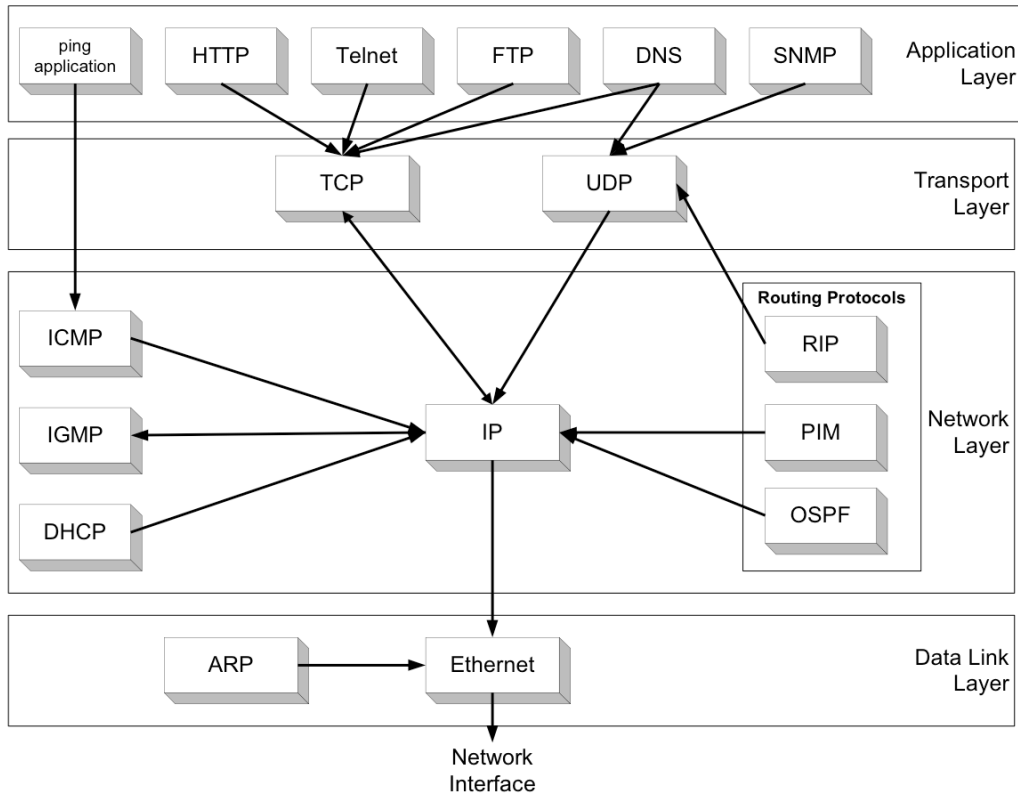


Транспортный уровень TCP/IP (продолжение).



5. Протокол TCP.

Transmission Control Protocol (TCP) (протокол управления передачей) - является обязательным протоколом стандарт TCP/IP. Он определён в стандарте RFC 793.

TCP — это протокол транспортного уровня, предоставляющий транспортировку (передачу) потока данных, с необходимостью предварительного установления соединения, благодаря чему гарантирует уверенность в целостности получаемых данных, также выполняет повторный запрос данных в случае потери данных или искажения. Помимо этого протокол TCP отслеживает дублирование пакетов и в случае обнаружения - уничтожает дублируемые пакеты.

В отличие от протокола UDP гарантирует целостность передаваемых данных и подтверждения отправителя о результатах передачи, и используется, например, при передаче файлов, где потеря одного пакета может привести к искажению всего файла.

При работе на хосте-отправителе протокол TCP рассматривает информацию, поступающую к нему от прикладных процессов, как неструктурированный поток байтов (см. рис.). Поступающие данные буферизуются средствами TCP. Для передачи на сетевой уровень из буфера «вырезается» некоторая непрерывная часть данных (**сегмент**), которая снабжается TCP-заголовком и отправляется в IP.

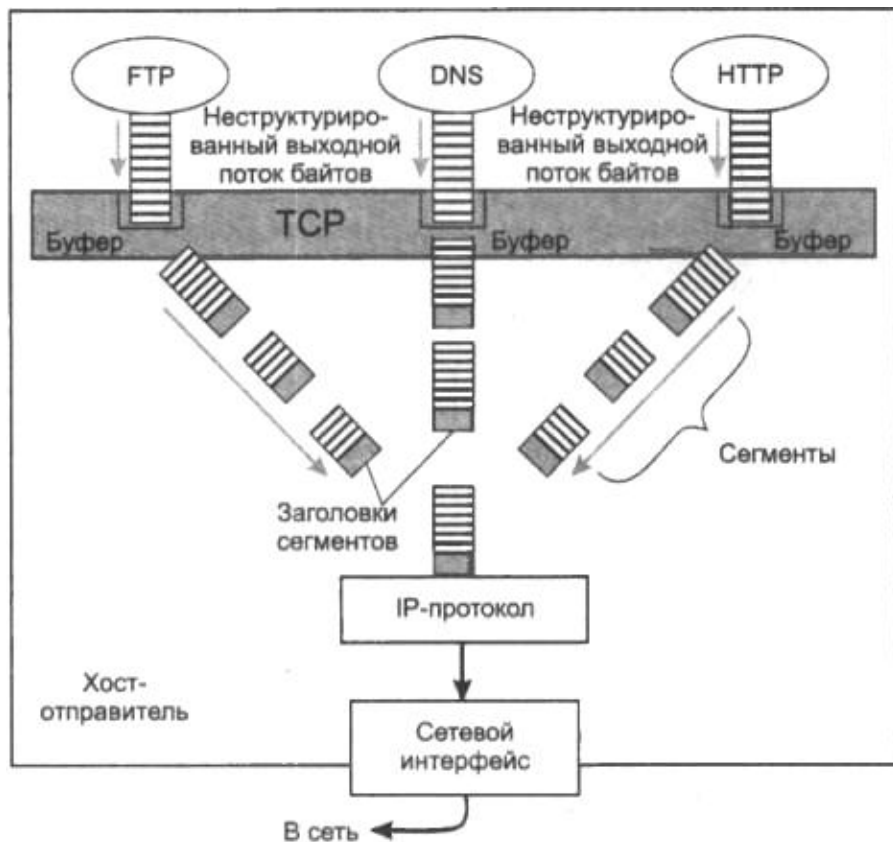


Рис. Формирование TCP-сегментов из потока байтов

5.1. Как обеспечивается надёжность TCP.

