Тема 11

Технологии транспортного уровня

Содержание темы

- Понятие порта и сокета.
- Протоколы UDP.
- Протокол TCP.
- Установление и разъединение логических соединений.
- Методы квитирования.
- Концепция скользящего окна при передаче данных.

Транспортный уровень

Транспортный уровень стека TCP/IP может предоставлять вышележащему уровню два типа сервиса:

- **гарантированную доставку** обеспечивает **протокол управления передачей (Transmission Control Protocol, TCP);**

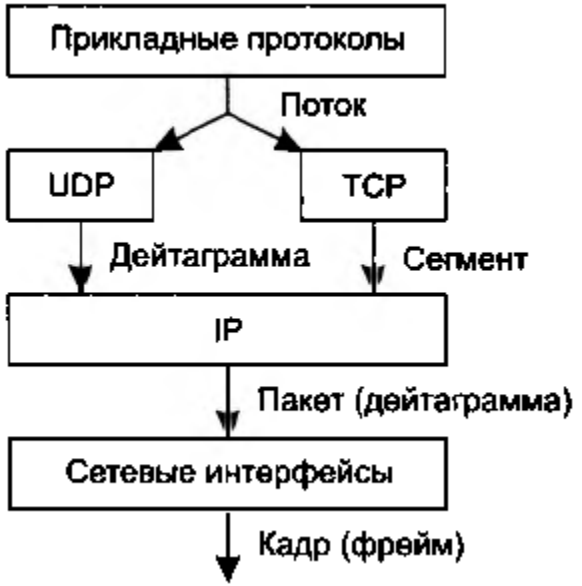
- **доставку по возможности**, или с максимальными усилиями, обеспечивает **протокол пользовательских дейтаграмм (User Datagram Protocol, UDP).**

Транспортный уровень

Потоком данных, информационным потоком, или просто **потоком,** называют данные, поступающие от приложений на вход протоколов транспортного уровня – **TCP** и **UDP**.

Протокол TCP «нарезает» из потока данных **сегменты.**

Единицу данных протокола UDP часто называют **дейтаграммой,** или **датаграммой.**



Дейтаграмма – это общее название для единиц данных, которыми оперируют протоколы без установления соединений.

Порты

Каждый компьютер может одновременно выполнять **несколько процессов**, даже отдельный прикладной процесс может иметь **несколько точек входа**, выступающих в качестве адресов назначения для пакетов данных.

Реализуемая протоколами **TCP** и **UDP** процедура распределения между прикладными процессами пакетов, поступающих от сетевого уровня, называется **демультиплексированием**.

Обратная задача – **мультиплексирование**.

Порты

Протоколы **TCP** и **UDP** ведут для **каждого** приложения две системные очереди:

- очередь данных, **поступающих к приложению из сети**;
- очередь данных, **отправляемых приложением в сеть**.

Эти системные очереди называются **портами** (входная и выходная очереди одного приложения рассматриваются как один порт).

Для идентификации портов им присваивают **номера**.

Порты

Если процессы представляют собой популярные системные службы (**FTP**, **telnet**, **HTTP**, **TFTP**, **DNS** и т. п.), то за ними **централизовано** закрепляются **стандартные назначенные номера**, называемые также **хорошо известными** (**well-known**) номерами портов в диапазоне от **0** до **1023**.

Порт	Name	Комментарий
1	tcpmux	TCP-порт службы мультиплексора
5	rje	Ввод удалённого задания
7	echo	Служба Echo
9	discard	Служба-пустышка для проверки соединения
11	systat	Служба системного состояния, показывающая подключенные порты
13	daytime	Возвращает запрашивающему узлу дату и время
17	qotd	Выдаёт подключенному узлу фразу дня
18	msp	Протокол отправки сообщений
19	chargen	Служба, генерирующая символы; посылает бесконечный поток символов
20	ftp-data	Порт данных FTP
21	ftp	Порт протокола передачи файлов (File Transfer Protocol, FTP); иногда используется протоколом файловой службы (File Service Protocol, FSP)
22	ssh	Служба безопасной оболочки (Secure SHell, SSH)
23	telnet	Служба Telnet

Порты

Для менее распространенных приложений номера портов назначаются **локально** разработчиками этих приложений или **динамически** операционной системой (она ведет список занятых и свободных номеров портов) из диапазона от **1024** до **65 535** в ответ на поступление запроса от приложения.

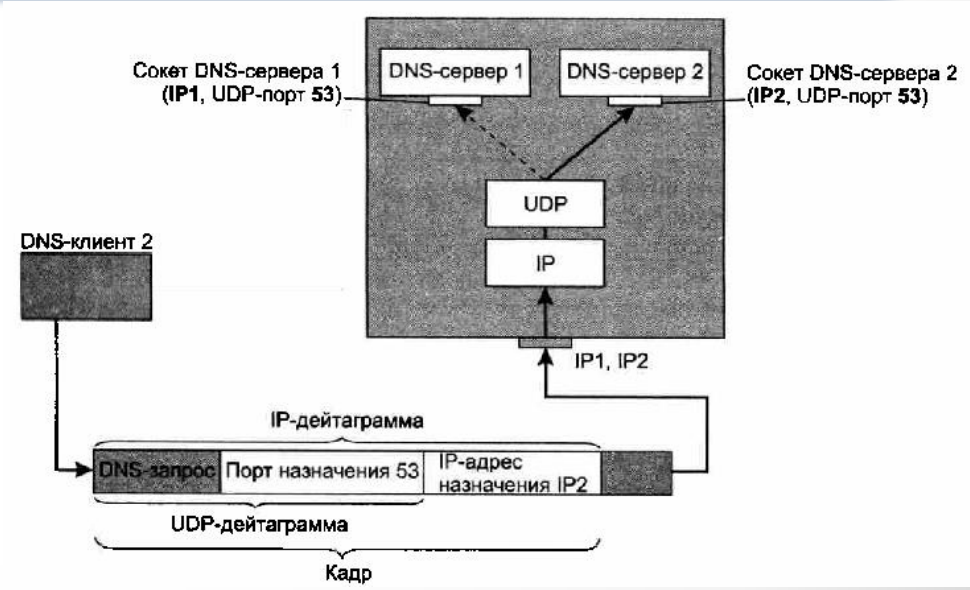
Приложения, которые передают данные на уровень IP по протоколу **UDP**, получают номера, называемые **UDP-портами.** Аналогично, приложениям, обращающимся к протоколу **TCP**, выделяются **TCP-порты.**

Сокеты

Одного номера порта не достаточно, чтобы однозначно определить прикладной процесс в пределах хоста.

