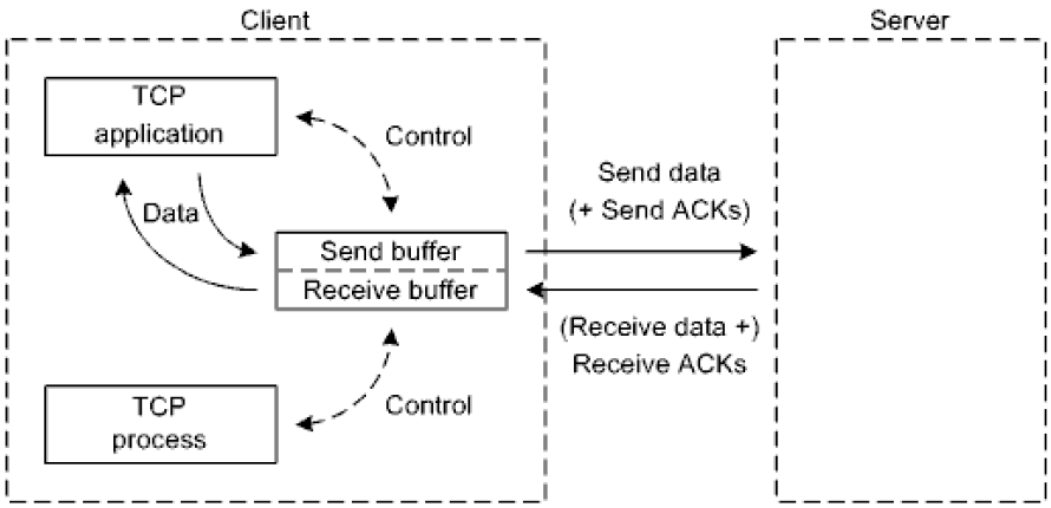
51 Структура системы TCP

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



Структура соединения 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.

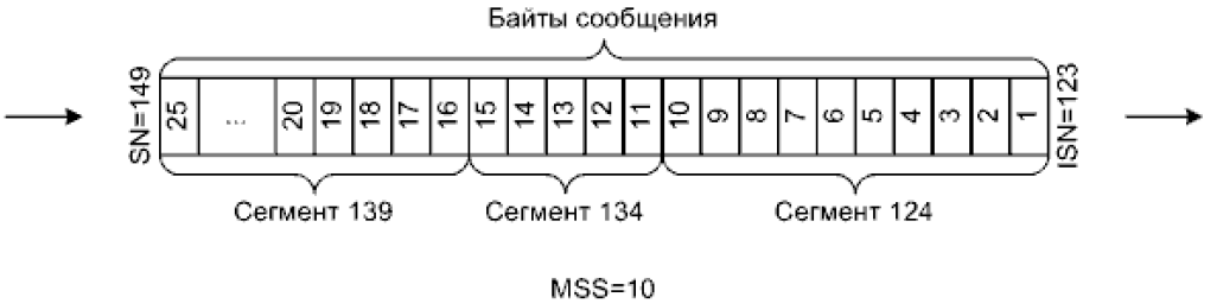
Все байты сообщения последовательно нумеруются так называемыми последовательными номерами -- SNs (Sequence Numbers).

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).

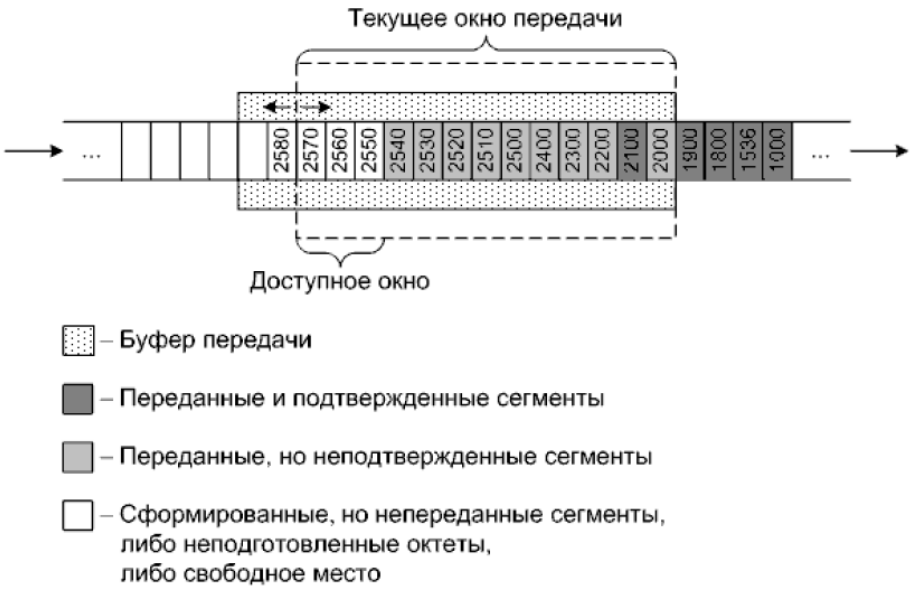


Пример сегментации сообщения

Дальше инфа которая мб и лишняя, но хз

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).



Организация буфера передачи

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.

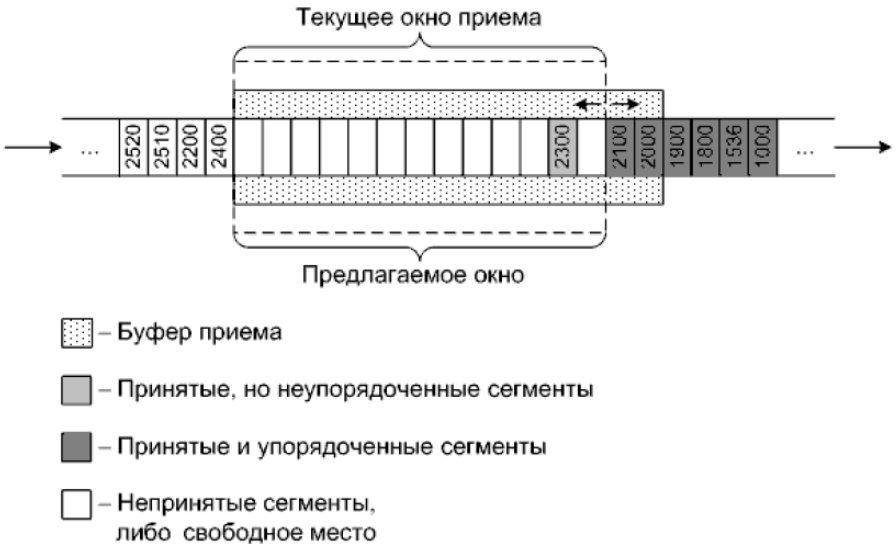
Переданные, но неподтвержденные сегменты с данными продолжают оставаться в буфере, так как возможно потребуется их повторная передача.

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.



Организация буфера приема

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.

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