TCP

> ***TCP*** -- протокол управления передачей. В отличие от UDP позволяет регулировать скорость, с которой эта передача происходит, средствами самого протокола, гарантирует доставку данных в том порядке, в котором они отправлены.

Формат заголовка TCP:

0 — 3	4 — 6	7 — 15	16 — 31
Порт источника, **Source Port**			Порт назначения, **Destination Port**
Порядковый номер, **Sequence Number (SN)**			
Номер подтверждения, **Acknowledgment Number (ACK SN)**			
Длина заголовка, (**Data offset**)	Зарезервировано	Флаги	Размер Окна, **Window size**
Контрольная сумма, **Checksum**			Указатель важности, **Urgent Point**
Опции (необязательное, но используется практически всегда)			
Данные			

Аналогично с UDP есть well-known порты и пользовательские.

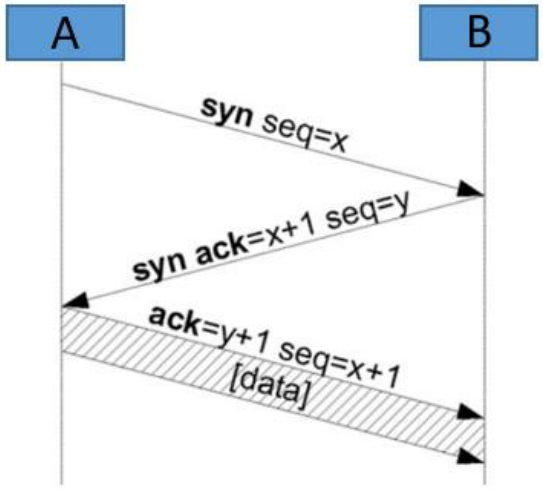
- ***Source/destination port*** -- по 16 бит, максимум мб 2^{16} портов.
- ***Sequence number*** -- номер первого байта в сегменте.
- ***Acknowledgement number*** -- до какого момента данные были получены от противоположной стороны (тоже номер байта).
- ***Data offset*** -- указатель на то, откуда начинаются данные, инкапсулированные в наш сегмент.

Установление соединения в TCP

> Называется ***тройным рукопожатием***.

Ориентация на соединение означает, что сначала устанавливается соединение и только потом передаются пользовательские данные.

Процесс установления соединения:



1. A отправляет к B сегмент с флагом SYN -- посмотри поле Sequence Number. Т.е. он отправляет данные с конкретного байта и просит B запомнить этот номер.
2. B отправляет к A сегмент с флагами SYN и ACK -- посмотри на моё поле Sequence Number, я буду нумеровать данные начиная с такого номера. Sequence Number могут быть разные с двух сторон.

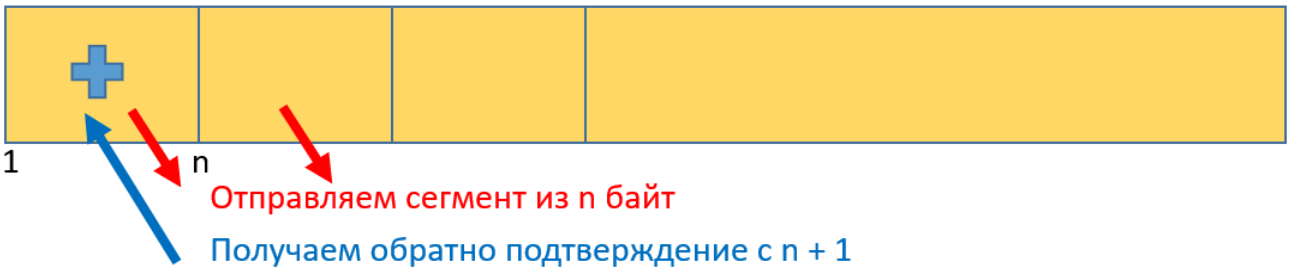
   Также B говорит: посмотри на мой Acknowledgement number -- подтверждает успешное получение данных от A. Если SN от A равно x, то AN = x + 1 -- номер следующего байта (верно только при установлении соединения) -- ожидаем получить следующий байт последовательности.

   В общем случае, если в данных содержится d байт, то AN = ((x - 1) + d) + 1 = x + d.
3. A отправляет к B сегмент с флагом ACK -- я видел твой Sequence Number, жду от тебя данные дальше.

Передача данных в TCP

Используется механизм ***скользящего окна***.

