ПРОТОКОЛЫ ТРАНСПОРТНОГО УРОВНЯ TCP/IP

Протоколы транспортного уровня направлены на передачу информации между приложениями прикладного уровня, выполняющимися на узлах, подключенных к сети.

Для однозначной идентификации сетевого приложения, выполняющегося на машине сети, для протоколов транспортного уровня вводится такое понятие как *порты*. Если IP адрес определяет адрес узла в сети, то порт определяет адрес сетевого приложения в узле. Таким образом, связка IP адреса и порта определяют оконечную точку для доставки информации в сети. Порты задаются 16-битным числом от 0 до 65535.

1.1. Протокол UDP

Заголовок пакета

Протокол UDP (User Datagram Protocol – протокол датаграмм пользователя, номер протокола 17; RFC 768 [17]) является транспортным протоколом и использует датаграммный способ передачи информации. В силу своей простоты, протокол не обеспечивает надежность доставки пакетов, но при этом их быстрее передает.

На рисунке 2.1 представлен формат заголовка UDP-пакета.

Биты	0 15 16 31	
0-31	Порт отправителя (Source Port)	Порт получателя (Destination Port)
32-63	Длина пакета (Length)	Контрольная сумма (Checksum)

Рис. 2.1 Формат заголовка UDP-пакета

Назначение полей UDP пакета

Номер порта отправителя – (16 бит) – содержит номер порта, с которого отправляется пакет. Если нет необходимости указывать номер порта отправителя, то это поле заполняется нулями.

Номер порта получателя – (16 бит) – содержит номер порта получателя, на который доставляется пакет.

Длина – (16 бит) – содержит длину датаграммы в байтах, включая заголовок и данные.

Поле контрольной суммы – (16 бит) – представляет собой побитное дополнение 16-битной суммы 16-битных слов. В вычислении суммы участвуют: данные пакета с полями выравнивания (заполненные нулями) по 16-битной границе, заголовок UDP-пакета, псевдозаголовок IP пакета.

1.2. Протокол UDP Lite

Существует облегченная версия протокола UDP – UDP Lite (RFC 3828) [18]. Номер протокола 136. Протокол UDP Lite отличается вычислением контрольной суммы, в расчете, которой рассматривается только часть датаграммы. Таким образом, могут быть доставлены частично поврежденные пакеты.

1.3. Протокол TCP

2.3.1. Заголовок пакета TCP

TCP (Transmission Control Protocol – протокол управления передачей, номер протокола 6; RFC 793 [19]) является транспортным протоколом и ориентирован на соединение [1]. Протокол предназначен для управления передачей данных, направлен на обнаружение ошибок и обеспечение целостности передаваемой информации.

TCP – является *байтовым последовательным протоколом*: присваивается последовательный номер каждому передаваемому байту пакета.

На рисунке 2.2 представлен формат заголовка TCP-пакета.

<table>
<tr><td>Строка</td><td colspan="10">0 7 8 15 16 31</td></tr>
<tr><td>0</td><td colspan="5">Порт отправителя (Source Port)</td><td colspan="5">Порт получателя (Destination Port)</td></tr>
<tr><td>1</td><td colspan="10">Порядковый номер последовательности (Sequence number)</td></tr>
<tr><td>2</td><td colspan="10">Номер подтверждения (Acknowledgment Number)</td></tr>
<tr><td rowspan="2">3</td><td rowspan="2">Длина заголовка (Data offset)</td><td rowspan="2">Зарезервировано (Reserved)</td><td colspan="6">Флаги</td><td rowspan="2" colspan="2">Окно (Window)</td></tr>
<tr><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td></tr>
<tr><td>4</td><td colspan="5">Контрольная сумма (Checksum)</td><td colspan="5">Указатель важности (Urgent Pointer)</td></tr>
<tr><td>5-15</td><td colspan="8">Опции (Options)</td><td colspan="2">Выравнивание</td></tr>
</table>

Рис. 2.2. Формат заголовка TCP-пакета

Назначение полей TCP пакета

Номер порта отправителя (Source Port) – содержит номер порта отправителя. Если нет необходимости указывать номер порта отправителя, то это поле заполняется нулями. Поле занимает 16 бит.

Номер порта назначения (Destination Port) – содержит номер порта, на который будет доставлен пакет. Поле занимает 16 бит.

