**Лекция за 11.04.2012, часть 1**

Протокол TCP действует на ***транспортном*** и, частично, ***сеансовом*** уровнях модели ISO/OSI. ***Основная его задача*** – это доставка пользовательских данных (доставка данных пользовательских протоколов).

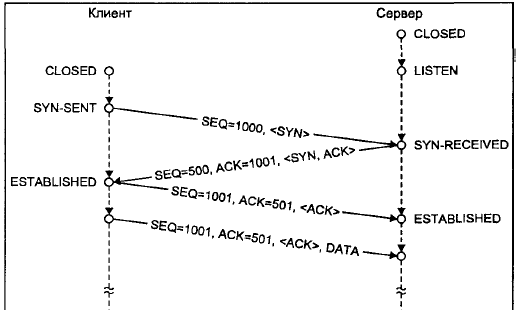
Протокол TCP ***поддерживает проверку целостности данных и установление виртуального соединения.*** То есть, с точки зрения пользовательского протокола, протокол TCP обеспечивает соединение “точка-точка” между отправителем и получателем данных.

Протокол TCP является ***дуплексным*** протоколом – он поддерживает передачу данных в обе стороны без открытия дополнительного соединения. Это является довольно существенным преимуществом, потому что многие другие реализации транспортных протоколов и, более того, ранние реализации самого протокола TCP, на самом деле, поддерживали передачу данных только в одну сторону, и для передачи данных в обе стороны требовалось открывать дополнительное соединение, что, естественно, влекло за собой дополнительные накладные расходы. Здесь же, в связи с тем, что мы используем одно соединение для передачи данных, гораздо полнее используется ширина канала, и в результате этого, как снижается нагрузка на канал, так и улучшаются такие параметры, как время задержки. Вообще протокол TCP, к сожалению, по логике своей работы характеризуется задержками, и это частично решается использованием механизма управления потоком в TCP (механизм будет изучен далее).

***Установка и разрыв TCP соединения.***

В логике работы TCP используется терминология ***клиент-сервер***, но при этом, понимание этих терминов немного отличается от того, которое присутствует в других областях. В TCP всегда ***клиентом*** называется та сторона, которая устанавливает TCP соединение. То есть в принципе, это может быть и программа-служба, которую мы привыкли в других случаях именовать *сервером*. В рамках данной лекции будем придерживаться новой концепции.

***Установка соединения.***

  
***Рис. 1 Установление связи, передача данных в TCP-сеансе***

* Любой TCP-сеанс всегда начинается с состояния ***CLOSED***, состояния, при котором сторона не может вести прием/передачу данных и не может устанавливать TCP соединение.
* *Сервер* из состояния ***CLOSED*** переходит в состояние ***LISTEN***, которое означает, что сервер открыл свой порт\* на “прослушку”: он фиксирует все TCP-сегменты, приходящие на этот порт и готов из них выделить те, которые отвечают за установку соединения.  
  *\* Как известно, пространство имен TCP – это пространство номеров портов.*
* *Клиент* также начинает с ***CLOSED***, переходит в состояние ***SYN-SENT*** и в этом состоянии посылает серверу TCP-сегмент, у которого установлен флаг SYN\*.  
  *\* Флаг SYN (Synchronized) означает, что данный сегмент используется для установления соединения и для процедуры начальной синхронизации порядковых номеров.  
  \*\* В заголовке TCP-сегмента имеются два 32-битных поля:* ***SequenceNum****,* ***AcknowledgementNum****. Они служат для подтверждения целостности потока данных и, в сущности, являются счетчиками переданных в TCP-сеансе октетов.*

Соответственно, сегмент с установленным флагом SYN говорит о том, что значение ***SequenceNum*** этого сегмента служит для первоначального согласования порядковых номеров.

* После того, как *сервер* получил сегмент с установленным флагом SYN, он переходит в состояние ***SYN-RECEIVED*** и отправляет ответный TCP-сегмент, в котором, во-первых, указывает собственный флаг SYN (в данном случае это означает, что в этом сегменте также для синхронизации устанавливается счетчик переданных октетов – ***SequenceNum***, но уже для передачи данных в обратную сторону, т.е. от сервера к клиенту), а во-вторых, устанавливает флаг ACK (Acknowledgement), означающий, что в данном сегменте содержится подтверждение о получении предыдущего сегмента со стороны клиента.