- **Сегмент.** Данные от приложения при отправке разбиваются на сегменты определенного размера.
- **Таймер.** Когда TCP посылает сегмент s_1 , он устанавливает таймер t_1 , ожидая, что с удаленного конца придёт подтверждение на этот сегмент. Если подтверждение не получено по истечении времени t_1 , сегмент передаётся повторно.
- **Подтверждение.** Когда TCP принимает данные от удаленной стороны соединения, он отправляет подтверждение.
- **Checksum.** TCP осуществляет расчет контрольной суммы для своего заголовка и данных. Контрольная сумма, рассчитывается на обоих концах соединения, с целью выявить любое изменение данных в процессе передачи. Если сегмент прибывает с неверной контрольной суммой, TCP отбрасывает его и подтверждение о получении не отправляется. (Ожидается, что отправитель отработает тайм-аут и осуществит повторную передачу.)
- **Упорядочивание.** Так как TCP сегменты передаются в виде IP дейтаграмм, а IP дейтаграммы могут прибывать беспорядочно, также беспорядочно могут прибывать и TCP сегменты. После получения данных TCP при необходимости изменяет их последовательность, в результате приложение получает данные в правильном порядке.
- **Дублирование.** Так как IP дейтаграмма может быть продублирована, узел, принимающий TCP, отбрасывает продублированные данные.
- **Поток.** TCP осуществляет контроль скорости потока данных. Каждая сторона имеет определенное пространство буфера. TCP на принимающей стороне позволяет удаленной стороне посылать данные только в том случае, если получатель может поместить их в буфер. Это предотвращает от переполнения медленных хостов быстрыми хостами.

5.2. Дейтаграмма и заголовок TCP.



5.2.1. Формат TCP заголовка.

Octet	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
00	Source Port															Destination Port																
04	Sequence Number (SN)																															
08	Acknowledgment Number (AN)																															
12	Data Offset		Reserved		N S	C W R	E C E	U R G	A C K	P S H	R S T	S Y N	F I N	Window Size																		
16	Checksum															Urgent Pointer																
20-n	Options (variable dimension) min=0																													PAD=0		
n+1-m	Data (variable dimension) min=0																															

- **Source port (Порт отправителя)** - в этом поле указывается номер порта отправителя, задает порт, на который при необходимости будет посылаться ответ. Если хостом-источником является клиент, то номер порта будет, скорее всего, эфемерным. Если источником является сервер, то его порт будет одним из "хорошо известных".
- **Destination port (Порт получателя)** - содержит порт получателя, если клиент - хост-получатель, то номер порта эфемерный, иначе (сервер - получатель) это "хорошо известный порт".
- **Sequence number – SN (Порядковый номер)** выполняет две задачи:
 - Если установлен флаг **SYN**, то это начальный порядковый номер (ISN - Initial Sequence Number), а первый байт данных, которые будут переданы в следующем пакете, будет иметь номер равный $ISN + 1$.
 - Если **SYN** не установлен, то первый байт данных, передаваемый в данном пакете, имеет этот указанный порядковый номер.

Поскольку поток TCP в общем случае может быть длиннее, чем число различных состояний этого поля, то все операции с порядковым номером должны выполняться по модулю 2^{32} . Это накладывает практическое ограничение на использование TCP. Если скорость передачи коммуникационной системы такова, что в течение MSL (максимального времени жизни сегмента) произошло **переполнение порядкового номера**, то в сети может появиться два сегмента с одинаковым номером, но, относящихся к разным частям потока, и приёмник может получить некорректно упорядоченные данные.

- **Acknowledgment number – AN (Номер подтверждения)** - если установлен флаг ACK, то это поле содержит номер последовательности, ожидаемый получателем в следующий раз. Это подтверждает получение всех предыдущих байт. Первый ACK, переданный каждой стороной подтверждает ISN другой стороны (без данных).

- **Data offset (Длина заголовка TCP)** – 4 бита, измеряется словами по 32 бита. Min размер составляет 5 слов, а Max - 15, что составляет заголовок 20 и 60 байт соответственно.
- **Reserved (Зарезервировано)** (3 бит) для будущего использования, должно устанавливаться в ноль.
- **Flags (Флаги)** – управляющие биты:
 - **NS** – ECN-nonce concealment protection (защита сокрытия experimental: RFC 3540).
 - **CWR** (Congestion Window Reduced - окно перегрузки уменьшено) - устанавливается отправителем, подтверждает получение пакета с установленным ECE (RFC 3168)
 - **ECE** (ECN-Echo - эхо ECN) — данный узел способен на ECN (явные уведомления о перегрузки) и для указания отправителю о перегрузках в сети (RFC 3168)
 - **URG** - флаг срочности, включает "Указатель срочности", иначе поле игнорируется.
 - **ACK** - подтверждение, включает "Номер подтверждения", иначе поле игнорируется.
 - **PSH** - флаг требует выполнения операции push, модуль TCP должен срочно передать пакет программе.
 - **RST** - флаг прерывания соединения, используется для отказа в соединении
 - **SYN** - флаг синхронизация ISN, используется при установлении соединения.
 - **FIN** - флаг окончание передачи со стороны отправителя.
- **Window Size (Размер окна)** - количество байт, которые готов принять получатель без подтверждения (для контроля потока).
- **Checksum (Контрольная сумма)** - включает псевдозаголовок, заголовок и данные, подсчитывается аналогично IP, ICMP, UDP – сложение 16-битных слов с дополнением до 1.
- **Urgent pointer (Указатель срочности)** - указывает последний байт срочных данных, на которые надо немедленно реагировать. Если URG установлен.
- **Options (Дополнительные параметры)** – необязательное поле имеющее переменную

длину, используется для решения вспомогательных задач, например, для согласования максимального размера сегмента для сеанса, параметр масштабирования окна и др.

