Транспортный уровень

- **Транспортный уровень** (англ. *Transport layer*) — 4-й уровень сетевой модели OSI предназначен для доставки данных. При этом не важно, какие данные передаются, откуда и куда, то есть он предоставляет сам механизм передачи. Блоки данных он разделяет на фрагменты, размер которых зависит от протокола, короткие объединяет в один, а длинные разбивает. Протоколы этого уровня предназначены для взаимодействия типа точка-точка.
  Пример: TCP, UDP, SCTP, RTP.

- Существует множество классов протоколов транспортного уровня, начиная от протоколов, предоставляющих только основные транспортные функции, например, функции передачи данных без подтверждения приема, и заканчивая протоколами, которые гарантируют доставку в пункт назначения нескольких пакетов данных в надлежащей последовательности, мультиплексируют несколько потоков данных, обеспечивают механизм управления потоками данных и гарантируют достоверность принятых данных.
- Некоторые протоколы транспортного уровня, называемые протоколами без установки соединения, не гарантируют, что данные доставляются по назначению в том порядке, в котором они были посланы устройством-источником. Некоторые транспортные уровни справляются с этим, собирая данные в нужной последовательности до передачи их на сеансовый уровень. Мультиплексирование (multiplexing) данных означает, что транспортный уровень способен одновременно обрабатывать несколько потоков данных (потоки могут поступать и от различных приложений) между двумя системами. Механизм управления потоком данных — это механизм, позволяющий регулировать количество данных, передаваемых от одной системы к другой. Протоколы транспортного уровня часто имеют функцию контроля доставки данных, заставляя принимающую данные систему отправлять подтверждения передающей стороне о приеме данных.

Transmission Control Protocol (TCP)

- транспортный механизм, предоставляющий поток данных, с предварительной установкой соединения, за счёт этого дающий уверенность в достоверности получаемых данных, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета. Реализация TCP, как правило, встроена в ядро ОС, хотя есть и реализации TCP в контексте приложения.

- Когда осуществляется передача от компьютера к компьютеру через Интернет, TCP работает на верхнем уровне между двумя конечными системами, например, браузером и веб-сервером. Также TCP осуществляет надежную передачу потока байтов от одной программы на некотором компьютере к другой программе на другом компьютере. Программы для электронной почты и обмена файлами используют TCP. TCP контролирует длину сообщения, скорость обмена сообщениями, сетевой трафик.

Заголовок сегмента TCP

Бит	0 — 3	4 — 9	10 — 15	16 — 31
0	Порт источника			Порт назначения
32	Номер последовательности			
64	Номер подтверждения			
96	Смещение данных	Зарезервировано	Флаги	Размер Окна
128	Контрольная сумма			Указатель важности
160	Опции (необязательное, но используется практически всегда)			
160/192 +	Данные			