*\* При установке флага ACK сервер устанавливает номер подтверждения -* ***AcknowledgementNum****, равный следующему непринятому номеру октета.*

* После приема сегмента с флагом SYNи ACK, *клиент* переходит в состояние ***ESTABLISHED***.   
  Здесь: поскольку никакие номера более согласовывать не требуется, то флаг SYNбольше не употребляется.
* Теперь сервер получает сегмент с только одним флагом ACK, и после приема, переходит в состояние ***ESTABLISHED***.

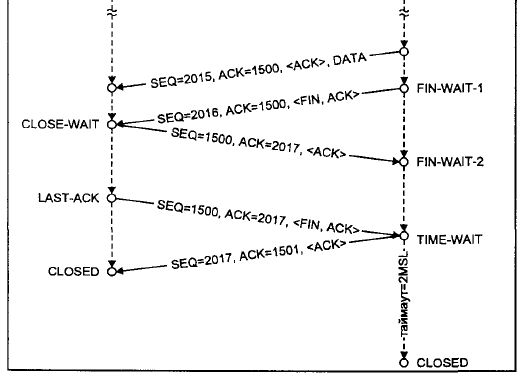
Вся вышеописанная процедура передачи трех сегментов, (из них два – с флагом SYN) носит название **“three-wayhandshake”** (трехстороннее рукопожатие). Данная процедура очень симметрична.

После процедуры тройного рукопожатия ***возможна передача данных***.   
 Строго говоря, *клиент* может начинать передавать данные на третьем сегменте, но обычно так не делают.

**Дальнейший учет счетчиков Sequence (SEQ#), Acknowledgement (ACK#):**(Считаем, что передача идет без потери сегмента)  
Ставим в качестве SEQ# номер предыдущего полученного ACK#.  
  
 ***Пример:*** Пусть сервер передает 200 октетов данных. Тогда:  
<ACK><ACK>  
SEQ#=501 SEQ#=1001   
[DATA]=200🡪 ACK#=701  
ACK#=1001[DATA]…

Вся схема установления и передачи данных называется “**синхронный TCP**”.   
На самом деле, изначально TCPрассматривался как синхронный протокол. Но данная схема показала большие проблемы, связанные с задержками. Фактически, мы не можем продолжать передавать данные, пока не получим подтверждение о приеме. Это может быть терпимо для быстрых линий, но для медленных линий, которые существовали в 80е годы, это было совершенно неприемлемо. Поэтому в настоящее время TCP классифицируется как ***полусинхронный*** или ***псевдосинхронный*** протокол.

***Разрыв соединения.***

(Считаем, что разрыв инициализирует сервер)  
  
***Рис.2 Завершение TCP-сеанса***

В отличие от установления соединения, которое совершает только клиент, разрывать соединение может любая сторона.

Закрытие TCP соединения выглядит сложнее его установки как раз засчет псевдосинхронности.

* Закрытие сеанса начинается с отправления сегмента с установленным флагом FIN (Finalize), после чего, сторона, его отправившая, переходит в состояние ***FIN-WAIT1***, а сторона, которая его приняла, переходит в состояние ***CLOSE-WAIT***.
* Сторона, принявшая сегмент, должна отправить подтверждение, но в принципе, она еще может продолжать передавать свои данные. Т.е. получение флага FIN не запрещает получившей стороне продолжать отправку данных, т.е. флаг FIN является индикацией того, что сторона, которая данные отправила, больше передавать данные не будет.
* После того, как подтверждение получено, сторона, инициализировавшая разрыв соединения, переходит в состояние ***FIN-WAIT2***, и в принципе может оставаться в этом состоянии неопределенно долгое время, пока соответственно, другая сторона еще, возможно, отправляет собственные данные.
* Для того чтобы полностью разрушить соединение, вторая сторона должна отправить свой сегмент с флагом FIN. Когда она это совершает, она переходит в состояние ***LAST-ACK***. После чего она уходит в состояние ***CLOSED***, и на этом ее деятельность заканчивается.
* При получении последнего FIN, сторона, инициализировавшая разрыв, переходит в состояние ***TIME-WAIT*** и отвечает своимсегментом с флагом ACK и еще некоторое время продолжает слушать порт на тот случай, если придут подтверждения предыдущих TCP-сегментов. Этот таймаут равен двум временам прохождения TCP-сегмента туда и обратно.

