МЕТОДЫ ВЗАИМОДЕЙСТВИЯ В ЗВЕНЕ СЕТИ ПЕРЕДАЧИ ДАННЫХ

При формировании полномасштабной сети, то есть при объединении разрозненных физических сегментов в СПД той или иной сложности, возникает ряд специфических задач, направленных на оптимизацию взаимодействия между абонентами.

Упомянутые задачи решают на третьем и четвертом уровнях модели OSI.

Новые задачи обусловлены серьезными отличиями процессов передачи-приема пакета в пределах сегмента и между сегментами. Основные отличия заключаются в необходимости ретрансляций пакетов, а так же в возможном наличии альтернативных путей.

Одним из ключевых терминов транспортного уровня является термин *соединение* (connection). По сути дела, понятие соединения связано с понятием готовности. Если абоненты находятся в состоянии «нормальной готовности» передавать или принимать данные, то считают, что между ними установлено соединение. С учетом абстрагирования от более низких уровней модели OSI и инкапсуляции, соединение может быть выражено неявно.

Нужно отличать *виртуальные соединения* (virtual connections) от *физических соединений* (physical connections). Абоненты-программы физически (явно) соединены быть не могут. Следовательно, применительно к ним, соединения являются сугубо виртуальными.

Следует также учитывать, что нормальная готовность может рассматриваться в двух ракурсах:
1. Организация взаимодействия абонентов-программ.
2. Настройка задействованного промежуточного оборудования.
В первом случае речь идет о собственно виртуальных соединениях транспортного уровня, во втором -- о *виртуальных цепях* (virtual circuits) сетевого или канального уровней.

В свою очередь, виртуальные цепи бывают:
1. PVCs (Permanent Virtual Circuits) -- *выделенные виртуальные цепи*.
2. SVCs (Switched Virtual Circuits) -- *коммутируемые виртуальные цепи* (в отечественной литературе иногда называют *виртуальными вызовами*).

Термин *виртуальный канал* (virtual channel) может в равной степени подходить как к виртуальным соединениям, так и к виртуальным цепям.

При разговоре о соединениях невозможно обойти стороной вопрос о надежности. Существуют два способа организации взаимодействия:
1. Без гарантированной доставки -- в СПД предпринимаются определенные усилия по доставке пакетов, но при этом ничего не гарантируется (при необходимости, соответствующий контроль возлагается на программы-абоненты).
2. С гарантированной доставкой -- алгоритм работы транспортной службы гарантирует доставку пакетов (программы-абоненты могут не контролировать наличие и очередность пакетов).
Однако, соединение без гарантированной доставки практического смысла не имеет.

Поэтому наличие соединения как правило говорит о надежности.

В общем случае, контроль передачи информации посредством СПД предотвращает не только потерю пакетов, но и искажение их содержимого. Отсутствие соединения не означает, что защита от сбойных пакетов отсутствует.

Простейшим подходом к обеспечению контроля доставки информационных пакетов является применение метода, который обобщенно можно назвать методом *запросов-подтверждений* (requests/acknowledges). Метод предполагает некоторое разнообразие и заключается в том, что вводят специальные служебные пакеты двух типов. Пакет-запрос используется при получении права принять или передать полезные данные, а также собственно при запросе данных. Пакет-подтверждение (в отечественной литературе часто называют *квитанцией*) передается в ответ на пакет-запрос или после приема полезных данных.

Кроме того, при реализации метода запросов-подтверждений следует учитывать следующие обстоятельства:
-- инициатором взаимодействия может быть передатчик либо приемник информационных пакетов;
-- контроль может осуществляться передатчиком либо приемником, либо передатчиком и приемником совместно;
-- запросы либо подтверждения могут отсутствовать вообще;
-- запросы могут комбинироваться с подтверждениями;
-- запрашиваться и подтверждаться может все сообщение либо каждый из пакетов;
-- подтверждаться могут не только информационные, а и служебные пакеты;
-- квитанции могут быть как положительными, так и отрицательными;
-- факт потери пакета может определяться и обрабатываться по-разному.

Таким образом, не смотря на сохранение идеологии, практические реализации метода запросов-подтверждений могут сильно различаться.

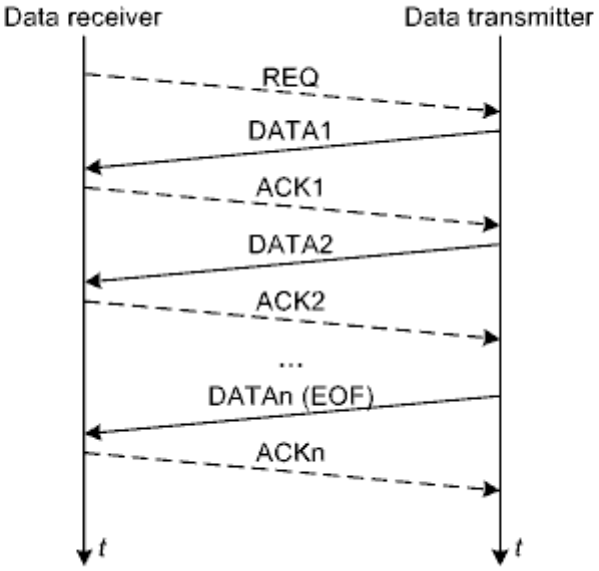

Рисунок -- Пример взаимодействия методом запросов-подтверждений

На практике, метод запросов-подтверждений невозможно реализовать без одного существенного дополнения.

Функционирование механизма запросов-подтверждений подразумевает ожидание определенных событий. Ожидание, в любом случае, не должно затягиваться до бесконечности. Ограничение ожидания во времени достигается за счет применения тайм-аута (time-out). После передачи некоторого служебного или информационного пакета, требующего подтверждения, запускается таймер с обратным отсчетом. Если в течение заданного интервала времени соответствующая квитанция не приходит, то пакет считается утерянным и передается повторно (retransmission). Если квитанция не приходит снова и снова, то после некоторого конечного количества попыток дальнейшая передача считается бесперспективной и прекращается.

