Computer Networks & IOT (18B11CS311)

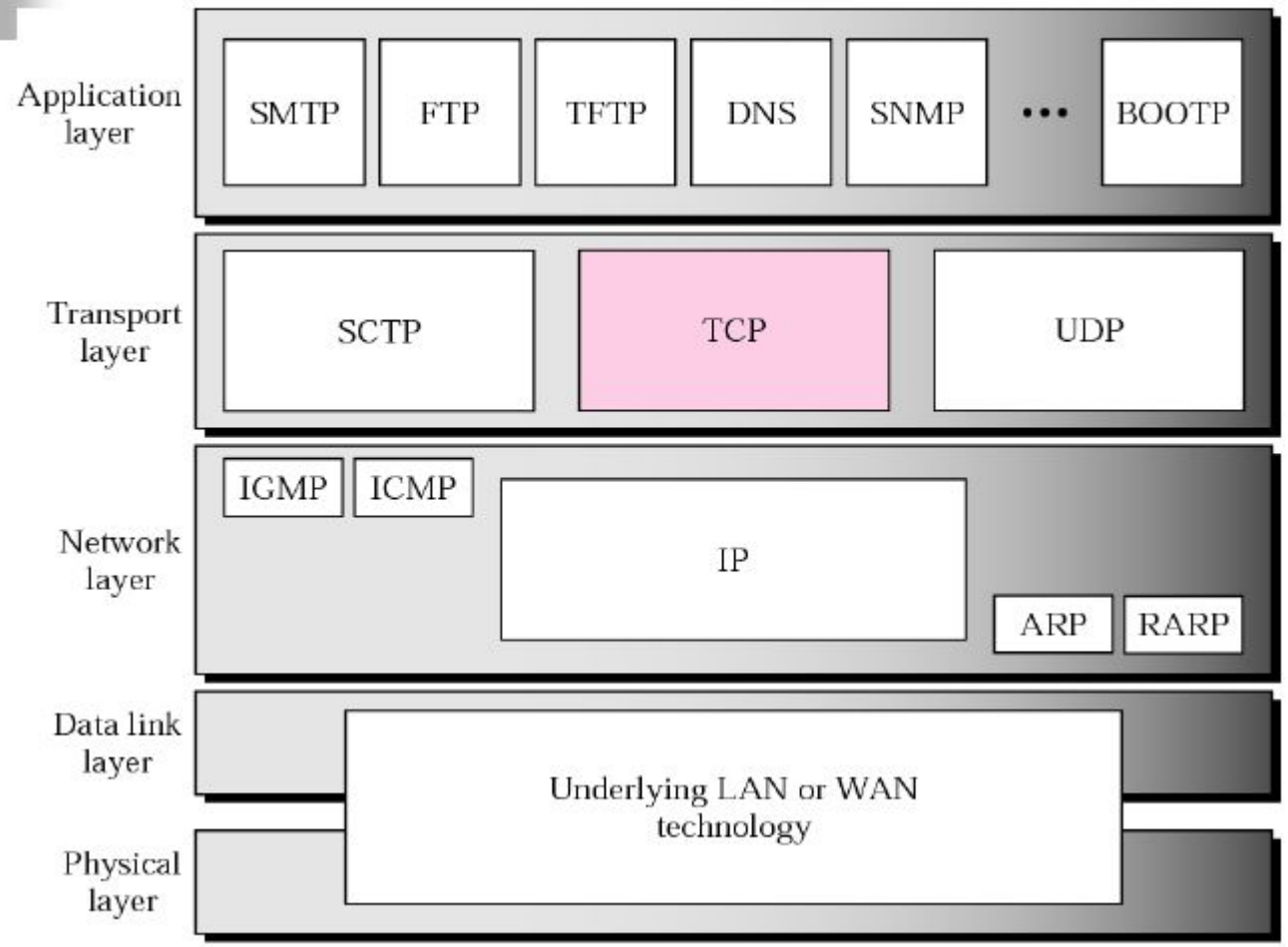
Even Semester_2024

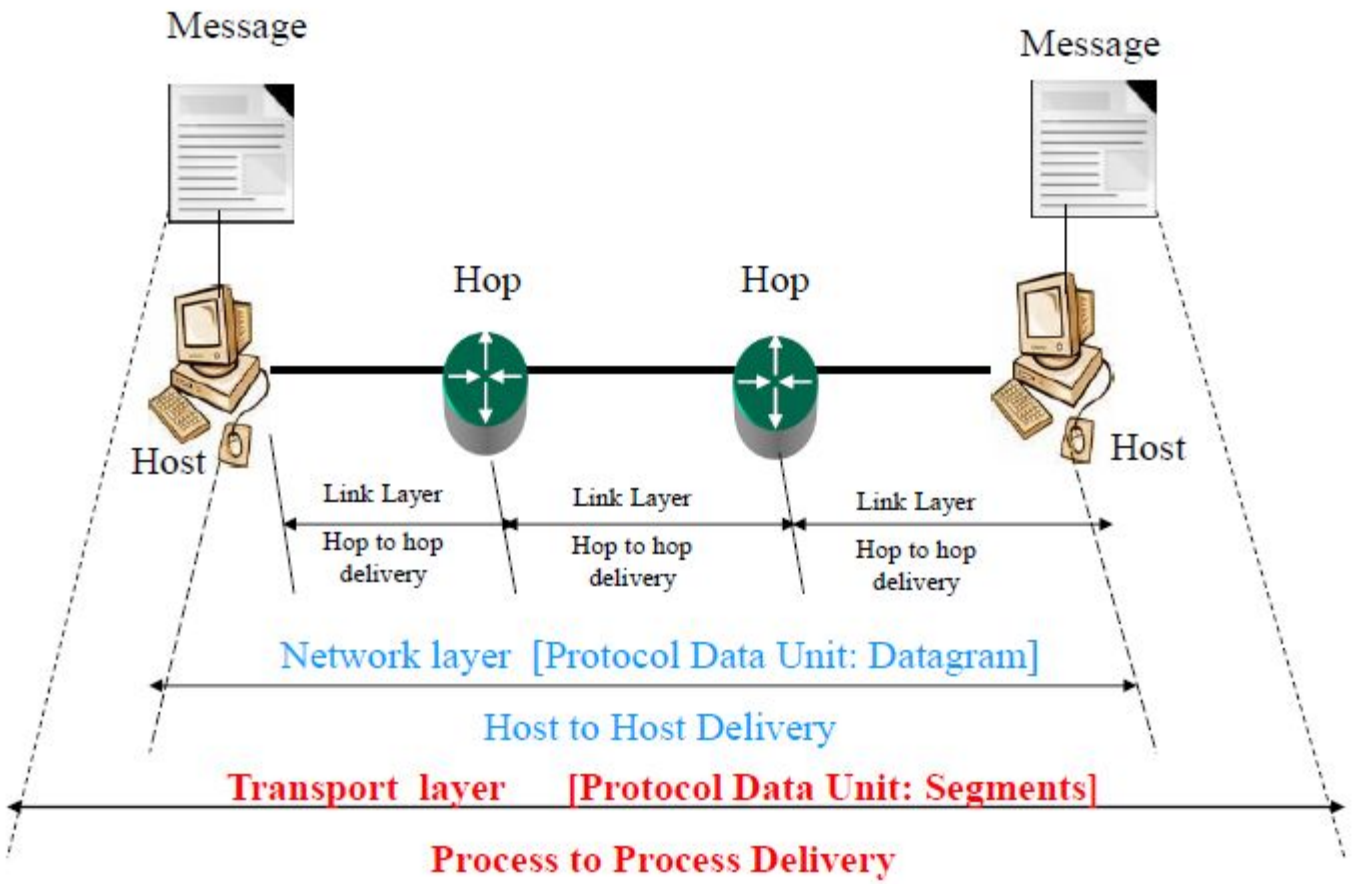
Transport Layer

outline

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- Principles of congestion control
- TCP congestion control

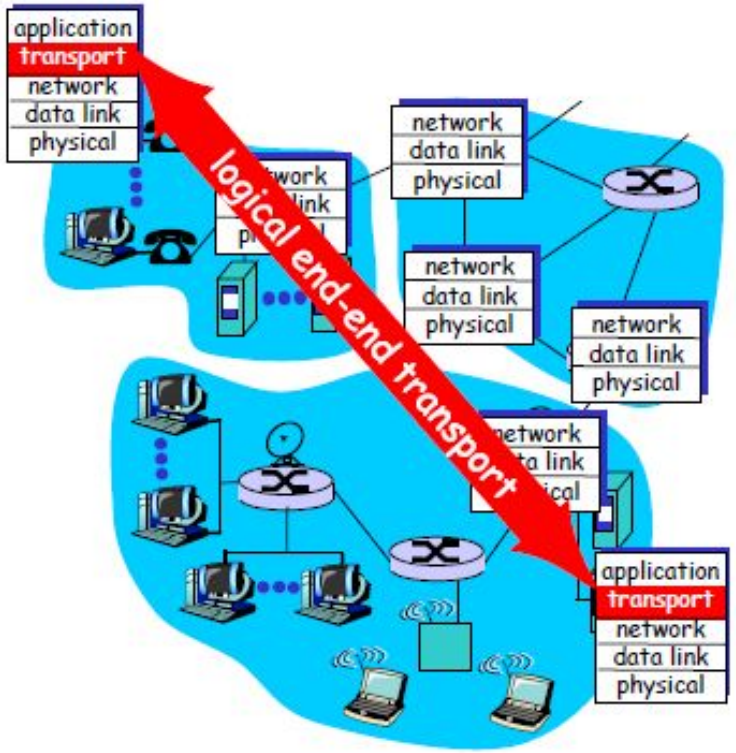
Figure 12.1 *TCP/IP protocol suite*




Message
Message
Hop
Hop
Host
Host
Link Layer
Hop to hop delivery
Link Layer
Hop to hop delivery
Link Layer
Hop to hop delivery
Network layer [Protocol Data Unit: Datagram]
Host to Host Delivery
Transport layer [Protocol Data Unit: Segments]
Process to Process Delivery

Transport services and protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



Transport vs. network layer

- *network layer:* logical communication between hosts
- *transport layer:* logical communication between processes
  - relies on, enhances, network layer services