Прикладной процесс однозначно определяется в пределах сети и в пределах отдельного компьютера парой (IP-адрес, номер порта), называемой **сокетом** (**socket**).

Сокет, определенный IP-адресом и номером **UDP**-порта, называется **UDP-сокетом,** а IP-адресом и номером **TCP**- порта – **TCP-сокетом.**



Сокеты

Порядок использования сокетов для получения приложениями данных:

1) получение IP- пакета от протокола канального уровня;

2) анализ протоколом IP содержимого заголовка пакета и его отбрасывание;

3) Запоминание IP-адреса назначения из заголовка IP-пакета;

4) передача UDP-дейтаграммы или TCP-сегмента в порт приложения, с использованием IP-адреса для однозначной его идентификации.

Протокол UDP

Протокол пользовательских дейтаграмм (**User Datagram Protocol**, **UDP)** – это протокол транспортного уровня, который обеспечивает доставку по возможности, или с максимальными усилиями**.**

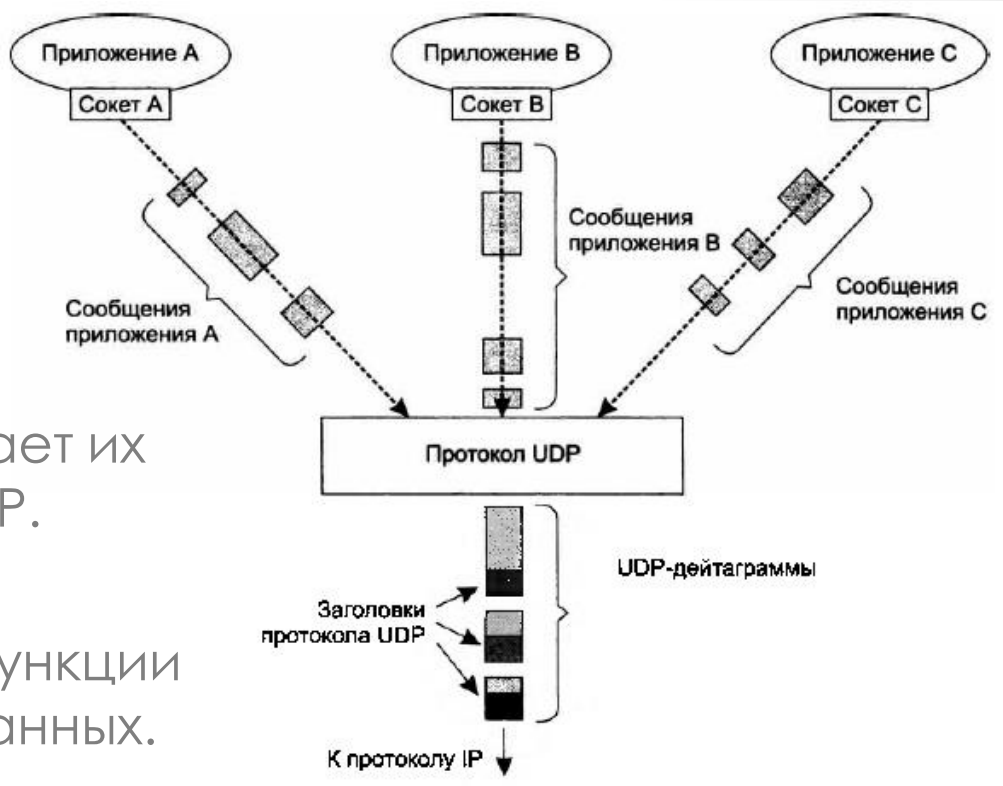
Подобно IP UDP является дейтаграммным протоколом, который **не гарантирует** доставку сообщений адресату.

Протокол UDP

При работе **на хосте-отправителе** данные от приложений поступают протоколу **UDP** через порт в виде сообщений.

Протокол UDP добавляет к каждому отдельному сообщению свой **8-байтный заголовок**, формируя из этих сообщений **UDP-дейтаграммы**, и передает их нижележащему протоколу IP.

В этом и заключаются его функции по **мультиплексированию** данных.



Процесс передачи UDP-дейтаграмм

Каждая дейтаграмма переносит **отдельное пользовательское сообщение**.

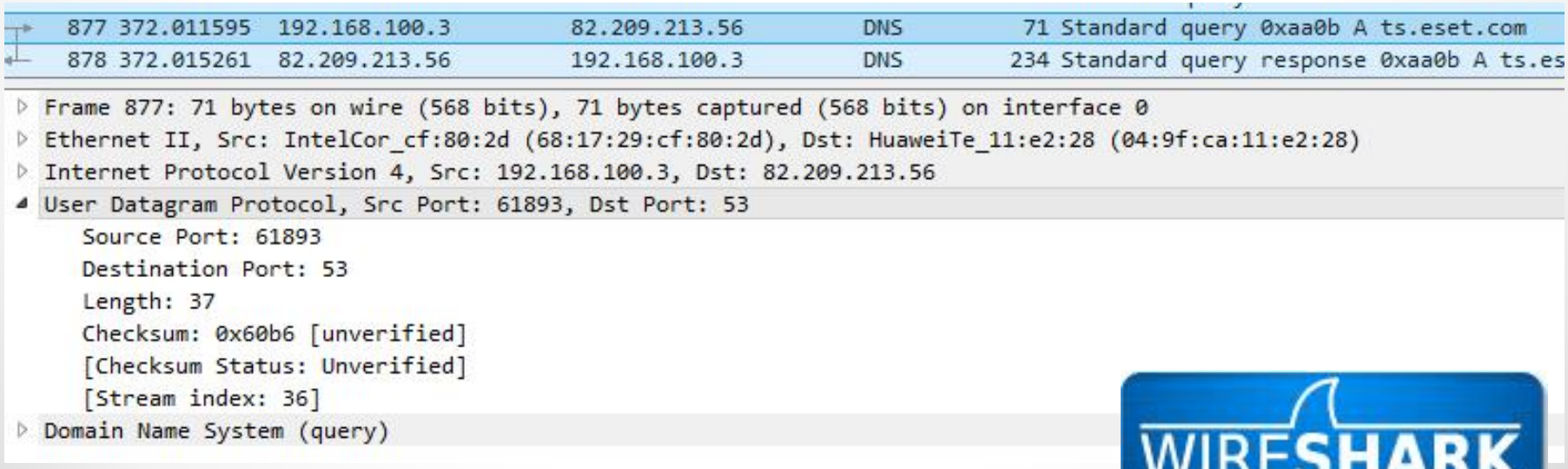
Сообщения могут иметь разную длину, не превышающую длину поля данных протокола IP, которое, в свою очередь, ограничено размером кадра технологии нижнего уровня.