Порядковый номер (Sequence number) Поле занимает 32 бита и определяет порядковый номер пакета, если не установлен бит SYN. Установленный флаг SYN используется на этапе соединения. Этот процесс известен как «*трехэтапное рукопожатие»* (three-way handshake), во время которого происходит синхронизация порядковых номеров и номеров подтверждений для каждой стороны. Предполагается, что информация о соединении хранится в структуре данных TCB (Transmission Control Block - блок управления передачей). Записи TCB включают информацию о порядковых номерах для приема и передачи, номера локального и удаленного сокетов, опции безопасности и предпочтения для сегмента, указатели на пользовательские буферы приема и передачи, указатели на очередь повторной передачи и текущий сегмент.

Если установлен флаг SYN, то порядковый номер является начальным порядковым номером (ISN) и первый байт данных, которые будут переданы в следующем пакете, будет иметь номер, равный ISN +1.

Номер подтверждения (Acknowledgment Number) – (заполняется, если установлен бит ACK) поле занимает 32 бита и содержит порядковый номер байта, который отправитель данного сегмента ожидает получить. При этом подтверждается успешный прием всех предыдущих байт, с номерами от ISN+1 до ACK-1 включительно, данных.

Смещение данных (Data Offset) (занимает 4 бита) – задает длину заголовка TCP в 4-байтовых словах. Смещение считается от начала заголовка TCP.

Резервное поле (Reserved) (занимает 6 бит) – зарезервировано. В настоящее время 5-й и 6-й биты определены (RFC 3168 [20]): CWR (Congestion Window Reduced) – поле «Окно перегрузки уменьшено» — флаг установлен отправителем, чтобы указать, что получен пакет с установленным флагом ECE; ECE (ECN-Echo) – поле «Эхо ECN» — указывает, что данный узел способен на ECN (явное уведомление перегрузки) и для указания отправителю о перегрузках в сети.

Флаги управления:

URG – флаг важности, применяется при отправке экстренной информации;

ACK – флаг пакета, содержащего подтверждение получения;

PSH – флаг выталкивания, стимулирует получателя протолкнуть данные, накопившиеся в приёмном буфере, в приложение пользователя.

RST – флаг переустановки соединения: сбросить соединение, очистить буфер;

SYN – флаг синхронизации номеров последовательности;

FIN – флаг окончания передачи со стороны отправителя, завершение соединения.

Окно (Window) (занимает 16 бит) – содержит количество байт данных, которые отправитель данного пакета может принять без подтверждения. Отсчет байт начинается с последнего номера подтверждения, указанного в поле Acknowledgment Number. T.e., отправитель пакета указывает, что располагает, для приема данных, буфером размером "Window" байт.

Поле контрольной суммы (Checksum) (занимает 16 бит) – представляет собой 16-битное дополнение к сумме всех 16-битных слов заголовка (включая псевдозаголовок IP пакета) и данных. Если пакет, по которому вычисляется контрольная сумма, имеет длину не кратную 16-ти битам, то длина сегмента увеличивается до кратной 16-ти, за счет дополнения к нему справа нулевых битов заполнения. При вычислении контрольной суммы *поле контрольной суммы* принимается равным нулю.

Указатель важности (Urgent Pointer) (занимает 16 бит) – поле указывает порядковый номер байта, которым заканчиваются важные (urgent) данные от порядкового номера в данном сегменте и интерпретируется только в пакетах с установленным флагом URG;

Опции (Options) – поле имеет переменную длину и может применяться в некоторых случаях для расширения протокола. Например, для тестирования. В настоящее время момент в опции практически всегда включают 2 байта NOP (NOOP, в данном случае 0x01) и 10 байт, задающих timestamps (временная метка). Определить размер поля опции можно через значение поля смещения.

Заполнение (Padding) – имеет переменную длину и используется для выравнивания заголовка по 32-битному слову нулевыми значениями.

2.3.2. Максимальный размер сегмента

В TCP для определения наибольшего размера принимаемых на обработку данных в одном пакете, вводится понятие *"максимальный размер сегмента"* (maximum segment size – MSS) [21], который измеряется в байтах и определяется как:

MSS = Размер наибольшей датаграммы, которую можно принять — (сумма длин заголовков IP и TCP).