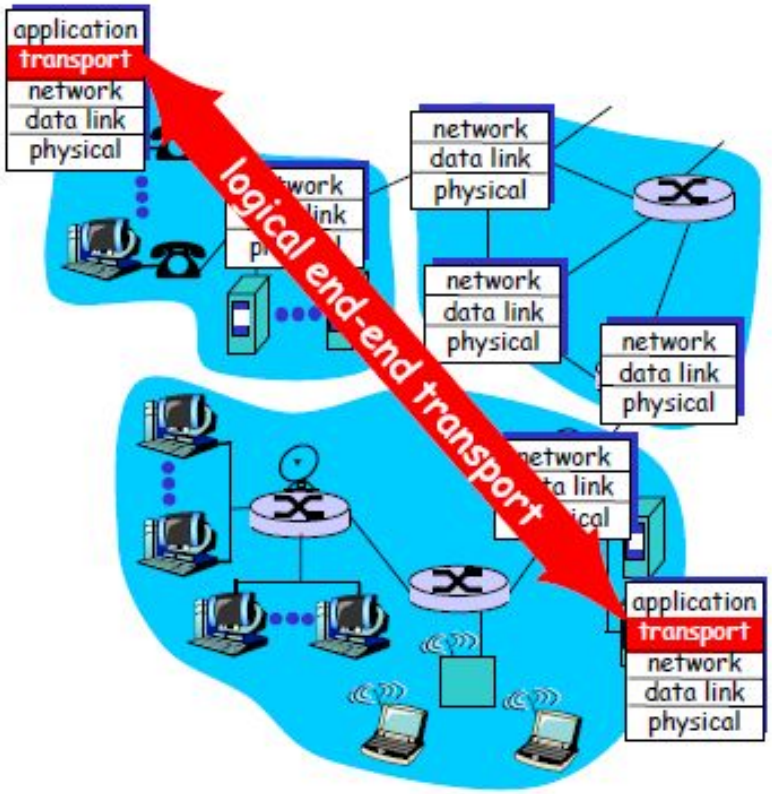
household analogy:

12 kids in Ann's house sending letters to 12 kids in Bill's house:

- hosts = houses
- processes = kids
- app messages = letters in envelopes
- transport protocol = Ann and Bill who demux to in-house siblings
- network-layer protocol = postal service

Internet transport-layer protocols

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - no-frills extension of "best-effort" IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



- Each port no. ranging from 0 to 65535.
- port no. ranging from 0 to 1023 are called well known port numbers.
- port no. ranging from 1024 to 49151 are called Registered port numbers.
- port no. ranging from 49152 to 65535 are called Ephemeral (dynamic) port numbers.

- The ports 0 to 1023 are called well-known ports or system ports, these ports are especially associated with particular services.
- The ports from 1024 to 49151 are called registered ports and this range port can be registered with the Internet Assigned Numbers Authority for a specific use.
- The ports from 49152 to 65535 are unassigned ports, called dynamic or ephemeral ports, and can be utilized for any type of service.
- **IANA (Internet Assigned Numbers Authority) maintains the official list of well-known and registered ranges.**

Well-known ports used by TCP

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FTP, Data	File Transfer Protocol (data connection)
21	FTP, Control	File Transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

Well-known ports used by UDP

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

Multiplexing/demultiplexing

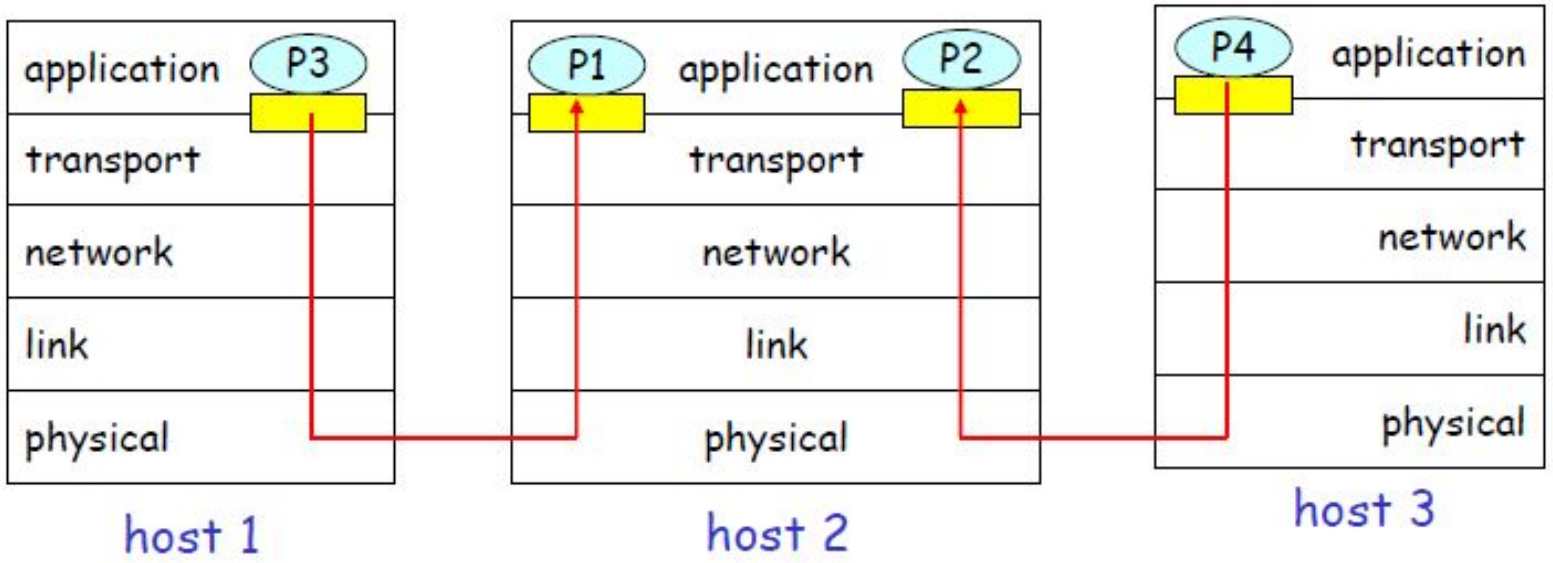
Demultiplexing at rcv host:

delivering received segments to correct socket

Multiplexing at send host:

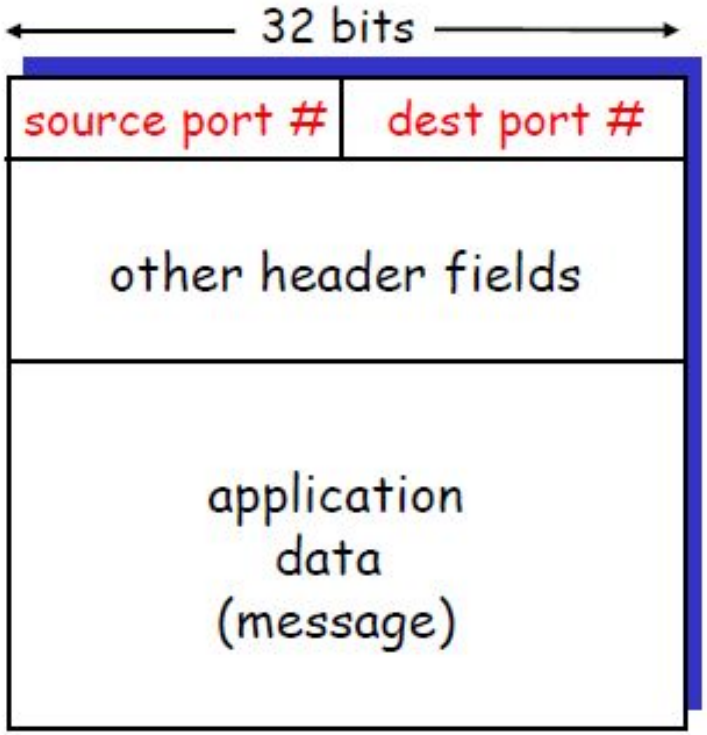
gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)

= socket = process



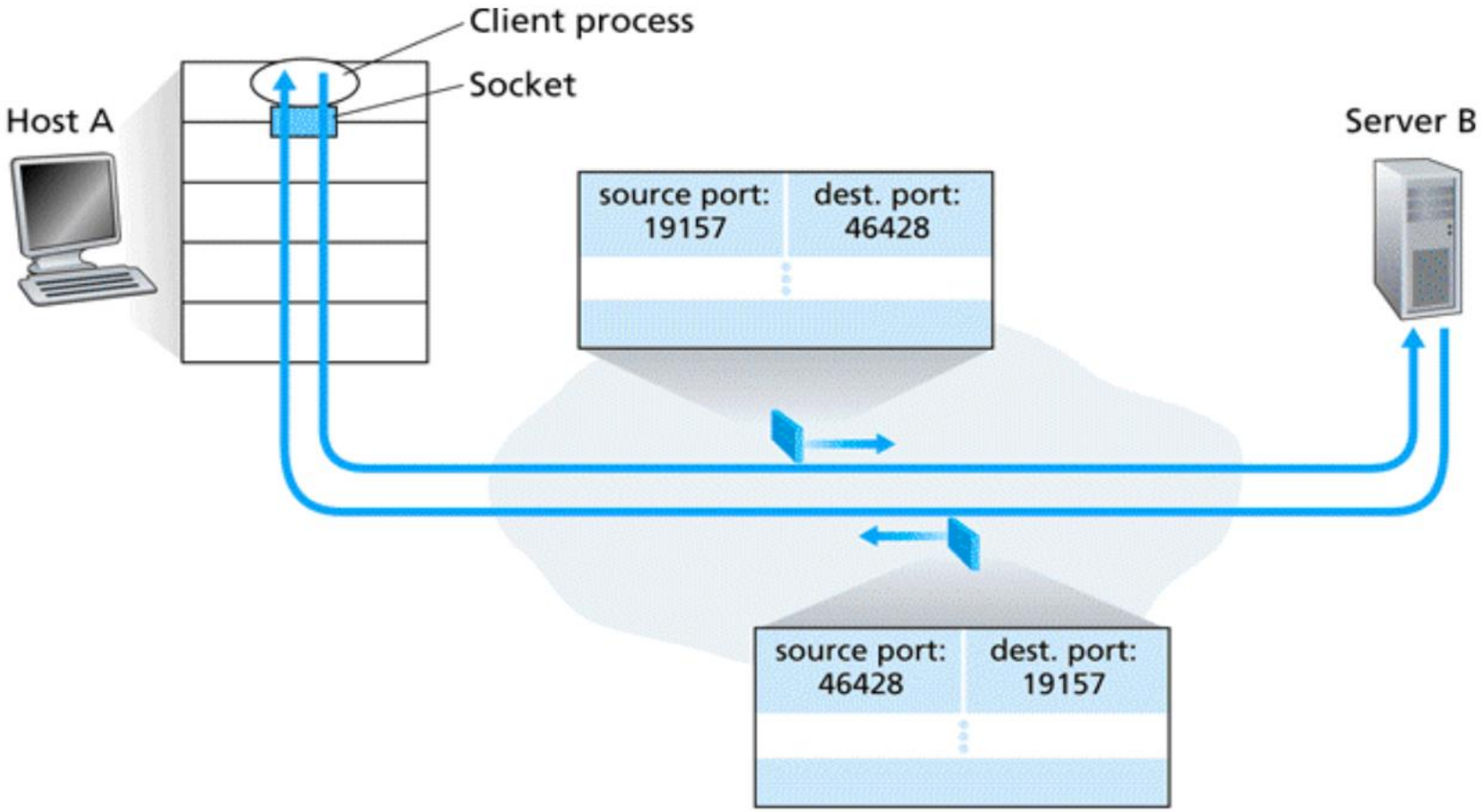
How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries 1 transport-layer segment
  - each segment has source, destination port number (recall: well-known port numbers for specific applications)
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP segment format