Если буфер UDP переполняется, то сообщение приложения **отбрасывается**.

Протокол UDP

Заголовок UDP состоит из четырех 2-байтных полей:

- **номер UDP-порта отправителя**;
- **номер UDP-порта получателя**;
- **длина дейтаграммы**;
- **контрольная сумма.**



Процесс передачи UDP-дейтаграмм

Функции **UDP** сводятся к простой передаче данных между прикладным и сетевым уровнями, а также к примитивному контролю искажений в передаваемых данных.

При контроле искажений протокол UDP только **диагностирует**, **но не исправляет ошибку**.

Если контрольная сумма показывает, что в поле данных UDP-дейтаграммы произошла ошибка, протокол UDP **отбрасывает поврежденную дейтаграмму**.

Протокол UDP

Работая **на хосте-получателе**, протокол UDP принимает от протокола IP извлеченные из пакетов UDP-дейтаграммы.

Полученные из IP-заголовка **IP-адрес назначения** и из UDP-заголовка **номер порта** используются для формирования UDP-сокета, однозначно идентифицирующего приложение, которому направлены данные.

Протокол UDP освобождает дейтаграмму от UDP-заголовка и полученное в результате сообщение передает приложению на соответствующий UDP-сокет.

Так протокол UDP выполняет **демультиплексирование** на основе сокетов.

Протокол UDP

Ряд приложений, использующих протокол UDP на транспортном уровне:

Приложение	Порт
DNS	53
BOOTP	клиент 68, сервер 67
DHCPv4	клиент 68, сервер 67
DHCPv6	клиент 546, сервер 547
TFTP	69
HTTPS	443
NTP	123
Syslog	514
SNMP	162

Протокол TCP

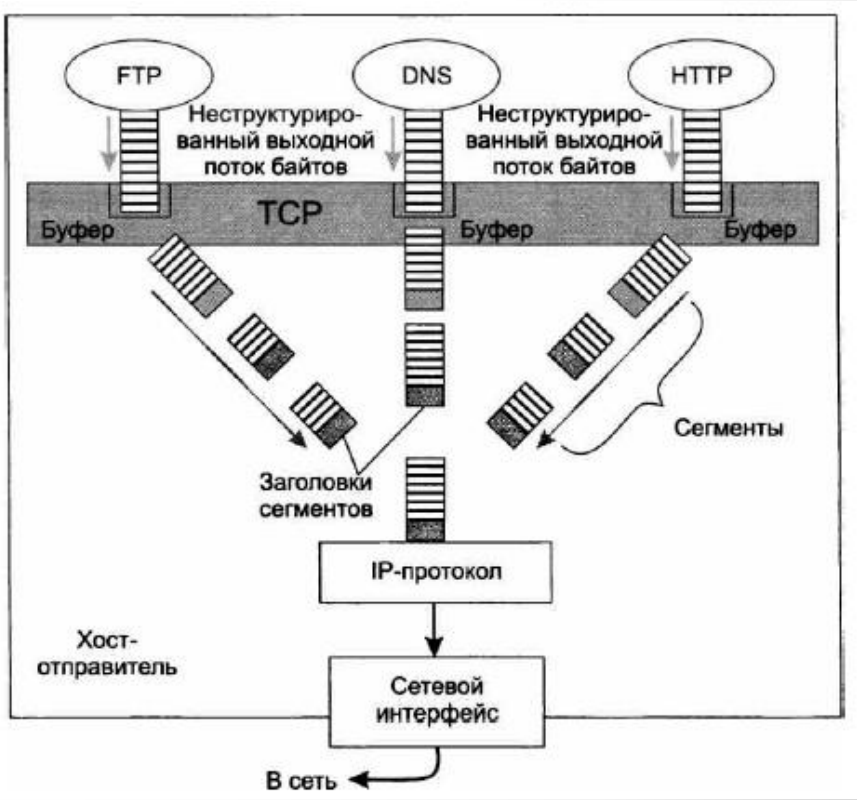
Протокол управления передачей (**Transmission Control Protocol**, **TCP)** предназначен для передачи данных между приложениями и основан на **логическом соединении**, что позволяет ему обеспечивать **гарантированную доставку** данных, используя в качестве инструмента ненадежный дейтаграммный сервис протокола IP.

Протокол TCP

При работе **на хосте-отправителе** протокол TCP рассматривает информацию, поступающую к нему от прикладных процессов, как **неструктурированный поток байтов**.

Поступающие данные буферизуются средствами TCP.

Для передачи на сетевой уровень из буфера «вырезается» некоторая непрерывная часть данных, которая называется **сегментом** и снабжается заголовком.



Протокол TCP

В отличие от протокола UDP, который создает свои дейтаграммы на основе **логически обособленных единиц данных – сообщений**, генерируемых приложениями, протокол TCP делит поток данных на сегменты **без учета их смысла или внутренней структуры**.

