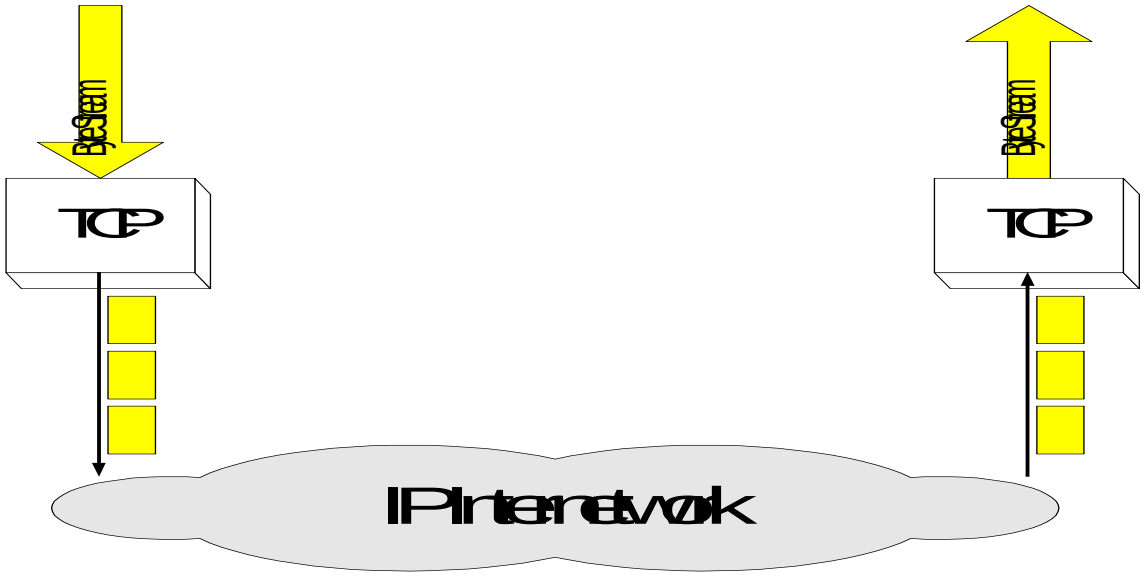
TCP

Overview

TCP = Transmission Control Protocol

- Connection-oriented protocol
- Provides a reliable unicast end-to-end byte stream over an unreliable internetwork.



TCP features

- Guaranteed service protocol [RFC 793]
  - Ensures that a packet has been received by the destination by using acknowledgments and retransmission
- Connection oriented
  - Applications need to establish a TCP connection prior to transfer
  - 3-way handshake
- Full duplex
  - Both ends can simultaneously read and write
- Byte stream
  - Ignores message boundaries
- Flow and congestion control
  - Source uses feedback to adjust transmission rate

TCP features

Acknowledgments from receiver

- Positive: “okay” or “ACK”
- Negative: “please repeat that” or “NACK” (not implemented for all TCP versions)

Timeout by the sender (“stop and wait”)

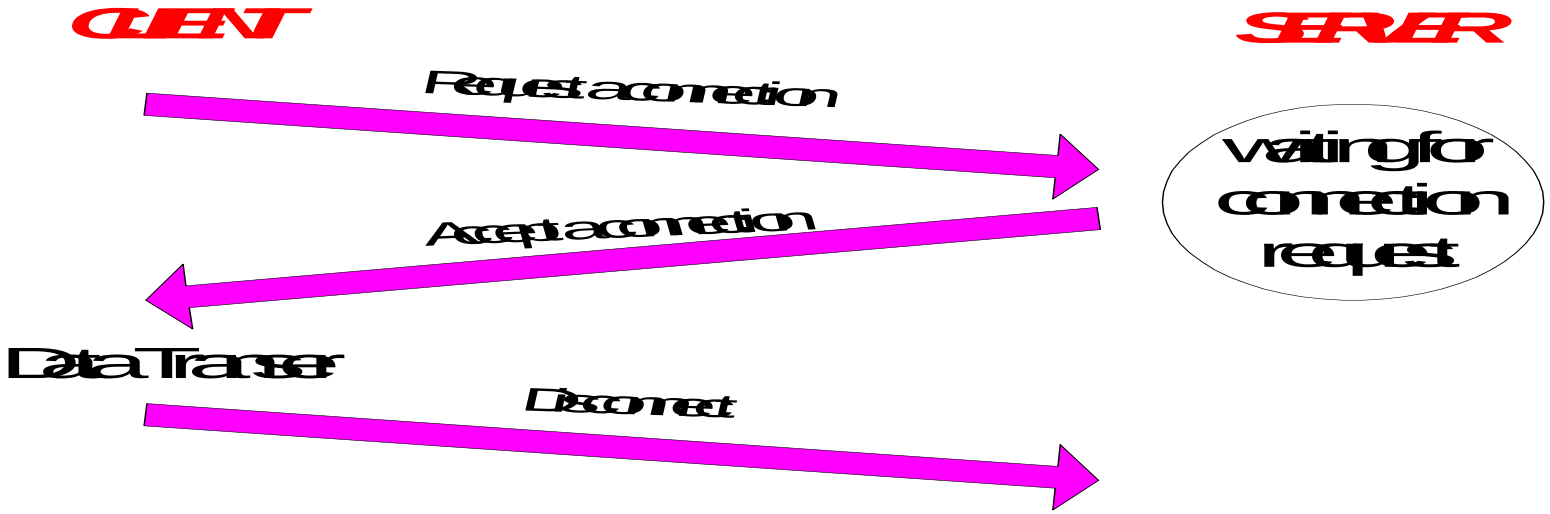
- Don’t wait indefinitely without receiving some response
- … whether a positive or a negative acknowledgment

Retransmission by the sender

- After receiving a “NACK” from the receiver
- After receiving no feedback from the receiver

Connection-Oriented

- Before any data transfer, TCP establishes a **connection**:
  - One TCP entity is waiting for a connection ("**server**")
  - The other TCP entity ("**client**") contacts the server
- Each connection is full duplex



Challenges of Reliable Data Transfer

- Over a perfectly reliable channel
  - All of the data arrives in order, just as it was sent
  - Simple: sender sends data, and receiver receives data
- Over a channel with bit errors
  - All of the data arrives in order, but some bits corrupted
  - Receiver detects errors and says “please repeat that”
  - Sender retransmits the data that were corrupted
- Over a lossy channel with bit errors
  - Some data are missing, and some bits are corrupted
  - Receiver detects errors but cannot always detect loss
  - Sender must wait for acknowledgment (“ACK” or “OK”)
  - … and retransmit data after some time if no ACK arrives

Reliable

• Byte stream is broken up into chunks which are called **segments**

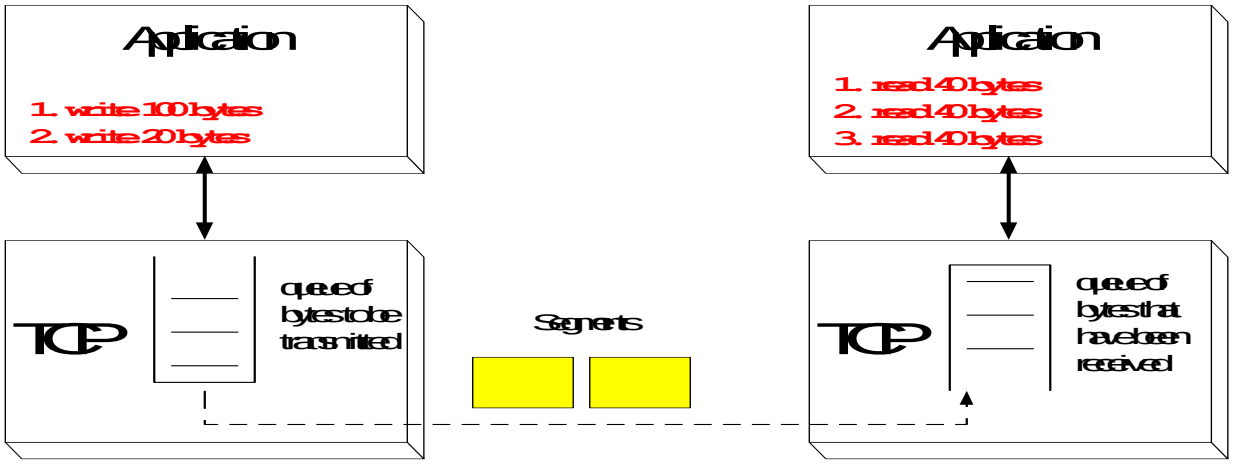
- Receiver sends acknowledgments (ACKs) for segments
- TCP maintains a timer. If an ACK is not received in time, the segment is retransmitted

•Detecting errors:

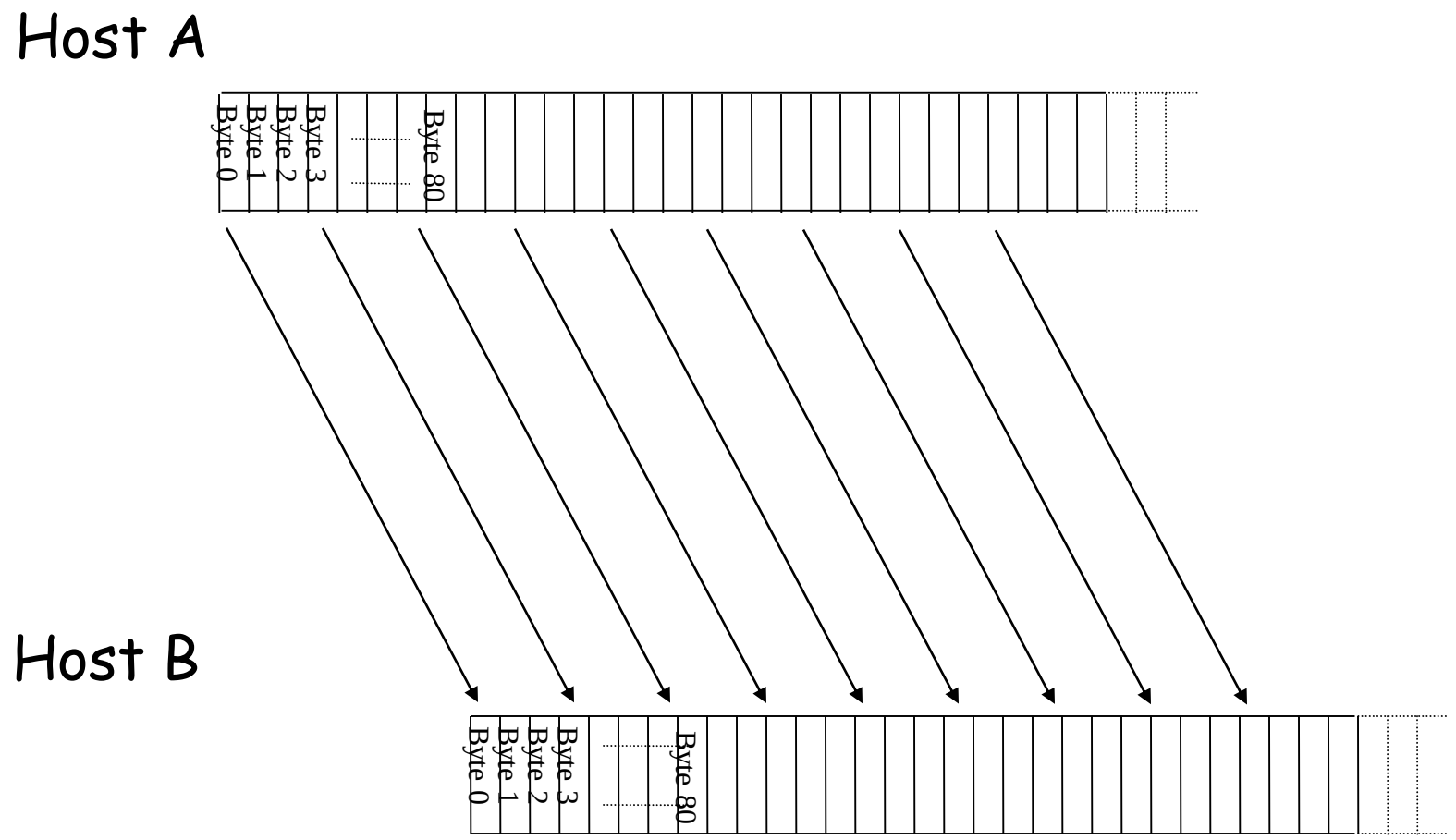
- TCP has checksums for header and data. Segments with invalid checksums are discarded
- Each byte that is transmitted has a sequence number