- **Pad (Заполнитель)** – для выравнивания заголовка до 32-битных слов. Биты заполнения (0) не передаются в сообщении и служат только для расчёта контрольной суммы.
- **Data (Данные)** – поле переменной длины с полезной нагрузкой, теоретически max размер поля составляет в отсутствии опций 65495 байт, но, практически max задаётся значением MSS (Maximum Segment Size). Значение MSS может быть задано при установлении соединения каждой из сторон независимо через опцию. Для Ethernet с учётом MTU MSS=1452 байта.

5.2.2. Опции TCP.



Рис. Формат опций для TCP-сегментов.

В поле вид записывается код опции, поле LEN содержит число октетов в описании опции, включая поля вид и LEN. В настоящее время определены опции вида:

- 0 - Конец списка опций.
- 1 - Нет опции. Используется для заполнения поля опции до числа октетов, кратного 4.
- 2 - Максимальный размер сегмента (MSS).
- В RFC определены опции вида=4,5,6,7 и опции 11, 12 и 13.

5.2.3. Освобождение от расчёта контрольной суммы.

Многие реализации стека TCP/IP предоставляют возможности использования аппаратной поддержки для автоматического расчёта контрольной суммы в сетевом адаптере до передачи в сеть или после приёма из сети. Это освобождает операционную систему от использования тактов процессора при вычислении контрольной суммы.

5.2.4. Псевдо-заголовок.

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. Эту задачу надёжной доставки 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 псевдо-заголовок:

Octet	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
00	Source IP address																															
04	Destination IP address																															
08	0								Protocol=6								TCP Length [Bytes]															

TCP-псевдо-заголовок IPv4.

Octet	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
00	Source IP address																															
16	Destination IP address																															
32	TCP Length [Bytes]																															
36	0																								Protocol=6							

TCP-псевдо-заголовок IPv6.

Псевдо-заголовок не включается в TCP-сегмент. Он используется для расчета контрольной суммы перед отправлением сообщения и при его получении (получатель составляет свой псевдо-заголовок, используя адрес хоста, с которого пришло сообщение, и собственный адрес, а затем считает контрольную сумму).

Псевдо-заголовок формируется не только для TCP, но и для UDP.

5.2.4. Пример структуры заголовка TCP в сетевом анализаторе Wireshark.

Wireshark packet capture showing a TCP ACK packet. The packet list shows a packet from 10.100.100.100 to 10.100.100.102, port 139 to 1423. The packet details show the TCP header with flags ACK, window size 255, and sequence number 56. The packet bytes show the raw data in hexadecimal and ASCII.

No.	Time	Source	Destination	Protocol	Length	Info
90	8.403457	10.100.100.100	10.100.100.102	TCP	54	watilapp > netbios-ssn [ACK] Seq=1 Ack=2
249	27.321783	10.100.100.100	10.100.100.100	SMB	109	Locking AndX Request, FID: 0x800b
250	27.321903	10.100.100.100	10.100.100.100	SMB	109	Locking AndX Request, FID: 0x800b
252	27.521618	10.100.100.100	10.100.100.100	TCP	60	netbios-ssn > essbase [ACK] Seq=56 Ack=56
344	40.521921	10.100.100.102	5.79.79.44	TCP	66	nrcabq-lm > http [SYN] Seq=0 win=13068
345	40.554112	10.100.100.100	10.100.100.100	DCERPC	186	Request: call_id: 6 Fragment: Single opnum
346	40.554622	10.100.100.100	10.100.100.100	DCERPC	186	Request: call_id: 3 Fragment: Single opnum

Frame 252: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0

Ethernet II, Src: Hewlett-1e:15:9e (00:21:5a:1e:15:9e), Dst: Msi_63:4f:39 (00:16:17:63:4f:39)

Internet Protocol Version 4, Src: 10.100.100.100 (10.100.100.100), Dst: 10.100.100.102 (10.100.100.102)

Transmission Control Protocol, Src Port: netbios-ssn (139), Dst Port: essbase (1423), Seq: 56, Ack: 56

Source port: netbios-ssn (139)

Destination port: essbase (1423)

[Stream index: 1]

Sequence number: 56 (relative sequence number)

Acknowledgment number: 56 (relative ack number)

Header length: 20 bytes

Flags: 0x010 (ACK)

Window size value: 255

[Calculated window size: 255]

[window size scaling factor: -1 (unknown)]

Checksum: 0x0aa7 [validation disabled]

[SEQ/ACK analysis]

0000 00 16 17 63 4f 39 00 21 5a 1e 15 9e 08 00 45 00 ...c09.! Z.....E.

0010 00 28 28 d3 40 00 80 06 f4 6a 0a 64 64 64 0a 64 .((.@... .j.ddd.d

0020 64 66 00 8b 05 8f 32 9a 84 ce 88 a4 80 75 50 10 df....2.uP.

0030 00 ff 0a a7 00 00 00 00 00 00 00 00 00 00 00 00

5.3. Механизм работы протокола TCP.

Перед передачей данных TCP устанавливает двунаправленное соединение. Хост, иницирующий соединение называют «клиентом», а другой узел «сервером». TCP сеанс связи можно разделить на 3 стадии:

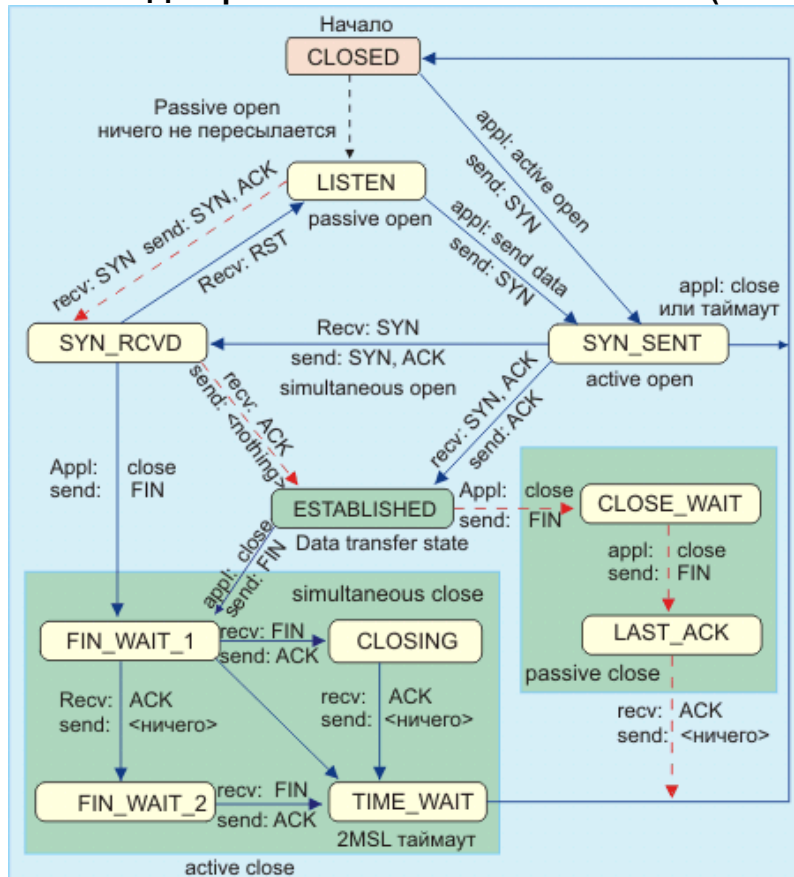
- Установка соединения
- Передача данных
- Завершение соединения