***Как вычисляется таймаут:***  
 *Начиная с отправки первого TCP-сегмента, обе стороны ведут счетчик времени, т.е. сколько времени прошло от момента отправки данных, до момента получения на них подтверждения, и это время постоянно корректируется. Соответственно, половина от этого времени носит название “****Maximumsegmentlifetime****” (максимальное время жизни сегмента). Считается, что скорости в обе стороны примерно одинаковы.*

* По истечении таймаута ***2MSL***сторона, инициировавшая завершение соединения, окончательно его закрывает, и после этого переходит в состояние ***CLOSED*** (или ***LISTEN***, в зависимости от настроек службы).

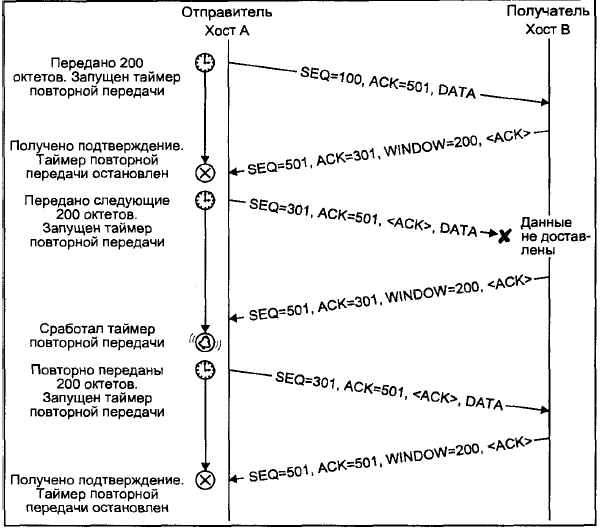
**Заметим:** схема разрыва TCP-соединения является симметричной со схемой установления соединения за исключением одного: мы оставляем возможность для продолжения передачи данных одной из сторон и для возможной отправки подтверждений на непринятые сегменты. Поэтому посылка первого флага FIN соответствует отправке первого флага SYN, после чего идет пауза на возможную передачу данных, а обмен последними TCP-сегментами при разрыве соединения соответствуют второму и третьему этапам ***three-wayhandshake***.

***Поддержка целостности передачи данных.***

На каждом запуске на передачу каждого сегмента MSL-таймер корректируется. Самый простой вариант корректировки – это каждый раз заменять значение MSLNull’ом. Но на самом деле, это не очень удобно, потому что могут произойти непредвиденные обстоятельства, например, в какой-то момент из-за нарушения целостности линии или из-за нарушения целостности маршрутизации значение MSLможет резко возрасти. Но резкие скачки MSL лучше предотвращать, поэтому используется некоторый взвешенный алгоритм, который зависит от реализации TCP-стека.  
 ***Ситуация:*** допустим, TCP-сеанс существовал в течение длительного времени, скажем, 100 секунд с малым значением MSL равным 0.1сек, но на 101 секунде MSL вырос до 1 секунды. Тогда:

***Как корректируется MSL:*** Берем среднее (для простоты арифметическое, геометрическое, есть варианты сложнее) взвешенное значение MSL по разным промежуткам времени. Т.е. в нашем случае 0.1 секунды с весом 100 и 1 секунда с весом 1. Получается общий вес: примерно 11/101. Т.е. вместо 0.1 сек получаем 0.11 сек.