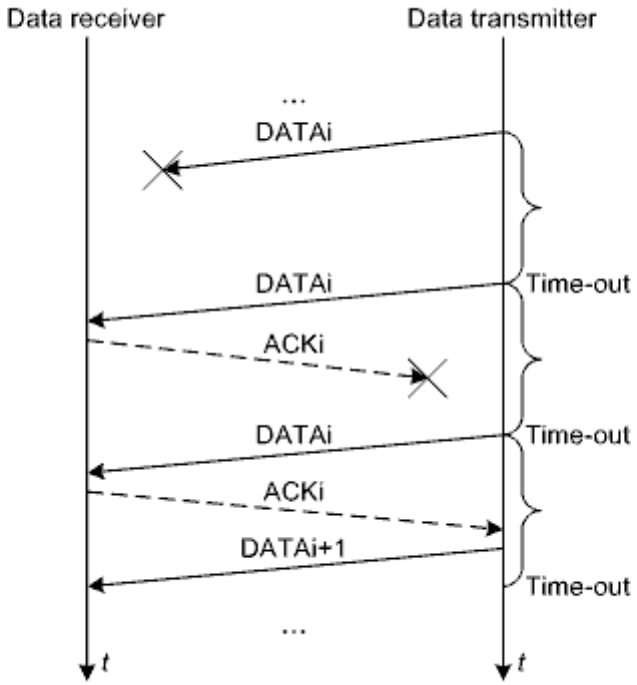

Рисунок -- Пример взаимодействия с учетом тайм-аута

Следует учитывать, что:
-- теряться могут как информационные пакеты, так и квитанции и пакеты-запросы;
-- если квитанция приходит позже наступления тайм-аута, то этот факт приравнивается к ее потере;
-- оптимальное время ожидания квитанций, применительно к некоторой СПД, зависит от ее особенностей.

В случае, когда СПД загружена незначительно, а взаимодействующие абоненты расположены далеко друг от друга, задействование классического механизма запросов-подтверждений приводит к неэффективному использованию ресурсов. Время, затрачиваемое на ожидание квитанций, становится недопустимо большим в сравнении с временем, затрачиваемым на передачу полезных данных. Оптимизировать обмен позволяет применение оконного (window) метода, суть которого состоит в том, что до перехода к ожиданию квитанций передается не один, а несколько пакетов.

Выделяют два основных критерия классификации оконных методов.

Исходя из количества пакетов, передаваемых в окне, оно может быть:
1. *Статическим* (static) -- неизменяемый размер окна заложен в протокол или устанавливается на весь сеанс обмена.

2. *Динамическим* (dynamic) -- размер окна может изменяться (увеличиваться или уменьшаться) в процессе передачи сообщения.

Исходя из способа обработки очереди пакетов, окно может быть:
1. *Фиксированным* (fixed) -- перед формированием следующего окна текущее должно быть полностью «закрыто», то есть должны быть приняты все необходимые квитанции.
2. *Скользящим* (sliding) -- существует возможность сдвигать окно относительно последовательности пакетов.

При реализации оконного метода следует учитывать следующие дополнительные обстоятельства:
-- должен быть установлен начальный размер окна;
-- нужна нумерация пакетов в том или ином виде;
-- подтверждаться может как все окно, так и каждый из пакетов;
-- размером окна может управлять как передатчик, так и приемник;
-- размером окна можно управлять посредством служебных полей, в том числе и в информационных пакетах;
-- окно, с которым работает передатчик, может отличаться от окна, с которым работает приемник;
-- иногда важен порядок доставки пакетов.

С точки зрения реализации, наиболее простым является статическое окно фиксированного размера.

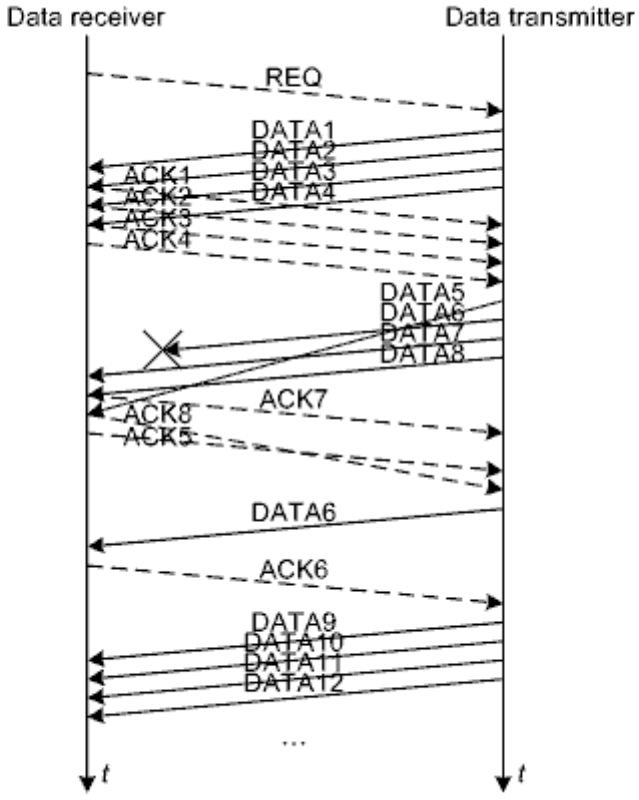

Рисунок -- Пример статического окна

Основной его недостаток состоит в отсутствии возможности адаптации к изменениям в СПД.

Первым вариантом усложнения является переход к динамическому окну.

