян при передаче, то необходимо посылать повторно все последующие сегменты.

При втором варианте может быть снижено количество повторных передач, но при этом требуется более сложный алгоритм приема и более сложная схема сохранения данных для буферизации и отслеживания порядка поступления данных.

8.9 Таймер повторной передачи

Протокол TCP не формирует явных негативных подтверждений (ACK), то есть подтверждений, явно указывающих на произошедшие нарушения. Вместо этого протокол полагается исключительно на положительные подтверждения и на повторную передачу, которая должна происходить, если подтверждение не поступает в определенный интервал времени. Два события приводят к повторной передаче сегмента:

- Сегмент может быть испорчен при передаче, но, тем не менее, доставлен получателю. Контрольная сумма, включенная в сегмент, позволяет получателю обнаружить ошибку и отбросить такой сегмент.
- Сегмент просто не поступает.

И в том, и в другом случае отправитель не сразу узнает о том, что посылка сегмента была не успешной.

Если сегмент на принимающей стороне принимается с ошибками либо не принимается вовсе, то при этом не формируется и не отсылается ACK. В таком случае потребуется повторная передача этого сегмента. Для принятия решения о том, когда следует осуществлять повторную передачу, вводится таймер, работающий с каждым посланным сегментом. Если время таймера истекает до момента получения ACK для этого сегмента, отправитель должен

выполнить повторную передачу. Важной особенностью протокола TCP является то, что время этого таймера можно регулировать. Это очень полезно, так как если время будет слишком малым, то часто будут осуществляться ненужные повторные передачи, снижающие эффективность использования полосы пропускания сети. Если время будет очень большим, протокол не сможет быстро адекватно реагировать на потерю сегмента. Время таймера следует выбрать чуть больше времени обращения RTT (Round Trip Time). Естественно, само время RTT (задержка в сети) зависит от множества факторов, вносящих свой вклад даже при постоянной загрузке сети.

Существуют два способа задания времени таймера повторной передачи.

- Фиксированный таймер. В первом случае используется фиксированное значение таймера, которое определяется по статистическим данным, характерным для «нормального» поведения распределенной сети. Иными словами, определяется среднее значение RTT и таймер выставляется с небольшим превышением. Так как такая политика основывается на устоявшемся режиме работы сети, она не способна адекватно и гибко реагировать на резкие изменения условий в сети.
- Адаптивный таймер. При втором способе используется адаптивная схема задания таймера, которая также имеет достоинства и недостатки. Предположим, что протокол TCP постоянно отслеживает время получения подтверждений на посланные сегменты и устанавливает свой таймер, основываясь на наблюдаемой задержке. Это значение не может быть признано самым оптимальным во всех возможных случаях по следующим причинам:

✓ посылка сегмента может не подтверждаться немедленно (напомним, что, как правило, используются совокупные подтверждения сразу нескольких сегментов);
✓ если сегмент был послан повторно, отправитель не всегда может узнать, был ли полученный АСК послан на начальный сегмент или на посланный повторно;
✓ условия в сети могут внезапно поменяться.

Следует отметить, что эта проблема не имеет удовлетворительного единственно правильного решения. Всегда будет существовать некоторая неуверенность в оптимальности установки таймера повторной передачи. Механизм вычисления таймера описан в документе RFC 793.

8.9.1 Адаптивный таймер повторной передачи

Если происходят какие-либо изменения в поведении сети, велика вероятность того, что статический (фиксированный) таймер повторной передачи отреагирует на них неадекватно: он, будет либо слишком мал, либо слишком велик. Поэтому во все реализации протокола TCP заложен механизм оценки текущего времени передачи, которая осуществляется по наблюдению за характером изменения задержки подтверждения на последние посланные сегменты. Затем таймер устанавливается в значение, которое немного превосходит оцененное время передачи.

Рассмотрим один из подходов, который учитывает среднее наблюдаемое время задержки подтверждения для некоторого количества посланных сегментов. Если это среднее время достаточно точно предсказывает будущие задержки, то полученное значение таймера повторной передачи приведет к очень хорошей производительности. Усредненное время можно определить, используя следующее выражение:

$$ARRT\ (K{+}1) = \frac{1}{K+1}\sum_{i=1}^{K+1} RTT\ (i)\ ,$$

где $RTT(i)$ — время обращения, наблюдаемое для i-го переданного сегмента;
$ARTT(K)$ — среднее время обращения для первых K сегментов.