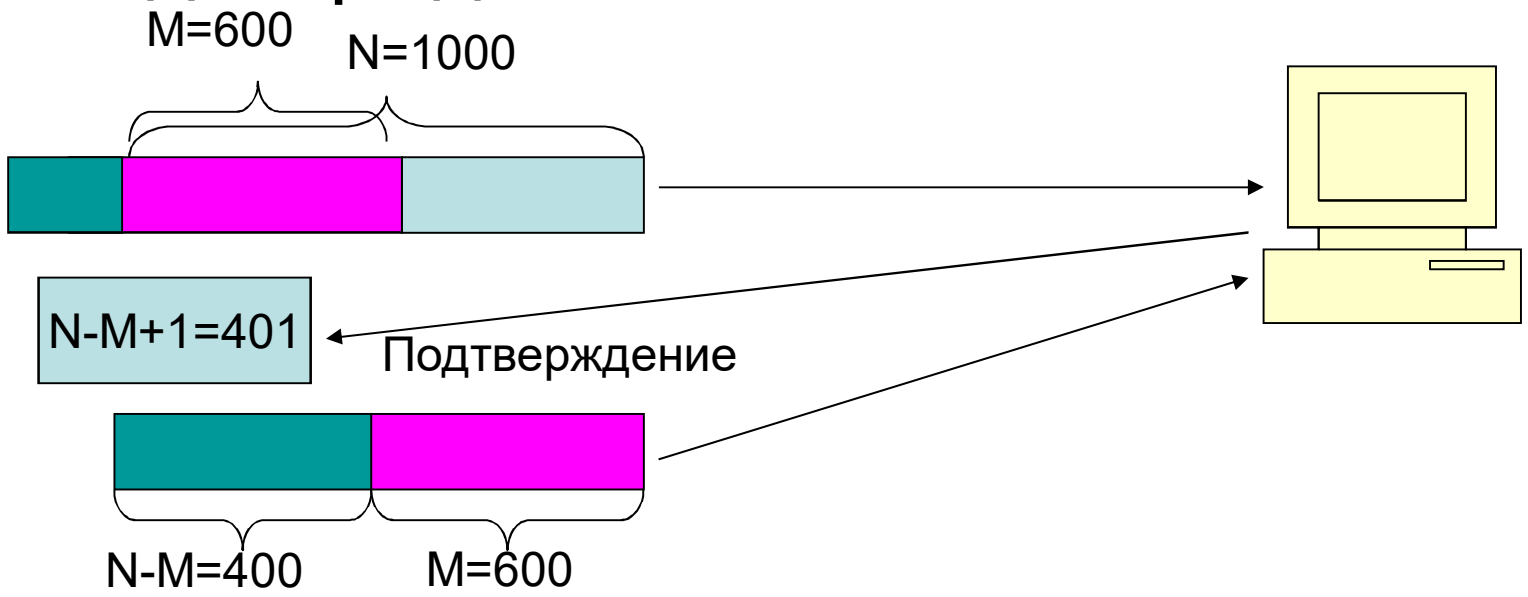
- Порт источника идентифицирует приложение клиента, с которого отправлены пакеты. По возвращении данные передаются клиенту на основании номера порта источника.
- Порт назначения идентифицирует порт, на который отправлен пакет.
- Существует набор служб (использующих для передачи данных TCP), за которыми закреплены определенные порты:
- 20/21 — FTP 22 — SSH 23 — Telnet 25 — SMTP 80 — HTTP 110 — POP3 194 — IRC (Internet Relay Chat)
- 443 — HTTPS (Secure HTTP) 1863 — MSN Messenger 2000 — Cisco SCCP (VoIP) 3389 — RDP
- 8080 — альтернативный порт HTTP

- **Номер последовательности**
- Номер последовательности выполняет две задачи:
- Если установлен флаг SYN, то это начальное значение номера последовательности — ISN (Initial Sequence Number), и первый байт данных, которые будут переданы в следующем пакете, будет иметь номер последовательности, равный ISN + 1.
- В противном случае, если SYN не установлен, первый байт данных, передаваемый в данном пакете, имеет этот номер последовательности.
- Поскольку поток TCP в общем случае может быть длиннее, чем число различных состояний этого поля, то все операции с номером последовательности должны выполняться по модулю 2^32. Это накладывает практическое ограничение на использование TCP. Если скорость передачи коммуникационной системы такова, чтобы в течение MSL (максимального времени жизни сегмента) произошло переполнение номера последовательности, то в сети может появиться два сегмента с одинаковым номером, относящихся к разным частям потока, и приёмник получит некорректные данные.

- **Номер подтверждения**
- Если установлен флаг ACK, то это поле содержит номер последовательности, ожидаемый получателем в следующий раз. Помечает этот сегмент как подтверждение получения.
- **Смещение данных**
- Это поле определяет размер заголовка пакета TCP в 4-байтных (4-октетных) словах. Минимальный размер составляет 5 слов, а максимальный — 15, что составляет 20 и 60 байт соответственно. Смещение считается от начала заголовка TCP.
- **Зарезервировано**
- Зарезервировано (6 бит) для будущего использования и должно устанавливаться в ноль. Из них два (5-й и 6-й) уже определены:
- **CWR** (Congestion Window Reduced) — Поле «Окно перегрузки уменьшено» — флаг установлен отправителем, чтоб указать, что получен пакет с установленным флагом ECE (RFC 3168)
- **ECE** (ECN-Echo) — Поле «Эхо ECN» — указывает, что данный узел способен на ECN (явное уведомление перегрузки) и для указания отправителю о перегрузках в сети (RFC 3168)

- **Флаги (управляющие биты)**
- Это поле содержит 6 битовых флагов:
- **URG** — Поле *«Указатель важности»* задействовано (англ. *Urgent pointer field is significant*)
- **ACK** — Поле *«Номер подтверждения»* задействовано (англ. *Acknowledgement field is significant*)
- **PSH** — (англ. *Push function*) инструктирует получателя протолкнуть данные, накопившиеся в приемном буфере, в приложение пользователя
- **RST** — Оборвать соединения, сбросить буфер (очистка буфера) (англ. *Reset the connection*)
- **SYN** — Синхронизация номеров последовательности (англ. *Synchronize sequence numbers*)
- **FIN** (англ. *final*, бит) — флаг, будучи установлен, указывает на завершение соединения (англ. *FIN bit used for connection termination*).

