# TCP/IP

| | |
|---|---|
| 7 | User/Application Level |
| 6 | Presentation Level |
| 5 | Seance/Session Level |
| 4 | Transport Level |
| 3 | Network Level |
| 2 | Data Link/Logical Channel Level |
| 1 | Media Access Control/Physical L. |

| | |
|---|---|
| 7 | User/Application Level |
| 6 | Presentation Level |
| 5 | Seance/Session Level |
| 4 | Transport Level |
| 3 | Network Level |
| 2 | Data Link/Logical Channel Level |
| 1 | Media Access Control/Physical L. |

| | |
|---|---|
| 4 | Application |
| 3 | TCP/UDP |
| 2 | IP |
| 1 | Physical |

| | |
|---|---|
| 7 | User/Application Level |
| 6 | Presentation Level |
| 5 | Seance/Session Level |
| 4 | Transport Level |
| 3 | Network Level |
| 2 | Data Link/Logical Channel Level |
| 1 | Media Access Control/Physical L. |

| | | |
|---|---|---|
| 7 | User/Application Level | Application |
| 6 | Presentation Level | Socket API |
| 5 | Seance/Session Level | TCP UDP |
| 4 | Transport Level | |
| 3 | Network Level | IP |
| 2 | Data Link/Logical Channel Level | Physical |
| 1 | Media Access Control/Physical L. | |

| | |
|---|---|
| 4 | Application |
| 3 | TCP/UDP |
| 2 | IP |
| 1 | Physical |

## Пространство имён IPv.4

- RFC 760, 791
- 32-разрядные двоичные числа
- 4 294 967 296 ($2^{32}$) уникальных адресов

## Форма записи

- 192.0.2.118 – тетрадная
- .118.2.0.192 – обратная тетрадная (реверсивная)
- 0xC0:0x00:0x02:0x76 или $C0000276_{16}$
- 00300:0000:0002:0114 или $03000000000020114_{8}$
- 11000000 00000000 00000010 01110110

# Сетевые и локальные адреса

IP адрес состоит из сетевой и локальной части
IP=Network IP + Local IP

| Network IP | Local IP |
|---|---|
| 101…010 (00…0) | (00…0) 110…101 |
| N bits | 32-N bits |

# Classful adressing


| Class | Starting bits | Address Structure | Nums of nets | Nums of hosts | Netmask |
|---|---|---|---|---|---|
| A | 0 | N.L.L.L | 128 | 16 777 216 | 255.0.0.0 |
| B | 10 | N.N.L.L | 16 386 | 65 536 | 255.255.0.0 |
| C | 110 | N.N.N.L | 2 097 154 | 256 | 255.255.255.0 |
| D | 1110 | Multicast addresses | | | |
| E | 1111 | Reserved by RFC 760 | | | |

# Classless addressing
## Начало 1990-х гг

Предпосылки:

- исчерпание пространства IP-адресов
- взрывной рост трафика рассылок маршрутных таблиц в связи с массовым использованием сетей класса C

# Address Structure

Граница сетевой и локальной части адреса могут проходить по любому биту адреса. Номер этого бита называется номером (числом) префикса подсети.

A.B.C.D/N

Двоичное число, имеющие N лидирующих единиц, называется маской подсети (netmask)

$$nm=(2^N-1)2^{32-N}$$

$$\text{Network Address}=nm \;\&\; ip$$

$$\text{Local Address} = \overline{nm} \;\&\; ip$$

# Netmask Samples

| Prefix | Mask tethrade | Nums of hosts |
|---|---|---|
| | | |
| 25 | 128 | 128-2=126 |
| 26 | 128+64=192 | 64-2=62 |
| 27 | 128+64+32=224 | 32-2=30 |
| 28 | 128+64+32+16=240 | 16-2=14 |
| 29 | 128+64+32+16+8=248 | 8-2=6 |
| 30 | 128+64+32+16+8+4=252 | 4-2=2 |
| 31 | 128+64+32+16+8+4+2=254 | 2-2=0 |
| 32 | 128+64+32+16+8+4+2+1=255 | 1-2=-1 |

# Reserved Address Blocks

| Network | Using | RFC |
|---|---|---|
| 0.0.0.0/8 | Current network (only valid as source address) | 6890 |
| 10.0.0.0/8 | Private network | 1918 |
| 127.0.0.0/8 | Loopback | 6890 |
| 169.254.0.0/16 | Link Local | 3927 |
| 172.16.0.0/12 | Private network | 1918 |
| 100.64.0.0/10 | Shared Address Space | 6598 |
| 192.0.0.0/24 | IETF reserved | 6890 |
| 192.0.2.0/24 | Documentation samles | 6890 |
| 192.168.0.0/16 | Private network | 1918 |
| 198.18.0.0/15 | Network benchmark tests | 2544 |
| 198.51.100.0/24 | Documentation samles | 5737 |
| 203.0.113.0/24 | Documentation samles | 5737 |
| 224.0.0.0/4 | IP multicast (former Class D network) | 5771 |
| 240.0.0.0/4 | Reserved (former Class E network) | 1700 |
| 255.255.255.255 | Local network broadcast address | 919 |

# Private networks

| Name | Address Range | Number of addresses | Classful description | Largest CIDR block |
|---|---|---|---|---|
| 24-bit block | 10.0.0.0–10.255.255.255 | 16777216 | Single Class A | 10.0.0.0/8 |
| 20-bit block | 172.16.0.0–172.31.255.255 | 1048576 | Contiguous range of 16 Class B blocks | 172.16.0.0/12 |
| 16-bit block | 192.168.0.0–192.168.255.255 | 65536 | Contiguous range of 256 Class C blocks | 192.168.0.0/16 |

# Маршрутизация

- Прямая (direct) – заранее известны все узлы, через которые пройдёт пакет
- Непрямая (indirect) – каждый узел решает куда переслать пакет на следующем шаге

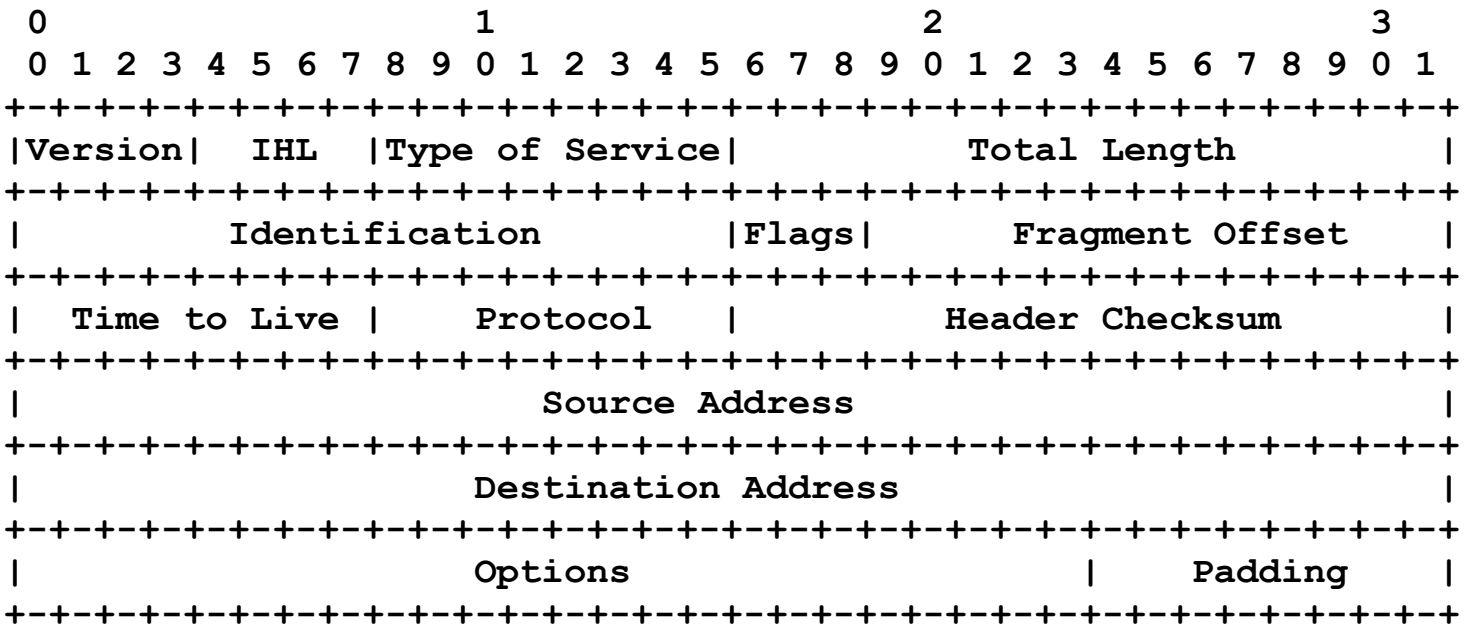
- Прямая маршрутизация характерна для слаборазветвлённых в т.ч. древовидных сетей. Недостатком является неразрешимость маршрутизации в случае выхода из строя узла, указанного в маршрутной таблице даже при наличии альтернативных путей доставки
- Непрямая маршрутизация характерна для структурированных сильноразветвлённых сетей. Недостатком является опасность зацикливания пакетов в случае ошибок маршрутизации

- Destination-based routing:
  next hop = f (last hop, destination ip)
- Source-based routing:
  next hop = f (last hop, destination ip, source ip)

# Routing tables

```
[root@fedora10 ~]# netstat -nr
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
60.49.199.72    0.0.0.0         255.255.255.248 U         0 0          0 eth1
172.16.163.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.162.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.161.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.160.0    0.0.0.0         255.255.255.0   U         0 0          0 eth0
172.16.167.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.166.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.165.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.164.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.170.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.169.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
172.16.168.0    172.16.160.1    255.255.255.0   UG        0 0          0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 eth1
0.0.0.0         60.49.199.73    0.0.0.0         UG        0 0          0 eth1
[root@fedora10 ~]#
```