Byte Stream Service

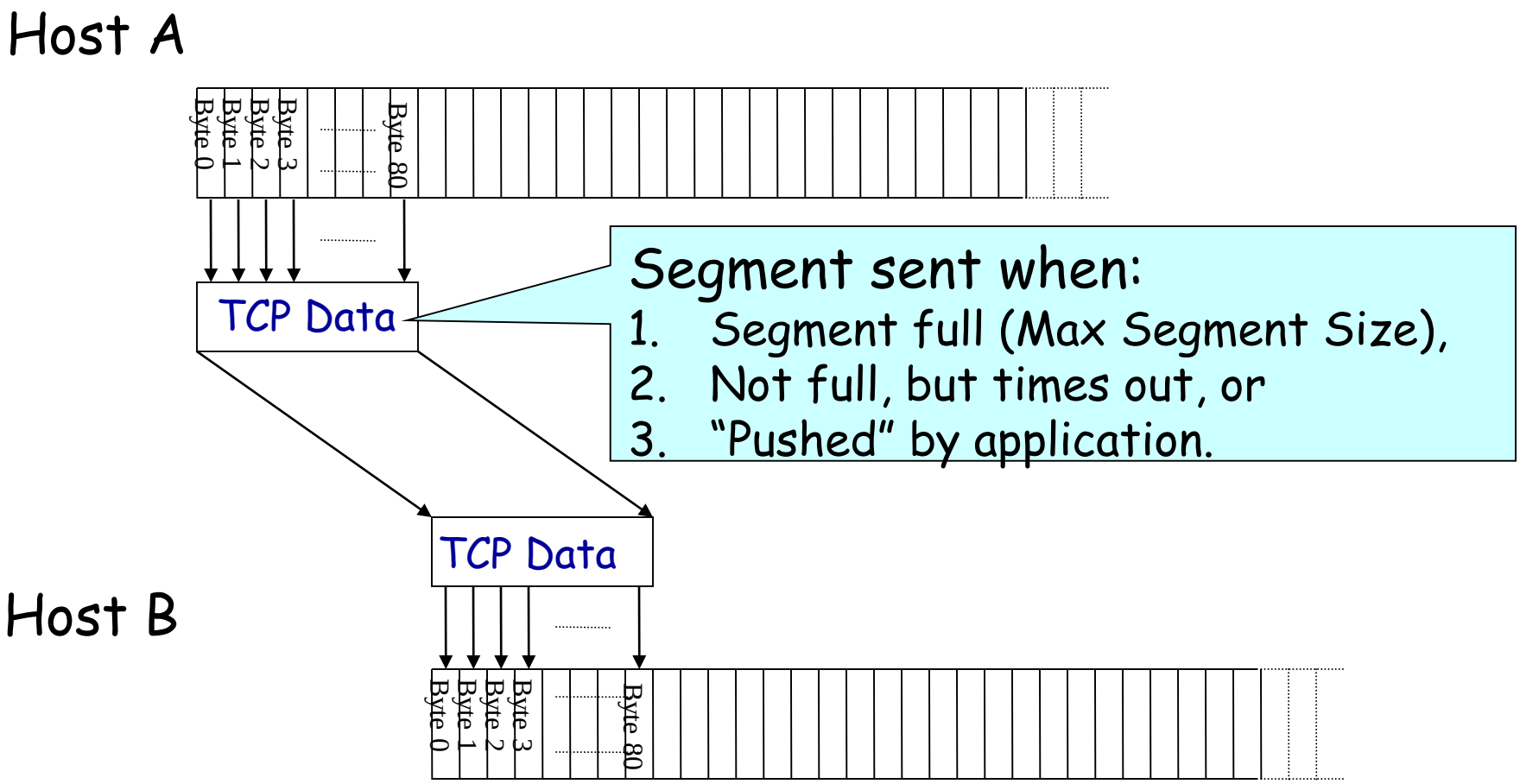
- To the lower layers, TCP handles data in blocks, the segments.
- To the higher layers TCP handles data as a sequence of bytes and does not identify boundaries between bytes
- So: Higher layers do not know about the beginning and end of segments !



Byte Stream Service



Emulated Using TCP "Segments"

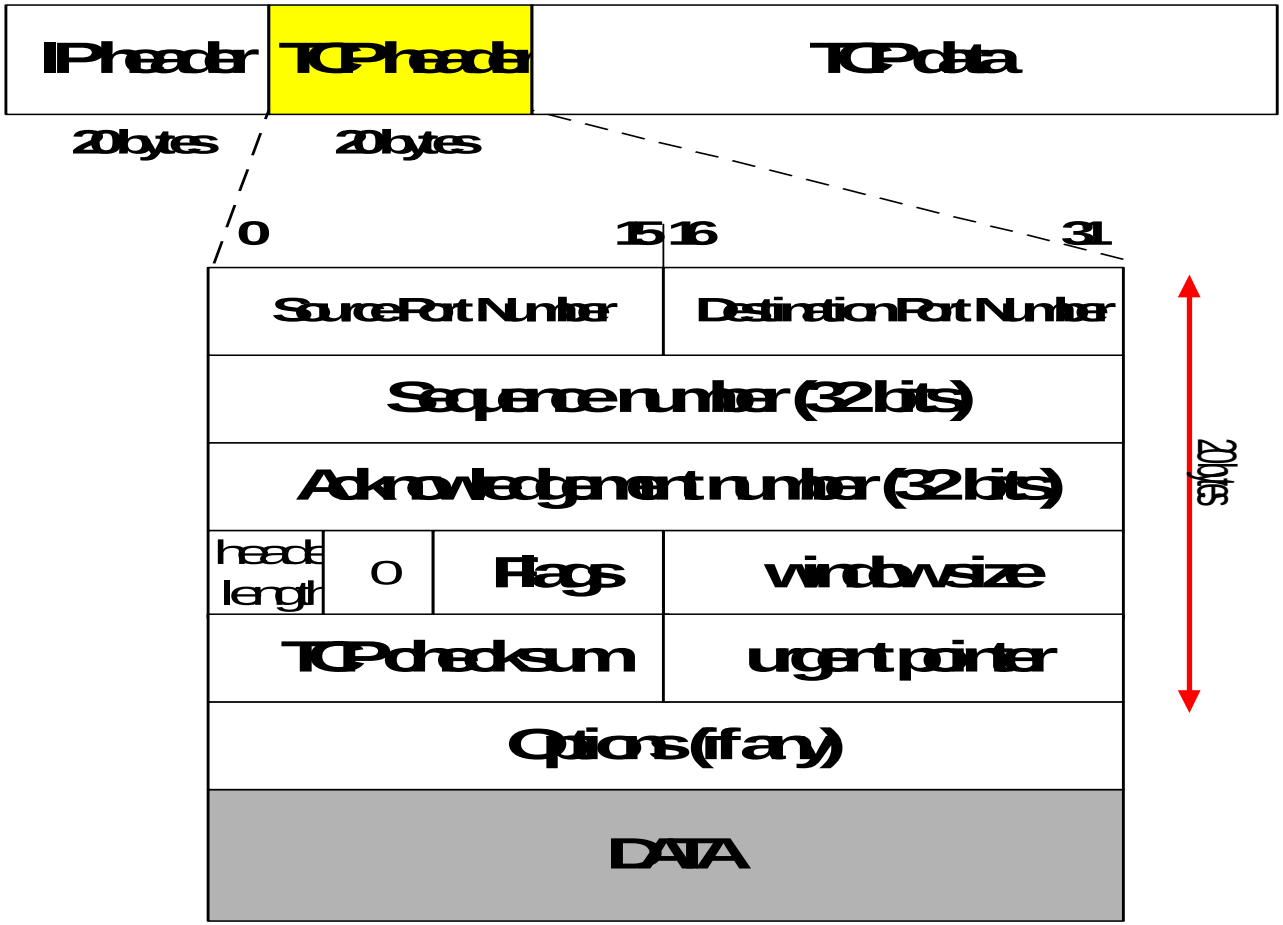


TCP Segment

- IP packet
  - No bigger than Maximum Transmission Unit (MTU)
  - E.g., up to 1500 bytes on an Ethernet
- TCP packet
  - IP packet with a TCP header and data inside
  - TCP header is typically 20 bytes long
- TCP segment
  - No more than Maximum Segment Size (MSS) bytes
  - E.g., up to 1460 consecutive bytes from the stream

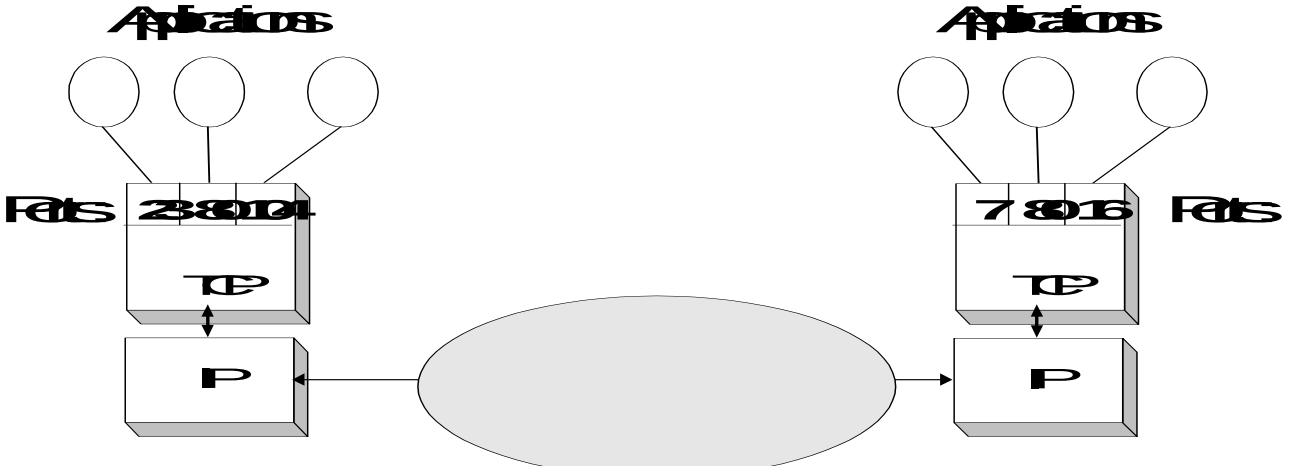
TCP Format

- TCP segments have a 20 byte header with >= 0 bytes of data.