В случае IPv4 длина заголовка по умолчанию равна 20, для IPv6 длина заголовка равна 40.

2.3.3. Процедура «окно»

Для начала поясним работу процедуры «окно».

Под *«окном» понимается максимальный объем данных, который может быть передан от отправителя к получателю без подтверждения о правильности приема*.

В каждой стороне соединения выделяется буферная память для приема данных, с последующей их обработкой. По мере обработки данных, буфер освобождается. Прежде чем принять очередную порцию данных от отправителя, *получатель* в очередном пакете (к *отправителю*) в заголовке протокола TCP указывает W – размер окна, свободного места в буферной памяти (см. рисунок 2.3). Исходя из размера W, *отправитель* определяет размер *нагрузочного окна* $W^* \le W$. После чего передается порция данных размером W^* байт с последующим ожиданием подтверждения их приема от *получателя*.

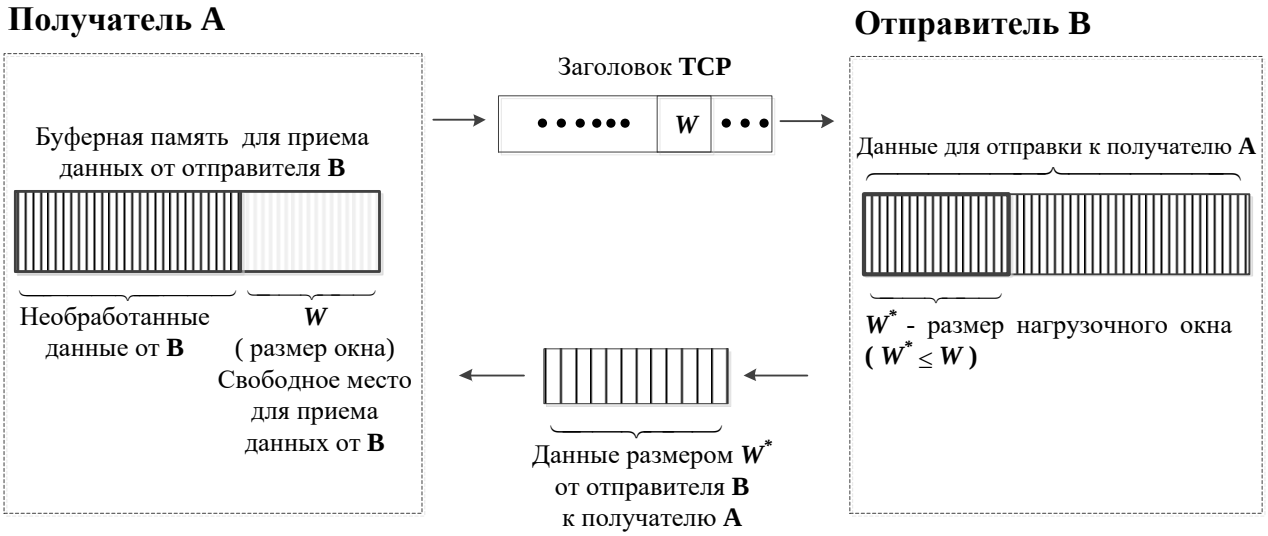

Рис. 2.3. Схема работы механизма процедуры «окно» протокола TCP

Производительность ресурсов может ограничиваться по многим причинам, из которых основными являются память и полоса пропускания.

Полоса пропускания, задержки, плохое качество пересылки данных приводит к снижению эффективности полосы пропускания.

Подстраиваясь под реалии сети, приемная сторона, должна обеспечить достаточное буферное пространство TCP, позволяющее отправителю выполнять пересылку данных без пауз в работе. Итак, в процедуре «окно» применяется простое правило:

Для поддержания устойчивого потока данных от источника принимающая сторона должна иметь окно размером не менее чем произведение полосы пропускания на задержку.

Например, если узел отправителя отсылает данные со скоростью 10 000 байт/с, а на подтверждение АСК тратится 0,5 с, то узлу приемника необходимо обеспечить приемное окно размером не менее 5 000 байт, иначе будут перерывы в потоке данных.

2.3.4. Проблемы и алгоритмы TCP

В работе [1] рассматриваются проблемы (и алгоритмы их решения), влияющие на производительность TCP, краткое описание которых представляем ниже.