# IP Datagramm Header



# Type of Service

```
Bits 0-2:  Precedence.
      Bit    3:  0 = Normal Delay,      1 = Low Delay.
      Bits   4:  0 = Normal Throughput, 1 = High Throughput.
      Bits   5:  0 = Normal Relibility, 1 = High Relibility.
      Bit  6-7:  Reserved for Future Use.

         0     1     2     3     4     5     6     7
      +-----+-----+-----+-----+-----+-----+-----+-----+
      |                 |     |     |     |     |     |
      |   PRECEDENCE    |  D  |  T  |  R  |  0  |  0  |
      |                 |     |     |     |     |     |
      +-----+-----+-----+-----+-----+-----+-----+-----+

        Precedence

          111 - Network Control
          110 - Internetwork Control
          101 - CRITIC/ECP
          100 - Flash Override
          011 - Flash
          010 - Immediate
          001 - Priority
          000 - Routine
```

# Fragmentation management flags

```
Bit 0: reserved, must be zero
Bit 1: (DF) 0 = May Fragment,  1 = Don't Fragment.
Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.

    0   1   2
  +---+---+---+
  |   | D | M |
  | 0 | F | F |
  +---+---+---+
```

# Protocols

| Hex | Dec | Protocol |
|---|---|---|
| 00 | 0 | Reserved |
| 01 | 1 | ICMP |
| 02 | 2 | IGMP |
| 03 | 3 | GGP |
| 04 | 4 | IP-in-IP encapsukation |
| 06 | 6 | TCP |
| 08 | 8 | EGP |
| 11 | 17 | UDP |
| 32 | 50 | Encapsulation Security Payload (ESP) Extention Header |
| 33 | 51 | Autentification Header (AH) |

# Options

Options: variable

The options may appear or not in datagrams. They must be implemented by all IP modules (host and gateways). What is optional is their transmission in any particular datagram, not their implementation.

In some environments the security option may be required in all datagrams.

The option field is variable in length. There may be zero or more options. There are two cases for the format of an option:

Case 1: A single octet of option-type.

Case 2: An option-type octet, an option-length octet, and the actual option-data octets.

The option-length octet counts the option-type octet and the option-length octet as well as the option-data octets.

# Options (continuouse)

The option-type octet is viewed as having 3 fields:

```
  1 bit   copied flag,
  2 bits  option class,
  5 bits  option number.
```

The copied flag indicates that this option is copied into all fragments on fragmentation.

```
  0 = not copied
  1 = copied
```

The option classes are:

```
  0 = control
  1 = reserved for future use
  2 = debugging and measurement
  3 = reserved for future use
```

Specifies one of 16 levels of security (eight of which are reserved for future use).

```
  00000000 00000000 - Unclassified
  11110001 00110101 - Confidential
  01111000 10011010 - EFTO
  10111100 01001101 - MMMM
  01011110 00100110 - PROG
  10101111 00010011 - Restricted
  11010111 10001000 - Secret
  01101011 11000101 - Top Secret
  00110101 11100010 - (Reserved for future use)
  10011010 11110001 - (Reserved for future use)
  01001101 01111000 - (Reserved for future use)
  00100100 10111101 - (Reserved for future use)
  00010011 01011110 - (Reserved for future use)
  10001001 10101111 - (Reserved for future use)
  11000100 11010110 - (Reserved for future use)
  11100010 01101011 - (Reserved for future use)
```

# Options (continuouse)

The following internet options are defined:

| CLASS | NUMBER | LENGTH | DESCRIPTION |
|---|---|---|---|
| 0 | 0 | - | End of Option list. This option occupies only 1 octet; it has no length octet. |
| 0 | 1 | - | No Operation. This option occupies only 1 octet; it has no length octet. |
| 0 | 2 | 11 | Security. Used to carry Security, Compartmentation, User Group (TCC), and Handling Restriction Codes compatible with DOD requirements. |
| 0 | 3 | var. | Loose Source Routing. Used to route the internet datagram based on information supplied by the source. |
| 0 | 9 | var. | Strict Source Routing. Used to route the internet datagram based on information supplied by the source. |
| 0 | 7 | var. | Record Route. Used to trace the route an internet datagram takes. |
| 0 | 8 | 4 | Stream ID. Used to carry the stream identifier. |
| 2 | 4 | var. | Internet Timestamp. |

Specific Option Definitions

End of Option List

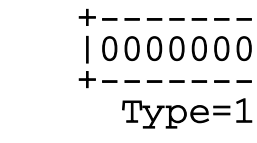



This option indicates the end of the option list. This might not coincide with the end of the internet header according to the internet header length. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the internet header.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

# Options (continuouse)

No Operation

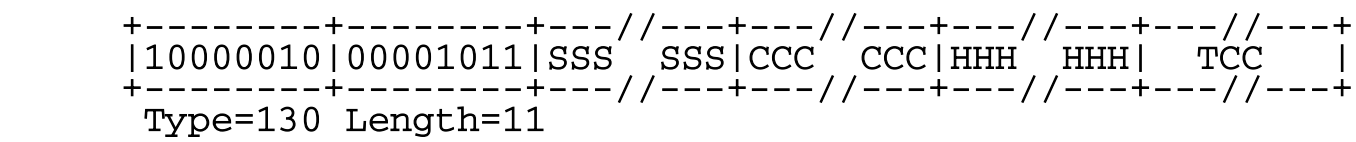



This option may be used between options, for example, to align the beginning of a subsequent option on a 32 bit boundary.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

Security

This option provides a way for hosts to send security, compartmentation, handling restrictions, and TCC (closed user group) parameters. The format for this option is as follows:




Security (S field): 16 bits

# Options (continuouse)

Compartments (C field): 16 bits

An all zero value is used when the information transmitted is not compartmented. Other values for the compartments field may be obtained from the Defense Intelligence Agency.

Handling Restrictions (H field): 16 bits

The values for the control and release markings are alphanumeric digraphs and are defined in the Defense Intelligence Agency Manual DIAM 65-19, "Standard Security Markings".

Transmission Control Code (TCC field): 24 bits

Provides a means to segregate traffic and define controlled communities of interest among subscribers. The TCC values are trigraphs, and are available from HQ DCA Code 530.

Must be copied on fragmentation. This option appears at most once in a datagram.

# Транспортные протоколы TCP и UDP

**TCP** — ориентированный на соединение протокол, что означает необходимость установки соединения между двумя хостами.

Как только соединение установлено, пользователи могут отправлять данные в обоих направлениях.

- **Надёжный** — TCP управляет подтверждением, повторной передачей и тайм-аутом сообщений. Производятся многочисленные попытки доставить сообщение. Если оно потеряется на пути, сервер вновь запросит потерянную часть. В TCP нет ни пропавших данных, ни (в случае корректно-установленных тайм-аутов) разорванных соединений

- **Упорядоченный** — если два сообщения последовательно отправлены, первое сообщение достигнет получателя первым. Если участки данных прибывают в неверном порядке, TCP отправляет неупорядоченные данные в буфер до тех пор, пока все данные не могут быть упорядочены и переданы приложению

- **Тяжеловесный** — TCP необходимо три пакета для установки соединения перед началом передачи данных. TCP следит за целостностью данных и перегрузками

- **Потоковый** — данные представляются потоком байтов, не существует особых границ сообщений или сегментов

**UDP** — более простой, основанный на сообщениях протокол без установления соединения. Такие протоколы не устанавливают соединения между двумя хостами. Связь осуществляется путем передачи информации в одном направлении от источника к получателю без проверки состояния и даже готовности получателя. Однако, основным преимуществом **UDP** над **TCP** являются приложения для голосовой связи через интернет-протокол (Voice over IP, VoIP), в котором любая буферизация помешала бы качеству голосовой связи. В VoIP считается, что конечные пользователи в реальном времени предоставят любое необходимое подтверждение о получении сообщения.

**Ненадёжный** — когда сообщение посылается, неизвестно, достигнет ли оно своего назначения — оно может потеряться по пути. Нет таких понятий, как подтверждение, повторная передача, тайм-аут

**Неупорядоченный** — если два сообщения отправлены одному получателю, то порядок их достижения цели не может быть предугадан.

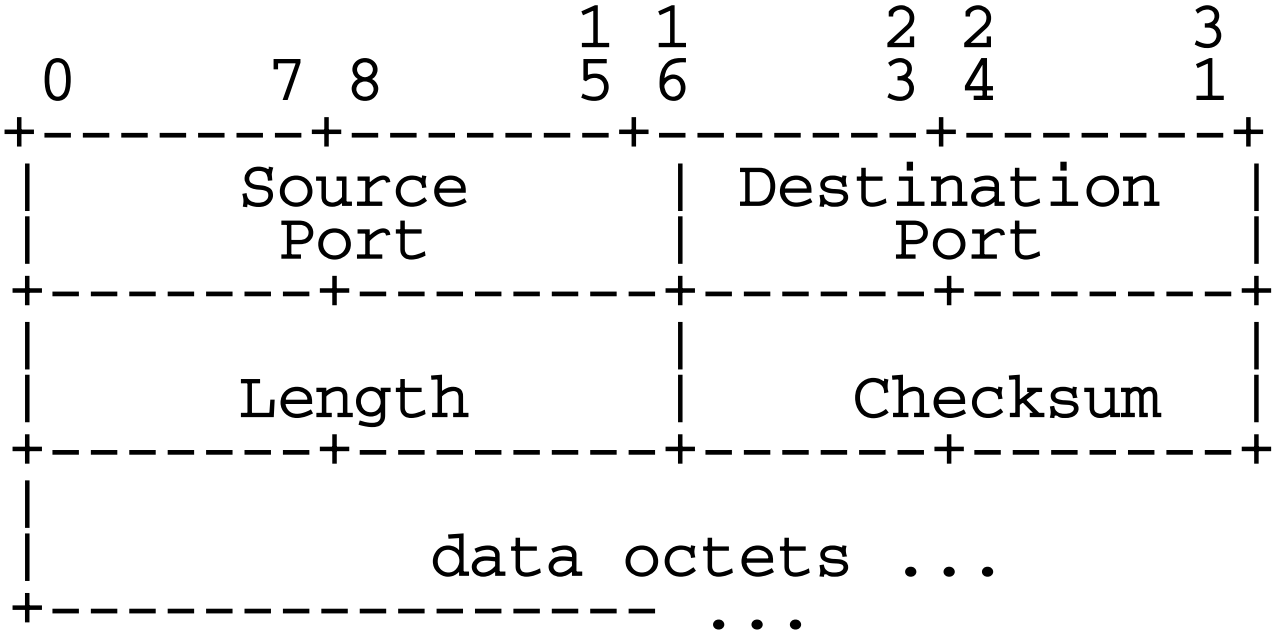
**Легковесный** — никакого упорядочивания сообщений, никакого отслеживания соединений и т. д. Это небольшой транспортный уровень, разработанный на IP.