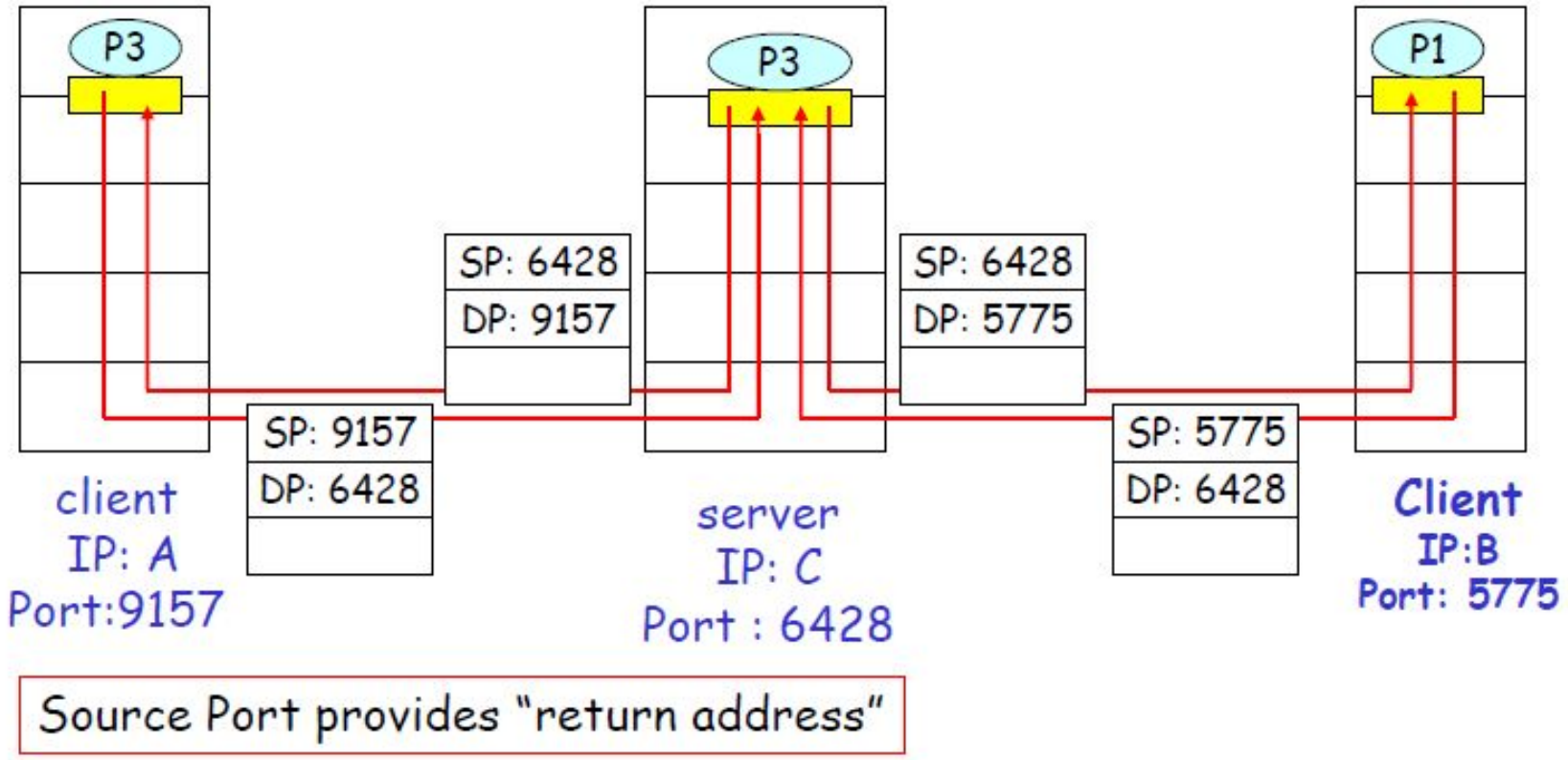
Inversion of source and destination port numbers



- say a process on Host A, with port number 19157, wants to send data to a process with UDP port 46428 on Host B
- transport layer in Host A creates a segment containing source port, destination port, and data
- passes it to the network layer in Host A
- transport layer in Host B examines destination port number and delivers segment to socket identified by port 46428
- **note: a UDP socket is fully identified by a two-tuple consisting of**
  - **a destination IP address**
  - **a destination port number**
- source port number from Host A is used at Host B **as "return address":**
-

Connectionless demux (cont)

```
DatagramSocket serverSocket = new DatagramSocket(6428);
```