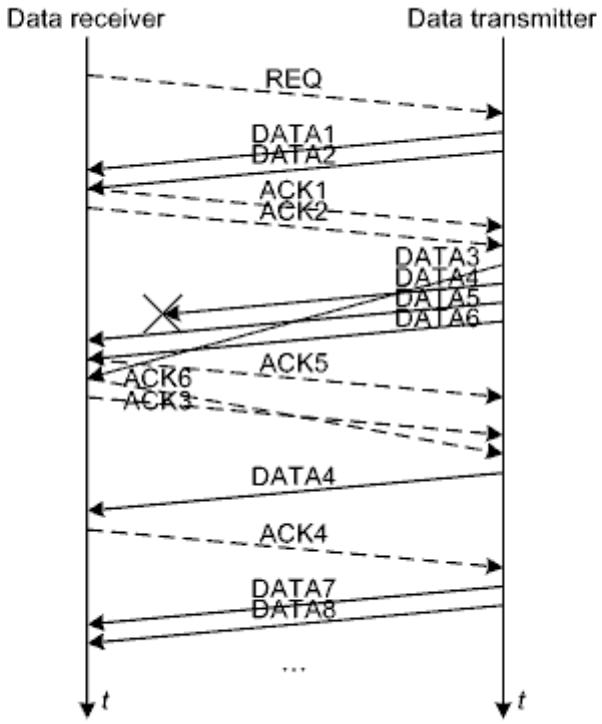

Рисунок -- Пример динамического окна

Динамическое окно позволяет успешно адаптироваться к изменениям в СПД. При увеличении загруженности окно целесообразно сужать, а при снижении -- расширять.

Вторым вариантом усложнения является переход к скользящему окну.

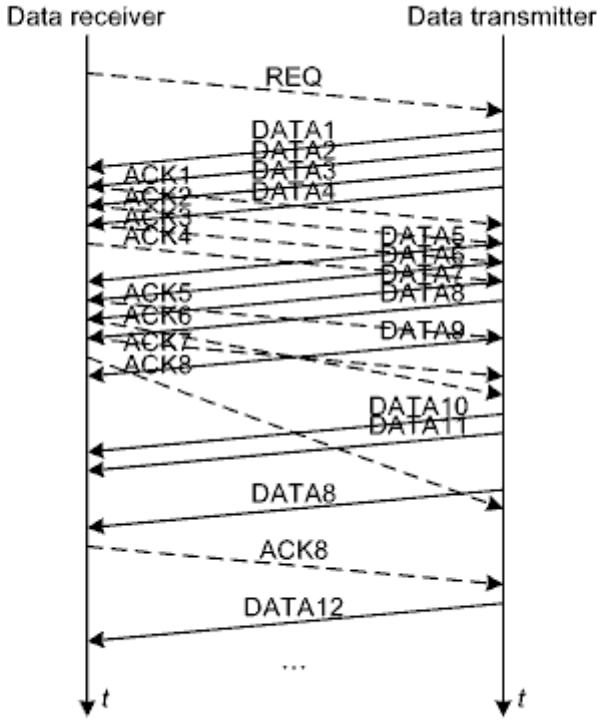

Рисунок -- Пример скользящего окна

Скользящее окно, особенно в сочетании с динамическим, позволяет ускорить адаптацию к топологическим и другим изменениям в СПД.

Таким образом, наиболее сложным является динамическое скользящее окно.

Классической реализацией оконного метода является оконный механизм протокола транспортного уровня TCP (Transmission Control Protocol) (основное RFC -- RFC 793). Протокол обеспечивает установление надежного соединения между сугубо пользовательскими или другими видами приложений, то есть доставка данных в правильном порядке гарантируется.

В стандарте TCP описано динамическое скользящее окно.

TCP соответствует клиент-серверной модели.

Сокет (socket) -- это «привязка» к виртуальному каналу, соединяющему между собой два взаимодействующих сетевых процесса, с точки зрения одного (любого) из этих процессов, причем с учетом всех трех уровней адресации.

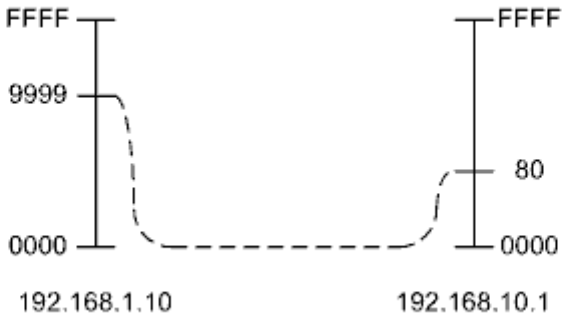

Рисунок -- TCP-соединение

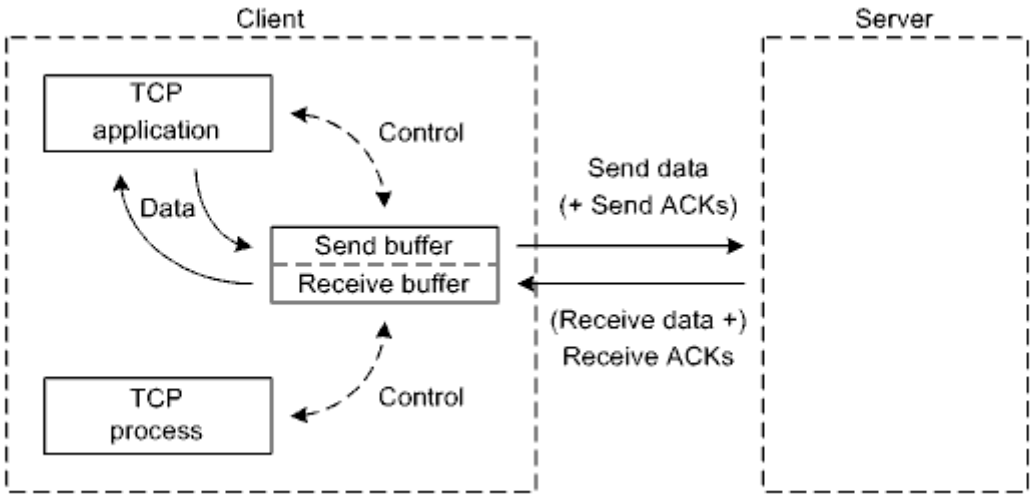

Рисунок -- Структура системы TCP

Применительно к каждому TCP-соединению нужно выделять приложение, производящее или потребляющее сетевые данные, и TCP-процесс, предоставляющий коммуникационные услуги (например, специальный драйвер ОС). Синхронизировать работу приложения и TCP-процесса можно только с помощью буферизации. TCP-интерфейс, которым пользуется приложение, состоит из примитивов для работы с буфером, позволяющих контролируя записывать или считывать данные. Доступ к буферу имеет и TCP-процесс, который отслеживает наполнение буфера и, используя ресурсы более низких уровней, организует прием или передачу данных.