- **Окно**
- В этом поле содержится число, определяющее в байтах размер данных, которые отправитель готов принять.
- Отправитель отправив данные не более размера окна, должен ждать прихода подтверждения.



- **Псевдозаголовок**
- TCP-заголовок не содержит информации об адресе отправителя и получателя, поэтому даже при совпадении порта получателя нельзя с точностью сказать, что сообщение пришло в нужное место. Поскольку назначением протокола TCP является надёжная доставка сообщений, то этот момент имеет принципиальное значение. Эту задачу можно было решить разными способами. Самый очевидный — добавить информацию об адресе назначения в заголовок TCP, однако это, во-первых, приводит к дублированию информации, что снижает долю полезной информации переносимой TCP-сегментом, а во-вторых, нарушает принцип инкапсуляции модели OSI. Поэтому разработчики протокола пошли другим путём и использовали дополнительный псевдозаголовок. Псевдозаголовок не включается в TCP-сегмент. Он используется для расчета контрольной суммы перед отправлением сообщения и при его получении (получатель составляет свой псевдозаголовок, используя адрес хоста, с которого пришло сообщение, и собственный адрес, а затем считает контрольную сумму).

TCP-псевдозаголовок IPv4

Биты	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
0-31	IP-адрес отправителя (Source address)																															
32-63	IP-адрес получателя (Destination address)																															
64-95	0	0	0	0	0	0	0	0	Протокол (Protocol)								Длина TCP-сегмента (TCP length)															

TCP-псевдозаголовок IPv6

Биты	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
0-96	IP-адрес отправителя (Source address)																															
128-224	IP-адрес получателя (Destination address)																															
256	Длина TCP-сегмента (TCP length)																															
288	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Протокол верхнего уровня (Next header)							

- **Контрольная сумма**
- Поле контрольной суммы — это 16-битное дополнение к сумме всех 16-битных слов заголовка(включая псевдозаголовок) и данных. Если сегмент, 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. Биты заполнения (0) не передаются в сообщении и служат только для расчёта контрольной суммы. При расчёте контрольной суммы значение самого поля контрольной суммы принимается равным 0.
- **Указатель важности**
- 16-битовое значение положительного смещения от порядкового номера в данном сегменте. Это поле указывает порядковый номер октета, которым заканчиваются важные (urgent) данные. Поле принимается во внимание только для пакетов с установленным флагом URG.

- **Опции**
- Могут применяться в некоторых случаях для расширения протокола. Иногда используются для тестирования. На данный момент в опции практически всегда включают 2 байта NOP (в данном случае 0x01) и 10 байт, задающих timestamps. Вычислить длину поля опции можно через значение поля смещения.

- Механизм действия протокола
- В отличие от традиционной альтернативы — UDP, который может сразу же начать передачу пакетов, TCP устанавливает соединения, которые должны быть созданы перед передачей данных. TCP соединение можно разделить на 3 стадии:
- Установка соединения
- Передача данных
- Завершение соединения

Состояния сеанса TCP - Упрощённая диаграмма состояний TCP.