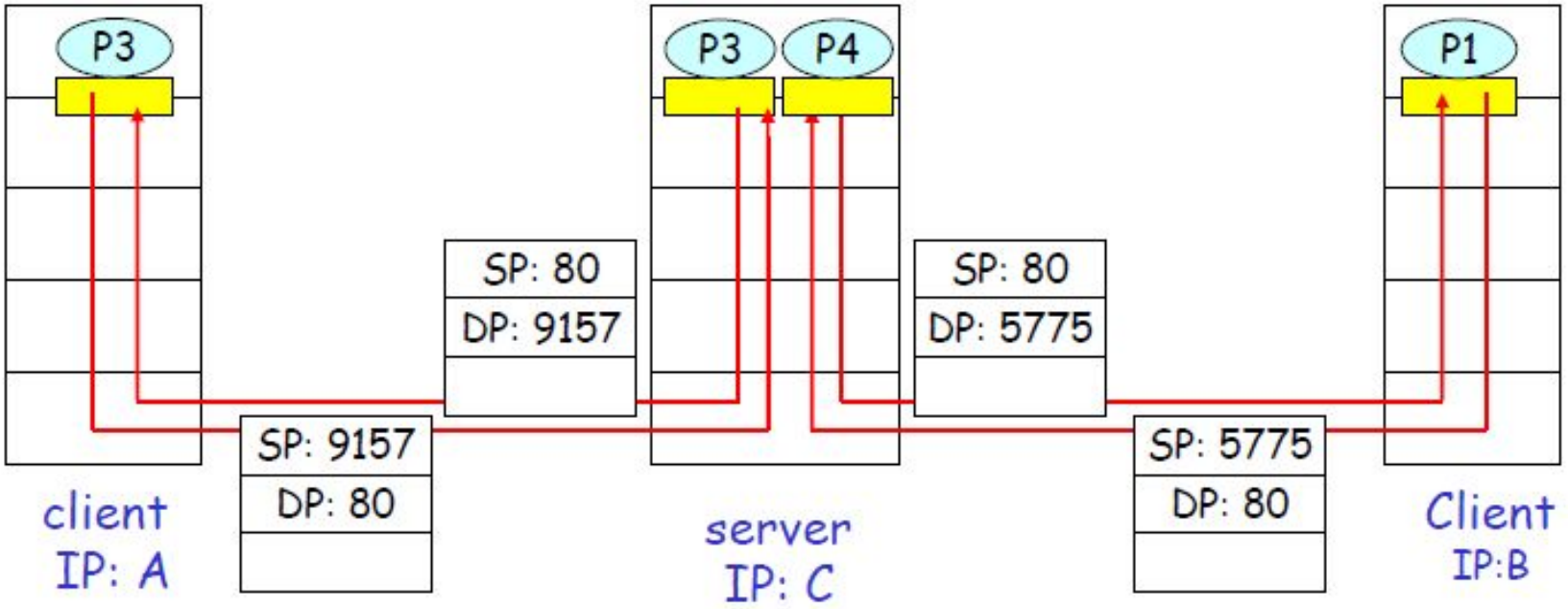


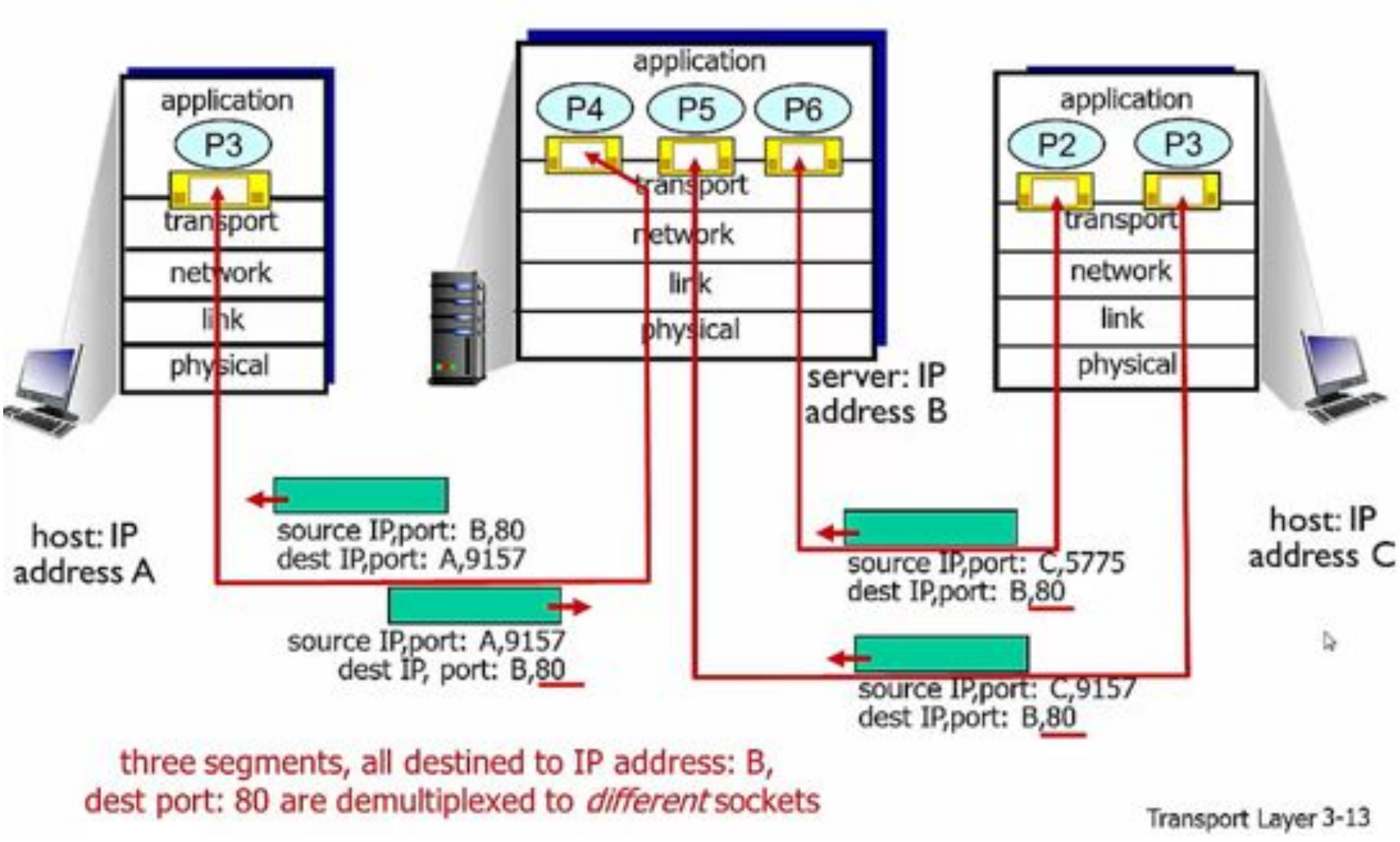
Source Port provides "return address"