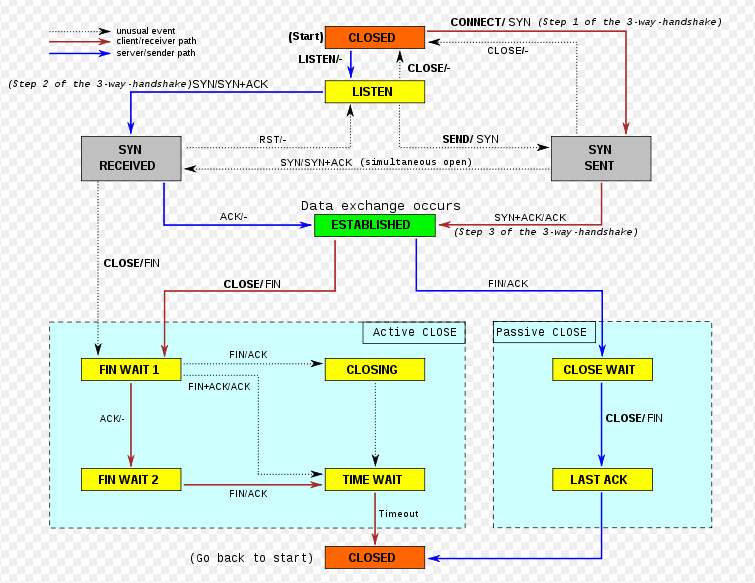
Фактически, резкие скачки вверх имеют чуть больший вес, а скачки вниз – чуть меньший, поэтому корректировка обычно заметно увеличивает значение MSL.

******  
***Рис. 3 Повторная передача данных***  
  
 Итак, когда начинается отправление данных, запускается так называемый ***таймер повторной передачи*** (retransmittingtimer). Обычно maxзначение этого таймеране должно превышать 4MSL.

- Если получается нормальное подтверждение на переданные данные, то просто корректируется таймер MSL.

- Теперь, предположим, что данные отправляются и уходят в DEVNULL.Что при этом получается: вновь запускается таймер повторной передачи, но, когда истекает время равное 4MSL, таймер срабатывает и таким образом сторона понимает, что отправленные данные не были доставлены. После срабатывания таймера повторной передачи он запускается заново, и повторно отправляются данные. И если удаленная сторона отвечает, то после этого возможно опять корректируется MSL. В сущности же, когда сработал таймер повторной передачи, время MSLне корректируется.

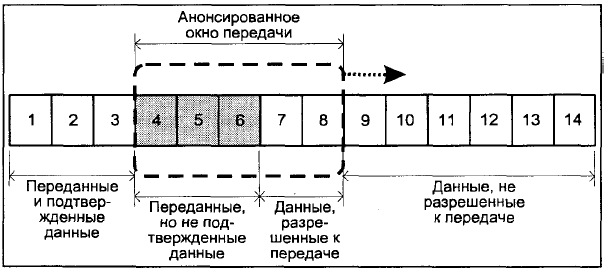
Если удаленная сторона не отвечает, то вечно отправлять повторно сегменты бесполезно. Поэтому кроме таймера о повторной передаче существует еще один параметр, который называется ***счетчик повторных передач***. В зависимости от реализации TCP-стека и от настроек конкретно, если сегмент безуспешно отправляется повторно 5-7 раз, то сторона, которая осуществляет передачу данных, считает, что удаленная сторона заглохла и закрывает соединение со своей стороны.   
Такое встречается в реальности и связано в основном с резким изменением механизма маршрутизации, при котором происходит резкое возрастание MSL. Поэтому рекомендуют настраивать таймер повторной передачи не на 4, а на большее число времен MSL, либо управлять этим динамически. С другой стороны, если установить слишком большое время для таймера или счетчика повторной передачи, то можно столкнуться с множеством заглохшихTCP соединений, которые не закрываются из-за большого таймаута.

***Рис.4TCP как конечный автомат***  
 ***Управление потоком в TCP.***

Любой синхронный протокол обладает существенными недостатками: он очень чувствителен к коротким проблемам со связью (помехи, короткие нарушения соединения и т.п.). Несмотря на то, что помехи могут занимать до 0.1% от любого заданного интервала, их влияние на поведение синхронного протокола очень существенно. Это касается даже не только передачи данных, но и любой синхронной процедуры обработки чего-либо.