5.3.1. Состояние сеансов TCP.

Состояние существующих сеансов TCP можно посмотреть командой `netstat -np tcp`. Например,

```
MBA-ys:~ ys$ netstat -np tcp
Active Internet connections
Proto Recv-Q Send-Q Local Address           Foreign Address         (state)
tcp4    0      0 192.168.111.4.54474    108.160.172.225.443    ESTABLISHED
tcp4    31      0 192.168.111.4.54473    54.230.228.44.443     CLOSE_WAIT
tcp4    0      0 192.168.111.4.54470    162.125.32.129.443    ESTABLISHED
tcp4    31      0 192.168.111.4.54466    108.160.173.130.443   CLOSE_WAIT
tcp4    0      0 192.168.111.4.54454    52.55.159.190.443     CLOSE_WAIT
tcp4    0      0 192.168.111.4.54422    108.160.172.193.443    LAST_ACK
tcp4    0      0 192.168.111.4.54301    64.233.161.108.993    ESTABLISHED
```

Диаграмма состояний TCP сеанса. (W.R. Stevens, TCP/IP Illustrated. 1993.)



Обозначения

- обозначает нормальный переход для клиента
- → обозначает нормальный переход для сервера
- appl: обозначает изменение состояния, когда приложение выдает команду
- recv: обозначает изменение состояния, когда получен сегмент
- send: обозначает то, что посылается при этом переходе

Машина конечных состояний протокола TCP имеет 11 состояний (см. RFC-793).

TCP не предусматривает изменения состояний при посылке или получении обычных пакетов с данными.

Четыре состояния в левом нижнем углу соответствуют активному закрытию.

Два справа относятся к пассивному закрытию.

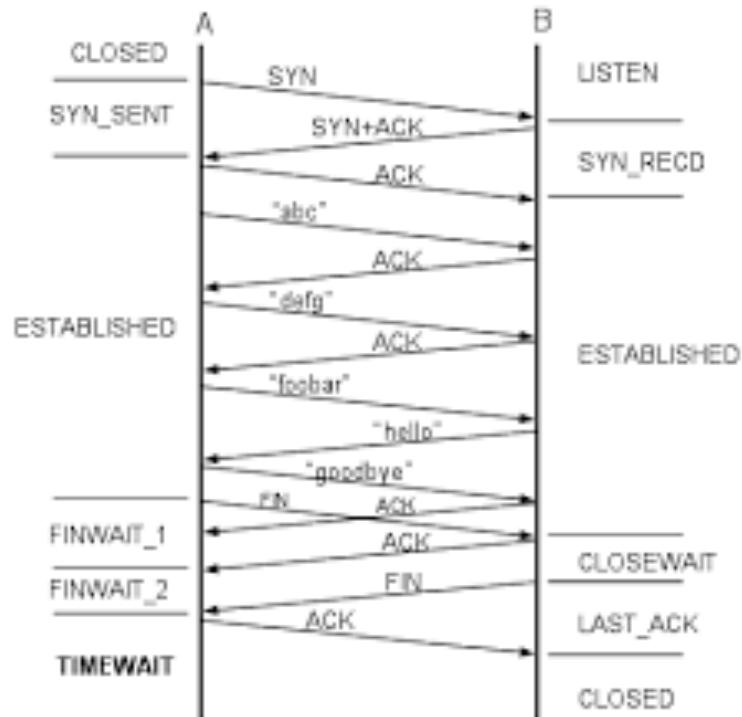
Переход из **SYN_RCVD** в **LISTEN** возможен, если переход в **SYN_RCVD** был из **LISTEN**, а не из **SYN_SENT** (при одновременном открытии двух соединений, был получен RST вместо финального ACK).

Состояние сеанса TCP	
CLOSED	Начальное состояние клиента. Фактически фиктивное.
LISTEN	Сервер ожидает запросов установления соединения от клиента
SYN-SENT	Клиент отправил запрос 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 (узел-1) завершает соединение, отправив сегмент с FIN
FIN-WAIT-2	Узел-1 получает ACK, продолжает чтение и ждёт получения сегмента с FIN. Состояние FIN_WAIT_2 сопряжено со случаем, когда одна сторона послала сегмент FIN, а другая сторона подтвердила его получение. Если данное соединение не нужно, можно ждать, когда приложение другой стороны получит код конца файла и пришлет свой флаг FIN. Только после этого система перейдет из состояний FIN_WAIT_2 в состояние TIME_WAIT. Теоретически такое ожидание может быть бесконечным. Другая сторона при этом остается в состоянии CLOSE_WAIT, пока приложение не вызовет функцию close. Для решения проблемы часто вводят дополнительный таймер.
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)