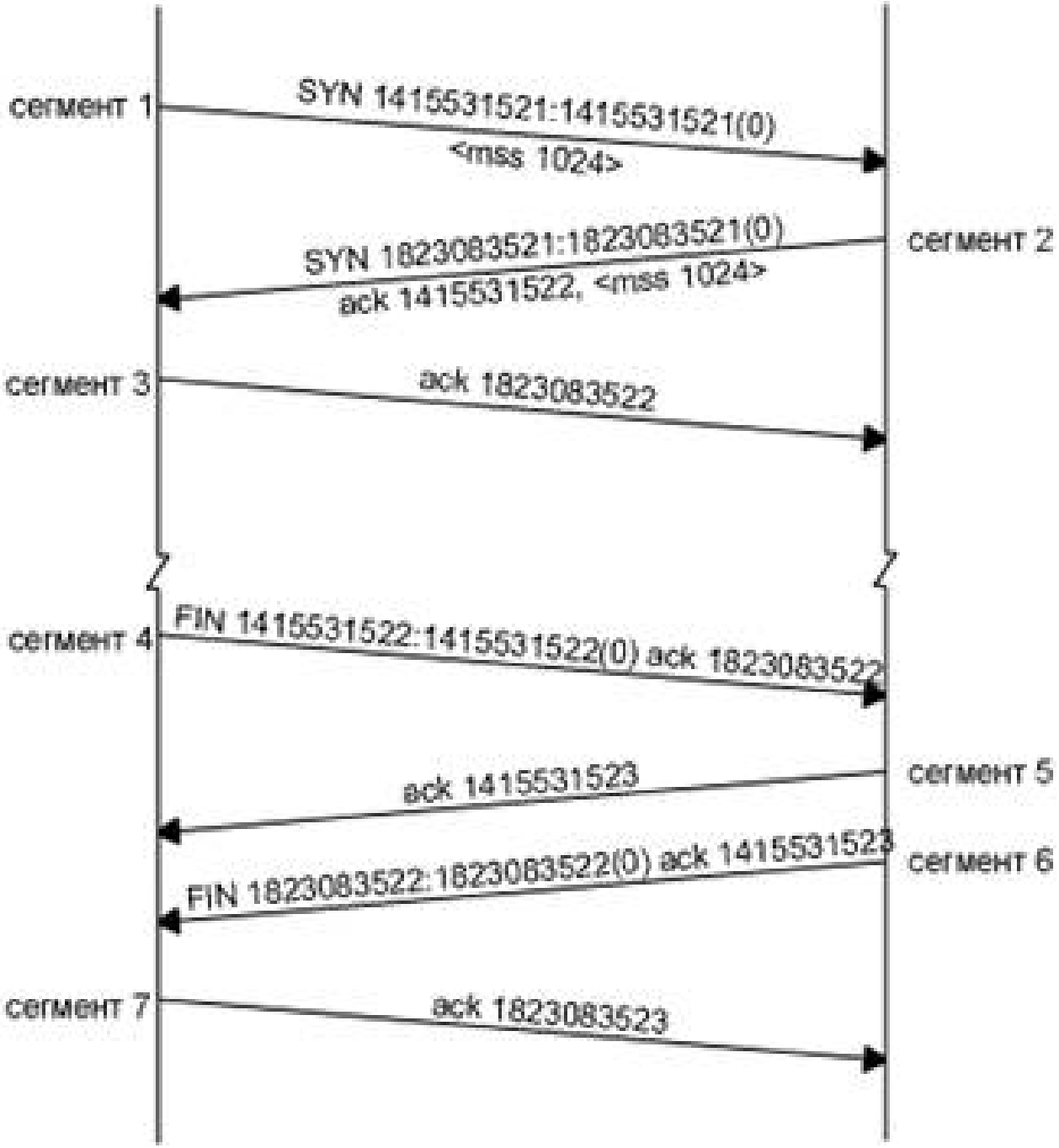
Состояния сеанса TCP	
CLOSED	Начальное состояние узла. Фактически фиктивное
LISTEN	Сервер ожидает запросов установления соединения от клиента
SYN-SENT	Клиент отправил запрос серверу на установление соединения и ожидает ответа
SYN-RECEIVED	Сервер получил запрос на соединение, отправил ответный запрос и ожидает подтверждения
ESTABLISHED	Соединение установлено, идёт передача данных
FIN-WAIT-1	Одна из сторон (назовём её узел-1) завершает соединение, отправив сегмент с флагом **FIN**
CLOSE-WAIT	Другая сторона (узел-2) переходит в это состояние, отправив, в свою очередь сегмент **ACK** и продолжает одностороннюю передачу
FIN-WAIT-2	Узел-1 получает **ACK**, продолжает чтение и ждёт получения сегмента с флагом **FIN**
LAST-ACK	Узел-2 заканчивает передачу и отправляет сегмент с флагом **FIN**
TIME-WAIT	Узел-1 получил сегмент с флагом **FIN**, отправил сегмент с флагом **ACK** и ждёт 2*MSL секунд, перед окончательным закрытием соединения
CLOSING	Обе стороны инициировали закрытие соединения одновременно: после отправки сегмента с флагом **FIN** узел-1 также получает сегмент **FIN**, отправляет **ACK** и находится в ожидании сегмента **ACK** (подтверждения на свой запрос о разъединении)