TCP header fields

- **Port Number:**
  - A port number identifies the endpoint of a connection.
  - A pair `<IP address, port number>` identifies one endpoint of a connection.
  - Two pairs `<client IP address, client port number>` and `<server IP address, server port number>` identify a TCP connection.



Port numbers

- 16-bit port numbers- 0 to 65535
  - 0 to 1023 set aside as well-known ports
  - Assigned to certain common applications
  - telnet uses 23, SMTP 25, HTTP 80 etc.
- 1024 to 49151 are registered ports
  - Assigned by IANA for specific service upon application by a requesting entity.
  - On most systems, registered ports can be used without superuser privileges.
    - 6000 through 6063 for X-window server
- 49152 through 65535 are dynamic or private ports, also called ephemeral ports

TCP header fields

- **Sequence Number (SeqNo):**
  - Sequence number is 32 bits long.
  - So the range of SeqNo is

    $0 <= SeqNo <= 2^{32} - 1 \approx 4.3$ Gbyte
  - Each sequence number identifies a byte in the byte stream
  - Initial Sequence Number (ISN) of a connection is set during connection establishment
    - A random number is taken as ISN
    - Avoids the confusion due to data loss during previous connection
    - Also better security is assured
    - Though the time-based random number generation by the OS poses security threats

TCP header fields

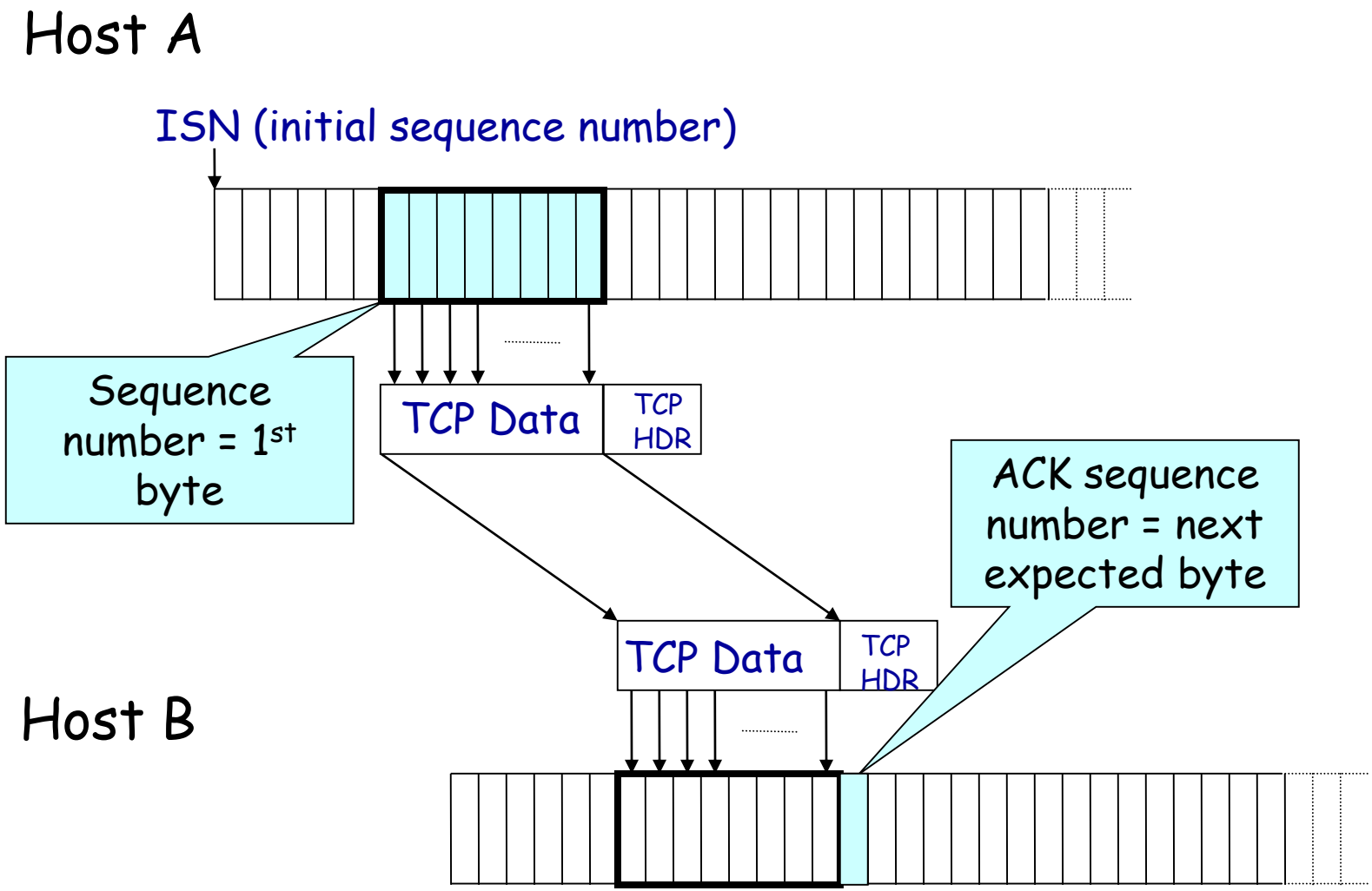
- **Acknowledgement Number (AckNo):**
  - Acknowledgements are piggybacked, I.e

    a segment from A -> B can contain an acknowledgement for a data sent in the B -> A direction

  - A host uses the AckNo field to send acknowledgements. (If a host sends an AckNo in a segment it sets the "**ACK flag**")
  - The AckNo contains the next SeqNo that a host wants to receive

    Example: The acknowledgement for a segment with sequence numbers 0-1500 is AckNo=1501

Sequence Numbers



TCP header fields

- **Acknowledge Number (cont'd)**
  - TCP uses the sliding window flow protocol to regulate the flow of traffic from sender to receiver
  - TCP uses the following variation of sliding window:
    - no NACKs (**N**egative **ACK**nowledgement)
    - only cumulative ACKs
- Example:

  Assume: Sender sends two segments with "1..1500" and "1501..3000", but receiver only gets the second segment.

  In this case, the receiver cannot acknowledge the second packet. It can only send AckNo=1

TCP header fields

- **Header Length (4bits):**
  - Length of header in 32-bit words
  - Note that TCP header has variable length (with minimum 20 bytes)

TCP header fields

- **Flag bits:**
  - **URG: Urgent pointer is valid**
    - If the bit is set, the following bytes contain an urgent message in the range:

      SeqNo <= urgent message <= SeqNo+urgent pointer
  - **ACK: Acknowledgement Number is valid**
  - **PSH: PUSH Flag**
    - Notification from sender to the receiver that the receiver should pass all data that it has to the application.
    - Normally set by sender when the sender's buffer is empty

TCP header fields

- **Flag bits:**
  - **RST: Reset the connection**
    - The flag causes the receiver to reset the connection
    - Receiver of a RST terminates the connection and indicates higher layer application about the reset
  - **SYN: Synchronize sequence numbers**
    - Sent in the first packet when initiating a connection
  - **FIN: Sender is finished with sending**
    - Used for closing a connection
    - Both sides of a connection must send a **FIN**

TCP header fields

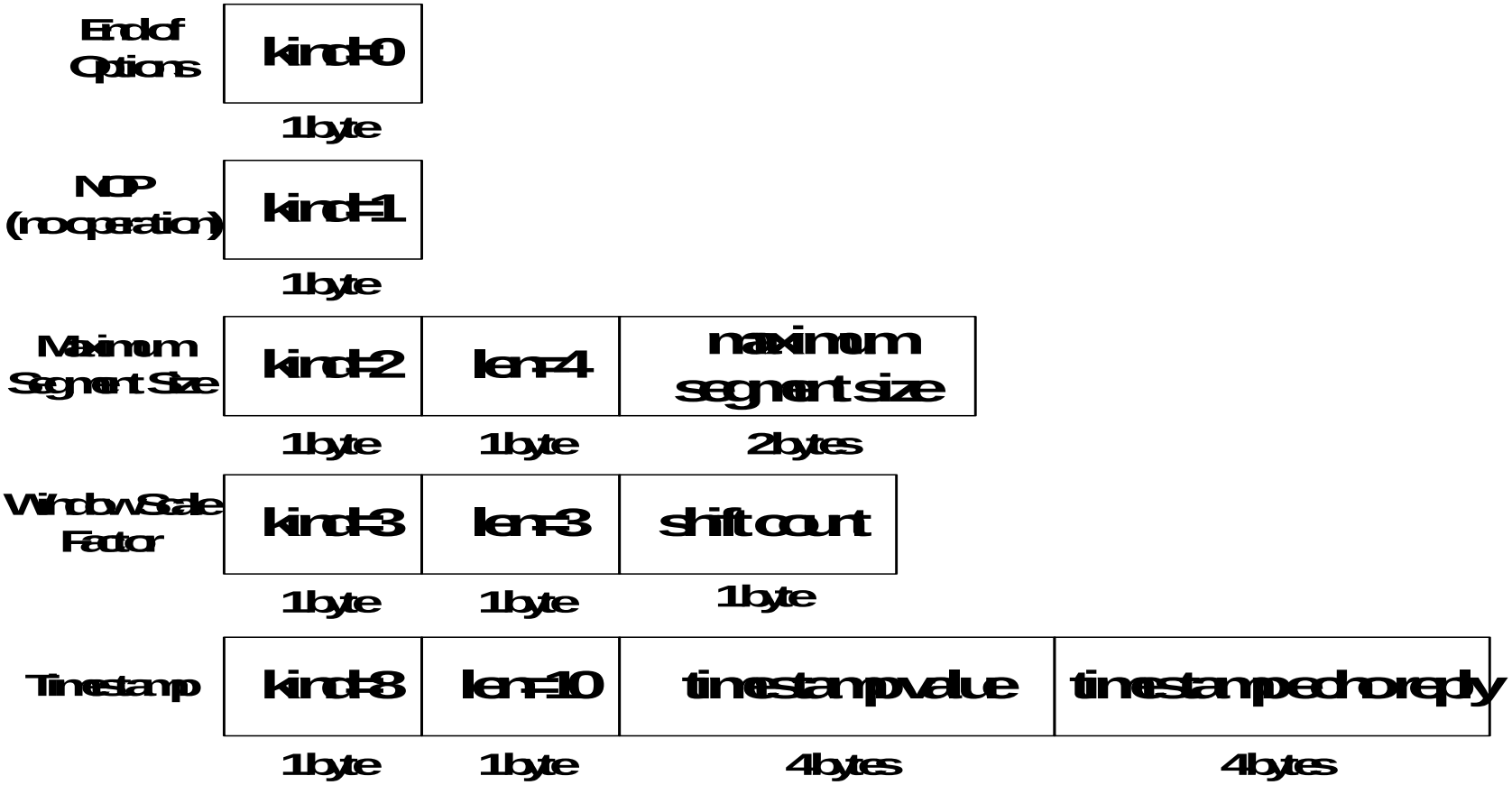
- **Window Size:**
  - Each side of the connection advertises the window size
  - Window size is the maximum number of bytes that a receiver can accept.
  - Maximum window size is 2^{16}-1= 65535 bytes
- **TCP Checksum:**
  - TCP checksum covers over both TCP header **and** TCP data (also covers some parts of the IP header)
- **Urgent Pointer:**
  - Only valid if **URG** flag is set