**Датаграммы** — пакеты посылаются по отдельности и проверяются на целостность только если они прибыли. Пакеты имеют определенные границы, которые соблюдаются после получения, то есть операция чтения на сокете-получателе выдаст сообщение таким, каким оно было изначально послано.

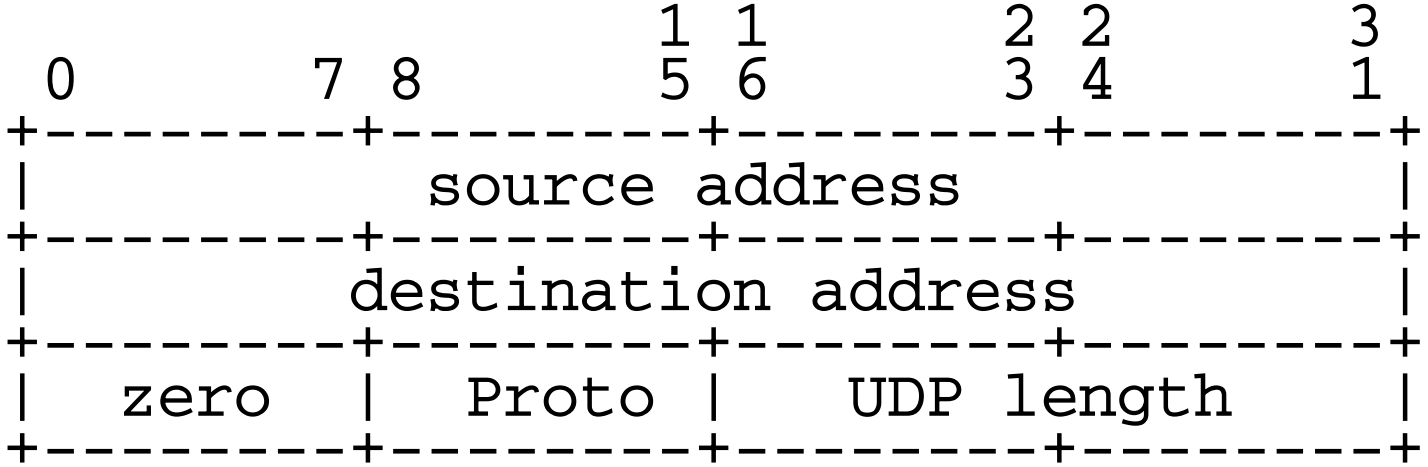
**Нет контроля перегрузок** — UDP сам по себе не избегает перегрузок. Для приложений с большой пропускной способностью возможно вызвать коллапс перегрузок, если только они не реализуют меры контроля на прикладном уровне.

# UDP Datagram Header

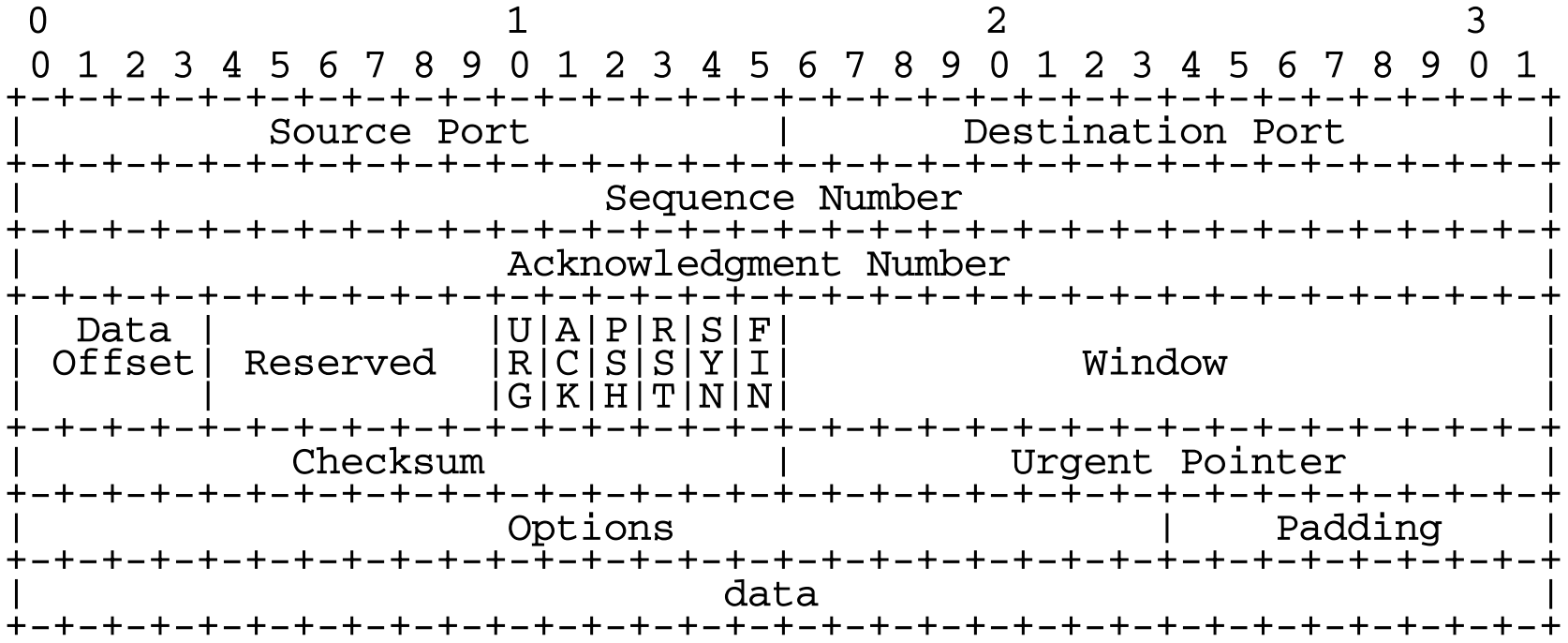
# UDP Header


0 7 8 15 16 23 24 31
Source Port
Destination Port
Length
Checksum
data octets ...

# UDP Pseudoheader Checksum



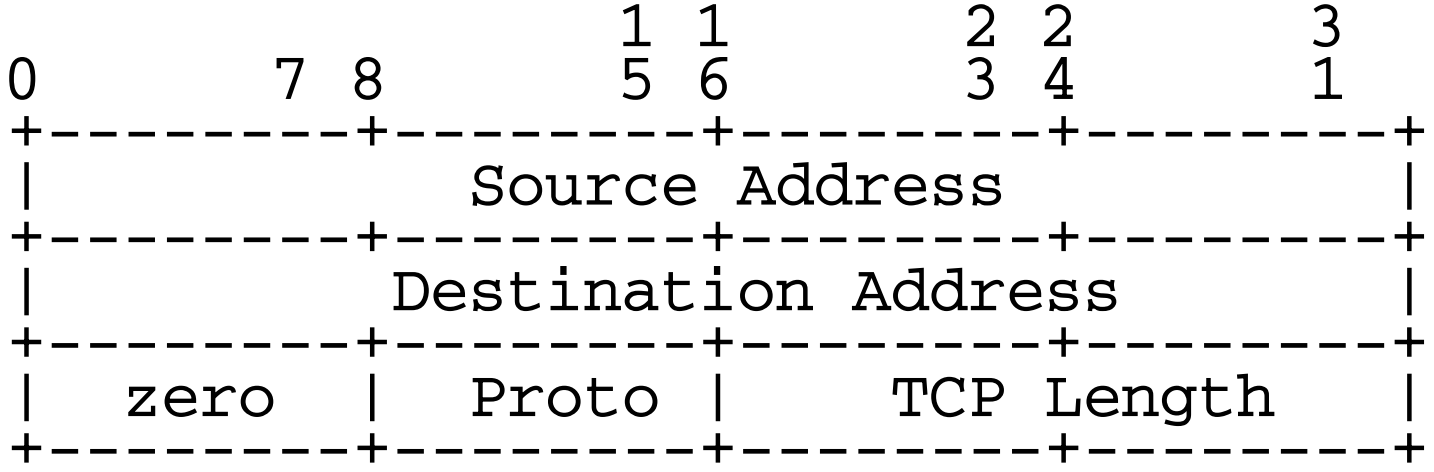
# TCP Segment Header



Control Bits (based):  6 bits (from left to right):

- **URG** – Urgent Pointer field significant
- **ACK** – Acknowledgment field significant
- **PSH** – Push Function
- **RST** – Reset the connection
- **SYN** – Synchronize sequence numbers
- **FIN** – No more data from sender

# TCP Pseudoheader Checksum



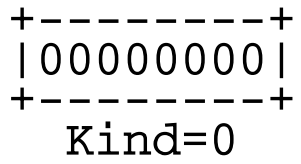
# Used Options

Currently defined options include (kind indicated in octal):

| Kind | Length | Meaning |
|---|---|---|
| 0 | - | End of option list. |
| 1 | - | No-Operation. |
| 2 | 4 | Maximum Segment Size. |

Specific Option Definitions

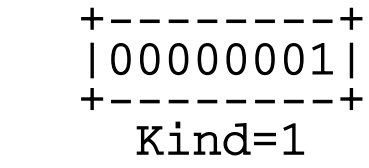
End of Option List




This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the TCP header.