	В TCP возможна ситуация, когда обе стороны запускают процедуру закрытия одновременно (посылают FIN), что в протоколе TCP вполне допустимо. Каждая из сторон при этом переходит из состояния ESTABLISHED в состояние FIN_WAIT_1 (после вызова операции closed). По получении FIN стороны переходят из состояния FIN_WAIT_1 в состояние CLOSING и посылают ACK. После получения ACK происходит переход в состояние TIME_WAIT.
TIME-WAIT	Узел-1 получил сегмент с флагом FIN, отправил сегмент с флагом ACK и ждёт $2 \cdot \text{MSL}$ (Maximum Segment Lifetime), перед окончательным закрытием соединения. RFC-793 рекомендует $\text{MSL} \sim 2 \text{ min}$. Сокет, использовавшийся данным соединением не сможет быть задействован другим соединением в течение $2 \cdot \text{MSL}$. Все сегменты данного соединения, задержавшиеся в пути, во время TIME_WAIT отбрасываются. Это гарантирует то, что сегменты старого соединения не будут восприняты новым соединением. Такая процедура препятствует перезапуску серверов в течение 1-4 минут, так как в течение данного времени не могут использоваться стандартные значения номеров портов.
CLOSE-WAIT	Другая сторона (узел-2) переходит в это состояние, отправив, в свою очередь сегмент ACK и продолжает одностороннюю передачу.
LAST-ACK	Узел-2 заканчивает передачу и отправляет сегмент с флагом FIN

5.3.2. Установка соединения TCP.

Соединение устанавливается по инициативе клиентской части приложения. Процесс начала сеанса TCP (называемый «рукопожатие» - *handshake*), состоит из трёх шагов.



1. Клиент посылает серверу запрос на соединение (сегмент с номером последовательности s-ISR и флагом SYN, но без данных).

- Сервер получает сегмент, запоминает s-ISR и пытается создать сокет (буферы и управляющие структуры памяти) для обслуживания нового клиента.
- В случае успеха сервер посылает клиенту сегмент с номером последовательности s-ISR 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 s-ISR и посылает сегмент с флагом ACK (но, без данных).

- Если клиент получает и флаг ACK, то он переходит в состояние ESTABLISHED.
- Если клиент не получает ответа за 10 сек, то он повторяет процесс соединения заново.
- Если клиент получает сегмент с флагом RST, то он прекращает попытки соединиться.

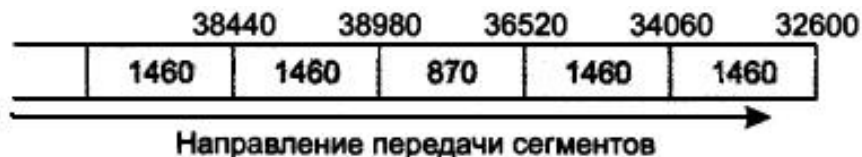
3. Если сервер в состоянии SYN-RECEIVED получает сегмент с флагом ACK, то он переходит в состояние ESTABLISHED.

- В противном случае после тайм-аута (величина регулируется в настройках стека) он закрывает сокет и переходит в состояние CLOSED.

После установления соединения TCP, эти два хоста могут передавать данные друг другу, так как TCP-соединение является полнодуплексным, они могут передавать данные одновременно.

5.3.3. Передача данных.

При обмене данными приёмник использует SN, содержащийся в получаемых сегментах, для восстановления исходного порядка сегментов, см. рис.



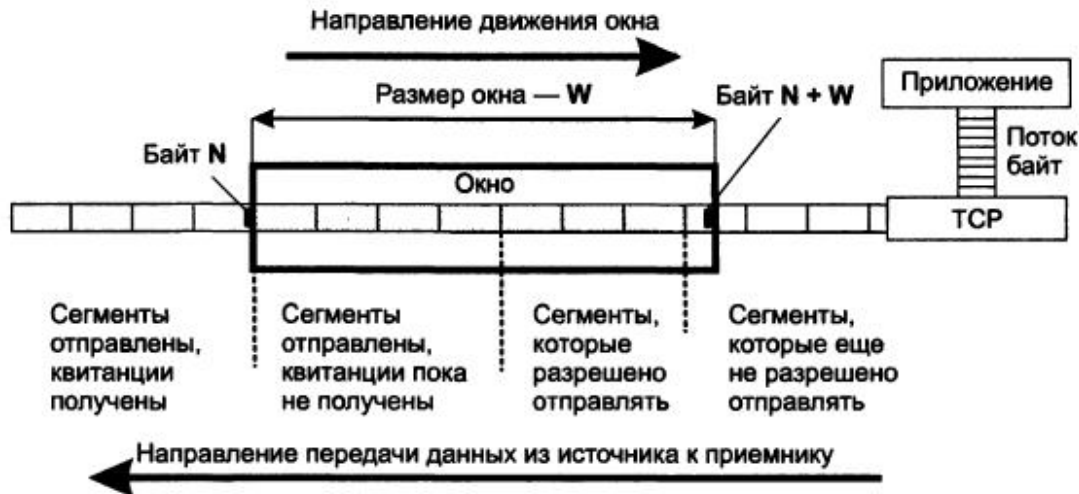
Приёмник уведомляет передающую сторону об успешно полученных данных, включая в поле AN следующий номер ещё не полученного сегмента (SN).

Все повторно получаемые данные, относящиеся к промежутку подтвержденных последовательностей, игнорируются.

Если получаемый сегмент содержит SN больший, чем ожидаемый, то данные из сегмента не отбрасываются, а буферизируются, но номер подтвержденной последовательности AN не изменяется. Если впоследствии будет принят сегмент, относящийся к ожидаемому номеру последовательности, то порядок данных будет автоматически восстановлен исходя из номеров последовательностей в сегментах.