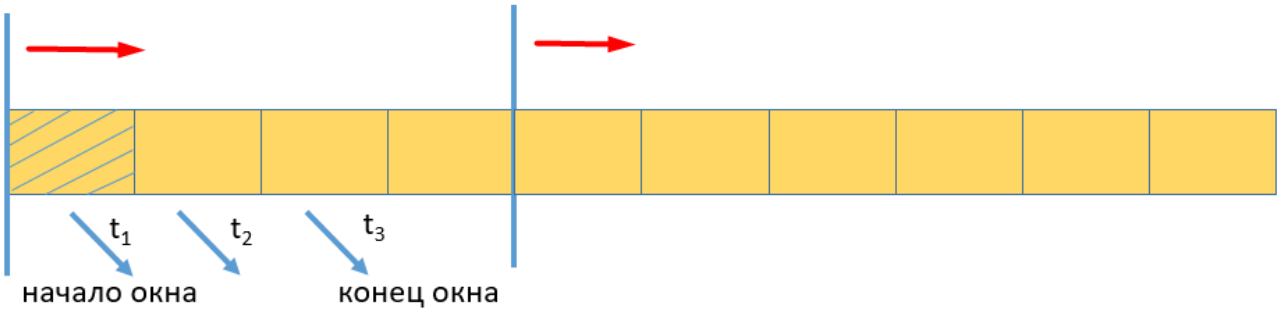
Более простая ситуация -- отправление данных по одному сегменту, каждый раз ждём подтверждения.



Пришедшие с верхних уровней данные делятся на сегменты. Отправляем один сегмент длиной n байт и ждём подтверждения. Принимающая сторона должна его получить, отправить нам сегмент с флагом ACK и в AN поставить n + 1. Затем выделяем следующий сегмент и повторяем.

Но бывает узлу A ждать ответа от узла B очень долго, в это время можно передавать данные. Это помогает разрешить механизм скользящего окна.

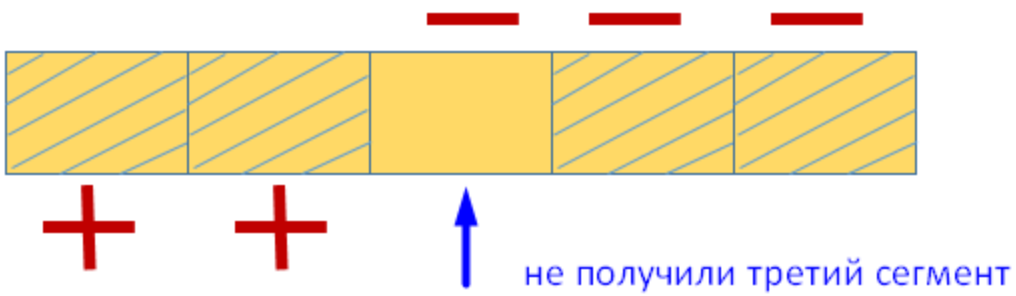
> ***Механизм скользящего окна*** -- отправление нескольких сегментов сразу (пачкой) до получения подтверждения их получения.



Берём сегменты (в нашем случае 4 шт), попадающие в окно, и отправляем последовательно, не дожидаясь ответа. После этого ждём подтверждение. Если подтверждение получения первого сегмента дошло раньше, чем отправили четвёртый сегмент, продолжаем отправлять, сдвинув окно, чтобы подтверждённый сегмент вышел за границу. Таким образом окно перемещается по данным.

А если узел B не получил данные?
Пусть потерялся 3 сегмент. Для каждого сегмента сторона A запускала свой таймер: $_{1\,2\,3}$.
По истечении таймера делаем вывод, что сегмент потерялся. Тогда происходит ***retransmission*** -- переотправка с новым таймером $_{3}$.

Принимающая сторона:

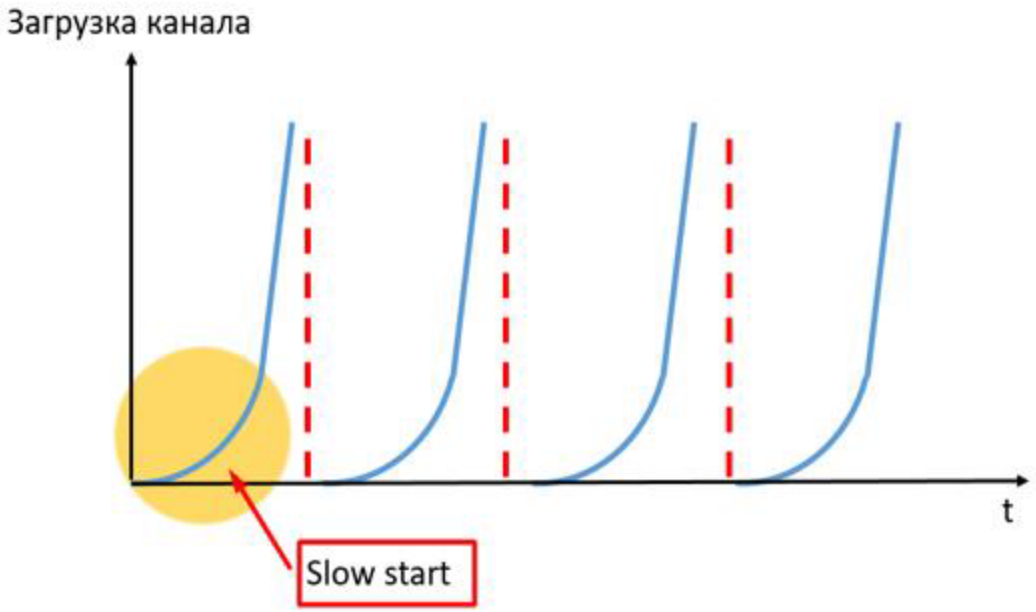


На узле B есть буфер, куда кладём полученные сегменты. Если не получили 3, но получили 4, 5, то последние мы не можем подтвердить. Мы подтверждаем всё до байта,

указанного в AN, т.е. данные без разрывов.

А если варьировать размер окна?
Чем больше размер окна, тем быстрее передача. Но B может не успевать принимать данные. На узле B дб механизм, регулирующий скорость поступления данных. Отправителю сообщается размер окна, который нужно использовать.

> ***Slow start*** -- при хорошей передаче мы увеличиваем размер окна, при потерях -- уменьшаем до одного.



Закрытие соединения в TCP

Один из вариантов закрытия:

Способы оптимизации TCP-соединения

- Задержанное подтверждение

Не выгодно отправлять от B большой поток пустых сегментов для подтверждения. Можно подтверждать через один сегмент: сначала получение первого сегмента, потом получение всех сегментов до третьего и самого третьего сегмента. То есть B будет задерживать отправку пустых сегментов, за это время могут прийти другие сегменты и мы сможем подтвердить сразу несколько.

- Алгоритм Нэйгла

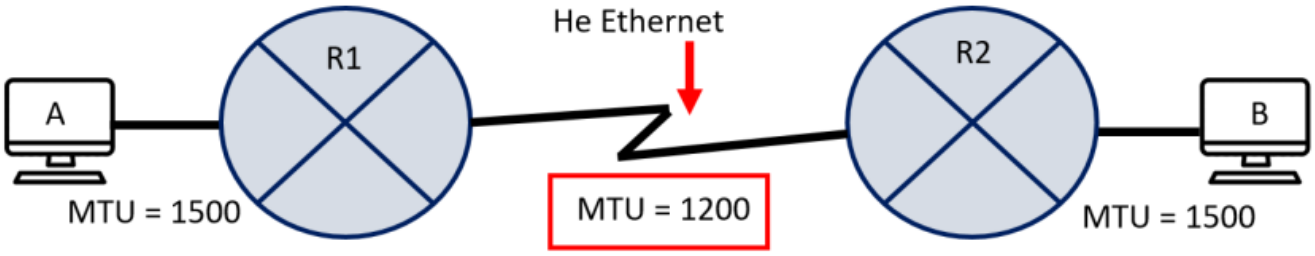
Не выгодно инкапсулировать малые порции данных. Делается задержка, чтобы отправить несколько порций данных в одном объёме.

- QoS -- Quality of Service

Использование флагов CWR и ECN-Echo для уведомления сторон о перегрузке (это делают роутеры).

- MSS -- Maximum Segment Size

Наш сегмент при инкапсуляции (>>> IP-пакет >>> Ethernet-фрейм) должен влезать в MTU.



Одно из решений -- использовать опцию ***MSS*** поля Options в заголовке TCP-сегмента. Допустим, у узла B MTU меньше, чем то, что может передавать A. При тройном рукопожатии A может вставить опцию MSS и сообщить максимальный размер сегмента со своей стороны. Аналогично делает B. Таким образом узлы договариваются о используемом размере сегмента.