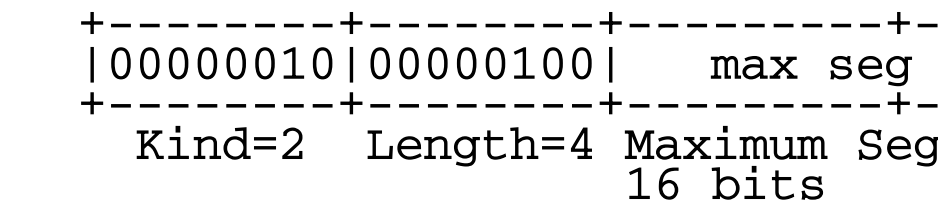
# Used Options (continuouse)

No-Operation



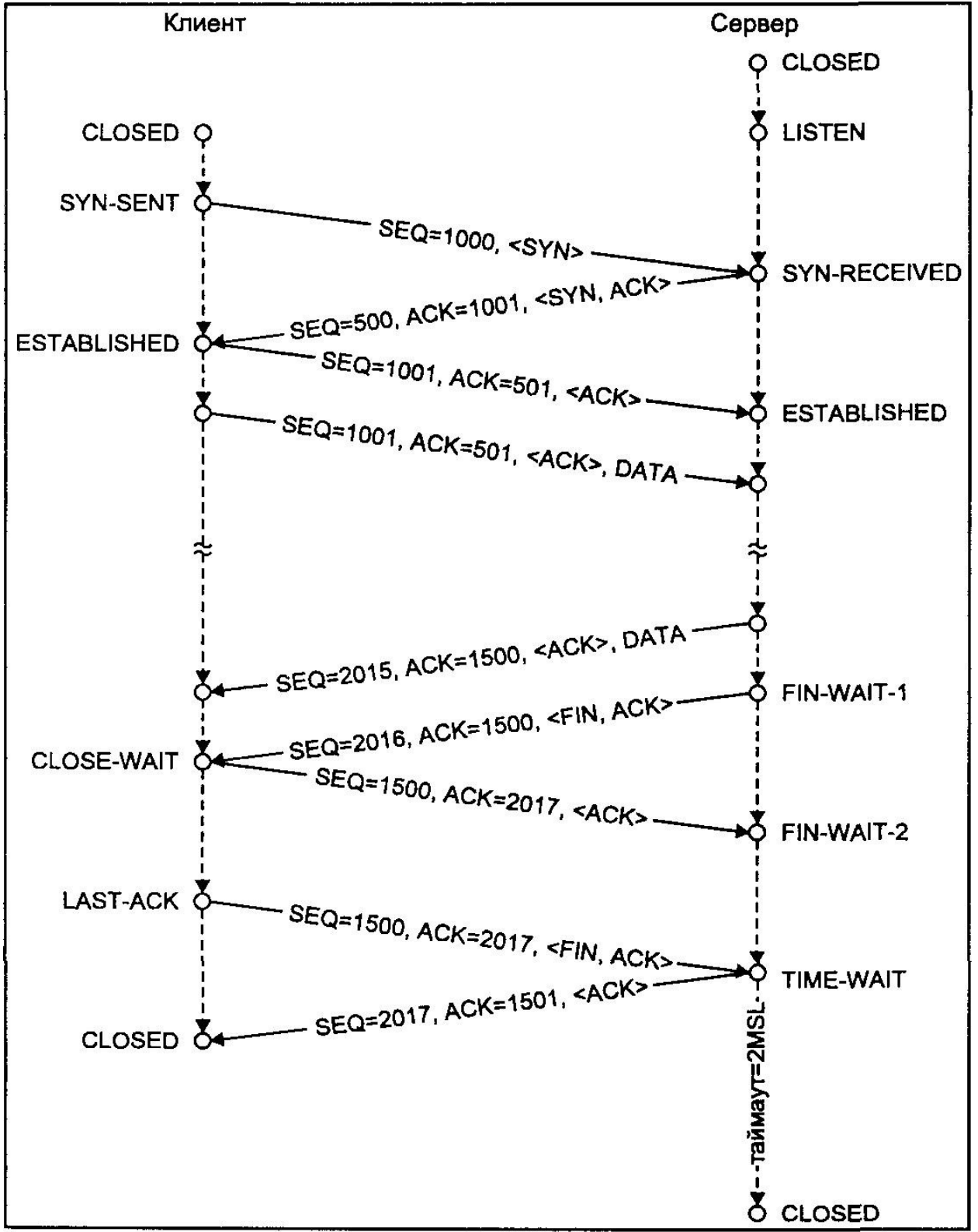

This option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary.