Предназначенное для передачи сообщение разбивается на сегменты.

Минимальной учитываемой в окне единицей данных является октет, то есть байт. Все байты сообщения последовательно нумеруются так называемыми последовательными номерами -- SNs (Sequence Numbers). Нумерация начинается с некоторого начального последовательного номера -- ISN (Initial Sequence Number), который как правило не равен нулю, а генерируется реализациями случайно (например, на основе текущего времени) для того чтобы лучше управлять соединениями (например, после их ненормальных завершений). Принято, что сам ISN в нумерацию байтов не включается, то есть номер первого байта сообщения больше ISN на единицу.

Номером сегмента является SN первого байта данных в нем. По разным понятным причинам длина сегмента может варьировать, но она имеет ограничение. Поэтому важное значение имеет конфигурационный параметр MSS (Maximum Segment Size) -- максимальная длина сегмента (по умолчанию 536 байтов).

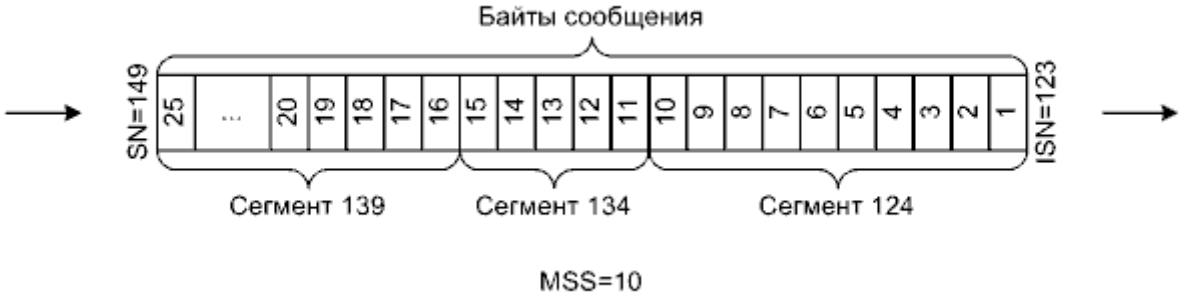

Рисунок -- Пример сегментации TCP-сообщения

В стандарте выделяют несколько видов окон, которые нужно различать.

Благодаря гибкости протокола, передающий и принимающий TCP-процессы работают с разными окнами, то есть, в первую очередь, следует отдельно рассматривать окно передачи (send window) и окно приема (receive window).

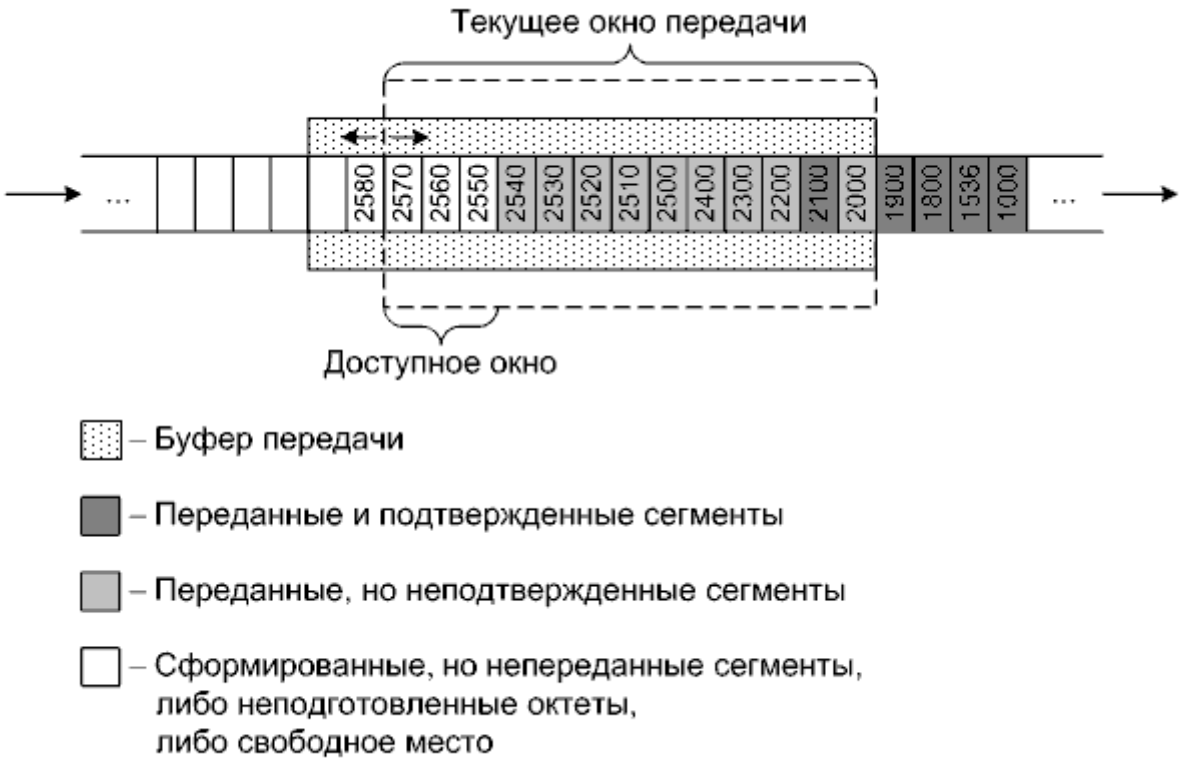

Рисунок -- Организация буфера передачи TCP

Передающее приложение последовательно, «порциями», записывает блоки байтов сообщения, возможно разной длины, в буфер передачи. Длина сообщения и размер буфера -- это вещи независимые, они почти всегда различаются. TCP-процесс формирует из имеющихся в буфере данных соответствующее количество сегментов и последовательно отправляет их. В любой момент времени текущее окно (current window) передачи имеет некоторый установленный размер и характеризуется тем, что все попадающие в него сегменты с данными можно передавать без ожидания подтверждений. Его правая (на рисунке) граница совпадает с правой границей буфера и скользит налево относительно последовательности сегментов с данными по мере поступления и упорядочивания подтверждений. Переданные, но неподтвержденные сегменты с данными продолжают оставаться в буфере, так как возможно потребуется их повторная передача. Левая граница «привязана» к правой в соответствии с размером текущего окна. Но поскольку размер подвержен динамической коррекции, положение левой границы относительно правой постоянно изменяется.