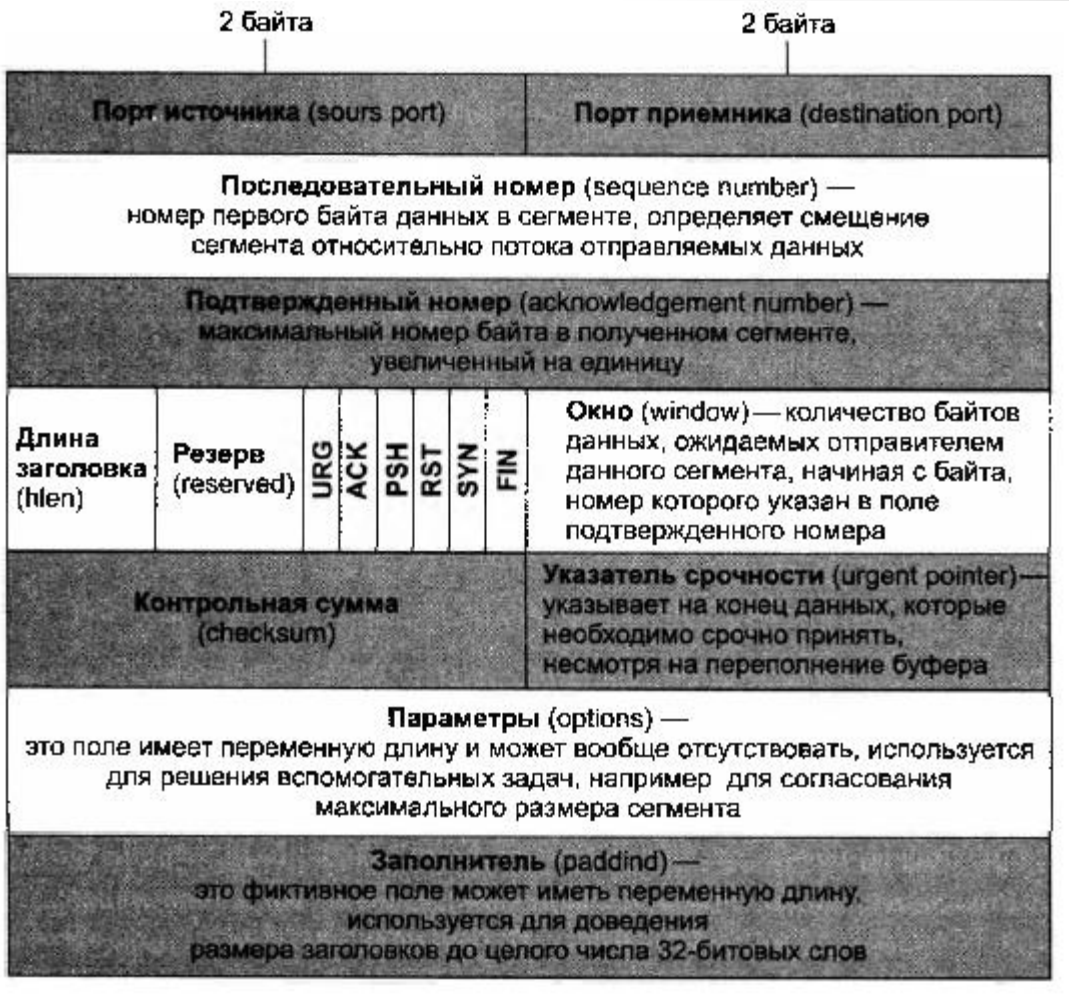
Протокол TCP

Ряд приложений, использующих протокол TCP на транспортном уровне:

Приложение	Порт
DNS	53
SMTP	25
POP	110
IMAP	143
FTP	20, 21
HTTP	80, 8080
HTTPS	443
SSH	22
Telnet	23

TCP-сегмент

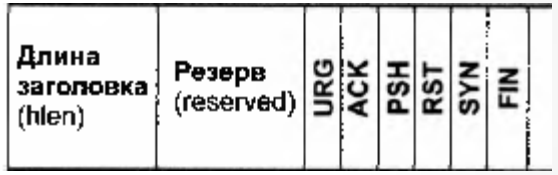
Заголовок TCP-сегмента содержит значительно больше полей, чем заголовок UDP, что отражает **более развитые возможности протокола TCP**.



TCP-сегмент

Однобитные поля, называемых **флагами,** или **кодовыми битами** (**code bits**) расположены сразу за резервным полем и содержат служебную информацию о типе сегмента.

Положительное значение сигнализируется установкой этих битов в единицу:



URG – срочное сообщение;

ACK – квитанция на принятый сегмент;

PSH – запрос на отправку сообщения без ожидания заполнения буфера;

RST – запрос на восстановление соединения;

SYN – сообщение, используемое для синхронизации счетчиков переданных данных при установлении соединения;

FIN – признак достижения передающей стороной последнего байта в потоке передаваемых данных.

TCP-сегмент

```
870 371.976240  91.190.216.58    192.168.100.3   TCP      54 12350 → 52094 [ACK] Seq=68 Ack=226 Win=8192 Len=0
871 371.981638  157.55.130.158   192.168.100.3   TCP     134 40031 → 52095 [PSH, ACK] Seq=1 Ack=76 Win=14848 Len=80
872 371.981971  91.190.216.58    192.168.100.3   TCP     357 12350 → 52094 [PSH, ACK] Seq=68 Ack=226 Win=8192 Len=303

▷ Frame 871: 134 bytes on wire (1072 bits), 134 bytes captured (1072 bits) on interface 0
▷ Ethernet II, Src: HuaweiTe_11:e2:28 (04:9f:ca:11:e2:28), Dst: IntelCor_cf:80:2d (68:17:29:cf:80:2d)
▷ Internet Protocol Version 4, Src: 157.55.130.158, Dst: 192.168.100.3
◢ Transmission Control Protocol, Src Port: 40031, Dst Port: 52095, Seq: 1, Ack: 76, Len: 80
    Source Port: 40031
    Destination Port: 52095
    [Stream index: 4]
    [TCP Segment Len: 80]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 81    (relative sequence number)]
    Acknowledgment number: 76    (relative ack number)
    Header Length: 20 bytes
  ◢ Flags: 0x018 (PSH, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 1... = Push: Set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·····AP···]
    Window size value: 29
    [Calculated window size: 14848]
    [Window size scaling factor: 512]
    Checksum: 0xa76c [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▷ [SEQ/ACK analysis]
▷ Data (80 bytes)
```