- **Установка соединения**
- Процесс начала сеанса TCP - обозначаемое как "рукопожатие" (handshake), состоит из 3 шагов.
- 1. Клиент, который намеревается установить соединение, посылает серверу сегмент с номером последовательности и флагом SYN.
- Сервер получает сегмент, запоминает номер последовательности и пытается создать сокет (буферы и управляющие структуры памяти) для обслуживания нового клиента.
  - В случае успеха сервер посылает клиенту сегмент с номером последовательности и флагами SYN и ACK, и переходит в состояние SYN-RECEIVED.
  - В случае неудачи сервер посылает клиенту сегмент с флагом RST.
- 2. Если клиент получает сегмент с флагом SYN, то он запоминает номер последовательности и посылает сегмент с флагом ACK.
- Если он одновременно получает и флаг ACK (что обычно и происходит), то он переходит в состояние ESTABLISHED.
- Если клиент получает сегмент с флагом RST, то он прекращает попытки соединиться.
- Если клиент не получает ответа в течение 10 секунд, то он повторяет процесс соединения заново.
- 3. Если сервер в состоянии SYN-RECEIVED получает сегмент с флагом ACK, то он переходит в состояние ESTABLISHED.
- В противном случае после тайм-аута он закрывает соккет и переходит в состояние CLOSED.

- Процесс называется "трехэтапным согласованием" ("three way handshake"), так как несмотря на то что возможен процесс установления соединения с использованием 4 сегментов (SYN в сторону сервера, ACK в сторону клиента, SYN в сторону клиента, ACK в сторону сервера), на практике для экономии времени используется 3 сегмента.
- Пример базового 3-этапного согласования:

TCP A TCP B

1. CLOSED L ISTEN
2. SYN-SENT --> <SEQ=100><CTL=SYN> --> SYN-RECEIVED
3. ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN,ACK> <-- SYN-RECEIVED
4. ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK> --> ESTABLISHED
5. ESTABLISHED <-- <SEQ=301><ACK=101><CTL=ACK> <-- ESTABLISHED


сегмент 1
SYN 1415531521:1415531521(0)
<mss 1024>
SYN 1823083521:1823083521(0)
ack 1415531522, <mss 1024>
сегмент 2
сегмент 3
ack 1823083522
сегмент 4
FIN 1415531522:1415531522(0) ack 1823083522
ack 1415531523
сегмент 5
FIN 1823083522:1823083522(0) ack 1415531523
сегмент 6
сегмент 7
ack 1823083523

- При обмене данными приемник использует номер последовательности, содержащийся в получаемых сегментах, для восстановления их исходного порядка. Приемник уведомляет передающую сторону о номере последовательности байт, до которой он успешно получил данные, включая его в поле «номер подтверждения». Все получаемые данные, относящиеся к промежутку подтвержденных последовательностей, игнорируются. Если полученный сегмент содержит номер последовательности больший, чем ожидаемый, то данные из сегмента буферизируются, но номер подтвержденной последовательности не изменяется. Если впоследствии будет принят сегмент, относящийся к ожидаемому номеру последовательности, то порядок данных будет автоматически восстановлен исходя из номеров последовательностей в сегментах.

- Для того, чтобы передающая сторона не отправляла данные интенсивнее, чем их может обработать приемник, TCP содержит средства управления потоком. Для этого используется поле «окно». В сегментах, направляемых от приемника передающей стороне в поле «окно» указывается текущий размер приемного буфера. Передающая сторона сохраняет размер окна и отправляет данных не более, чем указал приемник. Если приемник указал нулевой размер окна, то передача данных в направлении этого узла не происходит, до тех пор пока приемник не сообщит о большем размере окна.
- В некоторых случаях передающее приложение может явно затребовать протолкнуть данные до некоторой последовательности принимающему приложению, не буферизируя их. Для этого используется флаг PSH. Если в полученном сегменте обнаруживается флаг PSH, то реализация TCP отдает все буферизированные на текущий момент данные принимающему приложению. «Проталкивание» используется, например, в интерактивных приложениях. В сетевых терминалах нет смысла ожидать ввода пользователя после того, как он закончил набирать команду. Поэтому последний сегмент, содержащий команду, обязан содержать флаг PSH, чтобы приложение на принимающей стороне смогло начать её выполнение.