Это выражение можно представить в виде рекуррентной формулы:

$$ARIT\ (K+1) = \frac{K}{K+1}\ ARIT\ (K) + \frac{1}{K+1} RTT\ (i).$$

Обратите внимание на то, что оба параметра в этом выражении имеют равный вес (в оба знаменателя входит одно и то же выражение К+1). Как правило, больший вес — или большая степень доверия — дается более новым, последним замерам, так как они наиболее точно отражают будущее поведение сети. Общий метод предсказания следующего усредненного

значения времени обращения на основе предыдущей серии измерений времени приведен в документе RFC 793. Его суть сводится к обобщению предыдущей формулы, а именно:

SRTT(K+ 1) = αSRTT(K) + (1 - α) RTT(K+1),

где SRTT(K) (Smoothed Round Trip Time) — так называемое сглаженное оценочное время обращения. Давайте сравним это выражение с предыдущим. Используя константу а ($0 < \alpha < 1$), не зависящую от числа последних наблюдений, получаем, что все наблюдения учитываются, но более ранние наблюдения имеют меньший вес. Эту константу называют фактором сглаживания.

Наименьшее значение константы а дает больший вес самым последним наблюдениям. Для значения α=0.5 наиболее весомыми становятся четыре или пять последних наблюдений, в то время как для значения константы α=0.875 средний вес распределяется для десяти и более последних наблюдений. Достоинством выбора небольшого значения константы α является то, что при этом отражаются быстрые изменения в наблюдаемых величинах. Недостаток заключается в том, что если происходит короткое волнообразное изменение в наблюдаемых значениях с последующим переходом к усредненному значению, использование небольшого значения константы α приведет к резким изменениям вычисляемого времени обращения.

Последняя формула используется в документе RFC 793 для оценки текущего времени обращения. Как уже упоминалось, таймер повторной передачи должен быть установлен в значение, большее чем оцененное время обращения. Для вычисления этого превышения можно использовать формулу:

RTO (K+1) = SRTT(K + 1) + Δ,

где RTO (Retransmission Time Out) — контрольное время повторной передачи (его иногда называют тайм-аутом повторной передачи); Δ— константа.

Недостаток этой формулы в том, что значение Δ не пропорционально значению SRTT (то есть никак не учитывает его). Для больших значений SRTT любая константа Δ может оказаться относительно велика и вызовет ненужные задержки при повторных передачах потерянных сегментов. В этой связи документ RFC 793 определяет таймер, пропорциональный значению SRTT со следующими ограничениями:

RTO (K+1) = min(UBOUND , max(LBOUND, β*SRTT(K + 1))),

где UBOUND и LBOUND — фиксированные верхняя и нижняя границы значения таймера; β— константа.

Ее называют фактором изменения задержки. Документ RFC 793 не рекомендует каких-то фиксированных значений, но приводит диапазон изменений параметров: α— между 0.8 и 0.9 и β— между 1.3 и 2.0.

8.9.2 Узкие места в сети

Управление потоком с использованием плавающего окна дает возможность получателю задавать скорость работы отправителя. Получатель в этом случае только подтверждает полученные сегменты и расширяет свое окно приема в соответствии с наличием свободного буферного пространства. А если учесть, что данные (в конечном счете) не могут быть отосланы без подтверждения, то можно говорить, что скорость передачи данных определяется скоростью поступления подтверждений на прием предыдущих посланных сегментов.

Однако при использовании протокола TCP скорость поступления подтверждений определяется так называемыми узкими местами между отправителем и получателем. Под термином «узкое место» понимается либо какое-то устройство, либо часть канала, имеющие значительно меньшие скоростные параметры, чем весь канал в целом. Этим узким местом может быть либо получатель, либо сама сеть.

Узкое место между отправителем и получателем может располагаться где угодно в сети. Узкие места могут быть логическими и физическими. На рис. 8.7 показан пример образования логических и физических узких мест. В этом примере отправитель имеет пропускную способность 10 Мбит/с. Поэтому для работы протокола TCP канал со скоростью 1.5 Мбит/с между маршрутизаторами становится узким местом. Это физическое узкое место. Так как скорость передачи невелика, то после достижения устойчивого состояния протокол TCP будет эффективно использовать доступную скорость. Однако наиболее часто узкие места являются

логическими и образуются из-за очередей на маршрутизаторах, коммутаторах или получателе. Задержки в очередях, как правило, подвержены флуктуациям и усложняют процесс формирования устойчивого потока.