TCP-сегмент

```
911 372.165871  91.228.167.146        192.168.100.3         TCP         54 80 → 52097 [FIN, ACK] Seq=380 Ack=2482 Win=31064 Len=0

▷ Frame 911: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
▷ Ethernet II, Src: HuaweiTe_11:e2:28 (04:9f:ca:11:e2:28), Dst: IntelCor_cf:80:2d (68:17:29:cf:80:2d)
▷ Internet Protocol Version 4, Src: 91.228.167.146, Dst: 192.168.100.3
◢ Transmission Control Protocol, Src Port: 80, Dst Port: 52097, Seq: 380, Ack: 2482, Len: 0
    Source Port: 80
    Destination Port: 52097
    [Stream index: 6]
    [TCP Segment Len: 0]
    Sequence number: 380    (relative sequence number)
    Acknowledgment number: 2482    (relative ack number)
    Header Length: 20 bytes
  ◢ Flags: 0x011 (FIN, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
    ▷ .... .... ...1 = Fin: Set
      [TCP Flags: ·····A···F]
    Window size value: 31064
    [Calculated window size: 31064]
    [Window size scaling factor: -2 (no window scaling used)]
    Checksum: 0x916d [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▷ [SEQ/ACK analysis]
```

Логические соединения

Основным отличием TCP от **UDP** является то, что на протокол TCP возложена дополнительная задача – **обеспечить надежную доставку сообщений**, используя в качестве основы **ненадежный дейтаграммный протокол IP**.

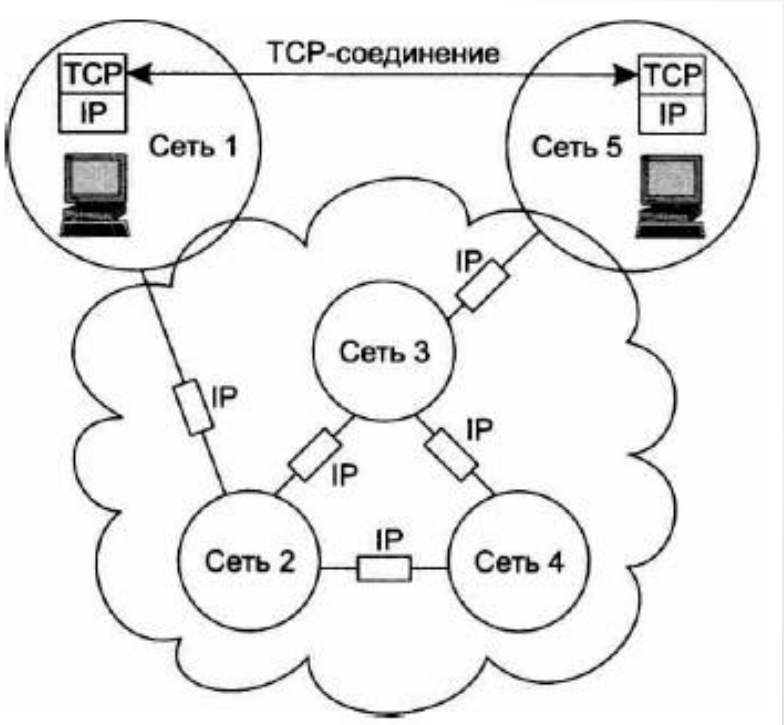
Для решения этой задачи протокол TCP использует метод продвижения данных с установлением **логического соединения**.

Логическое соединение дает возможность участникам обмена следить за тем, чтобы данные **не были потеряны**, **искажены** или **продублированы**, а также чтобы они **пришли к получателю в том порядке**, **в котором были отправлены**.

Логические соединения

Протокол TCP устанавливает логические соединения между **прикладными процессами**, причем в каждом соединении участвуют только **два** процесса.

TCP-соединение является **дуплексным**, то есть каждый из участников этого соединения может одновременно получать и отправлять данные.



Логическое TCP-соединение **однозначно** идентифицируется парой сокетов, определенных для этого соединения двумя взаимодействующими процессами.

Логические соединения

При **установлении логического соединения** модули TCP договариваются между собой о **параметрах процедуры обмена данными**.

В протоколе TCP каждая сторона соединения посылает противоположной стороне следующие параметры:

- максимальный размер сегмента, который она готова принимать;
- максимальный объем данных (количество сегментов), которые она разрешает другой стороне передавать в свою сторону, даже если та еще не получила квитанцию на предыдущую порцию данных (размер окна);
- начальный порядковый номер байта, с которого она начинает отсчет потока данных в рамках данного соединения.

Логические соединения

Соединение устанавливается по инициативе клиентской части приложения.

Клиент – это модуль, предназначенный для формирования и передачи сообщений-запросов к ресурсам удаленного компьютера от разных приложений с последующим приемом результатов из сети и передачей их соответствующим приложениям.

1) При необходимости выполнить обмен данными с серверной частью приложение-клиент обращается к нижележащему протоколу TCP, который в ответ на это обращение посылает **сегмент-запрос** (в запросе содержится флаг **SYN**, **установленный в 1**) на установление соединения протоколу TCP, работающему на стороне сервера.

Логические соединения

2) Получив запрос, модуль TCP на стороне сервера пытается создать «инфраструктуру» для обслуживания нового клиента.

Сервер – это модуль, который постоянно ожидает прихода из сети запросов от клиентов и, приняв запрос, пытается его обслужить.

Он обращается к операционной системе с просьбой о выделении определенных системных ресурсов для организации буферов, таймеров, счетчиков (они закрепляются за соединением с момента создания и до момента разрыва).

Если на стороне сервера все необходимые ресурсы были получены и все необходимые действия выполнены, то модуль TCP посылает клиенту **сегмент** с флагами **ACK** и **SYN**.

Логические соединения

3) В ответ клиент посылает **сегмент** с флагом **ACK** и переходит в состояние установленного логического соединения (состояние ESTABLISHED).

Когда сервер получает флаг **ACK**, он также переходит в состояние ESTABLISHED.

На этом процедура установления соединения заканчивается, и стороны могут переходить к обмену данными.