TCP Options

- Maximum Segment Size (MSS)
- Window scaling
- Selective Acknowledgements (SACK)
- Timestamps
- NOP

TCP header fields

Options: In all cases, len is the total length in bytes of option



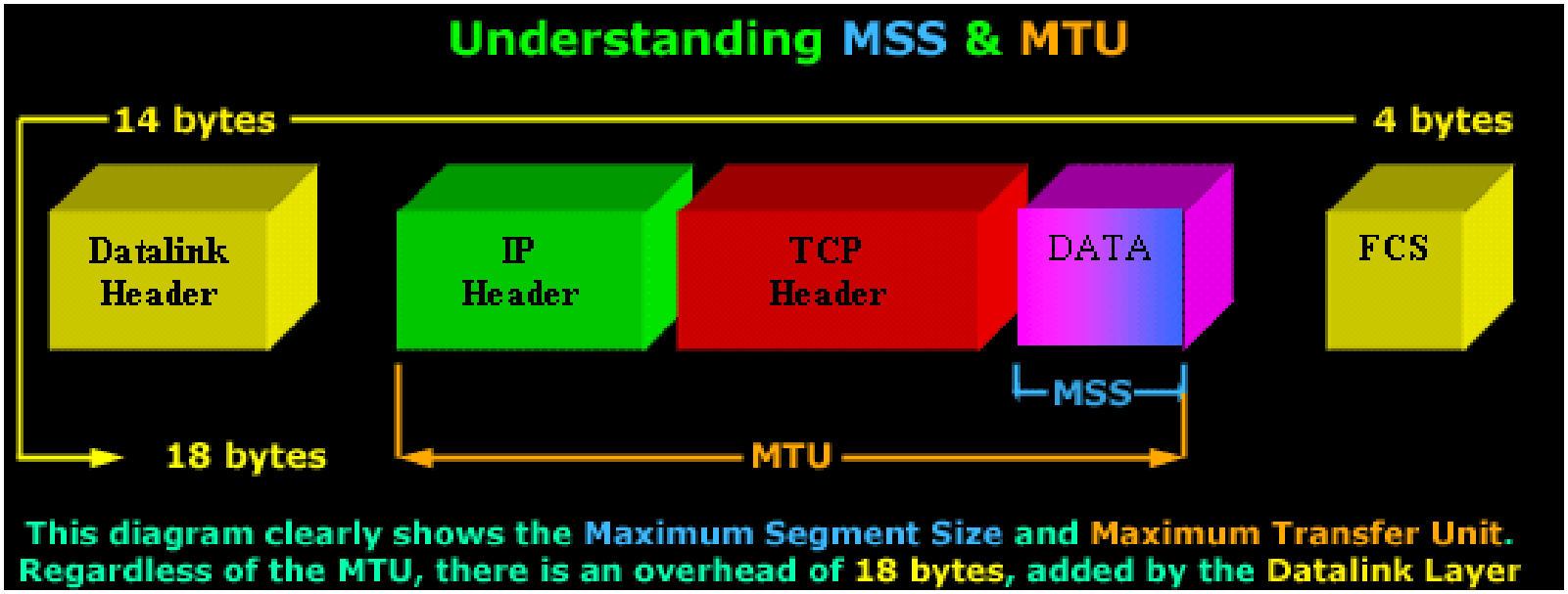
TCP header fields

- **Options**:
  - **NOP** is used to pad TCP header to multiples of 4 bytes
  - **Maximum Segment Size**
    - defined as the largest block of data that a sender using TCP will send to the receiver.
    - when a connection is initiated a SYN segment is sent (discussed later),
    - in the process of sending a SYN segment, the sender has the option of announcing its MSS,
    - TCP assumes a default of 536bytes (minus the 20 byte TCP header) (when a sender doesn't use the options field to declare the MSS)

TCP Options

Maximum Segment Size

If the MSS option is omitted by one or both ends of the connection, then the value of 536 bytes will be used (defined by RFC 1122 and is calculated by taking the default value of an IP Datagram, 576 bytes, minus the standard length of the IP and TCP Header (40 bytes), which gives us 536 bytes).



TCP header fields

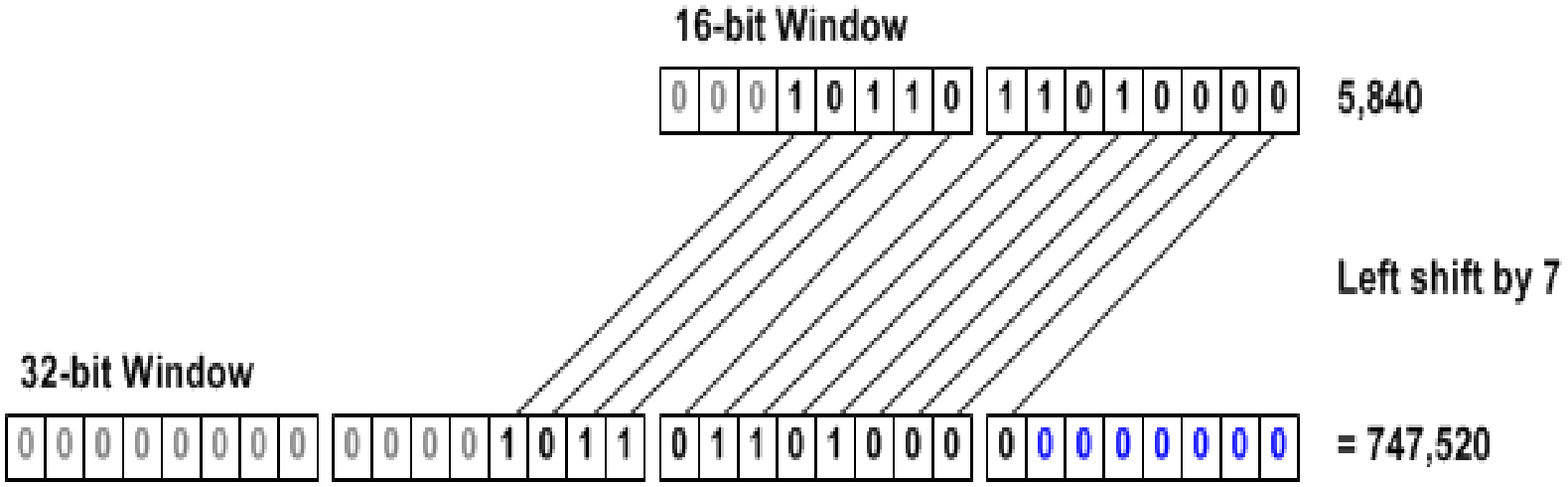
- **Options**:
  - **NOP**
  - **Maximum Segment Size**
  - **Window Scale Options**
    - » Increases the16-bit window field to 32 bits in length, i.e., the window size is interpreted differently
    - » This option can only be used in the SYN segment (first segment) during connection establishment time

TCP Options

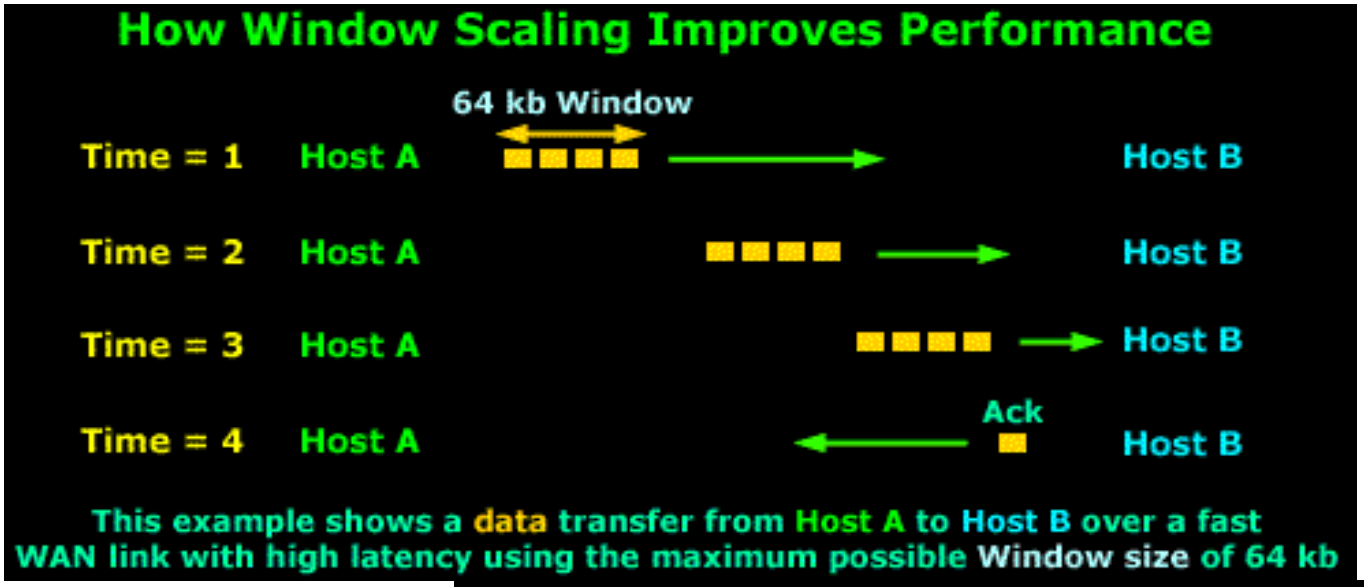
Window Scaling

Defined in RFC 1072 (redefined in 1323), which lets a system advertise 30-bit (16 from the original window + 14 from the TCP Options) Window size values, with a maximum buffer size of 1 GB