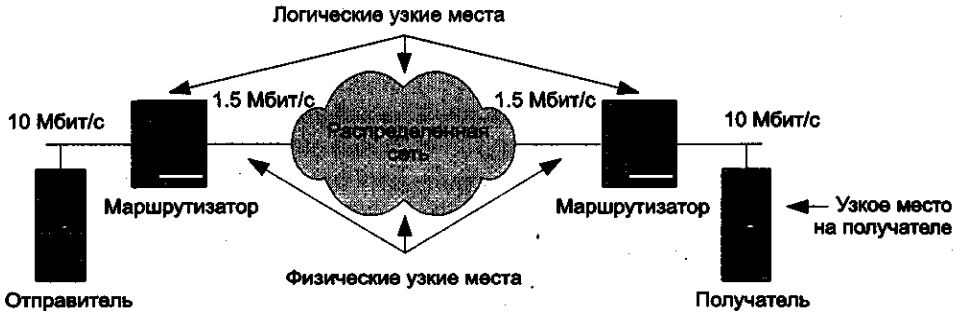

Рисунок 8.7 Пример логических и физических узких мест

Если множество отправителей TCP используют чрезмерно высокую скорость, то сегменты будут теряться при передачах, приводя к повторным передачам; или подтверждения будут сильно задерживаться, что приводит к ненужным повторным передачам. Более того, подобные повторные передачи могут иметь эффект положительный обратной связи: чем больше сегментов посылается повторно, тем больше растет перегрузка, что, в свою очередь, приводит к дальнейшему повышению задержек и к увеличению отброшенных сегментов. Все это вместе приводит к увеличению числа повторных передач, что в еще большей степени усугубляет перегрузку.

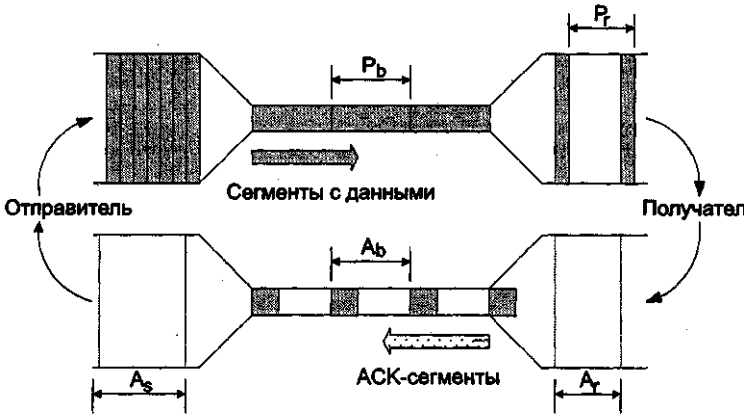

Рисунок 8.8. Пример узкого места в сети

Рассмотрим случай, когда узкое место находится где-то в сети. На рис. 8.8 показан низкоскоростной канал между отправителем и получателем. Ширина этого канала пропорциональна скорости передачи данных. Отправитель и получатель подключены к высокоскоростным сетям и поэтому каждый из них может работать с высокой скоростью. Канал из-за своей низкой скорости изображен на рис. 8.8 тонким. Он как раз и создает узкое место. На рис. 8.8 каждый посланный сегмент с данными изображается прямоугольником, чья площадь пропорциональна количеству байт информации в этом сегменте. Поэтому, когда сегмент проходит по узкому каналу, он как бы вытягивается в длину и, соответственно, увеличивается время его прохождения по каналу. Время прохождения сегмента через определенную точку медленного канала обозначим P_b: это разница между временами пересечения этой точки передней и задней границами сегмента. Предположим, что на медленном канале сегменты идут вплотную друг к другу, то есть задняя граница первого сегмента примыкает к передней границе второго сегмента. Поэтому P_b, определяет и время прохождения через точку в канале передних границ обоих сегментов. При поступлении сегментов в высокоростной канал это время (между передними границами) сохраняется даже с учетом повышения скорости передачи данных, так как время между поступлениями не меняется, А так как сегмент на высокоскоростном канале как бы сжимается по длине и увеличивается в ширину, то теперь это время состоит из времени прохождения самого сегмента и времени на паузу до передней границы следующего сегмента, то есть $P_r = P_b$. Если получатель подтверждает сегменты в момент поступления, то время между отсылками подтверждающих

сегментов (АСК-сегментов) A_r на выходе от получателя равно P_r. И, наконец, мы получаем, что АЬ будет равно A_r, а A_s равно A_b .

При переходе системы в устойчивое состояние после начальной попытки работать с большой скоростью скорость передачи сегментов сравняется со скоростью поступления подтверждений. Очевидно, что скорость отправки сегментов будет равна скорости самого медленного канала. Можно сказать, что протокол TCP автоматически определяет узкое место в сети и регулирует свой поток. Этот процесс называется самосинхронизацией (self-clocking). Самосинхронизация работает хорошо, если узким местом является получатель.