Логические соединения

Соединение может быть **разорвано** в **любой момент** по инициативе **любой стороны**.

Для этого **клиент** и **сервер** должны обменяться сегментами **FIN** и **ACK**.

Соединение считается закрытым по прошествии некоторого времени, в течение которого сторона-инициатор убеждается, что ее завершающий сигнал **ACK** дошел нормально и не вызвал никаких «аварийных» сообщений со стороны сервера.

Методы квитирования

Передача с квитированием – это один из наиболее естественных приемов, используемых для организации надежного обмена данными.

Отправитель **отсылает данные**, а получатель **подтверждает** их получение **квитанциями**.

Если отправитель вовремя не получает квитанции на переданные данные, то он передает их **повторно**, выполняя **запрос повторной передачи** (**Automatic Repeat reQuest**, **ARQ**).

Методы квитирования

Все многообразие **методов квитирования** может быть разделено на два класса:

- методы **простоя источника** (**Stop-and-Wait**);
- методы **скользящего окна**:
  - методы, использующие **окно передачи** – метод **передачи с возвращением на N пакетов** (**Go-Back-N**);
  - методы, использующие **окно передачи** и **окно приема** – метод **передачи с выборочным повторением** (**Selective Repeat**).

Методы квитирования

Общие черты, присущие всем методам квитирования:

- отправитель и получатель, работающие **асинхронно**, осуществляют передачу пакетов по **ненадежной** линии связи, в которой возможны **искажения**, большие **задержки** и **потери пакетов**;
- отправитель принимает данные от **протокола верхнего уровня** (приложения), получатель передает полученные данные на **верхний уровень** (приложению);
- получатель располагает механизмом определения искаженных пакетов (по контрольной сумме);
- после успешного получения пакета получатель посылает отправителю **квитанции** (**acknowledgment**, **ACK**);
- для отслеживания задержек пакетов используется **таймер**, тайм-аут которого устанавливается равным предельному времени ожидания квитанции.

Метод передачи данных с простоем источника

В методе **простоя источника** отправитель передает последовательность **ненумерованных** пакетов.

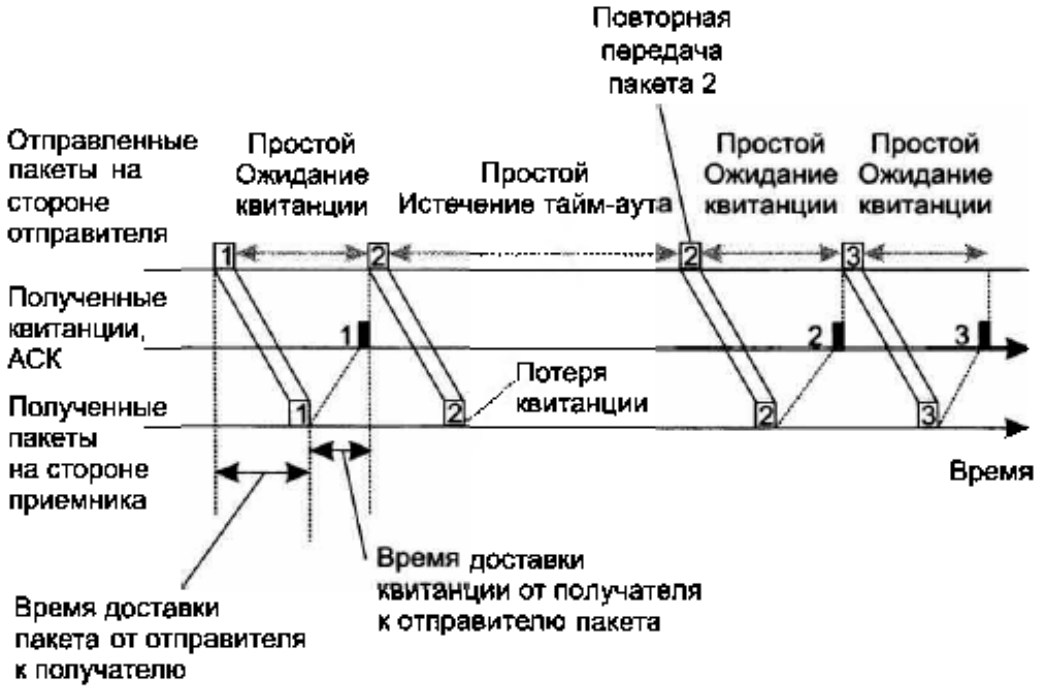
Метод требует, чтобы отправитель дожидался от получателя квитанции и только **после этого** посылал следующий пакет.

С переданным пакетом отправитель связывает **таймер**.

Если в течение тайм-аута квитанция не пришла, то пакет (или квитанция на него) считается утерянным или искаженным и его передача повторяется.

Метод передачи данных с простоем источника

Принимающая сторона должна уметь распознавать дублированные пакеты и отбрасывать их.



Отправитель также должен иметь возможность распознавать дубликаты квитанций.

Метод передачи данных с простоем источника

В заголовок пакета включается **специальный бит** и устанавливается отправителем так, что нуль и единица в качестве его значения чередуются в пределах всей последовательности отправляемых пакетов.

Приемник проверяет значение бита: если у двух последовательно пришедших пакетов значения данного бита равны **двум единицам** или **двум нулям** то эти пакеты считаются **дубликатами**.

Аналогично поступает отправитель с квитанциями.

Метод передачи данных с простоем источника

Такой способ распознавания дубликатов **не является абсолютно надежным**, поскольку он основывается на предположении, что **вероятность прихода в приемник дубликатов друг за другом достаточно высока**.

В данном методе **коэффициент использования линии связи очень низкий** – основную часть времени передатчик простаивает в ожидании прихода квитанции.

Концепция скользящего окна

Концепция **скользящего окна** (**sliding window**) заключается в том, что для повышения скорости передачи данных отправителю разрешается передать некоторое количество пакетов, **не дожидаясь прихода на эти пакеты квитанций**.

Для **идентификации** пакетам присваиваются **уникальные последовательные номера**, которые размещаются в заголовках пакетов.