Область текущего окна передачи за вычетом переданных, но неподтвержденных сегментов с данными, является доступным окном (useable равно effective window). TCP-процесс должен последовательно отправить все сегменты с данными, попавшие в эту область. Если размер текущего окна передачи равен нулю, то передача приостанавливается полностью.

Рисунок -- Организация буфера приема TCP

На другой стороне соединения, возможно уже разупорядоченные при преодолении СПД сегменты поступают в буфер приема (размер может не совпадать с размером буфера передачи). При этом они размещаются там согласно своим номерам.

Текущее окно приема охватывает часть буфера, в которой можно размещать еще неупорядоченные сегменты с данными. Как и текущее окно передачи, в любой момент времени оно так же имеет некоторый определенный размер. Левая (на рисунке) граница текущего окна приема совпадает с левой границей буфера. Правая граница проходит слева за последним упорядоченным сегментом с данными и поэтому динамически меняет свое положение относительно левой границы. По мере считывания принимающим приложением упорядоченных байтов из буфера окно скользит относительно последовательности сегментов с данными. Если размер текущего окна приема равен нулю, а сегменты с данными продолжают поступать, то возникает переполнение.

Вполне закономерно, что именно на принимающий TCP-процесс, как на более подверженный влиянию недетерминированности СПД, возложен контроль «поведения» оконного механизма. Это делается посредством «обратной связи». Принимающий TCP-процесс пытается информировать передающий о состоянии своего буфера, точнее о наличии в нем свободного места. Для этого он при подтверждениях сообщает предлагаемое окно (announced равно advertised равно offered window). В качестве размера предлагаемого окна указывается размер текущего окна приема. Последствия разупорядочивания сегментов с данными такому подходу не противоречат.

Максимальный размер любого из окон не может превышать размер соответствующего буфера (например, 8 килобайтов).

В результате, можно сделать вывод о том, что на работу соединения влияют приложения, TCP-процессы и сетевой уровень.

В идеале, при полностью сбалансированной работе, размер текущего окна передатчика равен размеру предлагаемого окна, то есть равен размеру текущего окна приемника. А если еще и буферы освобождаются «мгновенно», то этот размер совпадает с размером доступного окна и размерами буферов. Алгоритмы TCP направлены на «уравнивание» всех упомянутых окон.

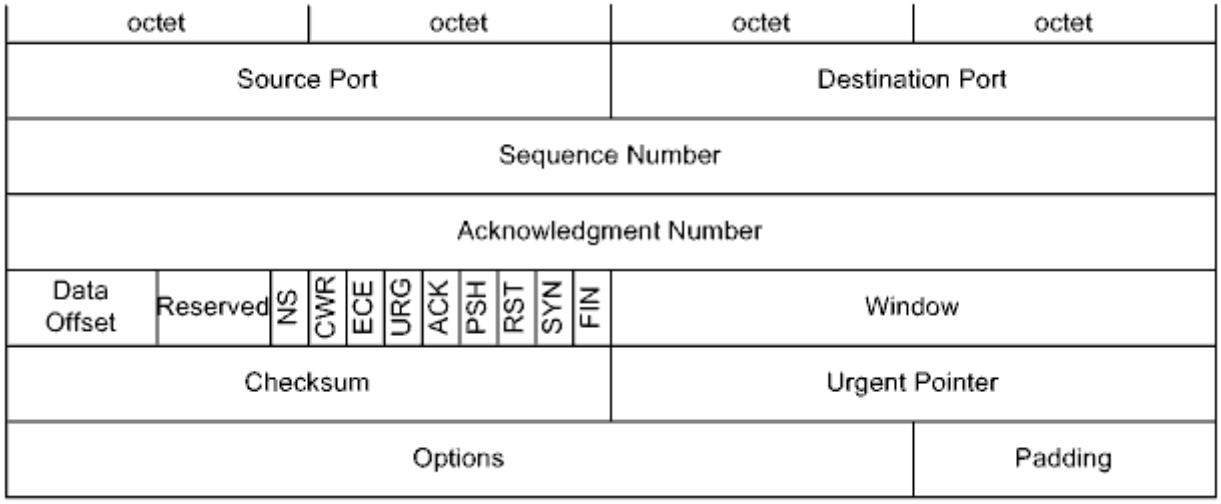

Рисунок -- Формат заголовка TCP-сегмента

Поля:
1. Source Port -- программный порт источника.
2. Destination Port -- программный порт назначения.
3. Sequence Number (SN) -- последовательный номер (сегмента).
4. Acknowledgment Number (AN) -- подтверждающий номер.
5. Data Offset -- смещение данных (в 32-ухбитных словах).
6. Reserved -- зарезервировано (должно равняться нулю).
7. URG (URGent Pointer field significant) -- флаг значимости указателя на экстренные данные.
8. ACK (ACKnowledgment field significant) -- флаг значимости подтверждающего номера.
9. NS (Nonce Sum) -- флаг -- контрольная сумма для проверки правильности кодов явных уведомлений о заторах (связан с QoS, связан с IP-заголовком) (RFC 3540).
10. CWR (Congestion Window Reduced) -- флаг уменьшения окна затора при явном уведомлении о заторе (RFC 3168).
11. ECE (Explicit Congestion Notification Echo) -- флаг подтверждения явного уведомления о заторе (RFC 3168).
12. PSH (PuSH Function) -- флаг принудительной доставки данных (без буферизации).
13. RST (ReSeT the connection) -- флаг разрыва соединения (например, из-за сбоя на одной из взаимодействующих сторон).
14. SYN (SYNchronize sequence numbers) -- флаг синхронизации последовательных номеров.
15. FIN (No more data from sender) -- флаг последних данных.
16. Window (W) -- предлагаемое окно.
17. Checksum -- контрольная сумма.
18. Urgent Pointer -- указатель на экстренные данные (RFC 6093).
19. Options -- опции (например, MSS).
20. Padding -- наполнитель.