Поэтому были предприняты успешные попытки отказаться от абсолютной синхронности TCP протокола. Это было осуществлено с помощью дополнительного поля в заголовке TCP-сегмента, называемого ***WINDOW*** (announcingtransmitwindow) –анонсированное окно передач. WINDOW – это число октетов, которое может быть передано противоположной стороной до получения подтверждения данных. Поэтому на самом деле, когда данные буферизуются в TCP буфере, они всегда делятся на 4 категории:

(считаем, что данные буферизуются по 100 октетов)

  
***Рис.5 Окно передачи TCP***

Соответственно, как только приходит подтверждение о переданных данных анонсированное окно передачи смещается вправо.

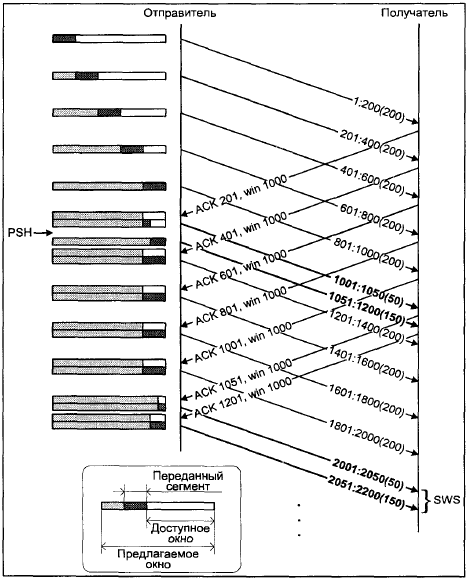
Можно передавать данные, пока они не выходят за правую границу анонсированного окна передачи. Как только приходит подтверждение данных, окно смещается вправо так, чтобы его левая граница приходилась на уже переданные октеты. Так, соответственно можно передавать данные без получения подтверждения. Все, что находится правее анонсированного окна передачи, все равно будет не разрешено.

Слева данные из буфера будут постепенно удаляться.

Если бы мы не могли влиять на размеры окна, то эта схема не имела бы никаких преимуществ по сравнению с синхронным TCP, а именно, мы бы очень быстро выбрали первоначальный размер окна и после этого схема работала бы точно так же как и без него. Но в связи с тем, что сторона, принимающая данные, может выставить в ответном TCP-сегменте новое значение для размера окна, то таким образом мы можем влиять на передачу и скорость передачи данных так, чтобы эта передача велась более равномерно. Если происходит уменьшение размера окна, то мы получаем подтверждение тем или иным принятым октетам, но после этого устанавливаем новый размер окна. Если окажется, что новый размер окна таков, что уже переданные данные вылезают за правую границу окна, то передача данных вообще приостанавливается до получения подтверждения не влезающих октетов.

К сожалению, управление потоком посредством изменения размеров окна все равно имеет определенные проблемы. Наиболее распространенная из них – это ***синдром глупого окна***.

***Синдром глупого окна.***

***  
Рис.6 Возникновение СГО (размер окна не изменяется и равен 1000 октетов)***

Если одна сторона никак не управляет размерами окна, а другая стремится отправить как можно больше данных, то достаточно возникнуть ситуации, при которой передается малое количество данных, как эта ситуация будет воспроизводиться далее неопределенно долгое время.  
  
 ***Рассмотрим пример (см. рис.6):*** отправитель данных передает данные порциями по 200 октетов, при установленном размере окна – 1000 октетов. (5 порций данных). Пока он не получит подтверждения на приход первой порции данных, он после отправки 100 октетов должен остановиться. А после этого происходит **страшное**: вместо того, чтобы отправить 200 октетов целиком, мы отправляем сначала сегмент в 50 октетов, а потом сегмент в 150 октетов (это характерно для любых интерактивных приложений, которые осуществляют интерактивное взаимодействие по сети). После этого ждем подтверждение и потом опять осуществляем передачу данных по 200 октетов.