- *Медленный старт*(slow start), это процесс аккуратного входа в передачу информации, чтобы сеть адаптировалась под новый сеанс, и тем самым избежала непроизводительные потери.
- *Алгоритмы борьбы с синдромом "бестолкового окна"* (silly window syndrome) предохраняют от перегрузки сети сообщениями.
- *Задержка АСК* (delayed АСК) позволяет снизить перегрузку за счет сокращения количества сообщений подтверждения пересылки данных.
- *Вычисляемый тайм-аут повторной пересылки* (computing retransmission timeout), в реальном времени, позволяет найти оптимальное решение для сведения к минимуму ненужных повторных пересылок.
- *Торможение пересылки TCP* при перегрузках в сети позволяет маршрутизаторам вернуться в исходный режим, адаптироваться, и совместно использовать сетевые ресурсы для всех сеансов.
- Отправка *дублированных АСК* (duplicate АСК) при получении сегмента вне последовательности отправки позволяет сторонам выполнить повторную пересылку до наступления тайм-аута.

Здесь же рассмотрим проблему *синдрома "бестолкового окна",* представленную следующим сценарием (см. рисунок 2.4):

1. Передающая сторона (отправитель В) быстро отсылает данные.
2. Принимающая сторона (получатель А) медленно читает данные из буфера по 1 байту.
3. В силу быстрого поступления и медленной обработки буфер быстро заполняется.
4. Из заполненного буфера принимающая сторона читает 1 байт, и TCP отправляет АСК, означающий "У меня есть свободное место равное 1 байт данных".
5. Передающая сторона отправляет по сети пакет TCP из 1 байта.
6. В результате буфер принимающей стороны заполнен и по TCP посылается АСК – "Пакет получен, свободного места нет!".
7. Далее, как освободится место, например, в 1 байт, отправляется АСК "У меня есть свободное место равное 1 байт данных".
8. Принимающее приложение опять читает 1 байт и отправляет АСК и весь процесс повторяется с шага 6.

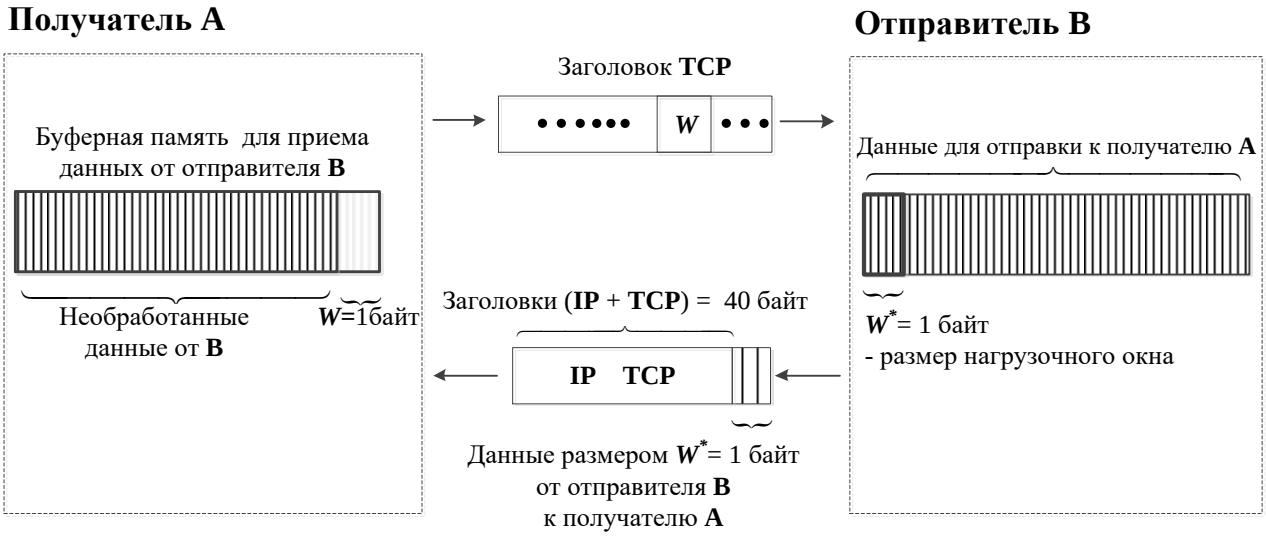

Рис. 2.4. Пример феномена синдрома «бестолкового окна»

Одним из решений является следующий алгоритм. Как только размер окна уменьшается до определенной величины, TCP начинает извещать отправителя о том, что свободного места нет. После того как буферное пространство освободится свыше определенного размера, получатель сообщает отправителю о наличии свободного места для приема данных.