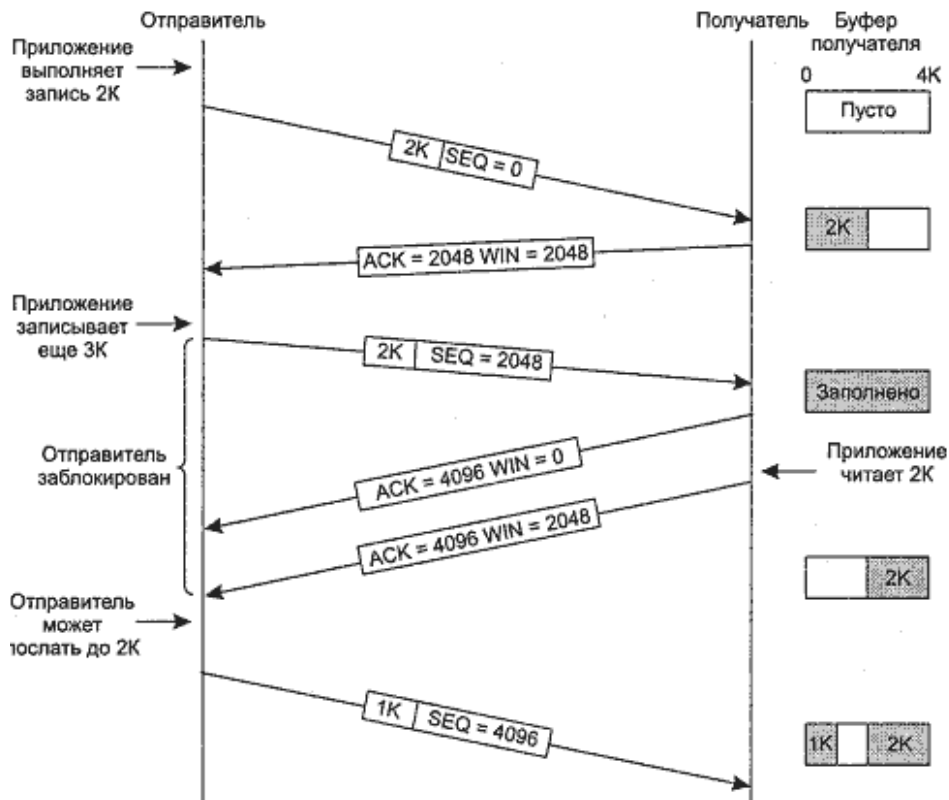
5.3.4. Управление перегрузками.

Поле Window (WIN). Для того, чтобы передающая сторона не отправляла данные интенсивнее, чем их может обработать приёмник, **TCP содержит средства управления потоком**. Для этого в сегментах, направляемых от приёмника передающей стороне, в поле Window («окно») указывается текущий размер приёмного буфера W .



Передающая сторона сохраняет размер окна и отправляет неподтверждённых данных не более, чем размер W . Размер окна в сегментах определяется соотношением: **$WIN > RTT * B / MSS$** , где B - полоса пропускания канала бит/с, MSS - max размер сегмента в битах.

Если приёмник указал WIN=0, то передача данных в направлении этого узла не происходит, пока приёмник не сообщит о большем размере окна, см. рис.



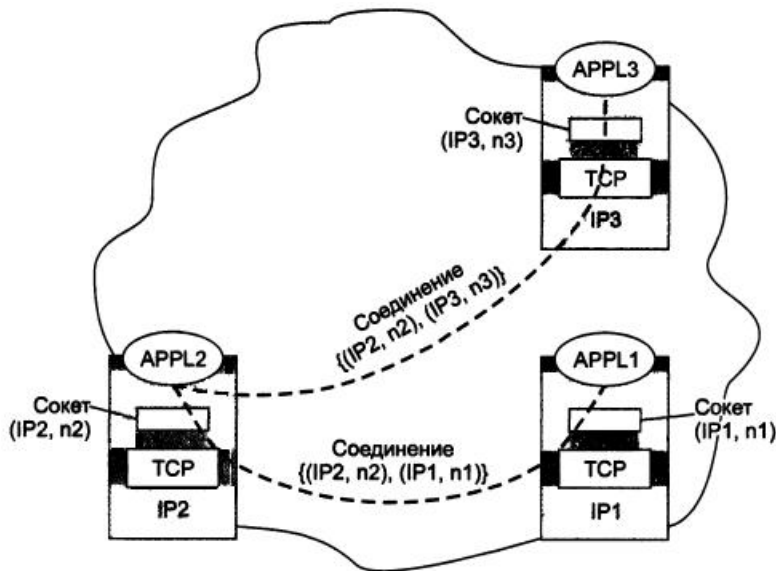
Реализовано множество алгоритмов управления потоком, определяющих баланс между скоростью передачи и перегрузкой. Например: медленный старт, увеличение прибавлением, мультипликативное уменьшение и другие алгоритмы. Эти алгоритмы могут анализировать и регулировать такие параметры как RTT, MTU, MSS, WIN, MSL, таймеры и др.

5.3.5. Флаг PSH. В некоторых случаях передающее приложение может явно затребовать протолкнуть данные принимающему приложению. Если в сегменте обнаруживается флаг PSH, то TCP отдает все буферизированные на этот момент данные принимающему приложению.

«Проталкивание» используется, 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.

5.3.6. Сокеты и соединения.

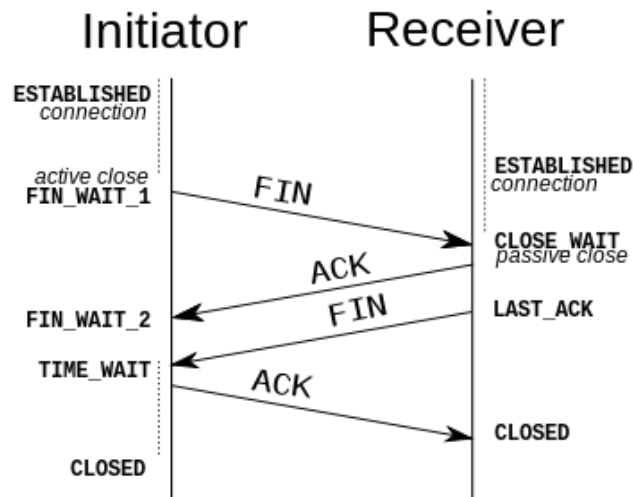
Один сокет может участвовать в нескольких соединениях, см. рис.