При этом роутеры могут менять значение MSS, чтобы через них эти данные прошли.

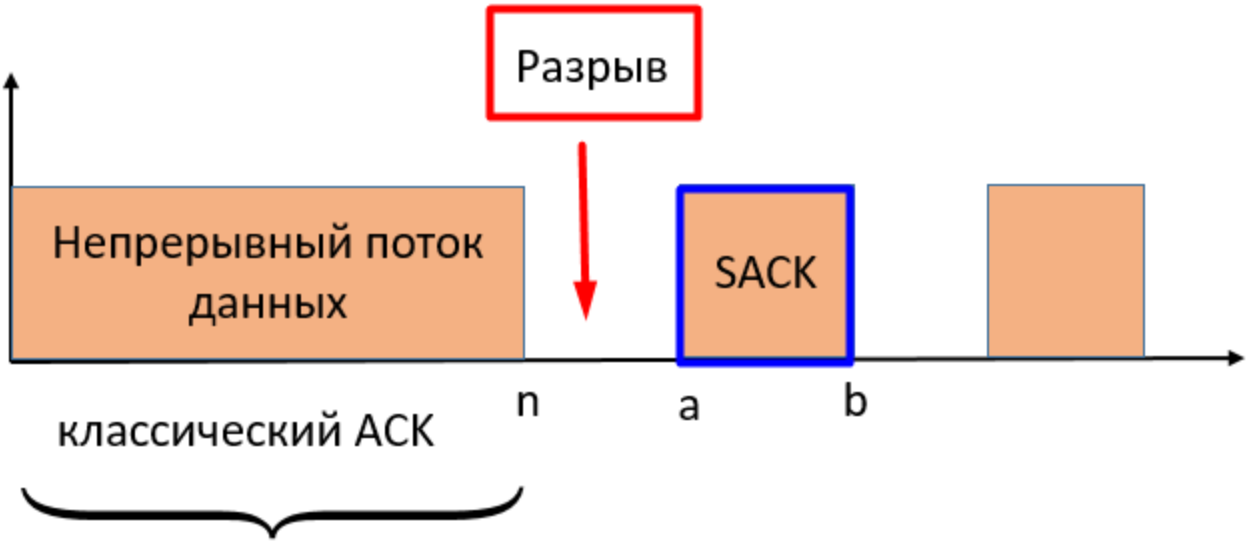
- SACK и NACK

Служат для подтверждения данных, которые пришли с разрывами.

> ***SACK*** -- подтверждение части данных, которая была доставлена после потери какого-то сегмента.

В поле Options можно подтвердить получение:

- [0, n] -- ACK
- [a, b] -- SACK



> ***NACK*** -- указание в явном виде отрезка, на котором данные не дошли.

TCP не использует в явном виде NACK. NACK позволяет не переотправлять дошедшие данные.

- Scaling factor

Актуально для широких ***RTT***.

> ***Scaling Factor*** -- число n, показывающее, во сколько раз нужно увеличить размер окна (поле Window Size), чтобы получить реальное окно.

2

- Fast Retransmission

> ***Fast Retransmission*** -- быстрая повторная отправка.

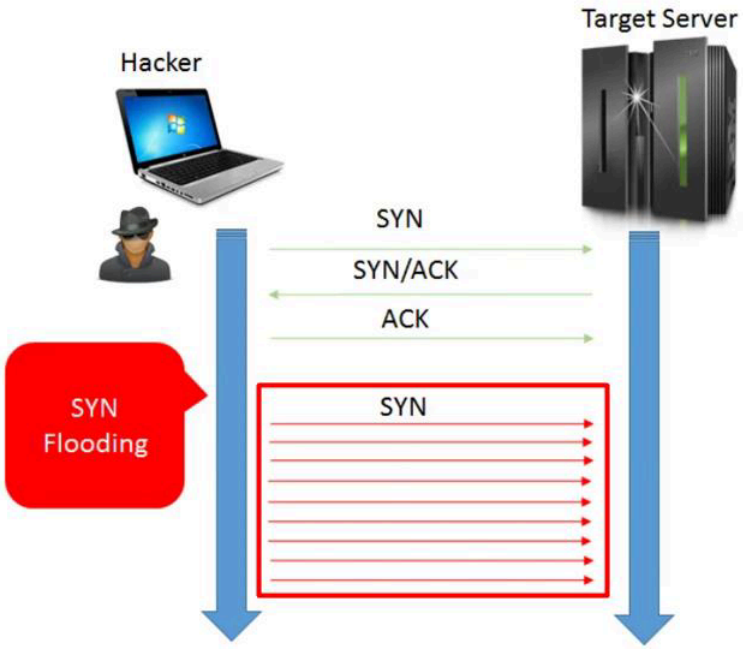
Применяется, когда не пришёл какой-то кусочек, но мы продолжаем принимать данные.