Размер свободного места для принятия данных, о котором сообщается получателем TCP отправителю, можно выразить как:

$$min(1/2 \text{ буфера, } MSS)$$

Таким образом, TCP не сообщает о наличии свободного места в буфере, пока размер окна не станет больше этого размера.

Вернемся к заголовку протокола TCP. Проанализируем возможности протокола TCP в рамках полей «Порядковый номер последовательности» и «Окно».

Пространство имен в поле «Порядковый номер последовательности» ограничено 32 битами, т.е. максимум имеем 4 Гига (2^{32}) имен. Предположим, что у нас есть возможность пересылать данные по соединению со скоростью 4 Гбайт/с, то для сохранения целостности информации порядковые номера должны обновляться в течение каждой секунды. Таким, образом каждую секунду порядковые номера будут повторяться. В случае появления отсроченных датаграмм более чем на секунду, то теряется возможность отделить эти датаграммы от новых с одинаковыми порядковыми номерами стандартными средствами TCP.

Далее, пусть полоса пропускания позволяет передавать информацию со скоростью 10 млн. бит/с, а время цикла составляет 100 мс (1/10 секунды). Тогда, на основании правила для поддержания устойчивого потока данных в заданном соединении, приемное окно должно быть размером, по крайней мере, 1 000 000 бит = 125 000 байт. Но наибольшее число, которое можно записать в поле «Окно» заголовка TCP, равно 65 535. Таким образом, при заданных начальных условиях, максимально возможная скорость получения снизится почти вдвое и протокол TCP имеет барьер на скорость передачи данных.

Алгоритм Нейгла

Отправитель должен независимо от получателя исключить пересылку очень коротких сегментов, аккумулируя данные перед отправлением [1]. Алгоритм Нейгла (*Nagle*) реализует очень простую идею, позволяющую снизить количество пересылаемых по сети коротких датаграмм.

Алгоритм рекомендует задержать пересылку данных (и их выталкивание) на время ожидания АСК от ранее переданных данных. Аккумулируемые данные пересылаются после получения АСК на ранее отправленную порцию информации, либо после получения для отправки данных в размере полного сегмента или по завершении тайм-аута. Этот алгоритм не следует применять для приложений реального времени, которые должны отправлять данные как можно быстрее.

3.2.7.Задержанный АСК

Еще одним механизмом повышения производительности является способ задержки АСК [1]. Сокращение числа АСК снижает полосу пропускания, которую можно использовать для пересылки другого трафика. Если партнер по TCP чуть задержит отправку АСК, то:

- можно подтвердить прием нескольких сегментов одним АСК;
- принимающее приложение способно получить некоторый объем данных в пределах интервала тайм-аута, т.е. в АСК может попасть выходной заголовок, и не потребуется формирование отдельного сообщения.

С целью исключения задержек при пересылке потока полноразмерных сегментов (например, при обмене файлами) АСК должен отсылаться, по крайней мере, для каждого второго полного сегмента.

Многие реализации используют тайм-аут в 200 мс. Но задержанный АСК не снижает скорость обмена. При поступлении короткого сегмента во входном буфере остается еще достаточно свободного места для получения новых данных, а отправитель может продолжить пересылку (кроме того, повторная пересылка обычно выполняется гораздо медленнее). Если же поступает целый сегмент, нужно в ту же секунду ответить на него сообщением АСК.

3.2.8.Тайм-аут повторной пересылки

После отправки сегмента TCP устанавливает таймер и отслеживает поступление АСК[1]. Если АСК не получен в течение периода тайм-аута, TCP выполняет повторную пересылку сегмента (ретрансляцию). Однако каким должен быть период тайм-аута?

Если он слишком короткий, отправитель заполнит сеть пересылкой ненужных сегментов, дублирующих уже отправленную информацию. Слишком же большой тайм-аут будет препятствовать быстрому исправлению действительно разрушенных при пересылке сегментов, что снизит пропускную способность.