Функционирование оконного механизма TCP базируется на использовании трех

полей в заголовке сегмента: SN, AN, W, и трех флагов (из шести стандартизованных изначально): SYN, ACK, FIN.

Установление TCP-соединения, известное как «тройное рукопожатие» (three-way handshake), основывается на использовании флагов SYN и ACK.

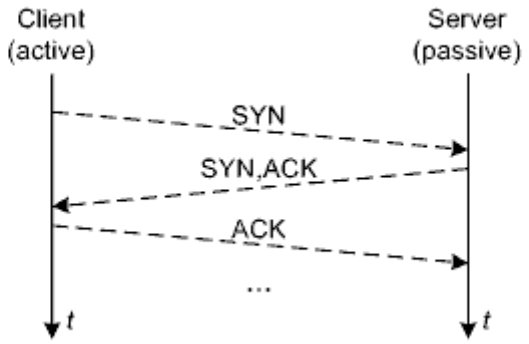

Рисунок -- Установление TCP-соединения

(На этом и последующих рисунках указаны ключевые задействованные флаги и поля. Сплошной линией обозначены сегменты с данными, пунктирной -- сугубо служебные.)

Сначала TCP-процесс -- инициатор взаимодействия (на стороне клиента) отправляет служебный сегмент с установленным флагом SYN, тем самым сообщая о своих намерениях (первое «рукопожатие»). Затем запрашиваемый TCP-процесс (на стороне сервера), если он согласен взаимодействовать, подтверждает это ответным служебным сегментом с двумя установленными флагами SYN и ACK (второе «рукопожатие»). Наконец, инициатор отвечает еще одним служебным сегментом с установленным флагом ACK, тем самым подтверждая подтверждение (третье «рукопожатие»).

Не смотря на то, что процесс установления соединения несимметричен, в дальнейшем, в общем случае, оно используется в полнодуплексном режиме.

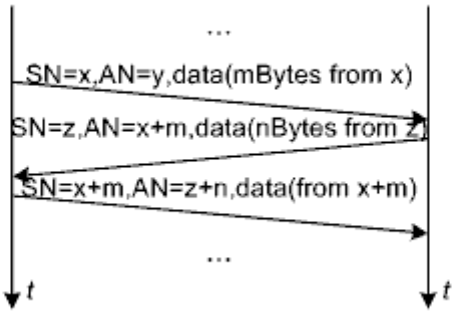

Рисунок -- Пересылка данных по TCP-соединению

Очень важно, что на обмен сегментами нужно смотреть с двух сторон. При этом один и тот же TCP-процесс, находящийся по одну сторону соединения, одновременно может выступать в качестве как передатчика данных, так и приемника данных. Полнодуплексность самого соединения достигается за счет того, что передаваемый в определенном направлении сегмент служит одновременно для транспортировки как данных и связанных с ними служебных полей от передающей составляющей TCP-процесса, так и подтверждений и связанных с ними других служебных полей от

принимающей составляющей TCP-процесса.

В СПД одновременно могут находиться множество сегментов, относящихся к одному соединению. Применительно к данным в одном сегменте, соединение является полудуплексным, так как сегмент не может содержать более одного поля с ними.

По правилу протокола, поле SN пересылаемого сегмента отражает собственный SN этого сегмента. По другому правилу, в поле AN указывается SN ожидаемого сегмента, коим является следующий по порядку сегмент.

При установлении соединения данные не пересылаются. Поэтому, для того чтобы не нарушать указанные правила, в качестве SNs используют невключенные в нумерацию байтов сообщения ISNs, а в качестве ANs -- просто инкрементированные SNs. Обойтись без передачи SNs при установлении соединения невозможно, так как стороны должны однозначно идентифицировать это соединение. После синхронизации SNs соединение считается установленным (established).

Флаг SYN используется только при установлении соединения, а флаг ACK -- в каждом ответном сегменте.

Не смотря на предоставляемые возможности, данные вполне могут пересылаться только в одном направлении, то есть в симплексном режиме. При этом в направлении, попутном направлению пересылки данных, в качестве AN используется SN следующего по порядку несуществующего (вообще, либо уже, либо пока) сегмента, что никоим образом не противоречит уже приведенным правилам. Если сегментов с данными пересылается несколько, то ANs дублируются столько раз, сколько нужно. Это приводит к дублированию SNs в ответных сегментах без данных. Аналогичные дублирования возникают и при приостановке пересылки данных в определенном направлении.

Поскольку при установлении соединения оно всегда открывается в двух направлениях (по инициативе клиента, но может использоваться в одном любом направлении), для нормального завершения оно и закрыто должно быть в обоих направлениях.

Для закрытия соединения в своем направлении, сторона, в соответствующем сегменте (обычно с последними данными), устанавливает флаг FIN.