- **Завершение соединения**
- Завершение соединения можно рассмотреть в три этапа:
- Посылка серверу от клиента флагов FIN и ACK на завершение соединения.
- Сервер посылает клиенту флаги ответа ACK , FIN, что соединение закрыто.
- После получения этих флагов клиент закрывает соединение и в подтверждение отправляет серверу ACK , что соединение закрыто.

- **Известные проблемы**
- **Максимальный размер сегмента**
- TCP требует явного указания максимального размера сегмента (MSS) в случае, если виртуальное соединение осуществляется через сегмент сети, где максимальный размер блока (MTU) менее, чем стандартный MTU Ethernet (1500 байт).
- В протоколах туннелирования, таких как GRE, IPIP, а также PPPoE MTU туннеля меньше чем стандартный, поэтому сегмент TCP максимального размера имеет длину пакета больше, чем MTU. Поскольку фрагментация в подавляющем большинстве случаев запрещена, то такие пакеты отбрасываются.
- Проявление этой проблемы выглядит как «зависание» соединений. При этом «зависание» может происходить в произвольные моменты времени, а именно тогда, когда отправитель использовал сегменты длиннее допустимого размера.
- Для решения этой проблемы на маршрутизаторах применяются правила Firewall-а, добавляющие параметр MSS во все пакеты, инициирующие соединения, чтобы отправитель использовал сегменты допустимого размера.
- MSS может также управляться параметрами операционной системы.

- **Обнаружение ошибок при передаче данных**
- Хотя протокол осуществляет проверку контрольной суммы по каждому сегменту, используемый алгоритм считается слабым. Так в 2008 году не обнаруженная сетевыми средствами ошибка в передаче одного бита, привела к остановке серверов системыAmazon Web Services.
- В общем случае распределенным сетевым приложениям рекомендуется использовать дополнительные программные средства для гарантирования целостности передаваемой информации.
- **Атаки на протокол**
- *Основная статья:* Атаки на TCP
- Недостатки протокола проявляются в успешных теоретических и практических атаках, при которых злоумышленник может получить доступ к передаваемым данным, выдать себя за другую сторону или привести систему в нерабочее состояние.