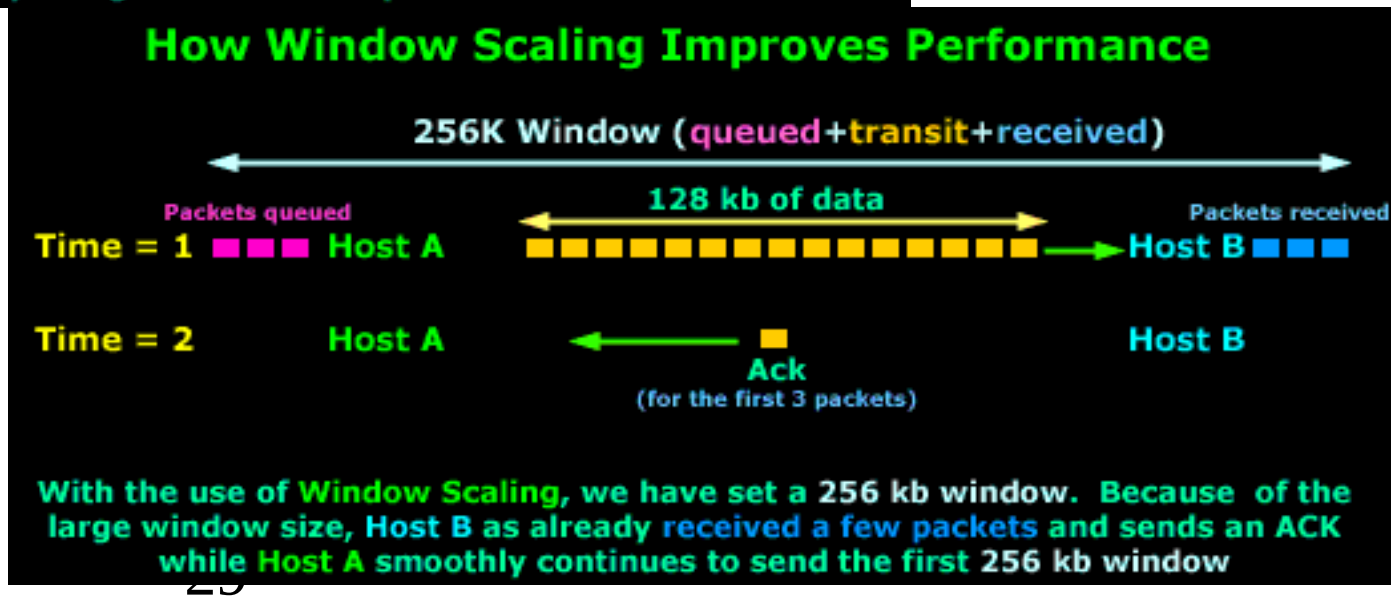
Primarily created for high-latency, high-bandwidth WAN links where a limited Window size can cause severe performance problems.



TCP Options



Without window scaling



With window scaling

TCP Options

Selective Acknowledgments (SACK)

Introduced with RFC 1072

Classical TCP does not allow “selective acknowledgment”

When a TCP connection is established using 3-way-handshake the hosts must send a "Selective Acknowledgments Permitted" in the TCP Options to indicate that they are able to use SACKs

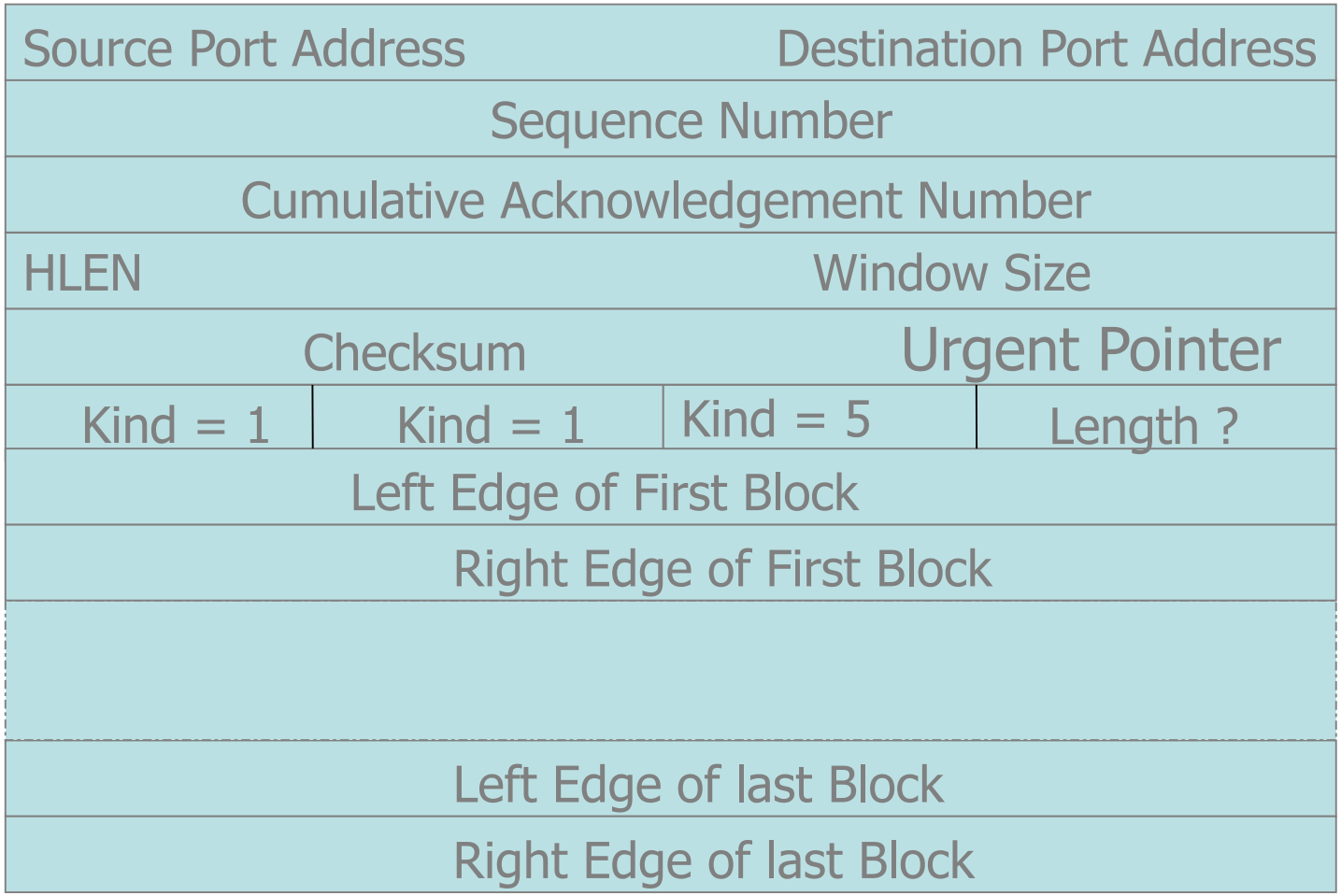
Kind = 5	Length = 2

SACK field in the TCP Options uses two 16 bit fields, to specify the range of bytes it received

In the case where Window Scaling is also used, these fields can be expanded to two 24 or 32 bit fields

Current Implementations : Windows XP/2000/ME/98, Solaris 7 and later, Free BSD & NetBSD have optional modules

SACK Message



TCP Options

Timestamp

Used to calculate *round trip time*

Timestamp option must be sent during the 3-way-handshake in order to enable its use during any subsequent segments

The Timestamp field consists of a Timestamp Echo and Timestamp Reply field

The reply field is always set to zero by the sender and completed by the receiver

NOP

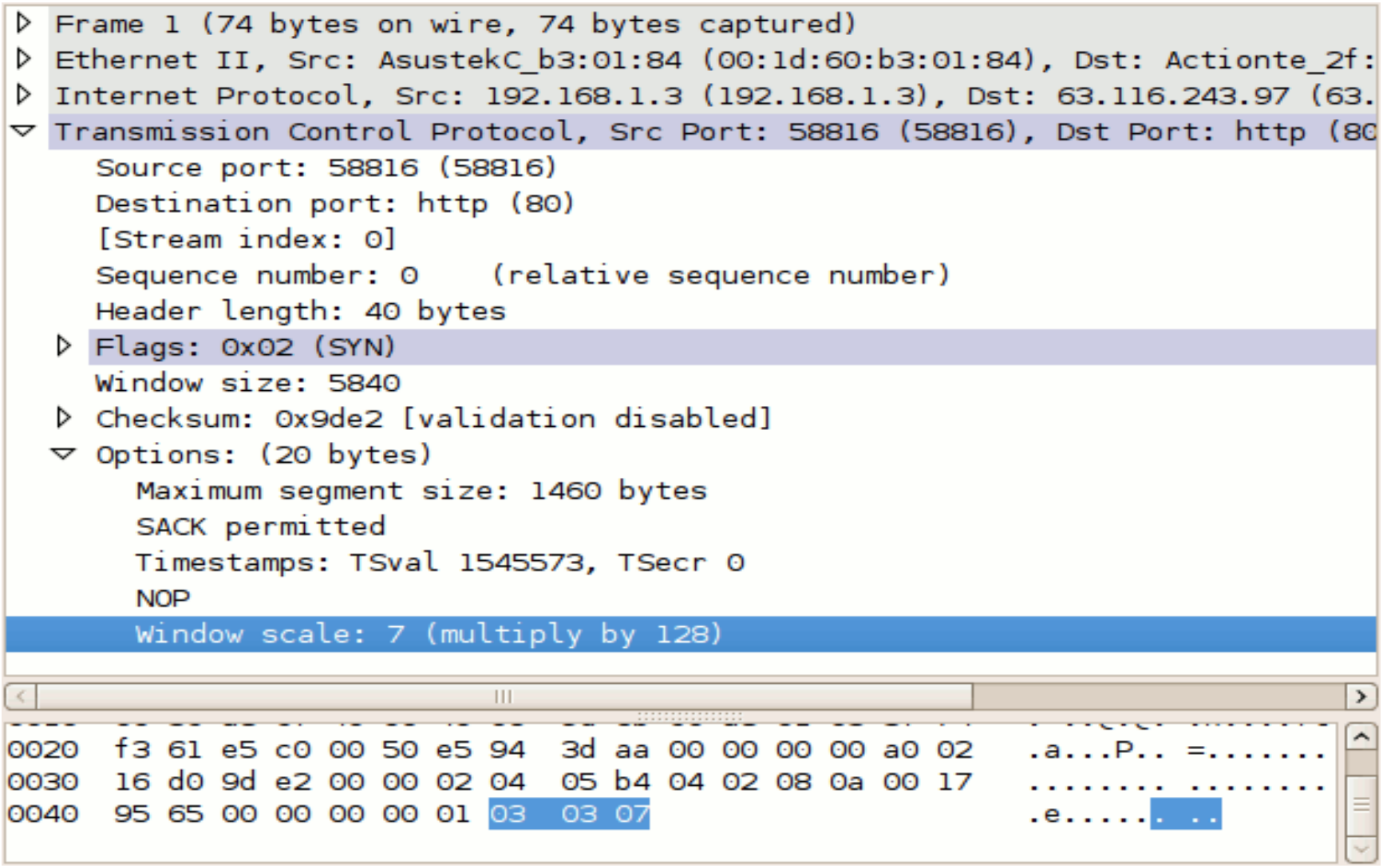
Used to pad TCP header to multiples of 4 bytes and to separate the different options