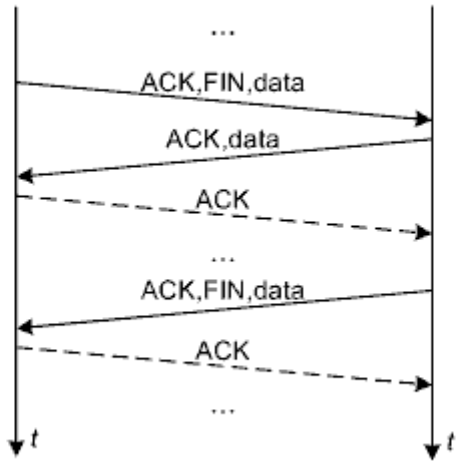

Рисунок -- Нормальное завершение TCP-соединения

Соединение, нормально закрытое только в одном направлении, или ненормально завершенное на одной из сторон без уведомления другой стороны (в результате сбоя) называют полуоткрытым (half-open).

Размер предлагаемого окна в поле W может изменяться каждый раз для соответствующей коррекции текущего окна передачи, в том числе и при установлении соединения для изменения размера текущего окна передачи по умолчанию.

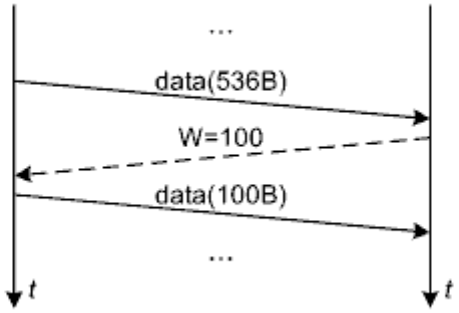

Рисунок -- Коррекция текущего окна передачи TCP-соединения

В случае задания нулевого значения поля W передача данных фактически запрещается. После освобождения места в буфере приема подтверждение обязательно повторяется с уже ненулевым полем W, что «разблокирует» передающую сторону.

Проблема возможной потери в СПД некоторых сегментов решается с помощью тайм-аутов.

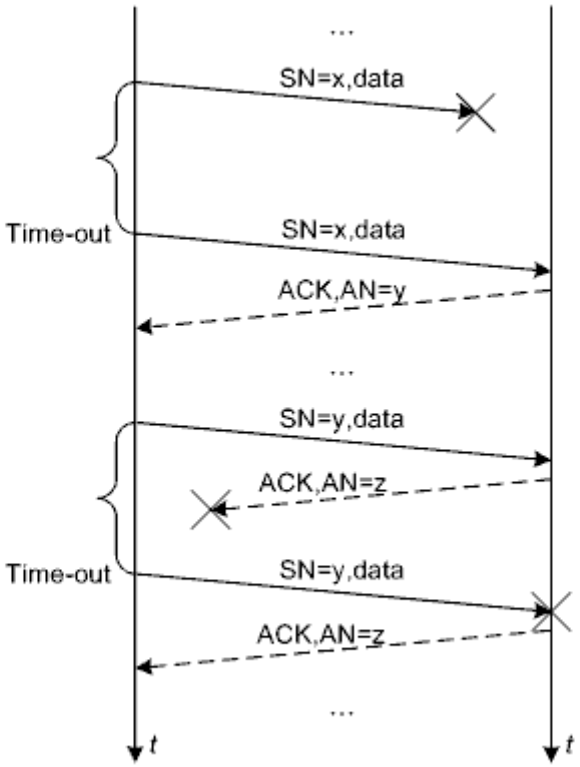

Рисунок -- Гарантия доставки по TCP-соединению

Передающий TCP-процесс определяет потерю сегмента с данными либо его подтверждения по отсутствию этого подтверждения в течение установленного интервала времени. После наступления тайм-аута сегмент с данными передается повторно.

Отрицательные подтверждения не предусмотрены вообще. Принимающий TCP-процесс подтверждает все принятые сегменты с данными, причем подтверждает всегда. При этом если принята копия (что говорит о потере подтверждения), то она удаляется.

Получение сегмента с SN больше ожидаемого говорит о возможной потере сегментов с данными или о разупорядочивании.

Важно правильно оценивать время «отклика системы». Поэтому время ожидания подтверждений рассчитывается на основе показаний таймеров и корректируется. При этом первостепенное значение имеет параметр RTT (Round-Trip Time) -- суммарное время пересылки по СПД сегмента с данными и его подтверждения.

Протокол TCP обладает несколькими дополнительными возможностями. Возможна пересылка экстренных данных (urgent data) и ускоренная пересылка (push function).

Состояния TCP-процесса стандартизированы и предусмотрена диаграмма переходов между состояниями.

Для обеспечения своей функциональности TCP-процесс должен хранить множество ассоциированных с каждым соединением служебных данных.

Как было сказано выше, несбалансированность соединений может возникать из-за разной производительности задействованных подсистем, плюс из-за изменения производительности этих подсистем.

Следует отметить, что базовая редакция стандарта TCP предоставила реализациям определенную вольность (как во многих других случаях со стандартами). Это привело к тому, что при полном соблюдении требований во многих случаях оконный механизм TCP может оказаться неэффективным. «Разношерстность» реализаций усугубляет ситуацию. Как следствие, потребовалась разработка дополнений к базовому алгоритму. Новые алгоритмы оперируют с новыми понятиями.

В частности, хорошо известна проблема, вошедшая в историю под обобщенным названием «синдром глупого окна» («silly window syndrome»), в свое время «стопорившая» значительную часть пространства Internet. Синдром может возникать по разным причинам и проявляется в том, что текущее окно передачи не соответствует состоянию приемника, тем самым не позволяя его как следует «нагрузить» либо, наоборот, «разгрузить».

Решение Нэгла (Nagle) позволяет побороть «синдром глупого окна» когда передающей стороне требуется часто отправлять небольшие сегменты с данными.

Решение Кларка (Clark) позволяет побороть «синдром глупого окна» когда принимающей стороной часто анонсируется небольшое предлагаемое окно.

Также стандартизированы четыре дополнения Ван Якобсона (Van Jacobson), призванные бороться с перегрузками в СПД (последнее RFC -- RFC 5681):

1. Медленный старт (slow start).
*И*дея заключается в том, что в начале передачи размер текущего окна передачи нужно увеличивать не «скачком», а плавно, пропорционально скорости получения подтверждений (не превышая размер предлагаемого окна).