Maximum Segment Size

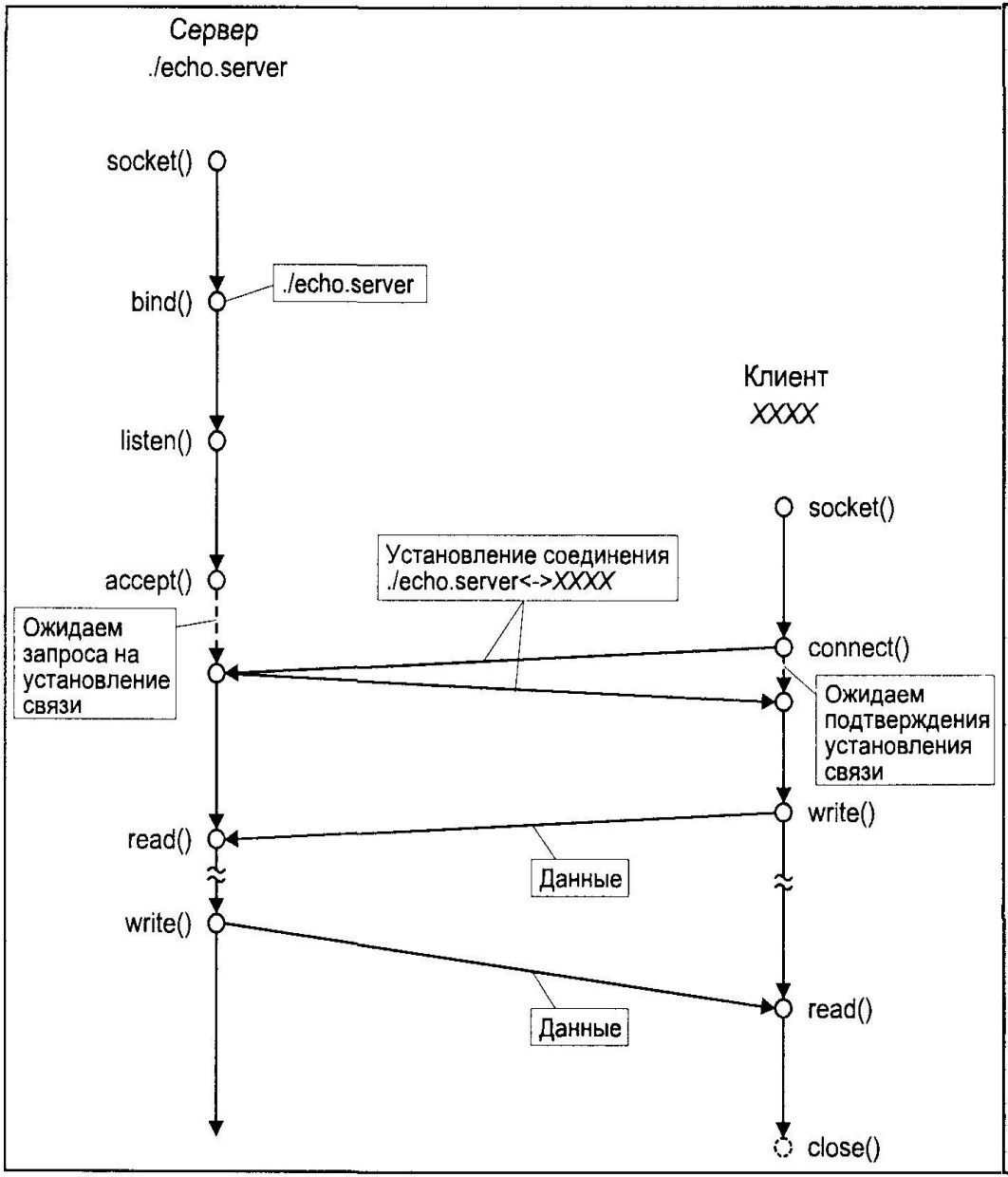


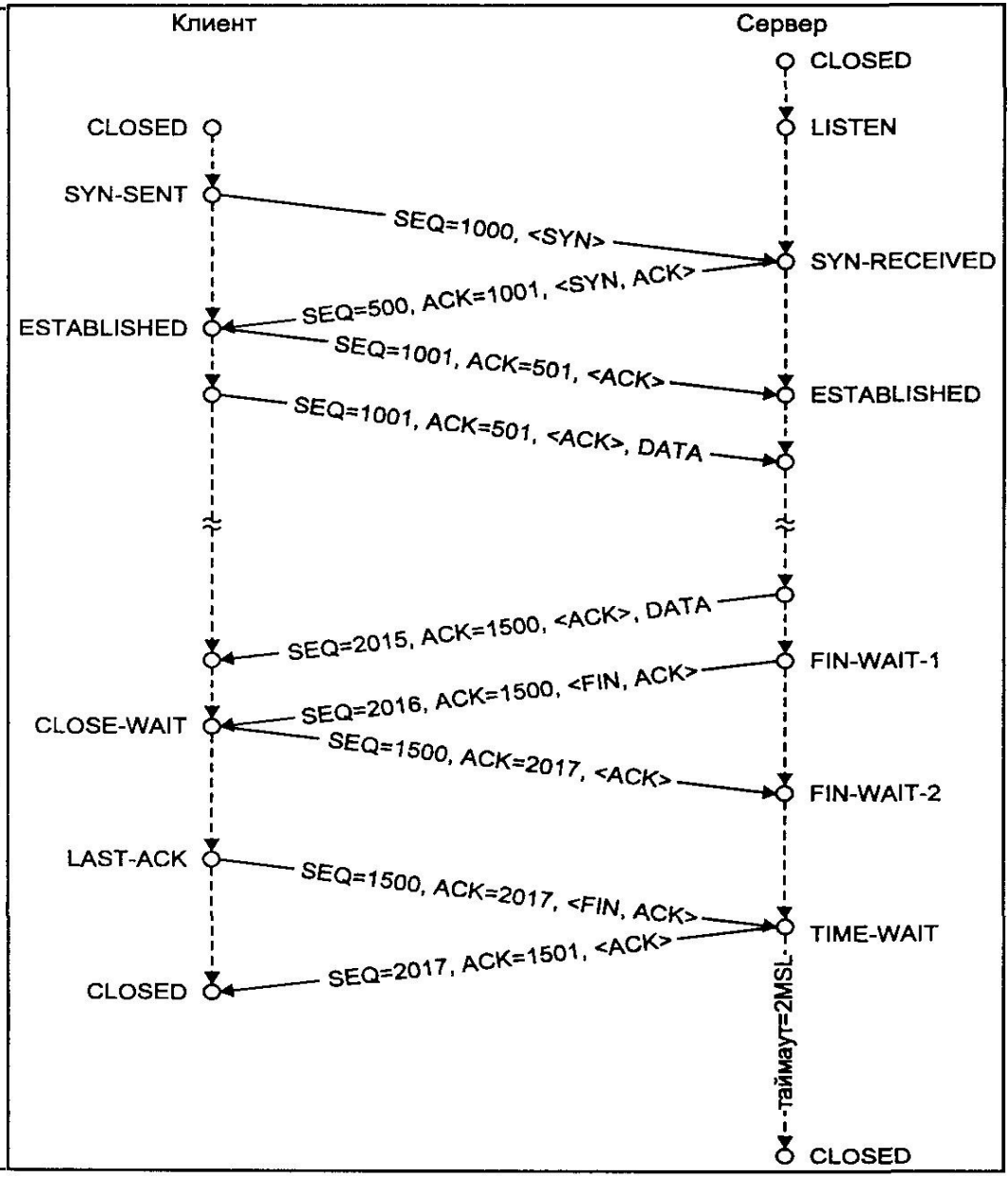

If this option is present, then it communicates the maximum receive segment size at the TCP which sends this segment. This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set). If this option is not used, any segment size is allowed.

# Установление и разрыв TCP соединения

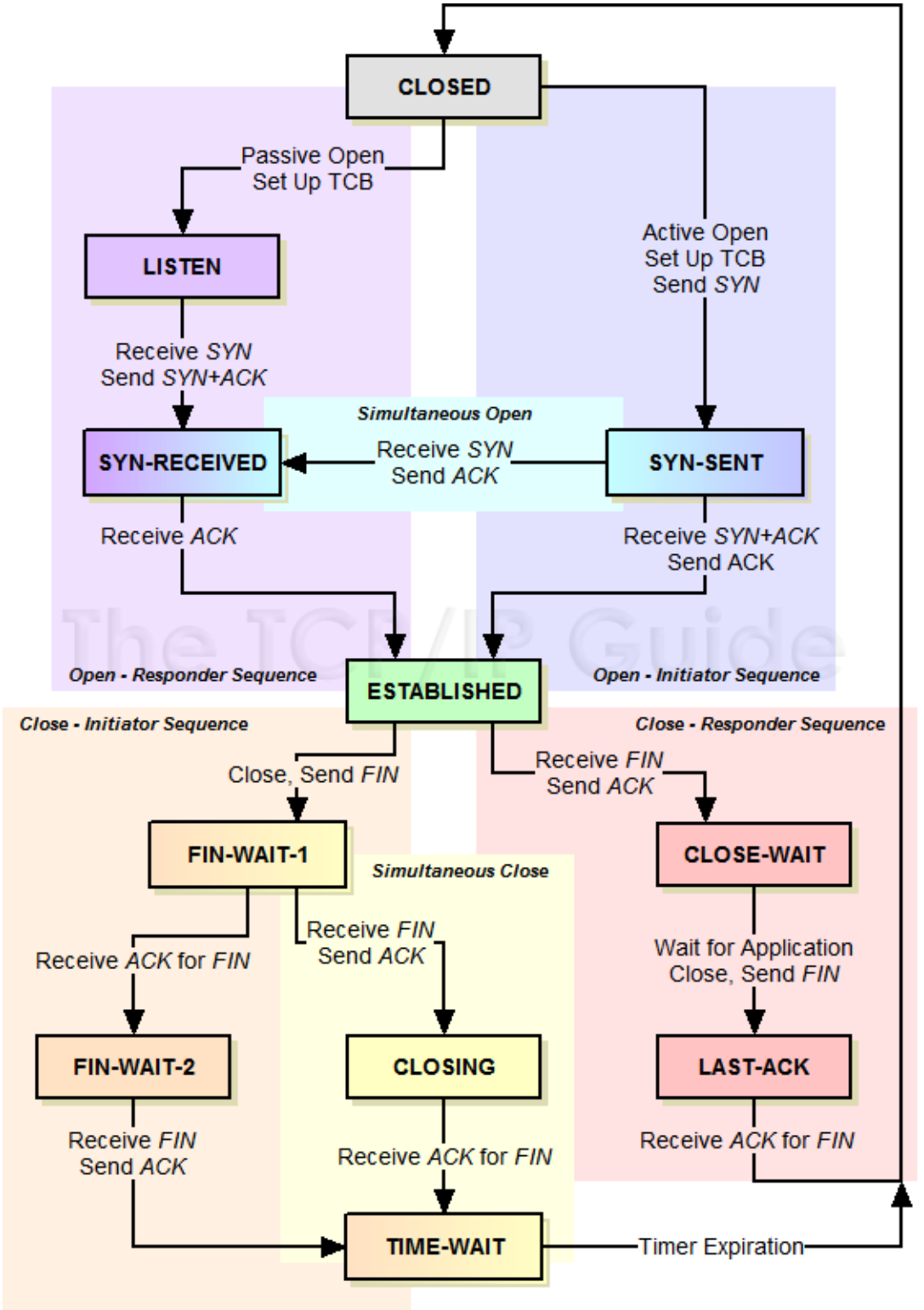

Клиент
Сервер
CLOSED
CLOSED
LISTEN
SYN-SENT
SEQ=1000, <SYN>
SYN-RECEIVED
SEQ=500, ACK=1001, <SYN, ACK>
ESTABLISHED
SEQ=1001, ACK=501, <ACK>
ESTABLISHED
SEQ=1001, ACK=501, <ACK>, DATA
SEQ=2015, ACK=1500, <ACK>, DATA
FIN-WAIT-1
SEQ=2016, ACK=1500, <FIN, ACK>
CLOSE-WAIT
SEQ=1500, ACK=2017, <ACK>
FIN-WAIT-2
LAST-ACK
SEQ=1500, ACK=2017, <FIN, ACK>
TIME-WAIT
SEQ=2017, ACK=1501, <ACK>
CLOSED
-таймаут=2MSL-
CLOSED

| | |
|---|---|
| LISTEN | Готовность узла к получению запроса на соединение от любого удаленного узла |
| SYN SENT | Ожидание ответного запроса на соединение |
| SYN RECEIVED | Ожидание подтверждения получения ответного запроса на соединение |
| ESTABLISHED | Состояние канала, при котором возможен дуплексный обмен данными между клиентом и сервером |
| CLOSE WAIT | Ожидание запроса на окончание связи от локального процесса, использующего данный коммуникационный узел |
| LAST ACK | Ожидание подтверждения запроса на окончание связи, отправленного удаленному узлу. Предварительно от удаленного узла уже был получен запрос на окончание связи и канал стал симплексным. |
| FIN WAIT 1 | Ожидание подтверждения запроса на окончание связи, отправленного удаленному узлу (инициирующий запрос, канал переходит в симплексный режим) |
| FIN WAIT 2 | Ожидание запроса на окончание связи от удаленного узла |
| CLOSING | Ожидание подтверждения от удаленного узла на запрос окончания связи. |
| TIME WAIT | Таймаут перед окончательным разрушением канала, достаточный для того, чтобы удаленный узел получил подтверждение своего запроса окончания связи. Величина тайм аута составляет 2 MSL (Maximum Segment Lifetime) |
| CLOSED | Фиктивное состояние при котором коммуникационный узел и канал фактически не существуют |