Its implementation depends on the operating system used

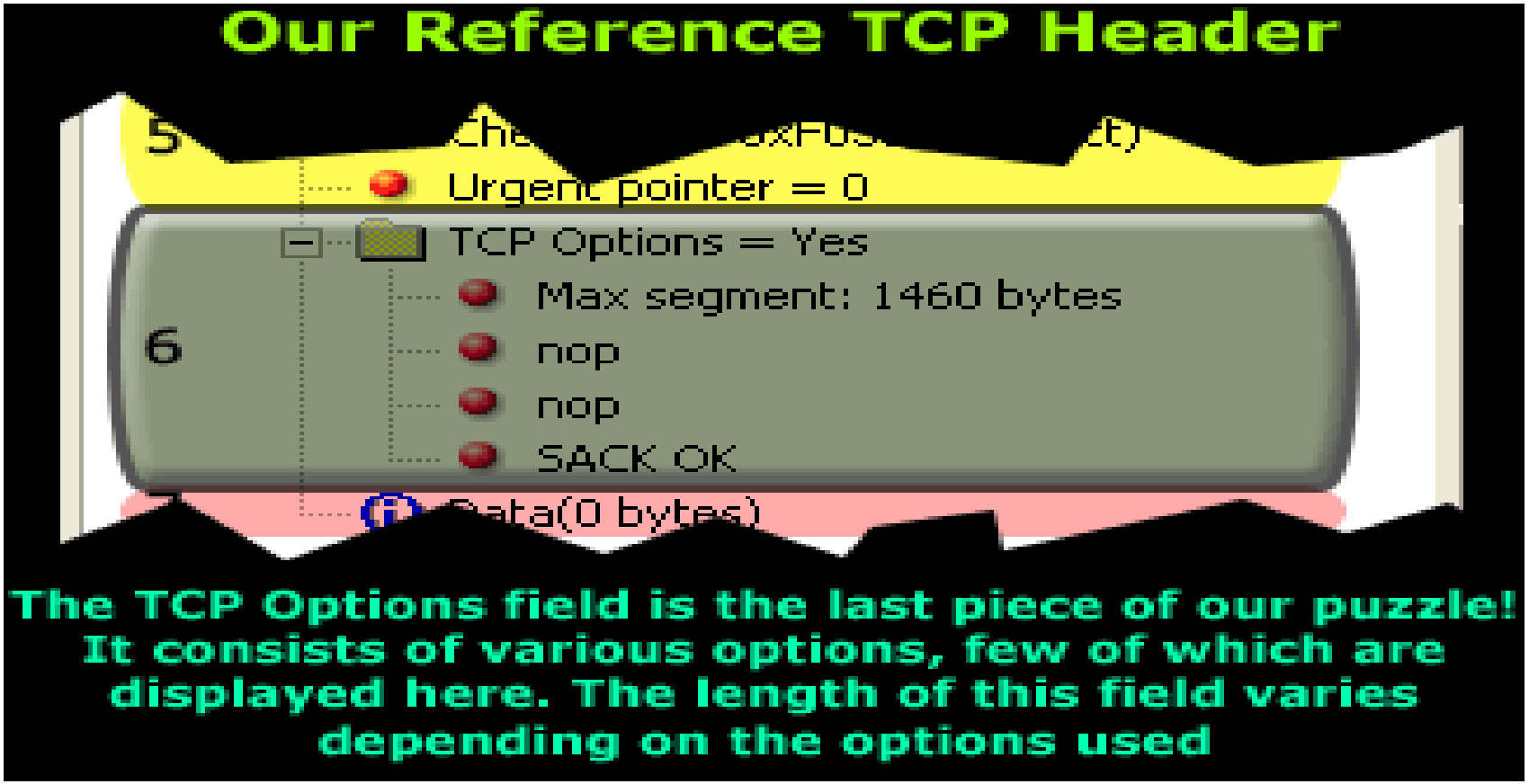
For example, if options MSS and SACK are used, Windows XP will usually place two nops between them

Often checked by hackers when trying to determine the remote host's operating system.

TCP header fields



TCP header fields



Ref: www.firewall.cx

Connection Management in TCP

- **Opening a TCP Connection**
- **Closing a TCP Connection**
- **Special Scenarios**
- **State Diagram**

TCP Connection Establishment

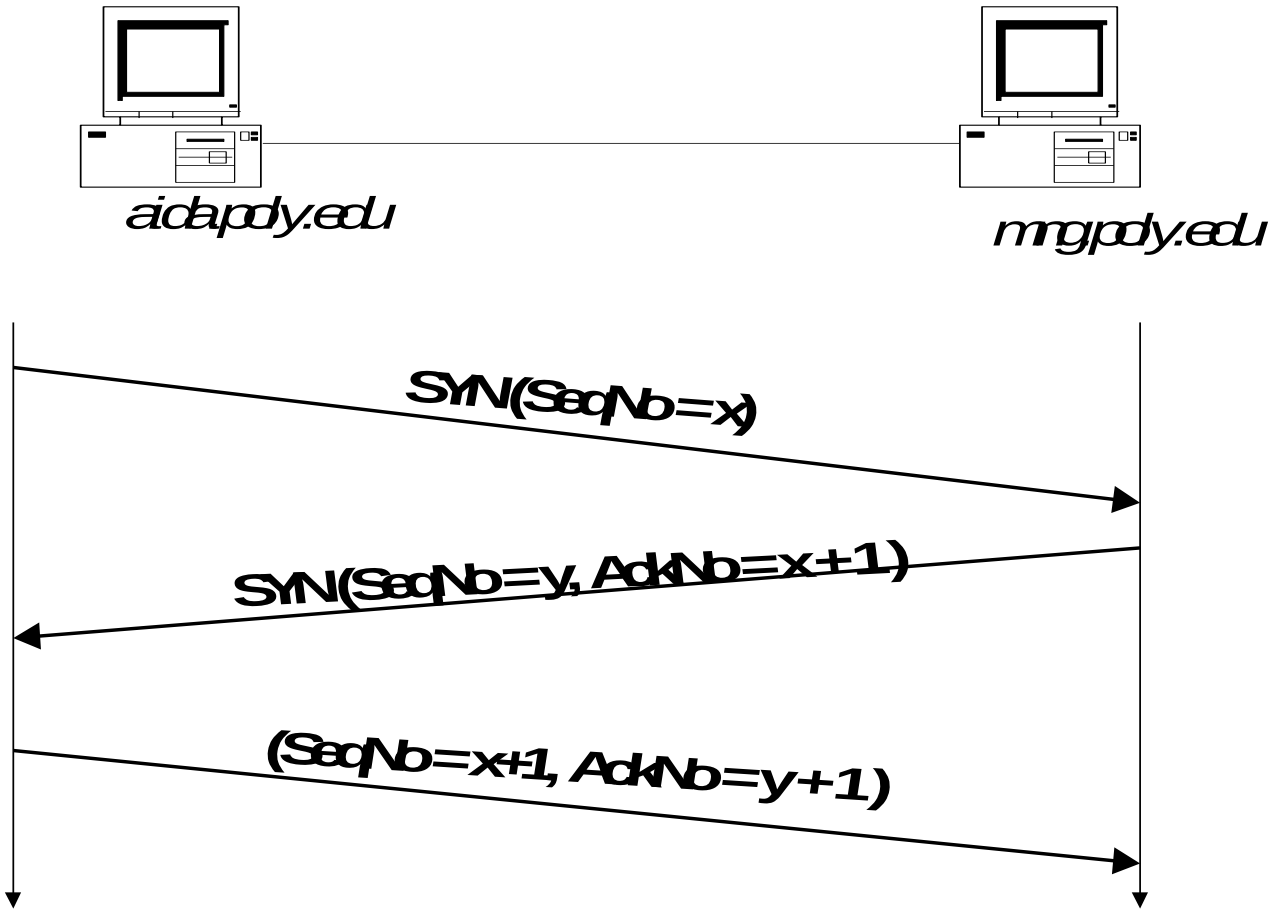
- TCP uses a **three-way handshake** to open a connection:

  (1) ACTIVE OPEN: Client sends a segment with
  - SYN bit set *
  - port number of client
  - initial sequence number (ISN) of client

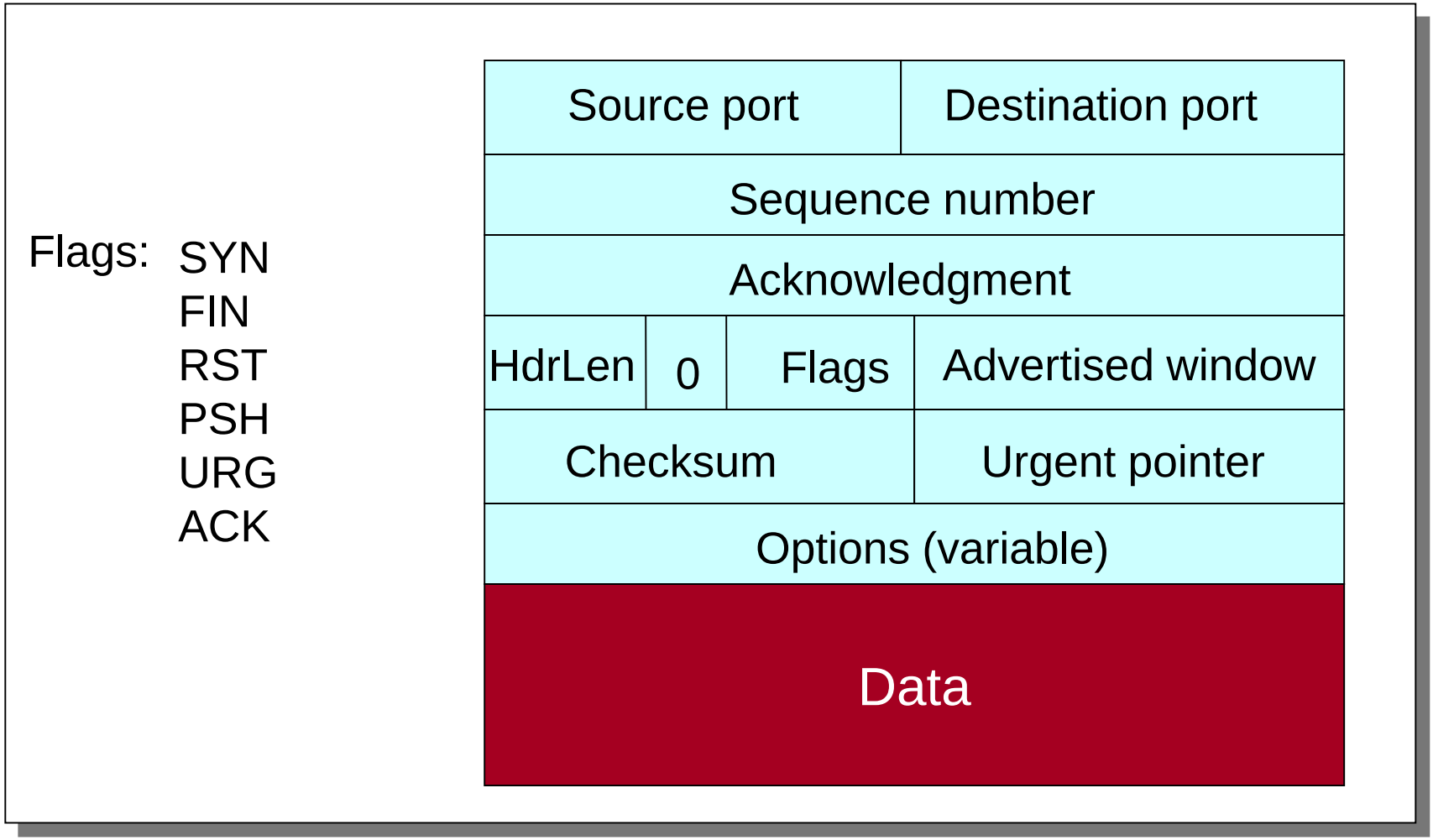
  (2) PASSIVE OPEN: Server responds with a segment with
  - SYN bit set *
  - initial sequence number of server
  - ACK for ISN of client