Как выбрать правильный промежуток для тайм-аута? Значение, пригодное для высокоскоростной локальной сети, не подойдет для удаленного соединения с множеством попаданий. Значит, принцип "одно значение для любых условий" явно непригоден. Более того, даже для существующего конкретного соединения могут измениться сетевые условия, а задержки — увеличиться или снизиться.

Алгоритмы Джекобсона, Керна и Партриджа (описанные в статьях CongestionAvoidanceandControl**,** VanJacobson, и ImprovingRound-TripTimeEstimatesinReliableTransportProtocols,Karn и Partridge) позволяют адаптировать

TCP к изменению сетевых условий. Эти алгоритмы рекомендованы к использованию в новых реализациях. Мы кратко рассмотрим их ниже.

Здравый смысл подсказывает, что наилучшей основой оценки правильного времени тайм-аута для конкретного соединения может быть отслеживание *времени цикла*(round-triptime) как промежутка между отправкой данных и получением подтверждения об их приеме.

Хорошие решения для следующих величин можно получить на основе элементарных статистических сведений, которые помогут вычислить время тайм-аута. Однако не нужно полагаться на усредненные величины, поскольку более половины оценок будет больше, чем среднестатистическая величина. Рассмотрев пару отклонений, можно получить более правильные оценки, учитывающие нормальное распределение и снижающие слишком долгое время ожидания повторной пересылки.

Нет необходимости в большом объеме вычислений для получения формальных математических оценок отклонений. Можно использовать достаточно грубые оценки на основе абсолютной величины разницы между последним значением и среднестатистической оценкой:

Последнее отклонение = |*Последний цикл* - *Средняя величина*|

Для вычисления правильного значения тайм-аута нужно учитывать еше один фактор — изменение времени цикла из-за текущих сетевых условий. Происходившее в сети в последнюю минуту более важно, чем то, что было час назад.

Предположим, что вычисляется среднее значение цикла для очень длинного по времени сеанса. Пусть вначале сеть была мало загружена, и мы определили 1000 небольших значений, однако, далее произошло увеличение трафика с существенным увеличением времени задержки.

Например, если 1000 значений дали среднестатистическую величину в 170 единиц, но далее были замерены 50 значений со средним в 282, то текущее среднее будет:

170 x 1000/1050 + 282 x 50/1050 = 175

Более резонной будет величина сглаженного времени цикла(*Smoothed Round-Trip Time — SRTT*), которая учитывает приоритет более поздних значений:

Новое SRTT = (1 — a) x *(старое SRTT) + a*x*Последнее значение цикла*

Значение*a* находится между 0 и 1. Увеличение*a*приводит к большему влиянию текущего времени цикла на сглаженное среднее значение. Поскольку компьютеры быстро могут выполнять деление на степени числа 2, сдвигая двоичные числа направо, для*a* всегда выбирается значение 1/2 в степени (обычно 1/8), поэтому:

Новое SRTT - 7/8 x *старое SRTT + 1/8* x *Последнее время цикла*

Теперь возникает вопрос о выборе значения для тайм-аута повторной пересылки. Анализ величин времени цикла показывает существенное отклонение этих значений от текущей средней величины. Имеет смысл установить границу для величины отклонений (девиаций). Хорошие величины для тайм-аута повторной пересылки (в стандартах RFC

эту величину именуют RetransmissionTimeOut — RTO) дает следующая формула с ограничением сглаженного отклонения (SDEV):

$$T = \text{Тайм-аут повторной пересылки} = SRTT + 2 \times SDEV$$

Именно эта формула рекомендована в RFC 1122. Однако некоторые реализации используют другую:

$$T = SRTT + 4 \times SDEV$$

Для вычисления *SDEV* сначала определяют абсолютное значение текущего отклонения:

$$DEV = | \text{Последнее время цикла} — \text{старое } SRTT$$

Затем используютформулу сглаживания, чтобы учесть последнее значение:

$$\text{Новое } SDEV = 3/4 \times \text{старое } SDEV + 1/4 \times DEV$$

Начальный тайм-аут = 3 с
Начальный SRTT = 0
Начальный SDEV = 1,5 с
Ван Джекобсон определил быстрый алгоритм, который очень эффективно вычисляет тайм-аут повторной пересылки данных.