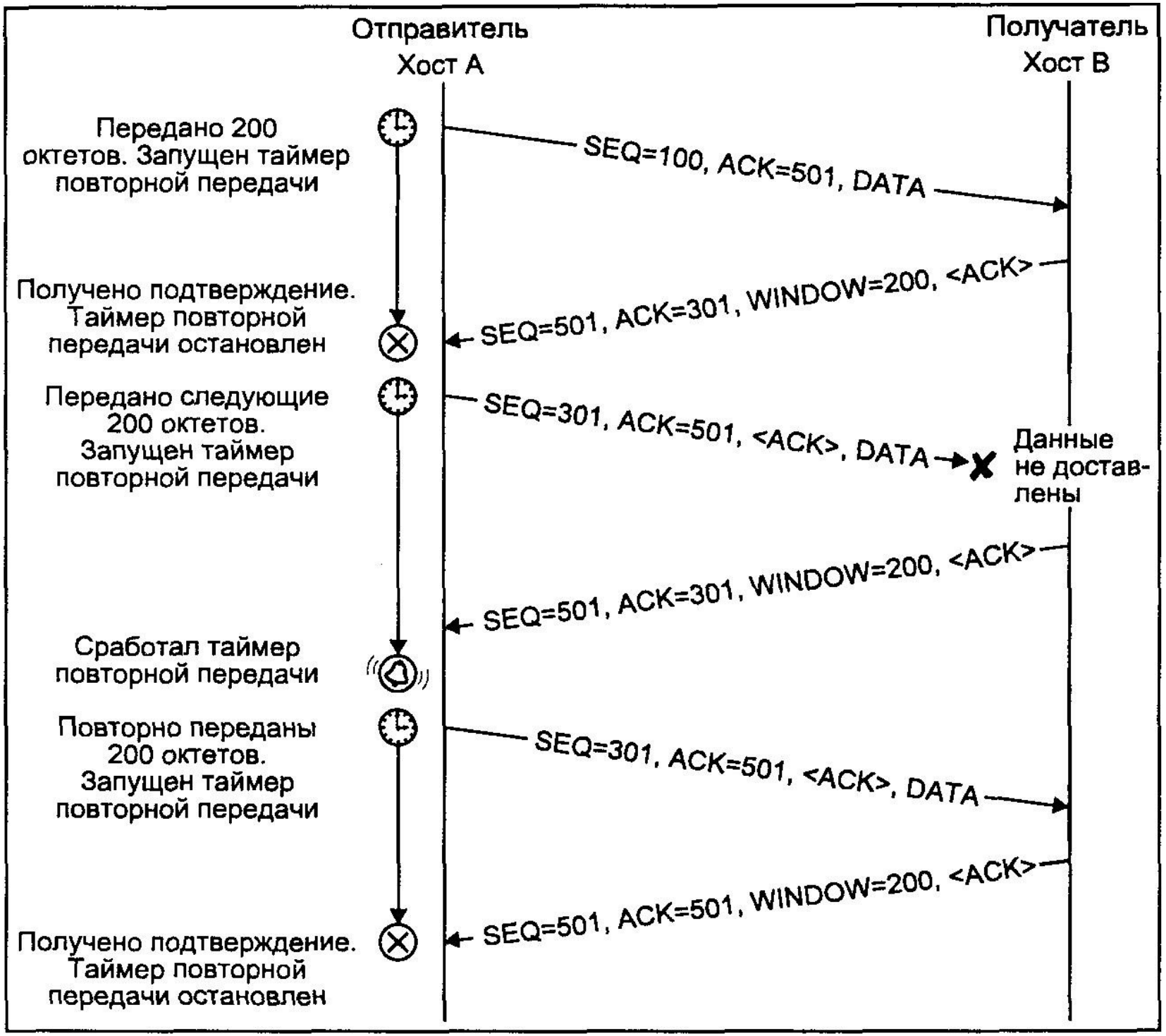

Сервер
./echo.server
socket()
bind()
./echo.server
listen()
accept()
Ожидаем запроса на установление связи
Клиент
XXXX
socket()
Установление соединения
./echo.server<->XXXX
connect()
Ожидаем подтверждения установления связи
write()
read()
Данные
write()
Данные
read()
close()


Клиент
Сервер
CLOSED
LISTEN
CLOSED
SYN-SENT
SEQ=1000, <SYN>
SYN-RECEIVED
SEQ=500, ACK=1001, <SYN, ACK>
ESTABLISHED
SEQ=1001, ACK=501, <ACK>
ESTABLISHED
SEQ=1001, ACK=501, <ACK>, DATA
SEQ=2015, ACK=1500, <ACK>, DATA
FIN-WAIT-1
SEQ=2016, ACK=1500, <FIN, ACK>
CLOSE-WAIT
SEQ=1500, ACK=2017, <ACK>
FIN-WAIT-2
LAST-ACK
SEQ=1500, ACK=2017, <FIN, ACK>
TIME-WAIT
SEQ=2017, ACK=1501, <ACK>
CLOSED
таймаут=2MSL
CLOSED

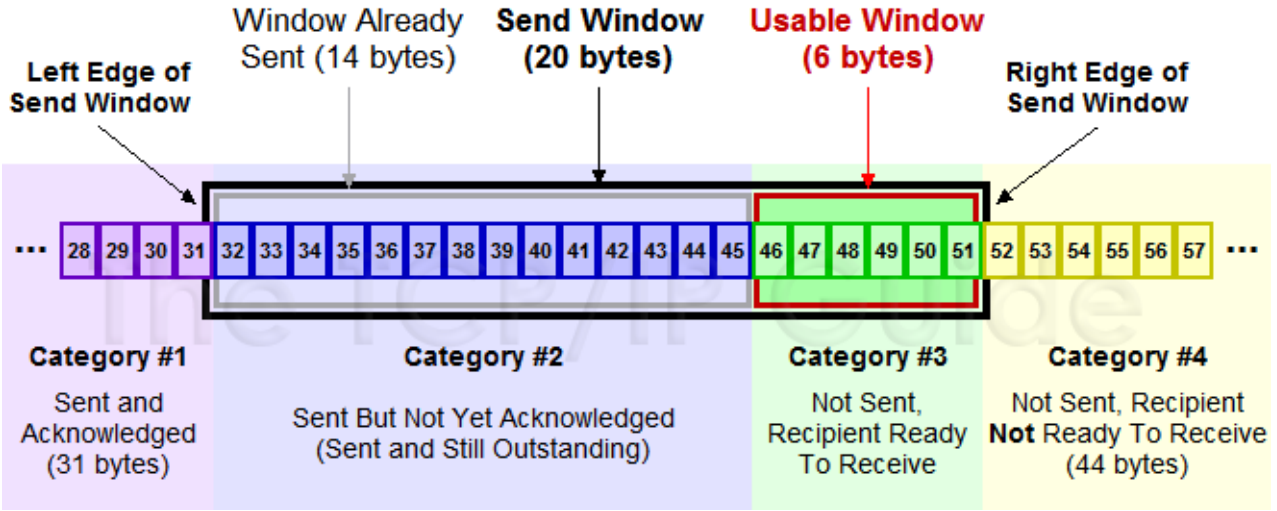
# TCP как конечный автомат



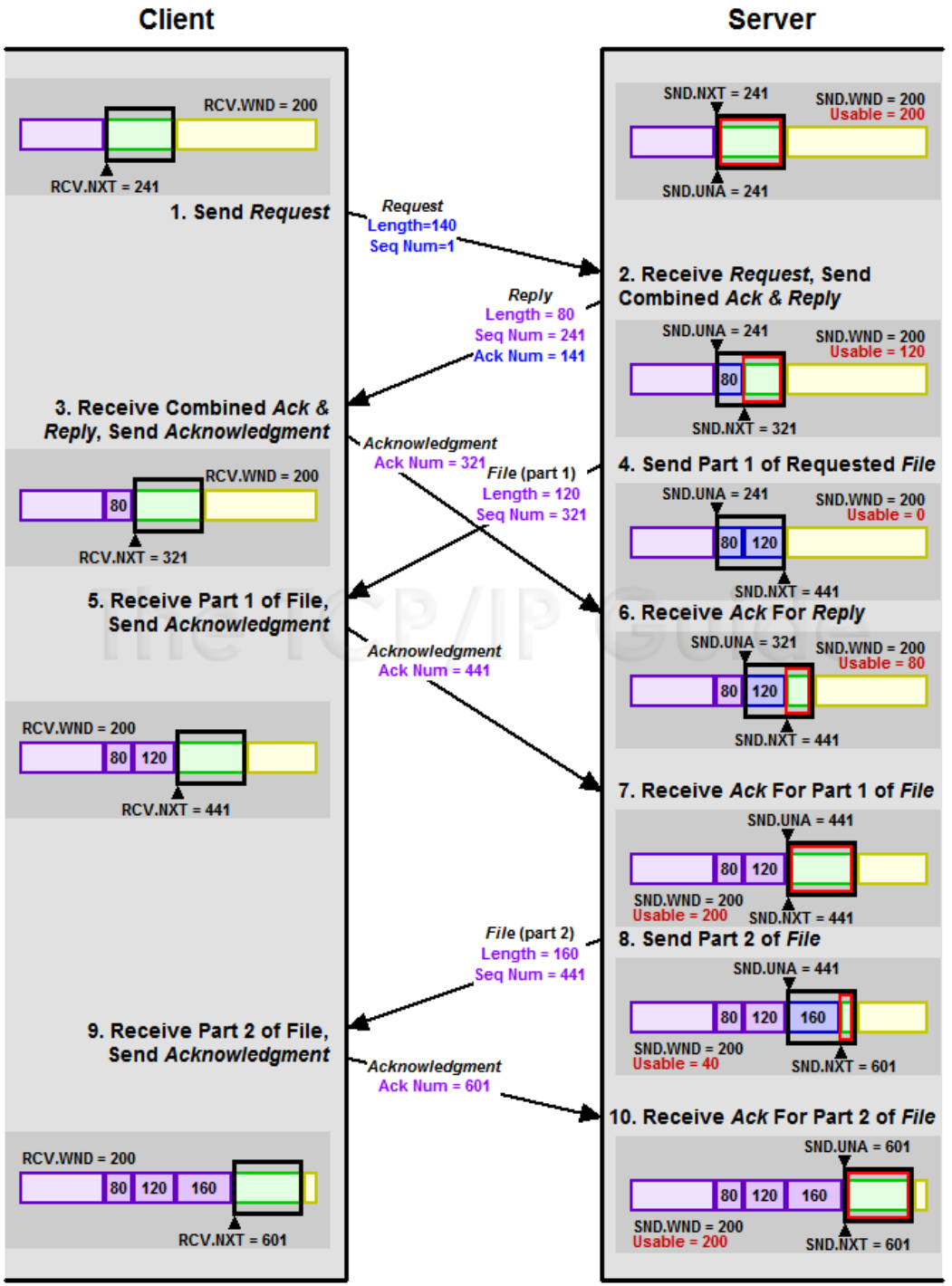
# Механизм повторной передачи


Отправитель
Хост А
Получатель
Хост В
Передано 200 октетов. Запущен таймер повторной передачи
SEQ=100, ACK=501, DATA
Получено подтверждение. Таймер повторной передачи остановлен
SEQ=501, ACK=301, WINDOW=200, <ACK>
Передано следующие 200 октетов. Запущен таймер повторной передачи
SEQ=301, ACK=501, <ACK>, DATA
Данные не доставлены
SEQ=501, ACK=301, WINDOW=200, <ACK>
Сработал таймер повторной передачи
Повторно переданы 200 октетов. Запущен таймер повторной передачи
SEQ=301, ACK=501, <ACK>, DATA
SEQ=501, ACK=501, WINDOW=200, <ACK>
Получено подтверждение. Таймер повторной передачи остановлен