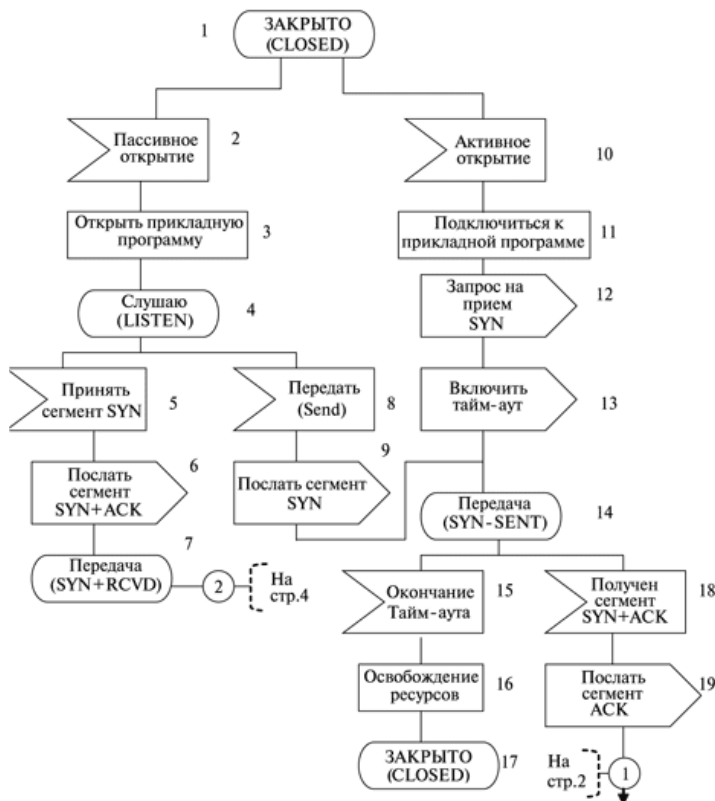
5.3.7. Завершение соединения.

Соединение может быть разорвано в любой момент по инициативе любой стороны. Завершение соединения происходит в три этапа:

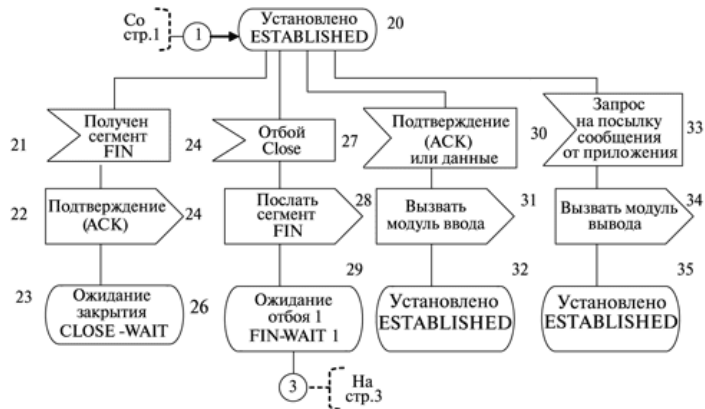
1. Сторона1 посылает стороне2 флаг FIN на завершение соединения.
2. Сторона2 посылает стороне1 флаги ответа ACK , FIN, что соединение закрыто.
3. После получения этих флагов сторона1 закрывает соединение и в подтверждение отправляет стороне2 ACK , что соединение закрыто.

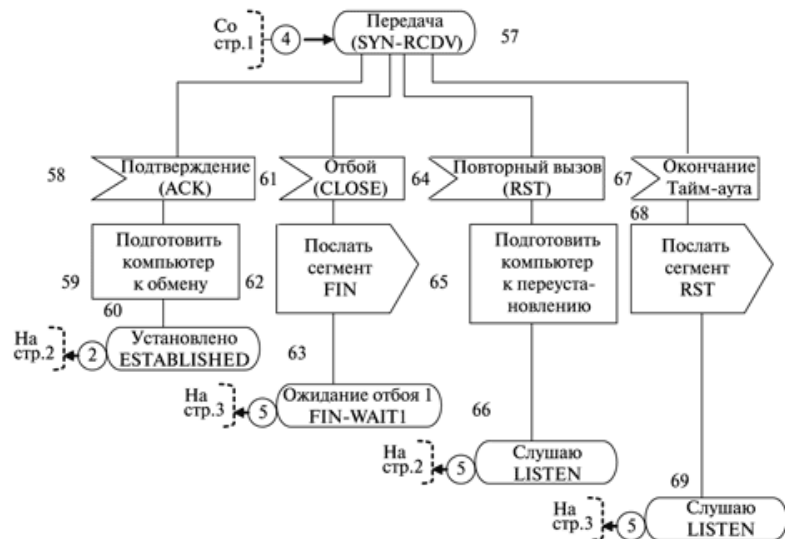
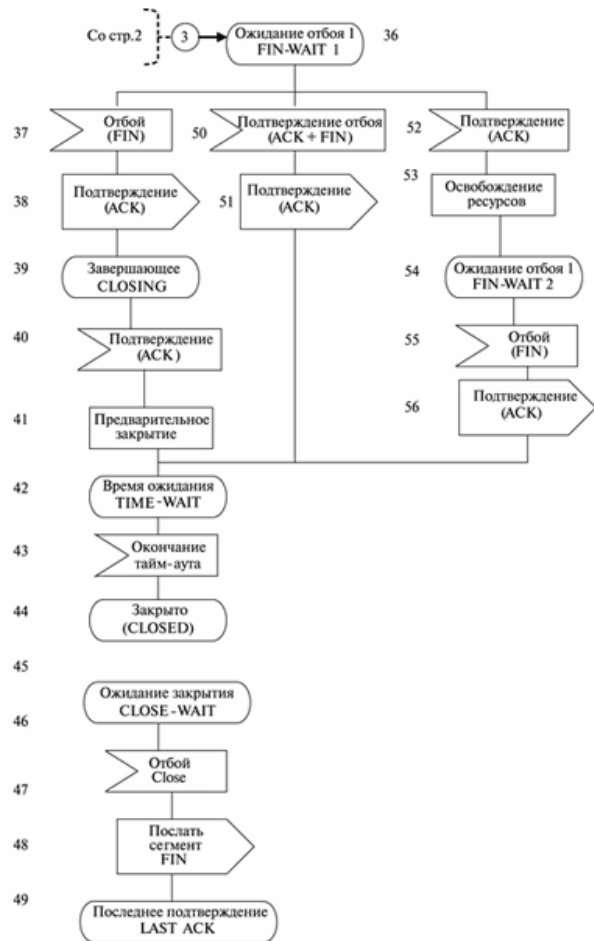


5.4. Алгоритм работы главного модуля ТСР.



Алгоритм нарисован в виде переходов конечного автомата от состояния к состоянию. Такой подход упрощает сетевое программирование сокетов.





7. Протокол SCTP.

SCTP (Stream Control Transmission Protocol — «протокол передачи с управлением потоком») — протокол транспортного уровня, появившийся в 2000г., описывается в RFC 3268 и RFC 4960.