Connection-oriented demux

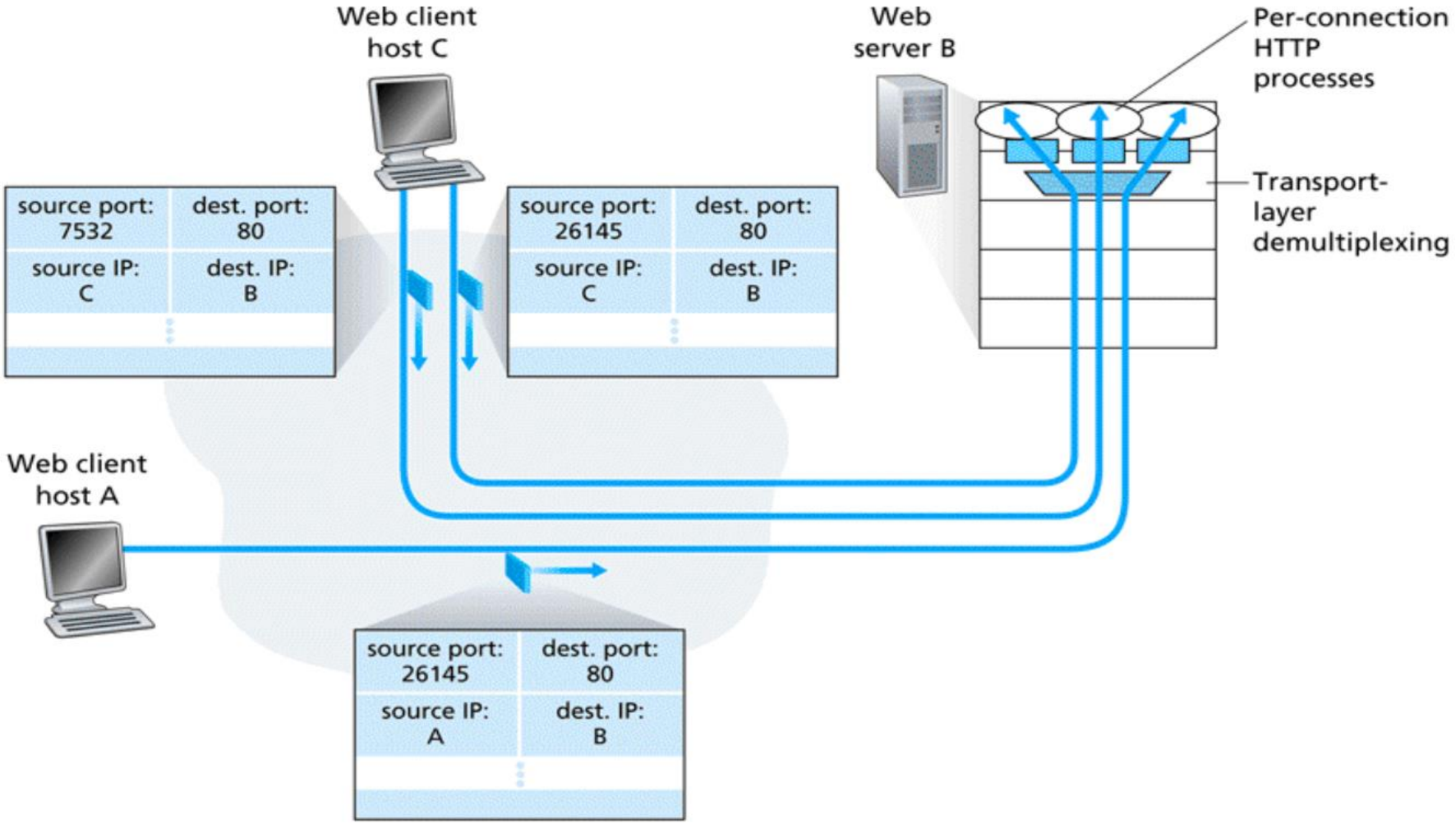
- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- recv host uses all four values to direct segment to appropriate socket
- Server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- Web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

Connection-oriented demux (cont)




application
P3
transport
network
link
physical
application
P4
P5
P6
transport
network
link
physical
server: IP address B
application
P2
P3
transport
network
link
physical
host: IP address A
source IP,port: B,80
dest IP,port: A,9157
source IP,port: A,9157
dest IP, port: B,80
source IP,port: C,5775
dest IP,port: B,80
source IP,port: C,9157
dest IP,port: B,80
host: IP address C
three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets
Transport Layer 3-13