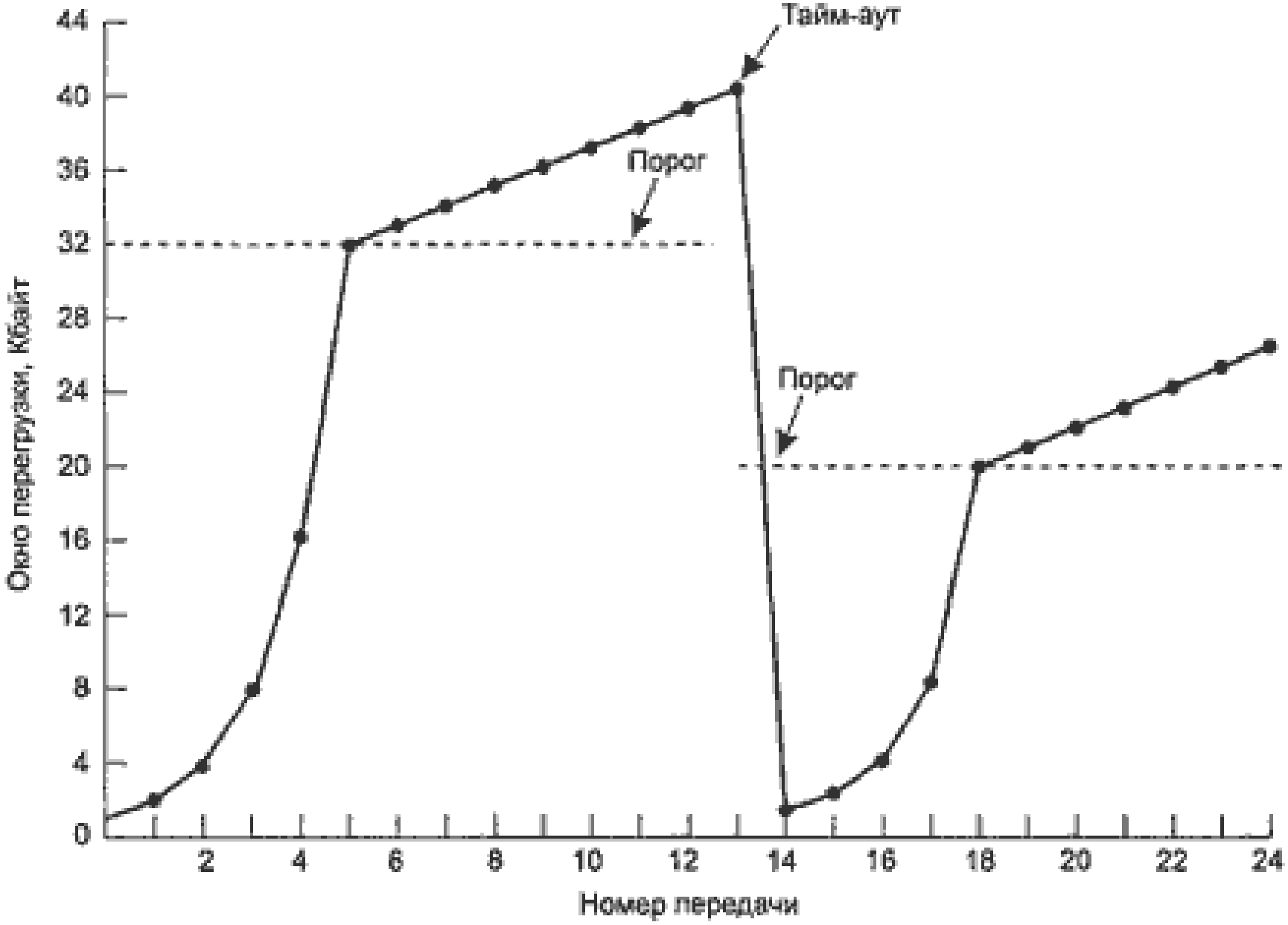
Алгоритм медленного пуска

- При установке соединения отправитель устанавливает размер окна перегрузки равным размеру максимального используемого в данном соединении сегмента. Затем он передает один максимальный сегмент. Если подтверждение получения этого сегмента прибывает прежде, чем истекает период ожидания, к размеру окна добавляется размер сегмента, то есть размер окна перегрузки удваивается, и посылаются уже два сегмента. В ответ на подтверждение получения каждого из сегментов производится расширение окна перегрузки на величину одного максимального сегмента. Допустим, размер окна равен п сегментам. Если подтверждения для всех сегментов приходят вовремя, окно увеличивается на число байтов, соответствующее п сегментам. По сути, подтверждение каждой последовательности сегментов приводит к удвоению окна перегрузки.

- Этот процесс экспоненциального роста продолжается до тех пор, пока не будет достигнут размер окна получателя или не будет выработан признак таймаута, сигнализирующий о перегрузке в сети. Например, если пакеты размером 1024, 2048 и 4096 байт дошли до получателя успешно, а в ответ на передачу па-кета размером 8192 байта подтверждение не пришло в установленный срок, окно перегрузки устанавливается равным 4096 байтам. Пока размер окна перегрузки остается равным 4096 байтам, более длинные пакеты не посылаются, независимо от размера окна, предоставляемого получателем. Этот алгоритм называется затяжным пуском, или медленным пуском. Однако он не такой уж и медленный (Jacobson, 1988). Он экспоненциальный. Все реализации протокола TCP обязаны его поддерживать.

Механизм борьбы с перегрузкой, применяемый в Интернете.

- Помимо окон получателя и перегрузки, в качестве третьего параметра в нем используется пороговое значение, которое изначально устанавливается равным 64 Кбайт. Когда возникает ситуация таймаута (подтверждение не возвращается в срок), новое значение порога устанавливается равным половине текущего размера окна перегрузки, а окно перегрузки уменьшается до размера одного максимального сегмента. Затем, так же как и в предыдущем случае, используется алгоритм затяжного пуска, позволяющий быстро обнаружить предел пропускной способности сети. Однако на этот раз экспоненциальный рост размера окна останавливается по достижении им порогового значения, после чего окно увеличивается линейно, на один сегмент для каждой следующей передачи. В сущности, предполагается, что можно спокойно урезать вдвое размер окна перегрузки, после чего постепенно наращивать его.



Пример алгоритма борьбы с перегрузкой, применяемого в Интернете

- Максимальный размер сегмента равен 1024 байт. Сначала окно перегрузки было установлено равным 64 Кбайт, но затем произошел таймаут, и порог стал равным 32 Кбайт, а окно перегрузки — 1 Кбайт (передача 0). Затем размер окна перегрузки удваивается на каждом шаге, пока не достигает порога (32 Кбайт). Начиная с этого момента, размер окна увеличивается линейно.