3.2.9.Вычисления после повторной отправки

В представленных выше формулах используется значение времени цикла как интервала между отправкой сегмента и получением подтверждения его приема [1]. Однако предположим, что в течение периода тайм-аута подтверждение не получено и данные должны быть оправлены повторно.

Алгоритм Керна предполагает, что в этом случае не следует изменять время цикла. Текущее сглаженное значение времени цикла и *сглаженное отклонение* сохраняют свои значения, пока не будет получено подтверждение на пересылку некоторого сегмента без его повторной отправки. С этого момента возобновляются вычисления на основе сохраненных величин и новых замеров.

Но что происходит до получения подтверждения? После повторной пересылки поведение TCP радикально меняется в основном из-за потери данных от перегрузки в сети. Следовательно, реакцией на повторную отправку данных будет:

- Снижение скорости повторной пересылки.
- Борьба с перегрузкой сети с помощью сокращения общего трафика

3.2.10.Экспоненциальное торможение

После повторной пересылки удваивается интервал тайм-аута. Однако, что произойдет при повторном переполнении таймера? Данные будут оправлены еще раз, а период повторной пересылки снова удвоится. Этот процесс называется *экспоненциальным торможением* (exponential backoff) [1].

Если продолжает проявляться неисправность сети, период тайм-аута будет удваиваться до достижения предустановленного максимального значения (обычно — 1

мин). После тайм-аута может быть отправлен только один сегмент. Тайм-аут наступает и при превышении заранее установленного значения для количества пересылок данных без получения АСК.

3.2.11.Снижение перегрузок за счет уменьшения пересылаемых по сети данных

Сокращение объема пересылаемых данных несколько сложнее, чем рассмотренные выше механизмы[1]. Оно начинает работать, как и уже упомянутый медленный старт. Но, поскольку устанавливается граница для уровня трафика, который может в начальный момент привести к проблемам, будет реально замедляться скорость обмена вследствие увеличения размера нагрузочного окна по одному сегменту. Нужно установить значения границы для реального сокращения скорости отправки. Сначала вычисляется граница Опасности (danger threshold):

Граница = 1/2minimum (текущее нагрузочное окно, приемное окно партнера)

Если полученная величина будет более двух сегментов, ее используют как границу. Иначе размер границы устанавливается равным двум сегментам. Полный алгоритм восстановления требует:

- Установить размер нагрузочного окна в один сегмент.
- Для каждого полученного АСК увеличивать размер нагрузочного окна на один сегмент, пока не будет достигнута граница (что очень напоминает механизм медленного старта).
- После этого с каждым полученным АСК к нагрузочному окну добавлять меньшее значение, которое выбирается на основе скорости увеличения по одному сегменту для времени цикла (увеличение вычисляется как *MSS/N*, где *N* — размер нагрузочного окна в сегментах).

Сценарий для идеального варианта может упрощенно представить работу механизма восстановления. Предположим, что приемное окно партнера (и текущее нагрузочное окно) имело до выявления тайм-аута размер в 8 сегментов, а граница определена в 4 сегмента. Если принимающее приложение мгновенно читает данные из буфера, размер приемного окна останется равным 8 сегментам.

1. Отправляется 1 сегмент (нагрузочное окно = 1 сегмент).
2. Получен АСК — отправляется 2 сегмента.
3. Получен АСК для 2 сегментов — посылается 4 сегмента, (достигается граница).
4. Получен АСК для 4 сегментов. Посылается 5 сегментов.
5. Получен АСК для 5 сегментов. Посылается 6 сегментов.
6. Получен АСК для 6 сегментов. Посылается 7 сегментов.
7. Получен АСК для 7 сегментов. Посылается 8 сегментов (нагрузочное окно по размеру снова стало равно приемному окну).

Поскольку во время повторной пересылки по тайм-ауту требуется подтверждение приема всех отправленных данных, процесс продолжается до достижения нагрузочным окном размера приемного окна. Размер окна увеличивается экспоненциально, удваиваясь во время периода медленного старта, а по достижении границы увеличение происходит по линейному закону.

3.2.12.Дублированные АСК

В некоторых реализациях применяется необязательная возможность — так называемая *быстрая повторная пересылка* (fast retransmit) — с целью ускорить

повторную отправку данных при определенных условиях[1]. Ее основная идея связана с отправкой получателем дополнительных АСК, указывающих на пробел в принятых данных.