Как и любой другой протокол передачи данных транспортного уровня, SCTP работает с портами аналогично TCP, но, имеет несколько нововведений, таких как:

- защита от DoS атак;
- multi-streaming - многопоточность;
- multi-homing - синхронное соединение между двумя хостами по двум и более независимым физическим каналам
- и др.

7.1. Безопасное установление соединения.

SCTP защищен от DoS-атак с помощью механизма четырёхэтапного квитирования (four-way handshake) и маркера (cookie). Клиент начинает процедуру установления соединения, посылая пакет INIT. В ответ сервер посылает пакет INIT-ACK, который содержит COOKIE (уникальный ключ, идентифицирующий новое соединение). Затем клиент отвечает посылкой пакета COOKIE-ECHO, в котором содержится маркер, полученный от сервера. Только после этого сервер выделяет свои ресурсы новому подключению и подтверждает это отправкой клиенту пакета COOKIE-ACK.

Для решения проблемы задержки пересылки данных при выполнении процедуры четырёхэтапного квитирования в протоколе SCTP допускается включение данных в пакеты COOKIE-ECHO и COOKIE-ACK.



7.2. Многопоточность.

TCP управляет последовательностью байт: данные, посланные приложением-отправителем, должны поступать приложению-получателю строго в том же порядке (в то время как протокол IP способен поменять последовательность пакетов; кроме того, пропавшие пакеты посылаются повторно и обычно прибывают к получателю с нарушением последовательности; для борьбы с этими явлениями данные накапливаются в буфере).

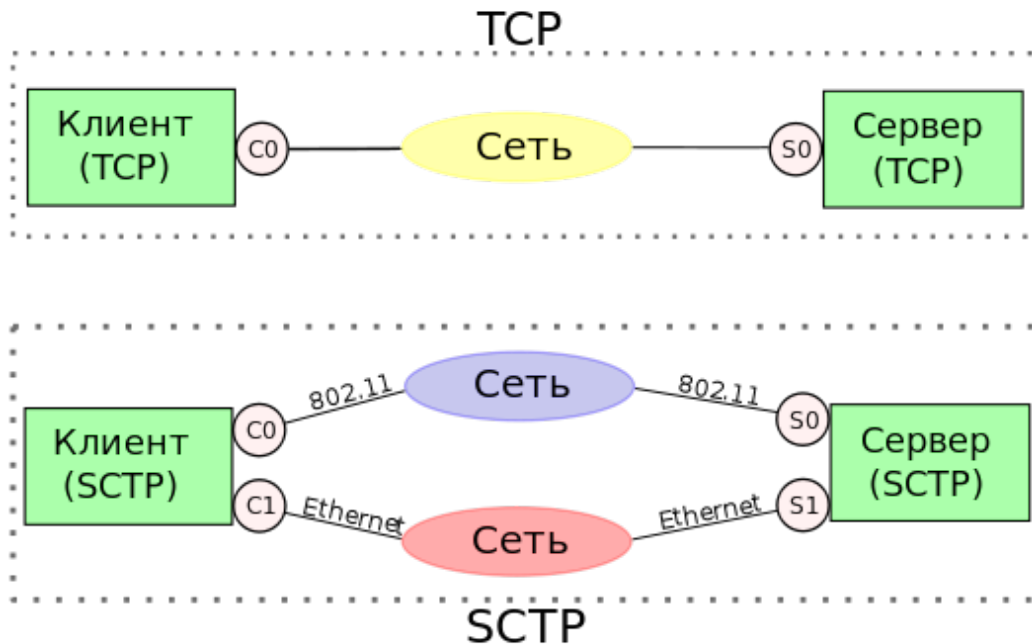
SCTP может транспортировать данные между двумя точками (узлами) одновременно по нескольким потокам сообщений. В противоположность к TCP, SCTP обрабатывает целые сообщения, а не обычные байты информации. Это означает, что если отправитель отправляет серверу сообщение, состоящее из 100 байт за первый шаг, а за ним ещё 50 байт, то получатель за первый шаг получит именно первые 100 байт в первом сообщении, а только затем 50 байт на второй операции чтения из сокета.



Термин «многопоточность» обозначает способность SCTP параллельно передавать по нескольким независимым потокам сообщений. Например, мы передаем несколько фотографий через HTTP-приложение (например, браузер). Можно использовать для этого связку из нескольких TCP-соединений на уровне приложения, но, удобнее использовать SCTP-ассоциация (SCTP-association), управляющая несколькими потоками сообщений для этой цели на уровне транспорта.

7.3. Множественные интерфейсы (Multihoming).

Допустим, у нас есть два хоста. И хотя бы один из них имеет несколько сетевых интерфейсов, и соответственно несколько IP-адресов. В TCP, понятие «соединение» означает обмен данными между двумя точками, в то время, как в SCTP имеет место концепция «ассоциации» (англ. association), обозначающая всё происходящее между двумя хостами.



7.4. Реализации.

Протокол SCTP реализован в следующих операционных системах:

- Linux 2.4 и выше
- CISCO IOS 12+
- DragonFly BSD начиная с версии 1.4, поддержка прекращена в версии 4.2
- QNX Neutrino,
- BSD UNIX (с внешним дополнением от проекта KAME)
- FreeBSD начиная с версии 7
- HP-UX
- AIX 5

8. DCCP.

DCCP (Datagram Congestion Control Protocol) — протокол транспортного уровня OSI, разрабатываемый IETF. Принят в качестве стандарта в марте 2006 г. Он предоставляет механизмы для отслеживания перегрузок в сети, избегая необходимости создавать их на прикладном уровне. Этот протокол не гарантирует доставку информации в нужном порядке.

DCCP очень эффективен для приложений, в которых данные, пришедшие не вовремя, становятся бесполезными. Например, потоковое медиа-вещание, онлайн-игры и интернет-телефония. Главная особенность этих приложений состоит в том, что старые сообщения очень быстро становятся бесполезными, поэтому лучше получить новое сообщение, чем пытаться переслать старое. Но на данный момент большинство таких приложений самостоятельно реализовывают отслеживание перегрузок, а в качестве протокола передачи используются TCP или UDP.

Протокол DCCP доступен в ядре Linux с версии 2.6.14 и улучшается с каждым выпуском.