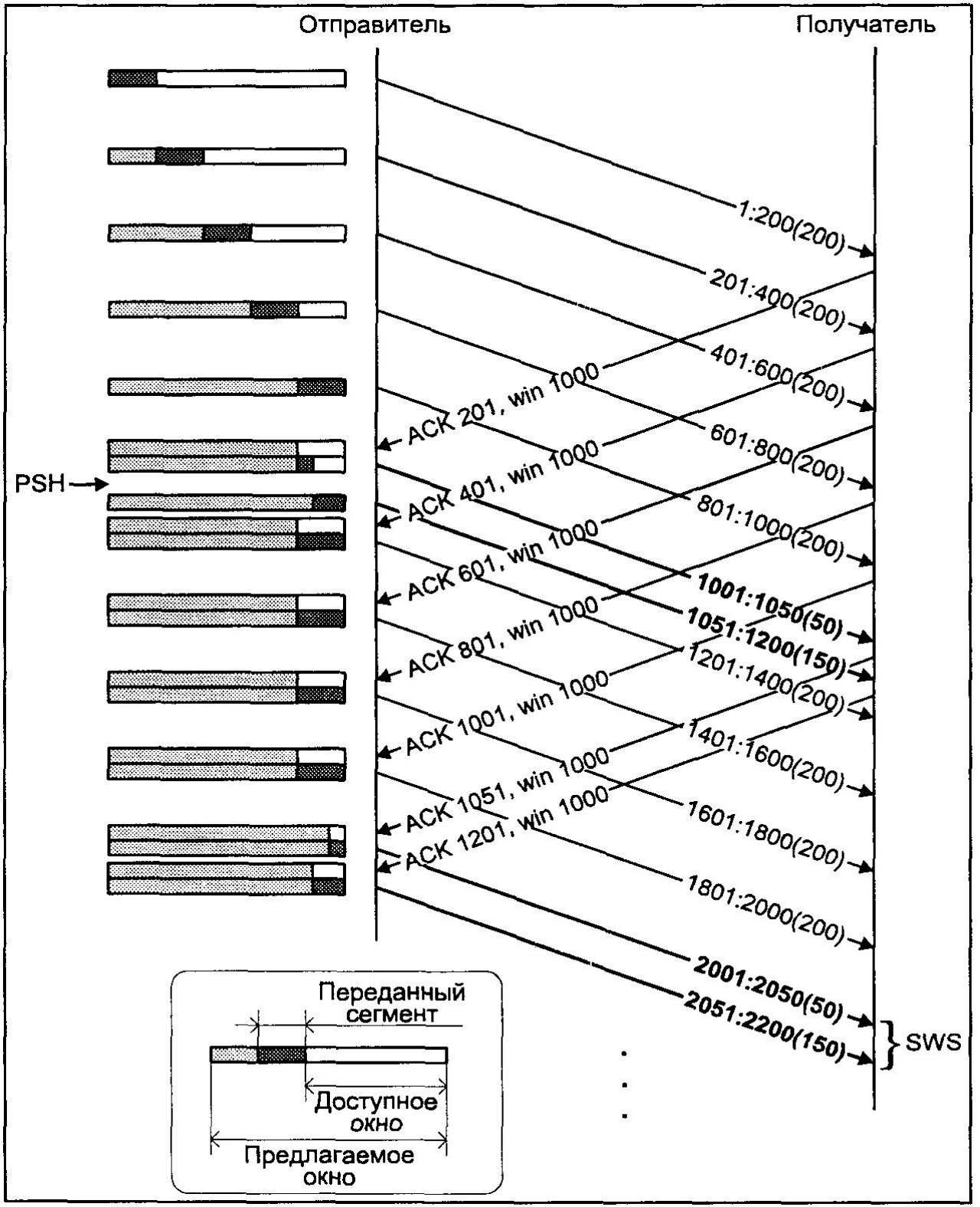
# Скользящее окно TCP




- Окно закрывается по мере смещения левого края вправо. Это происходит при отправлении данных
- Окно открывается по мере смещения правого края вправо. Это происходит в соответствии с получением подтверждений
- Окно сжимается, когда правый край смещается влево при срабатывании механизма повторной передачи. Модуль TCP должен быть готов к обработке этой ситуации.


Client
Server
RCV.WND = 200
RCV.NXT = 241
1. Send Request
Request
Length=140
Seq Num=1
SND.NXT = 241
SND.WND = 200
Usable = 200
SND.UNA = 241
2. Receive Request, Send Combined Ack & Reply
Reply
Length = 80
Seq Num = 241
Ack Num = 141
SND.UNA = 241
SND.WND = 200
Usable = 120
80
SND.NXT = 321
3. Receive Combined Ack & Reply, Send Acknowledgment
Acknowledgment
Ack Num = 321
4. Send Part 1 of Requested File
File (part 1)
Length = 120
Seq Num = 321
RCV.WND = 200
80
RCV.NXT = 321
SND.UNA = 241
SND.WND = 200
Usable = 0
80
120
SND.NXT = 441
6. Receive Ack For Reply
5. Receive Part 1 of File, Send Acknowledgment
Acknowledgment
Ack Num = 441
SND.UNA = 321
SND.WND = 200
Usable = 80
80
120
SND.NXT = 441
RCV.WND = 200
80
120
RCV.NXT = 441
7. Receive Ack For Part 1 of File
SND.UNA = 441
80
120
SND.WND = 200
Usable = 200
SND.NXT = 441
File (part 2)
Length = 160
Seq Num = 441
8. Send Part 2 of File
SND.UNA = 441
80
120
160
SND.WND = 200
Usable = 40
SND.NXT = 601
9. Receive Part 2 of File, Send Acknowledgment
Acknowledgment
Ack Num = 601
10. Receive Ack For Part 2 of File
RCV.WND = 200
80
120
160
RCV.NXT = 601
SND.UNA = 601
80
120
160
SND.WND = 200
Usable = 200
SND.NXT = 601

# Синдром глупого окна


Отправитель
Получатель
PSH
1:200(200)
201:400(200)
401:600(200)
601:800(200)
801:1000(200)
1001:1050(50)
1051:1200(150)
1201:1400(200)
1401:1600(200)
1601:1800(200)
1801:2000(200)
2001:2050(50)
2051:2200(150)
ACK 201, win 1000
ACK 401, win 1000
ACK 601, win 1000
ACK 801, win 1000
ACK 1001, win 1000
ACK 1051, win 1000
ACK 1201, win 1000
SWS
Переданный сегмент
Доступное окно
Предлагаемое окно

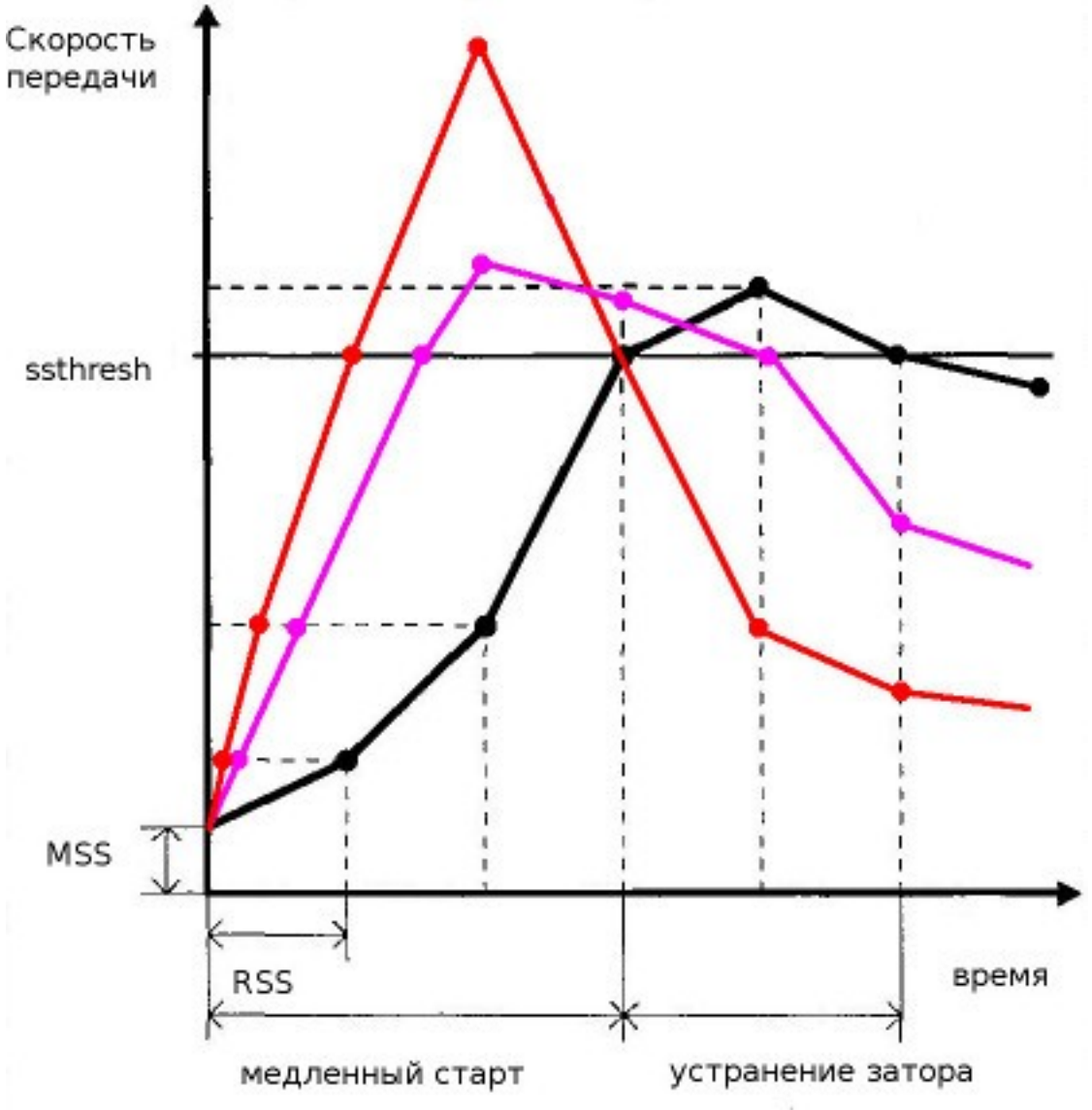
- Принимающая сторона не должна анонсировать маленькие окна. Адресат не должен анонсировать размер окна, больший текущего (который скорее всего равен 0), пока последний не может быть увеличен либо на размер максимального сегмента (Maximum Segment Size, MSS), либо на 1/2 размера буфера приёма, в зависимости от того, какое значение окажется меньшим