Рекомендуемые формулы:

IW = 2 * *SMSS*, если *SMSS* > 2190 Bytes ,
IW = 3 * *SMSS*, если 2190 Bytes >= *SMSS* > 1095 Bytes ,
IW = 4 * *SMSS*, если *SMSS* <= 1095 Bytes ,

где *IW* (initial window) -- начальное значение текущего окна передачи;

cwnd += min (*N*, *SMSS*) ,

где *cwnd* (congestion window) -- текущее окно передачи (в данном случае, окно затора), *N* -- количество подтвержденных байтов, *SMSS* (sender MSS) -- MSS передатчика.

2. *И*збегание затора (congestion avoidance).
Состоит в сдерживании экспоненциального роста размера текущего окна передачи после преодоления им некоторого порога. Как правило переход к избеганию затора происходит после медленного старта.

Рекомендуемые формулы:

ssthresh = max (*FlightSize* / 2, 2 * *SMSS*) ,

где *ssthresh* (slow start threshold) -- порог перехода от медленного старта к избеганию затора, *FlightSize* -- количество еще неподтвержденных байтов;

cwnd += *SMSS* * *SMSS* / *cwnd* .

3. Быстрая повторная передача (fast retransmit).
При получении принимающей стороной разупорядоченного сегмента с данными (возможно из-за потери ожидаемого сегмента с данными) незамедлительный повтор подтверждения с AN недостающего сегмента с данными. При получении передающей стороной трех одинаковых подтверждений незамедлительный повтор сегмента с данными согласно AN. Что, в некоторых ситуациях, позволяет успешно послать потерянный сегмент еще до наступления тайм-аута.

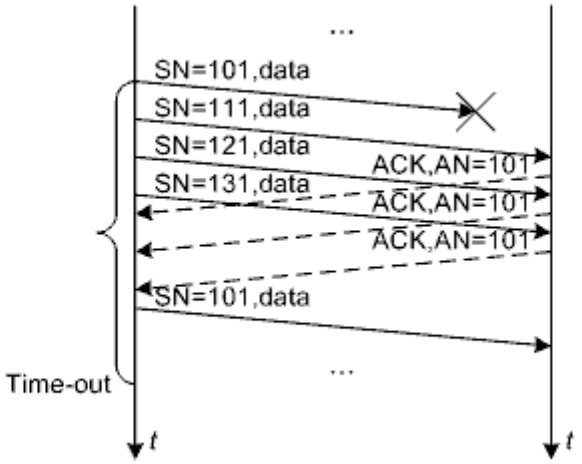

Рисунок -- Быстрая повторная передача TCP

4. Быстрое восстановление (fast recovery).
После обнаружения затора, переход сразу к избеганию коллизий, минуя стадию медленного старта. Как правило в связке с быстрой повторной передачей.

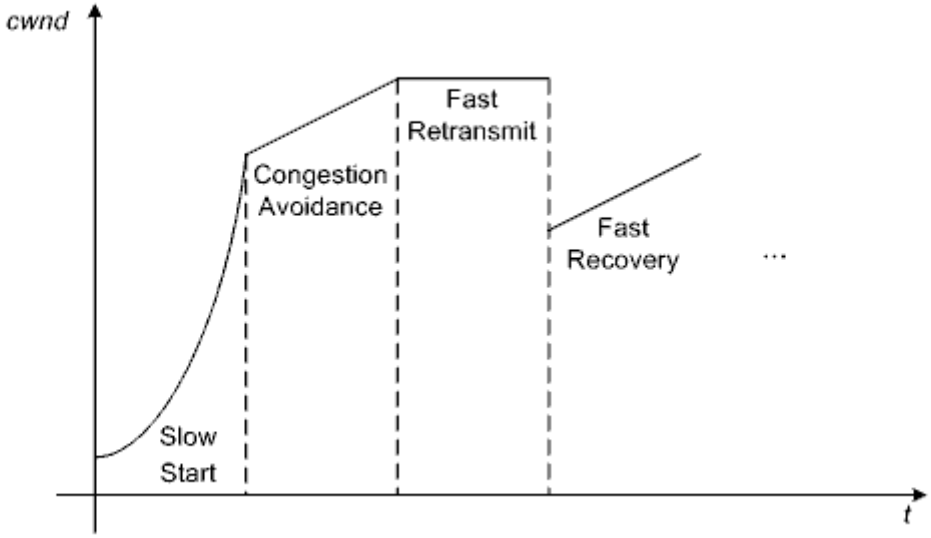

Рисунок -- TCP congestion control

Последствия потерь и разупорядочивания сегментов заключаются в разрушении «маятника» взаимодействия и приводят к необходимости еще одной важной оптимизации, четко проявляющейся при быстрой повторной передаче. Согласно базовому алгоритму все сегменты должны быть подтверждены, а значит, после быстрой повторной передачи принимающая сторона должна послать все недостающие подтверждения. Но стороны могут «договориться», что текущий AN отражает номер первого ожидаемого получателем сегмента и автоматически подтверждает все сегменты с меньшими номерами (cumulative acknowlegement).

Были разработаны и другие усовершенствования, основанные, например, на манипулировании с RTT (RFC 6298, RFC 7323).

Протокол транспортного уровня UDP (User Datagram Protocol) (RFC 768) реализует способ пересылки данных без гарантии доставки, часто называемый *дейтаграммным*

(datagram) (хотя user datagram -- это пакет с контролируемыми пользователем данными, а datagram -- это любой пакет с данными).

Рисунок -- Формат заголовка UDP-дейтаграммы

Поля:
1. Source Port -- программный порт источника.
2. Destination Port -- программный порт назначения.
3. Length -- длина дейтаграммы включая заголовок (в байтах).
4. Checksum -- контрольная сумма (псевдозаголовка, плюс заголовка, плюс данных).

При вкладывании UDP-дейтаграммы в IP-пакет (IPv4, IPv6), между UDP-заголовком и IP-заголовком вставляется дополнительный так называемый UDP-псевдозаголовок, в котором дублируются некоторые значения из основного IP-заголовка.