  (3) Client acknowledges by sending a segment with:
  - ACK ISN of server (* counts as one byte)

Three-Way Handshake

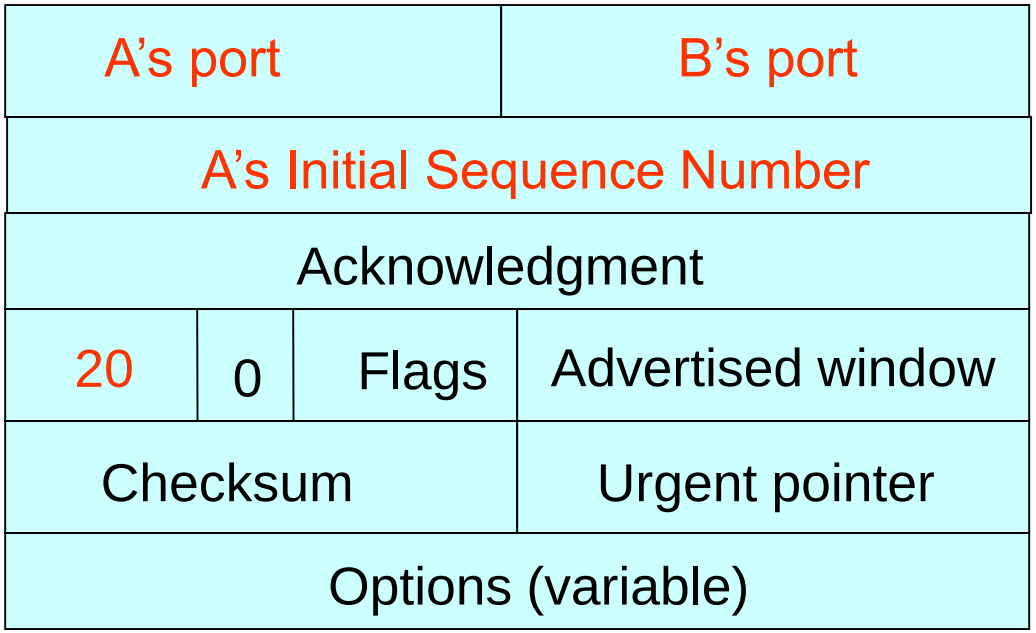


TCP Header


Flags:
SYN
FIN
RST
PSH
URG
ACK
Source port
Destination port
Sequence number
Acknowledgment
HdrLen
0
Flags
Advertised window
Checksum
Urgent pointer
Options (variable)
Data

Step 1: A's Initial SYN Packet

Flags: SYN
FIN
RST
PSH
URG
ACK



A tells B it wants to open a connection…

Step 2: B's SYN-ACK Packet

Flags: SYN
FIN
RST
PSH
URG
ACK

B's port			A's port
B's Initial Sequence Number			
A's ISN plus 1			
20	0	Flags	Advertised window
Checksum			Urgent pointer
Options (variable)			

B tells A it accepts, and is ready to hear the next byte…

… upon receiving this packet, A can start sending data

Step 3: A's ACK of the SYN-ACK

Flags: SYN
FIN
RST
PSH
URG
ACK

A's port			B's port
Sequence number			
B's ISN plus 1			
20	0	Flags	Advertised window
Checksum			Urgent pointer
Options (variable)			

A tells B it is okay to start sending

… upon receiving this packet, B can start sending data

What if the SYN Packet Gets Lost?

- Suppose the SYN packet gets lost
  - Packet is lost inside the network, or
  - Server rejects the packet (e.g., listen queue is full)
- Eventually, no SYN-ACK arrives
  - Sender sets a timer and waits for the SYN-ACK
  - … and retransmits the SYN if needed
- How should the TCP sender set the timer?
  - Sender has no idea how far away the receiver is
  - Hard to guess a reasonable length of time to wait
  - Some TCPs use a default of 3 or 6 seconds

TCP Connection Termination

- Each end of the data flow must be shut down independently **("half-close")**
- If one end is done it sends a FIN segment. This means that no more data will be sent

- Four steps involved:

  (1) X sends a FIN to Y **(active close)**

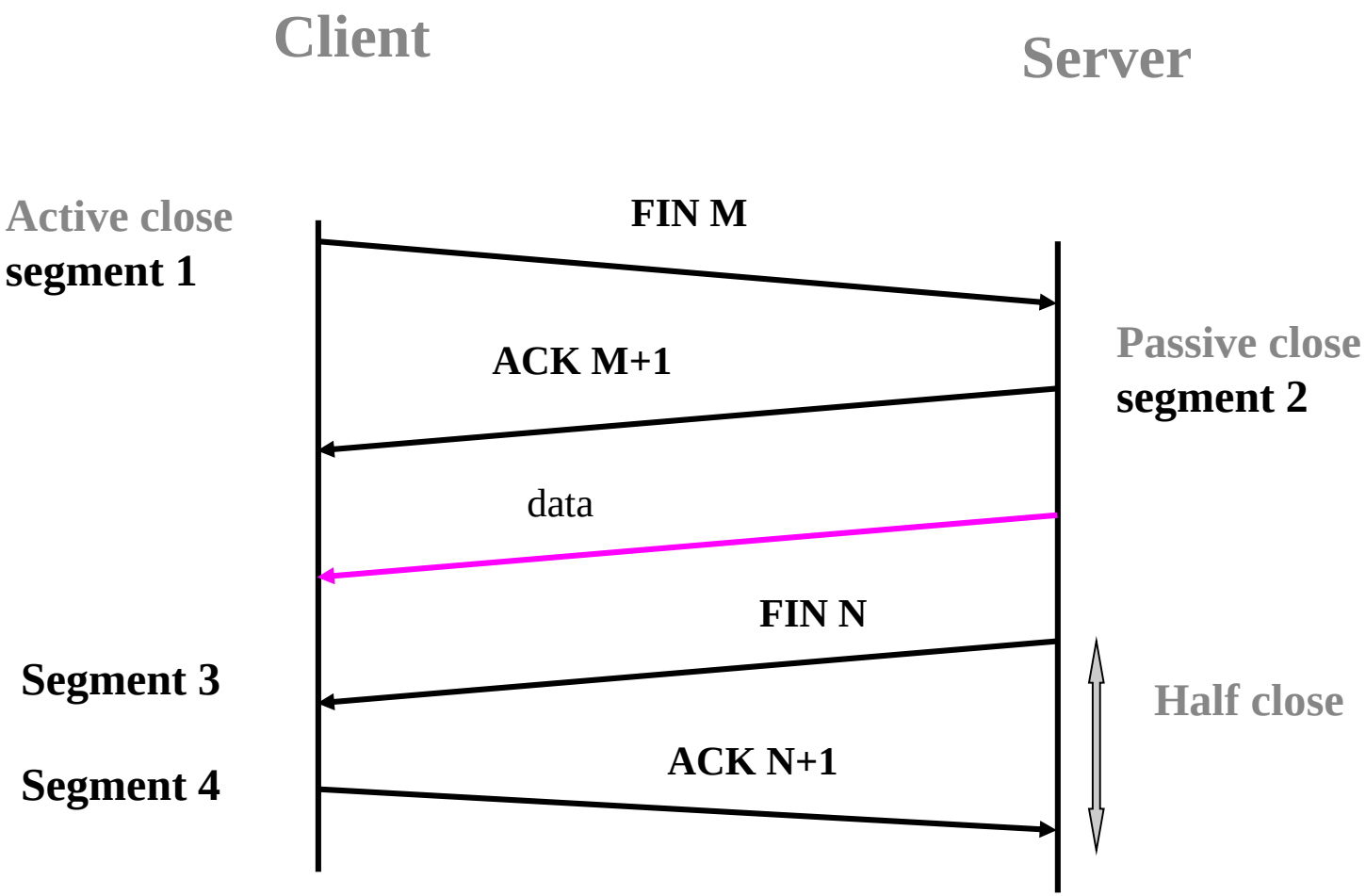
  (2) Y ACKs the FIN,

  (at this time: Y can still send data to X)