Отправляется сразу три подтверждения получения данных до текущего момента с выставленным флагом ACK. Отправитель сразу же переотправляет данные.

Также можно не схлопывать размер окна, а уменьшать понемногу:

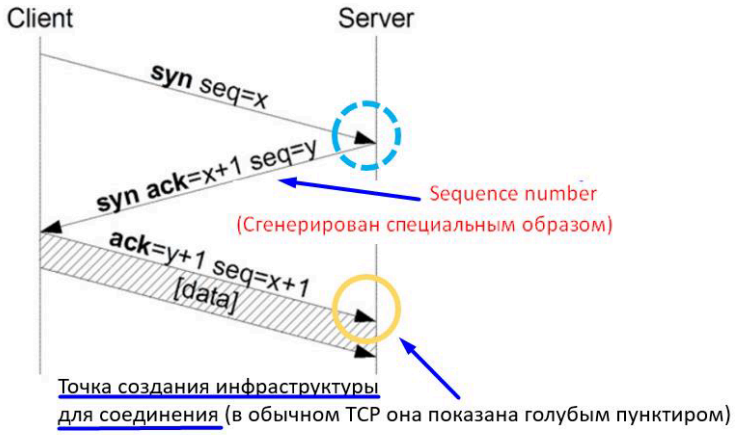
SYN cookie

Атака SYN flood:



Заваливая сервер кучей пустых сегментов с флагом SYN, мы заставляем сервер резервировать память, отправлять в ответ SYN + ACK >>> в итоге сервер замедляется.

> ***SYN cookie*** -- технология противодействия атакам SYN flood.

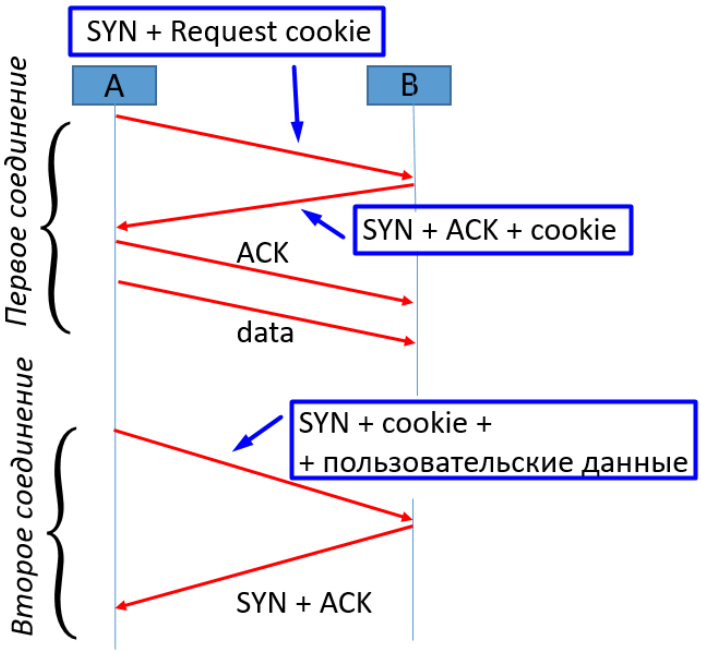


Сервер видит увеличение скорости открывающихся и незавершённых подключений. Сервер получает SYN, но не создаёт дополнительных структур. Он отвечает с SYN + ACK, генерируя SN особым образом, учтя адрес отправителя и номера портов. Если вдруг подключение настоящее, приходит ACK, сервер сравнивает его со своим SN и, если всё в порядке, принимает соединение.

При этом мы теряем все опции (MSS, Scaling Factor, SACK / NACK).

TFO

> ***TFO*** -- улучшает процедуру открытия повторного соединения.



Клиент отправляет вместе с SYN ещё и Request cookie. Обратно сервер отправляет свой cookie, сгенерированный особым образом. Клиент это число запомнит и при повторном подключении отправит вместе с пользовательскими данными.

Т.е. происходит то же тройное рукопожатие, но с обменом данными.