Разрядность поля «**номер пакета**» определяет диапазон возможных номеров и когда этот диапазон исчерпывается, нумерация пакетов снова начинается с нуля, поэтому **нельзя абсолютно исключить ситуацию**, **когда в сети существуют пакеты с одинаковыми номерами**.

Концепция скользящего окна

Окно определяется на **последовательности пронумерованных пакетов**.

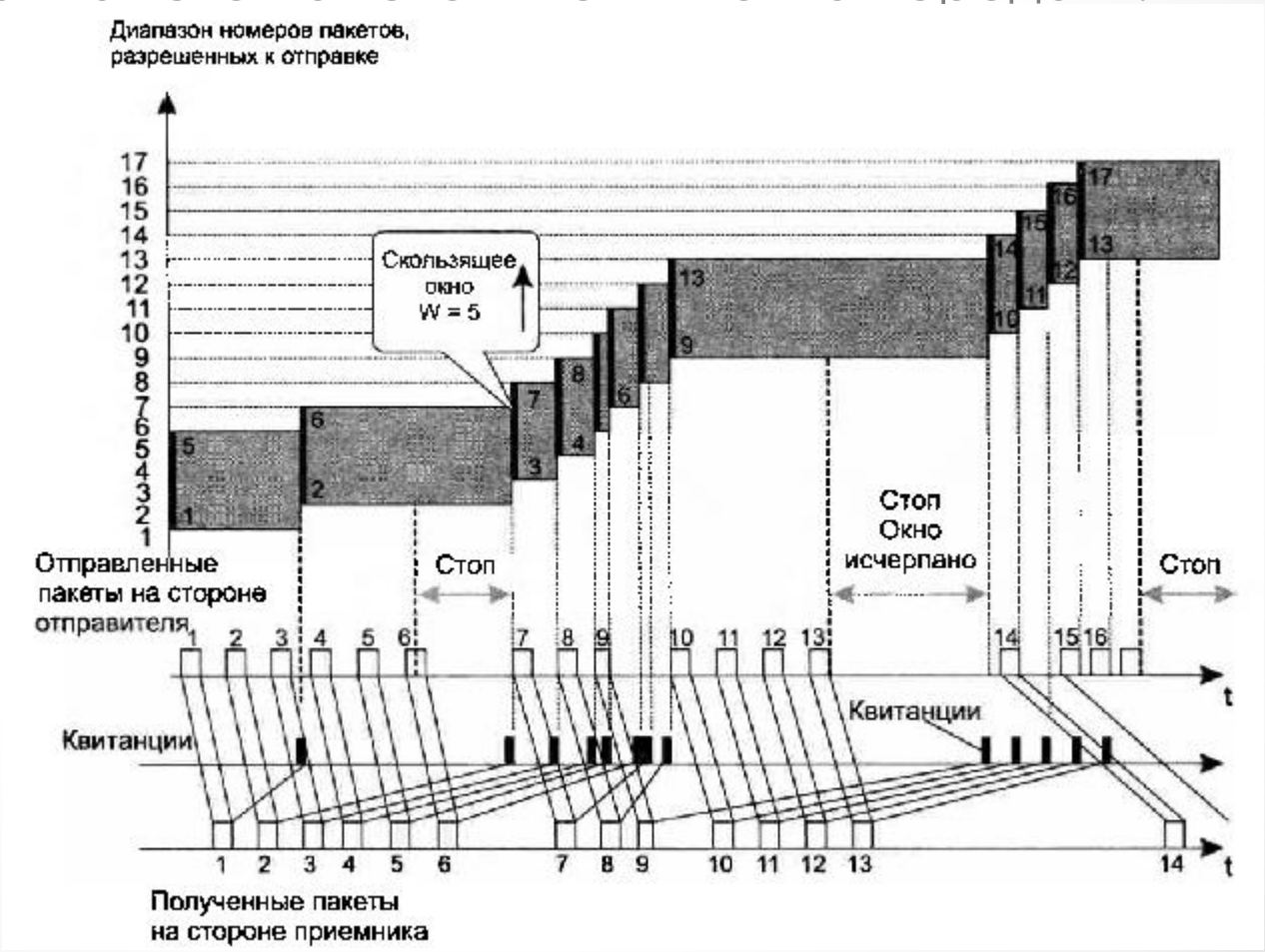
Окно всегда имеет **нижнюю границу**, называемую также **базой окна**, и **верхнюю границу**.

Количество номеров, попадающих в пределы окна, называют **размером окна**.

Окно **всегда перемещается в сторону больших номеров**.

Концепция скользящего окна

Идеализированная схема «скольжения» окна передачи.



Реализация метода скользящего окна в TCP

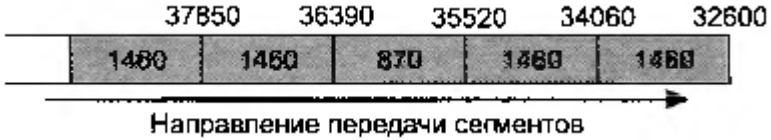
Алгоритм скользящего окна в протоколе TCP имеет существенную особенность – окно определено на множестве **нумерованных байтов** неструктурированного потока данных, передаваемого приложением протоколу TCP.

В ходе переговорного процесса модули TCP обеих участвующих в обмене сторон договариваются между собой о **начальном номере байта** (у каждой стороны он свой), с которого будет вестись отсчет в течение всего функционирования данного соединения.

Нумерация байтов в пределах сегмента осуществляется начиная от заголовка.

Реализация метода скользящего окна в TCP

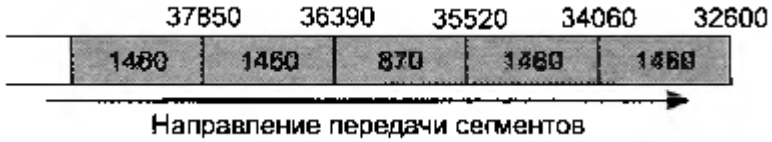
Когда отправитель посылает TCP-сегмент, он помещает в поле **последовательного номера** номер первого байта данного сегмента, который служит **идентификатором** сегмента.



На основании этих номеров получатель TCP-сегмента не только отличает данный сегмент от других, но и позиционирует полученный фрагмент относительно общего потока байтов (он может сделать вывод, например, что полученный сегмент является дубликатом или что между двумя полученными сегментами пропущены данные и т. п.).