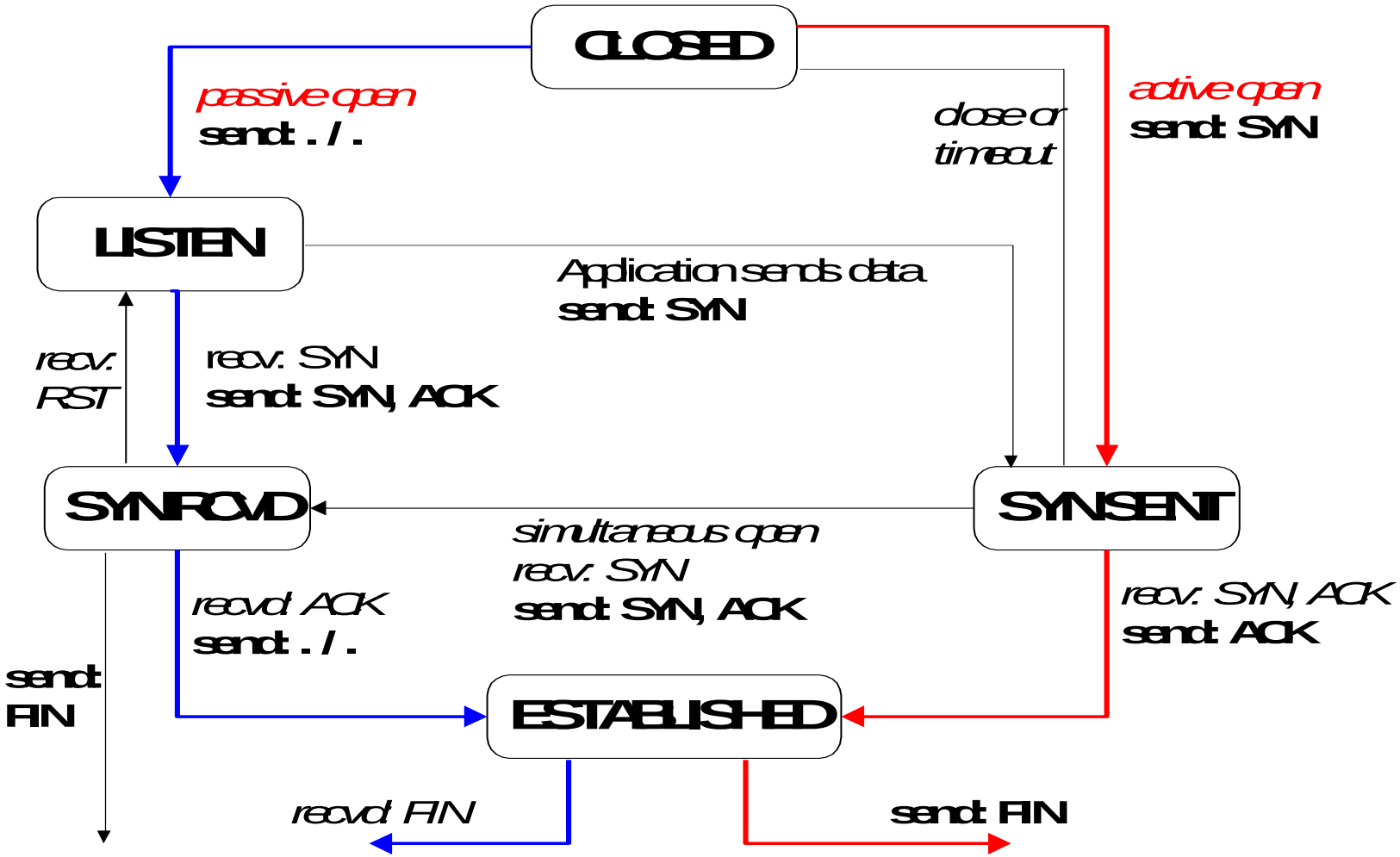
  (3) and Y sends a FIN to X **(passive close)**

  (4) X ACKs the FIN.

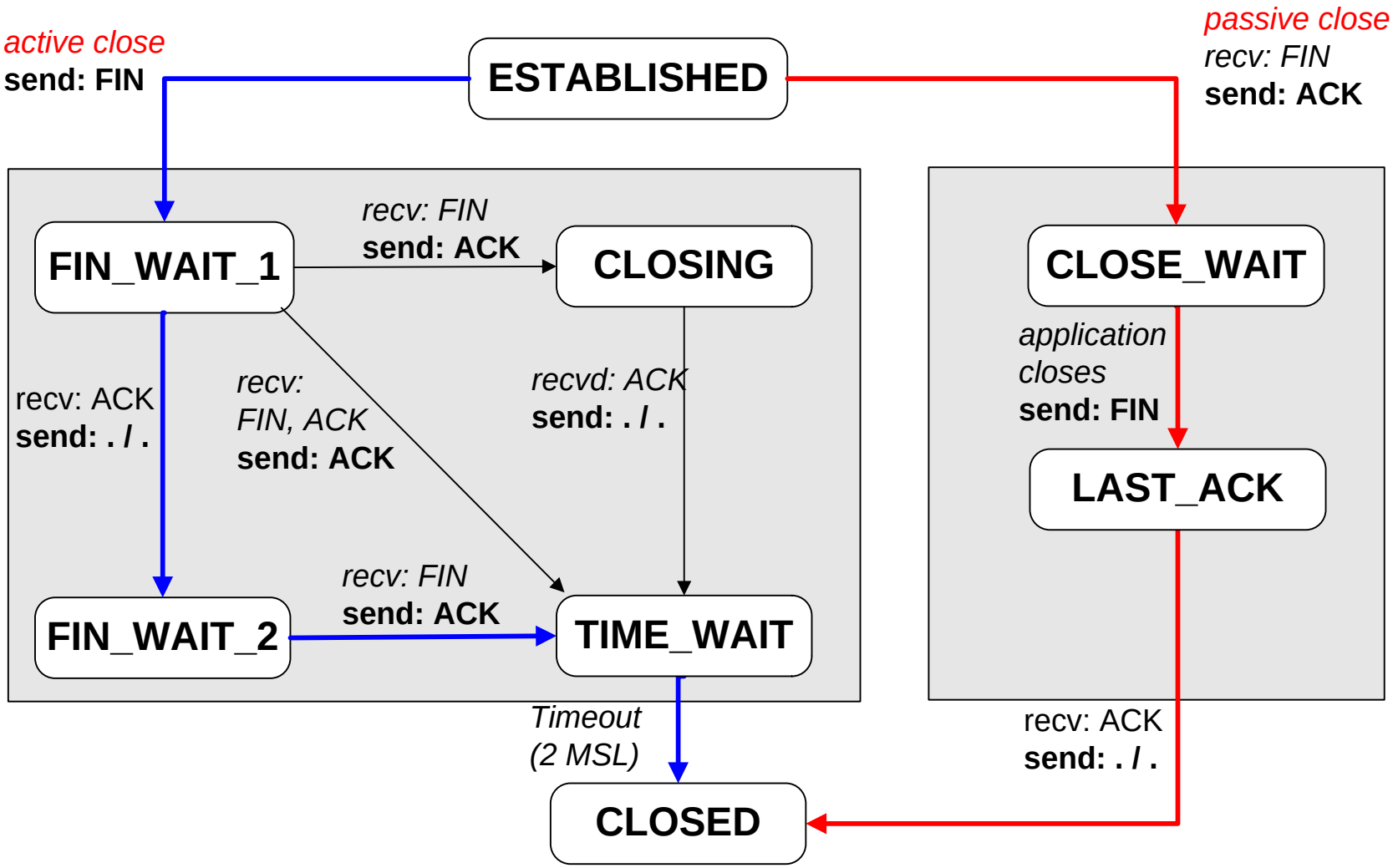
TCP Connection Termination



TCP State Transition Diagram
Opening A Connection



TCP State Transition Diagram
Closing A Connection



2MSL Wait State

2MSL Wait State = TIME_WAIT

- When TCP does an active close, and sends the final ACK, the connection **must stay in the TIME_WAIT state for twice the maximum segment lifetime**.

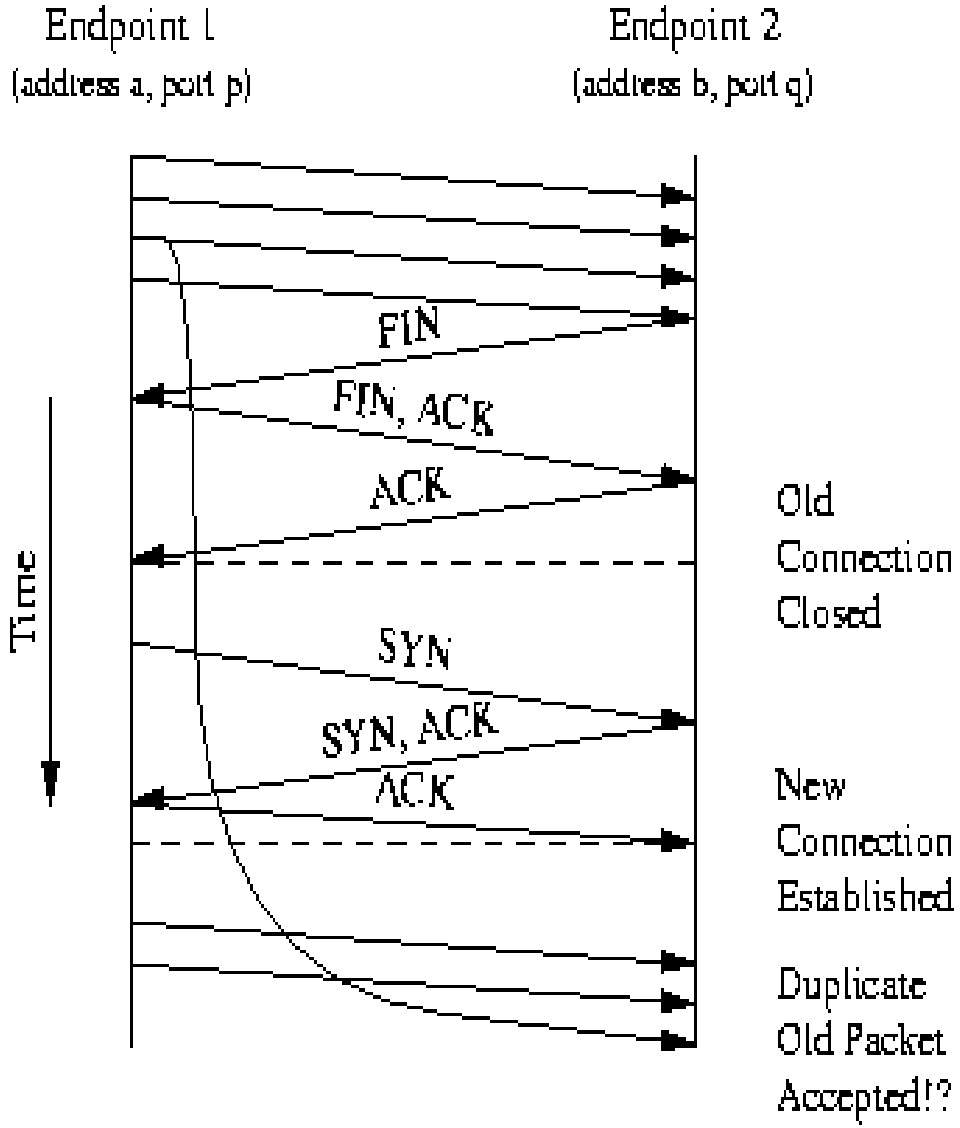
2MSL= 2 * Maximum Segment Lifetime

MSL is the longest period of time that a packet can remain undelivered in the network (an estimate rather than a guarantee)

The MSL is set to 2 minutes or 1 minute or 30 seconds.

- Why 2 MSL?
  TCP requires that the endpoint that initiates an active close of the connection eventually enters TIME-WAIT

2MSL Wait State



Though, the situation is quite unlikely as the address, port number and ISN may not match.

2MSL Wait State

The second reason for the TIME_WAIT state is to implement TCP's full-duplex connection termination reliably.

If the final ACK from end point 2 is dropped then the end point 1 will resend the final FIN.

If the connection had transitioned to CLOSED on end point 2 then the only response possible would be to send an RST

(the retransmitted FIN would be unexpected)

This would cause end point 1 to receive an error even though all data was transmitted correctly

Hence the TIME_WAIT state is introduced to avoid the above error situation

Resetting Connections

- Resetting connections is done by setting the RST flag
- **When is the RST flag set?**
  - Connection request arrives and no server process is waiting on the destination port
  - Abort (Terminate) a connection
    Causes the receiver to throw away buffered data. Receiver does not acknowledge the RST segment

  - 13-09-19 – 01, 43, 57, 06, 47, 76, 31, 74, 34