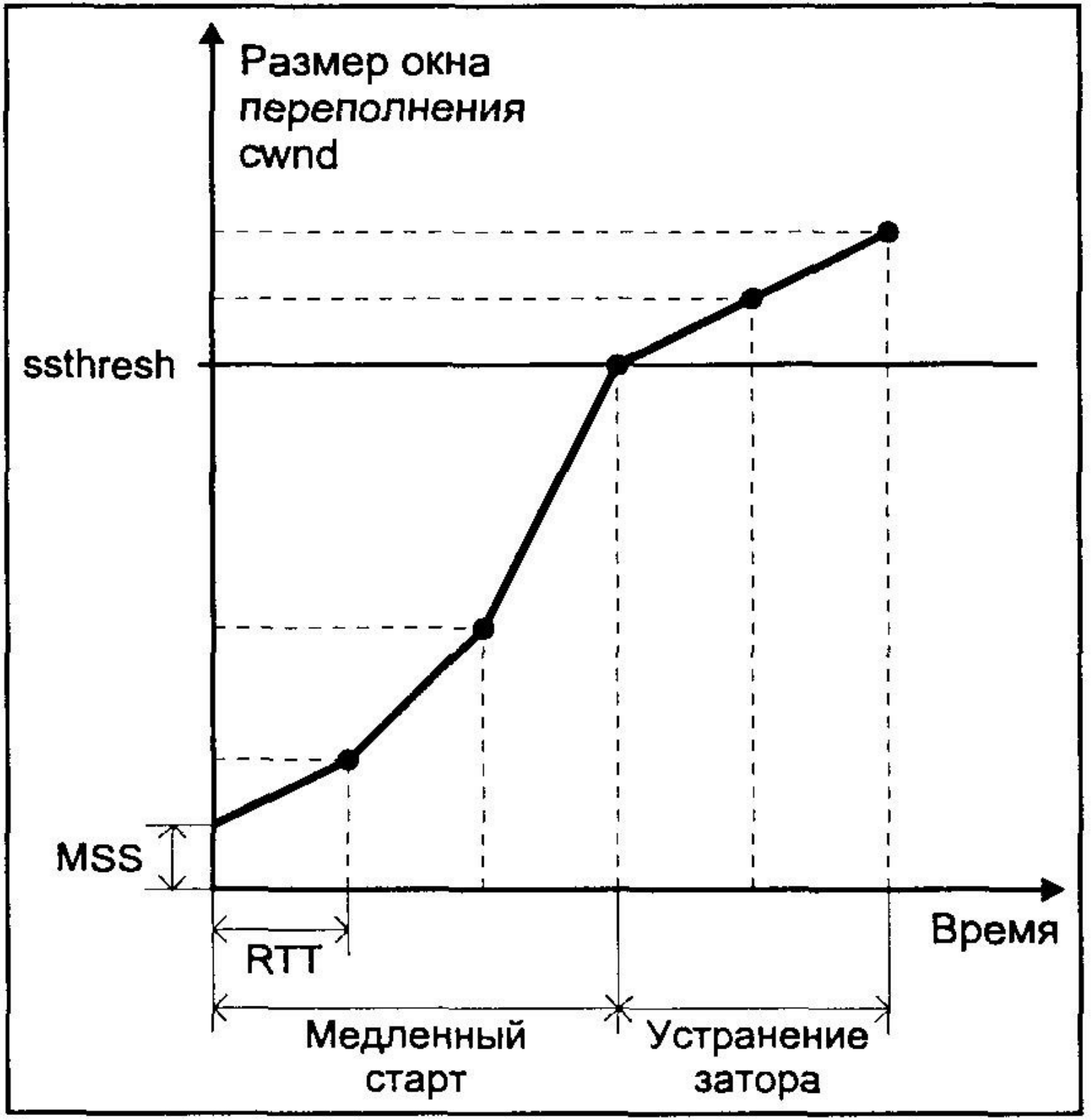
- Отправитель должен воздержаться от передачи, пока он не сможет передать сегмент максимального размера или сегмент, размер которого больше половины максимального размера окна, который когда либо анонсировался принимающей стороной

- При получении сегмента адресат не отправляет подтверждение, если, во первых, сегмент не содержит флага PSH (дающего основание полагать, что вслед за полученным сегментом вскоре последуют дополнительные данные), и, во вторых, отсутствует необходимость отправки обновлённого значения окна

- Тем не менее получатель должен установить таймер, который позволит послать подтверждение, если в передаче данных произошёл определённый перерыв, что может быть вызвано, например, потерей сегментов

# Стратегия медленного старта


Скорость
передачи
ssthresh
MSS
RSS
время
медленный старт
устранение затора


Размер окна
переполнения
cwnd
ssthresh
MSS
RTT
Время
Медленный
старт
Устранение
затора

- Вводится новая величина «окно переполнение» (congestion window), `cwnd`
- Начальное значение `cwnd` устанавливается равным MSS или меньше (536 или 512 октетов)
- При вычислении доступного окна отправитель использует меньшее из предлагаемого окна и окна переполнения.
- Каждый раз, когда отправитель получает подтверждение полученного сегмента, его окно переполнения увеличивается на величину этого сегмента

Легко заметить, что предлагаемое окно служит для управления потоком со стороны получателя, в то время как окно переполнения служит для управления со стороны отправителя.

Если первое из них связано с наличием свободного места в буфере приема адресата, то второе — с представлением о загрузке сети у отправителя данных.

- Если разрешённое окно передачи больше MSS, то отправитель передаёт сегмент размера MSS и ожидает подтверждения

- При приходе подтверждения отправитель увеличивает `cwnd` вдвое

- Следующее увеличение `cwnd` происходит при получении всех подтверждений на отправленные разрешённые сегменты

- По достижении скорости, близкой к скорости насыщения, включается механизм устранения затора

$$cwnd_o = sz$$
$$cwnd_1 = cwnd_o + (cwnd_o/sz)*sz = 2\ sz = 2\ cwnd_o$$
$$cwnd_n = cwnd_{n-1} + (cwnd_{n-1}/sz)*sz = 2\ cwnd_{n-1}$$

# Устранение затора

Будем считать, что вероятность ошибок на физическом уровне мала (меньше 0,5%)

Поэтому потеря данных у получателя будет свидетельствовать о заторе. В свою очередь, со стороны отправителя о заторе можно судить по значительной паузе в получении подтверждения и/или получении дубликатов подтверждений

## Наряду с окном переполнения, введём величину порога меденного старта (`sstresh`)

- Начальные значения `cwnd` и `ssthresh` инициализируются равными размеру одного сегмента и 65535 байтов соответственно
- Максимальное количество данных, которое может передать отправитель, не превышает меньшего из значений окна переполнения и предлагаемого окна
- При возникновении затора (что определяется по тайм ауту или получению дубликатов подтверждений) параметр `ssthresh` устанавливается равным половине текущего окна, но не меньше размера двух сегментов. Если же свидетельством затора является тайм аут, то дополнительно размер `cwnd` сбрасывается до одного сегмента, или, другими словами, включается медленный старт
- Когда отправитель получает подтверждение, он увеличивает размер `cwnd`, однако новый размер зависит от того, выполняет ли модуль медленный старт или устранение затора. При этом до устранения затора, размер окна, вызвавший этот затор будет в диапазоне от `ssthresh до 2*ssthresh`

В фазе устранения затора

$$cwnd_n = cwnd_{n-1} + s/cwnd_{n-1},$$

s – число одновременных подтверждений
Эта формула обеспечивает приращение скорости передачи не выше, чем MSS/RTT.
(RTT – Round Time Trip)

Пусть в какой-то момент времени размер окна равен $cwnd_n$, отправитель может передать максимум сегментов размером sz, на которые он получит такое же число подтверждений.

$$cwnd_{n+1} < cwnd_n + (s*cwnd_n*sz)/(s*cwnd_n) = cwnd_n + sz$$

# Расширенное дифференцированное управление потоком

Дублирование подтверждений рассматривалось как признак затора в сети, однако, согласно RFC 1122 это также может служить признаком прихода неупорядоченных сегментов. Поэтому требуется дополнительный механизм подтверждения затора. Для этого вводятся новые флаги:

- ECN bits (from reserved bits 6,7 in TOS field in IP header)

- NS – маскировка ECN-nonce (RFC 3540).
- CWR (Congestion Window Reduced) — Поле «Окно перегрузки уменьшено» — флаг установлен отправителем, чтобы 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) (from 4-6 bits from reserved in TCP header before flags fields)

- 00 – Non ECN-Capable Transport, Non-ECT
- 10 – ECN Capable Transport, ECT(0)
- 01 – ECN Capable Transport, ECT(1)
- 11 – Congestion Encountered, CE

Если и отправитель и получатель поддерживают ECN, они могут использовать как алгоритм ECN(0), так и ECN(1)

Маршрутизатор в случае перегрузки вместо отбрасывания пакетов, может изменить значения 01 и 10 ECN битов до 11, тем самым сообщая получателю данных о перегрузке

Поскольку качественная обработка перегрузки возможна только на транспортном уровне, то требуется поддержка со стороны протокола TCP

- При установлении TCP соединения, сторона инициирующая соединение и поддерживающая ECN, устанавливает флаги ECN в 11, и также устанавливая в 1 любые из используемых флагов NS, CWR, ECE. Отвечающая сторона поступает аналогично
- В случае перегрузки, детектируемой получателем данных, он устанавливает флаги ECN в значение 11
- Если соединение установлено по протоколу TCP, удалённая сторона должна подтвердить приём битов ECN флагом ECE в случае включения алгоритма устранения затора
- CWR-флаг устанавливается отправителем в 1 при явном сбросе значения cwnd при устранении затора. Информирует получателя о фактическом включении медленного старта
- В случае перегрузки, детектируемой отправителем данных, получатель данных должен ответить флагом NS, если использовал повтор подтверждений как индикатором нарушения порядка приёма данных и флагом ECE во всех остальных случаях

Спасибо за внимание!