- Передача 13 оказывается несчастливой (как и положено), так как срабатывает таймаут. При этом пороговое значение устанавливается равным половине текущего размера окна (40 Кбайт пополам, то есть 20 Кбайт), и опять происходит затяжной пуск. После достижения порогового значения экспоненциальный рост размера окна сменяется линейным.

- Если таймаутов больше не возникает, окно перегрузки может продолжать расти до размера окна получателя. Затем рост прекратится, и размер окна останется постоянным, пока не произойдет таймаут или не изменится размер окна получателя. Прибытие ICMP-пакета SOURCE QUENCH (гашение источника) обрабатывается таким же образом, как и таймаут. Альтернативный (и более современный) подход описан в RFC 3168.

Управление таймерами в TCP

- В протоколе TCP используется много различных таймеров (по крайней мере, такова концепция). Наиболее важным из них является таймер повторной передачи. Когда посылается сегмент, запускается таймер повторной передачи. Если подтверждение получения сегмента прибывает раньше, чем истекает период таймера, таймер останавливается. Если же, наоборот, период ожидания истечет раньше, чем прибудет подтверждение, сегмент передается еще раз (а таймер запускается снова). Соответственно возникает вопрос: каким должен быть интервал времени ожидания?

- Предсказать, сколько потребуется времени для прохождения данных от отправителя до получателя и обратно, весьма непросто. Если выбрать значение интервала ожидания слишком малым, возникнут излишние повторные передачи, забивающие Интернет бесполезными пакетами. Если же установить значение этого интервала слишком большим, то из-за увеличения времени ожидания в случае потери пакета пострадает производительность. Более того, среднее значение и величина дисперсии времени прибытия подтверждений может изменяться всего за несколько секунд при возникновении и устранении перегрузки

- Решение заключается в использовании крайне динамичного алгоритма, постоянно изменяющего величину периода ожидания, основываясь на измерениях производительности сети. Алгоритм, применяемый в TCP, разработан Джекобсоном (Jacobson) в 1988 году и работает следующим образом. Для каждого соединения в протоколе TCP предусмотрена переменная RTT (Round-Trip Time — время перемещения в оба конца), в которой хранится наименьшее время получения подтверждения для данного соединения. При передаче сегмента запускается таймер, который измеряет время, требуемое для получения подтверждения, а так же запускает повторную передачу, если подтверждение не приходит в срок. Если подтверждение успевает вернуться прежде чем истечет период ожидания, TCP-сущность измеряет время, потребовавшееся для его получения (M). Затем значение переменной RTT обновляется по следующей формуле:

- RTT=a*RTT+(1-a)*M,

- где a — весовой коэффициент, обычно равный 7/8.

- Даже при известном значении переменной RTT выбор периода ожидания подтверждения оказывается задачей нетривиальной. Обычно в протоколе TCP это значение вычисляется как b*RTT. Остается только выбрать каким-нибудь хитрым образом соответствующее значение коэффициента b. В ранних реализациях протокола использовалось b = 2, однако экспериментально было показано, что постоянное значение b является негибким и плохо учитывает ситуации, при которых разброс значений времени прибытия подтверждения увеличивается.

- В 1988 г. Джекобсон предложил использовать значение b, грубо пропорциональное среднеквадратичному отклонению (дисперсии) функции плотности вероятности времени прибытия подтверждения. Таким образом, при увеличении разброса значений времени прибытия увеличивалось бы и значение b, и наоборот. В частности, он предложил использовать среднее линейное отклонение в качестве легко вычисляемой оценки среднеквадратичного отклонения. В его версии алгоритма для каждого соединения вводилась еще одна сглаженная переменная D, отклонение. При получении каждого подтверждения вычислялась абсолютная величина разности между ожидаемым и измеренным значениями |RTT- M| Сглаженное значение этой величины сохранялось в переменной D, вычисляемой по формуле

- D = aD + (1-a)|RTT-M|

- где b в общем случае могло выбираться отличным от значения, используемого в предыдущей формуле. Хотя значение переменной D и отличается от среднеквадратичного отклонения, оно достаточно хорошо подходит для данного алгоритма. Кроме того, Джекобсон показал, как можно вычислить значение этой переменной, используя только целочисленные сложение, вычитание и сдвиг, что явилось большим плюсом. В настоящее время этот алгоритм применяется в большинстве реализаций протокола TCP, а значение интервала ожидания устанавливается по формуле

- Время ожидания = RTT+ 4D.