Принимая сегмент, поступивший не по порядку, получатель отсылает обратно АСК, указывающий на первый байт *потерянных* данных

Отправитель не выполняет мгновенной повторной пересылки данных, поскольку IP может и в нормальном режиме доставлять данные получателю без последовательности отправки. Но когда получено несколько дополнительных АСК на дублирование данных (например, три), то отсутствующий сегмент будет отправлен, не дожидаясь завершения тайм-аута.

Отметим, что каждый дублирующий АСК указывает на получение сегмента данных. Несколько дублирующих АСК позволяют понять, что сеть способна доставлять достаточный объем данных, следовательно, не слишком сильно нагружена. Как часть общего алгоритма выполняется небольшое сокращение размера нагрузочного окна при реальном увеличении сетевого трафика. В данном случае процесс радикального изменения размера при восстановлении работы не применяется.

3.2.13. Барьеры для производительности

TCP доказал свою гибкость, работая в сетях со скоростью обмена в сотни или миллионы бит за секунду. Этот протокол позволил достичь хороших результатов в современных локальных сетях с топологиями Ethernet, Token-Ring и Fiber Distributed DataInterlace (FDDI), а также для низкоскоростных линий связи или соединений на дальние расстояния (подобных спутниковым связям).

TCP разработан так, чтобы реагировать на экстремальные условия, например на перегрузки в сети. Однако в текущей версии протокола имеются особенности, ограничивающие производительность в перспективных технологиях, которые предлагают полосу пропускания в сотни и тысячи мегабайт [1]. Чтобы понять возникающие проблемы, рассмотрим простой (хотя и нереалистичный) пример.

Предположим, что при перемещении файла между двумя системами необходимо выполнять обмен непрерывным потоком настолько эффективно, насколько это возможно. Допустим, что:

- максимальный размер сегмента приемника — 1 Кбайт;
- приемное окно — 4 Кбайт;
- полоса пропускания позволяет пересылать по два сегмента за 1 с.;
- принимающее приложение поглощает данные по мере их поступления;
- сообщения АСК прибывают через 2 с.

Отправитель способен посылать данные непрерывно. Ведь когда выделенный для окна объем будет заполнен, прибывает АСК, разрешающий отправку другого сегмента:

SEND SEGMENT 1.
SEND SEGMENT 2.
SEND SEGMENT 3.
SEND SEGMENT 4.

Через 2 с:
RECEIVE ACK OF SEGMENT 1, CAN SEND SEGMENT 5.

RECEIVE ACKOF SEGMENT 2, CAN SEND SEGMENT 6.
RECEIVE ACKOF SEGMENT 3, CAN SEND SEGMENT 7.
RECEIVE ACKOF SEGMENT 4, CAN SEND SEGMENT 8.

Ещечерез 2 с:
RECEIVE ACK OF SEGMENT 5, CAN SEND SEGMENT 9.

Если приемное окно было только в 2 Кбайт, отправитель будет вынужден ждать одну секунду из каждых двух перед отправкой следующих данных. Фактически, чтобы удержать непрерывный поток данных, приемное окно должно иметь размер не менее:

Окно = Полоса пропускания x *Время цикла*

Хотя пример несколько преувеличен (для обеспечения более простых чисел), малое окно может привести к проблемам при соединениях через спутниковые связи с большой задержкой.

Теперь рассмотрим, что происходит с высокоскоростными соединениями. Например, если полоса пропускания и скорость пересылки измеряются 10 млн. бит в секунду, но время цикла составляет 100 мс (1/10 секунды), то для непрерывного потока приемное окно должно хранить, по крайней мере, 1 000 000 бит, т.е. 125 000 байт. Но наибольшее число, которое можно записать в поле заголовка для приемного окна TCP, равно 65 536.

Другая проблема возникает при высоких скоростях обмена, поскольку порядковые номера очень быстро закончатся. Если по соединению можно будет пересылать данные со скоростью 4 Гбайт/с, то порядковые номера должны обновляться в течение каждой секунды. Не будет возможности различить старые дублирующие датаграммы, которые были отсрочены более чем на секунду при перемещении по Интернету, от свежих новых данных.

Сейчас активно проводятся новые исследования для улучшения TCP/IP и устранения упомянутых выше препятствий.