Реализация метода скользящего окна в TCP

В качестве **квитанции** получатель сегмента отсылает ответное сообщение (сегмент), в поле **подтвержденного номера** которого он помещает число, **на единицу превышающее максимальный номер байта в полученном сегменте**.



Для первого отправленного сегмента квитанцией о получении (подтвержденным номером) будет число **34060**, для второго – **35520** и т. д.

Подтвержденный номер часто интерпретируют не только как оповещение о благополучной доставке, но и как **номер следующего ожидаемого байта данных**.

Реализация метода скользящего окна в TCP

Квитанция в протоколе TCP посылается только в случае **правильного приема данных** – отсутствие квитанции означает либо потерю сегмента, либо потерю квитанции, либо прием искаженного сегмента.

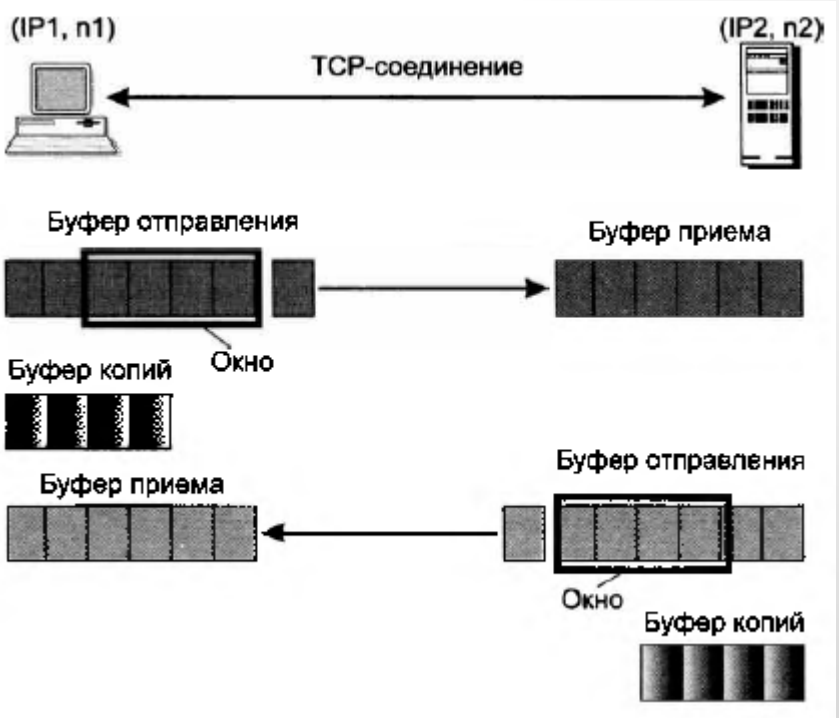
В соответствии с определенным форматом **один и тот же TCP-сегмент** может нести в себе как **пользовательские данные** (в поле данных), так и **квитанцию** (в заголовке), которой подтверждается получение данных от другой стороны.

Реализация метода скользящего окна в TCP

Поскольку протокол TCP является **дуплексным**, каждая сторона одновременно выступает и как **отправитель**, и как **получатель**.

У каждой стороны есть **набор буферов**:

- для **хранения принятых сегментов**;
- для **сегментов, которые только еще предстоит отправить**;
- буфер для **хранения копий сегментов, которые были отправлены, но квитанции о получении которых еще не поступили.**



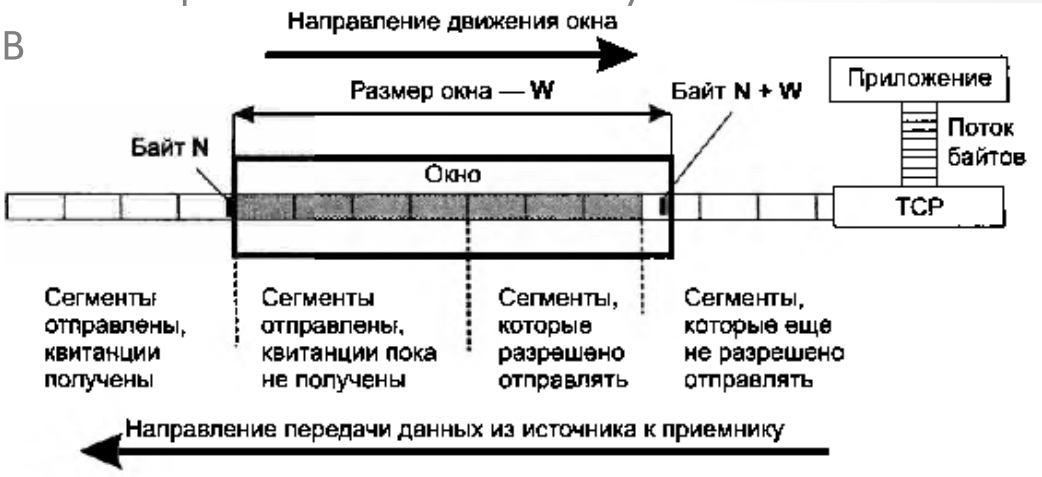
Реализация метода скользящего окна в TCP

1) сегменты, которые уже были отправлены и на которые уже пришли квитанции (последняя квитанция пришла на байт с номером *N*);

2) часть байтов, входящих в окно размером *W* байт, составляют сегменты, которые также уже отправлены, но квитанции на которые пока не получены;

3) сегменты, которые пока не отправлены, но могут быть отправлены, так как входят в пределы окна;

4) сегменты, ни один из которых не может быть отправлен до тех пор, пока не придет очередная квитанция и окно не будет сдвинуто вправо.



Реализация метода скользящего окна в TCP

Накопительный принцип квитирования.

Возможны ситуации, когда сегменты приходят к получателю не в том порядке, в каком были посланы, то есть в приемном буфере может образоваться «**прогалина**».

Получатель может только **еще раз повторить квитанцию** на максимальный байт плюс один в последнем принятом сегменте из непрерывной цепочки сегментов, говоря тем самым, что **все еще ожидает поступления пропущенного блока байтов**.