2 clients using the same port (80) – web server application



UDP
User Datagram Protocol

UDP: User Datagram Protocol

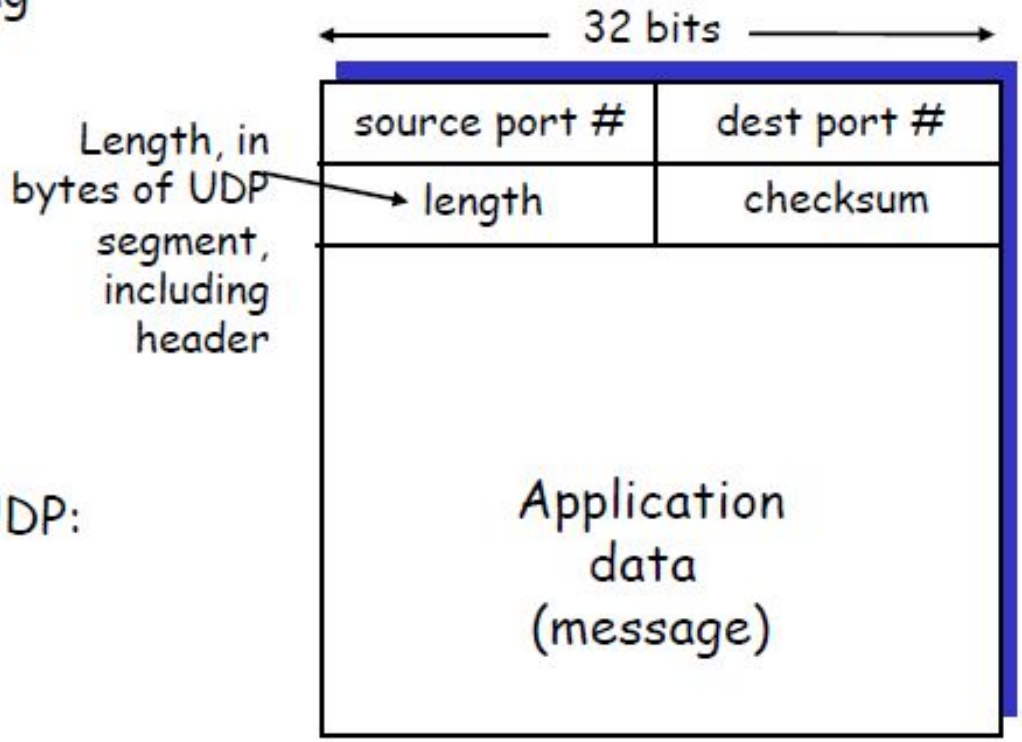
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- connectionless:
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

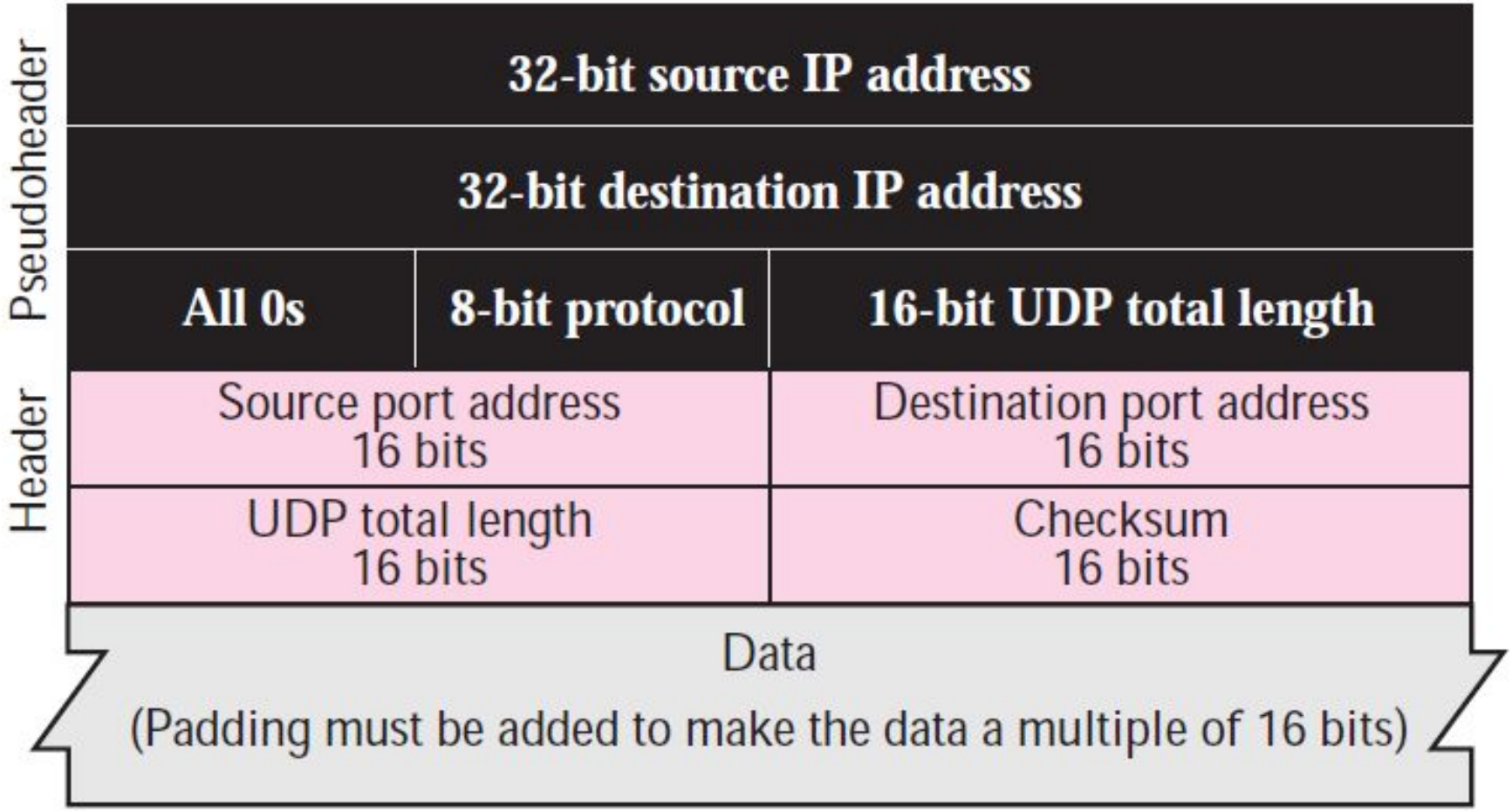
UDP: more

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!



UDP segment format

UDP segment format



UDP checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

Sender:

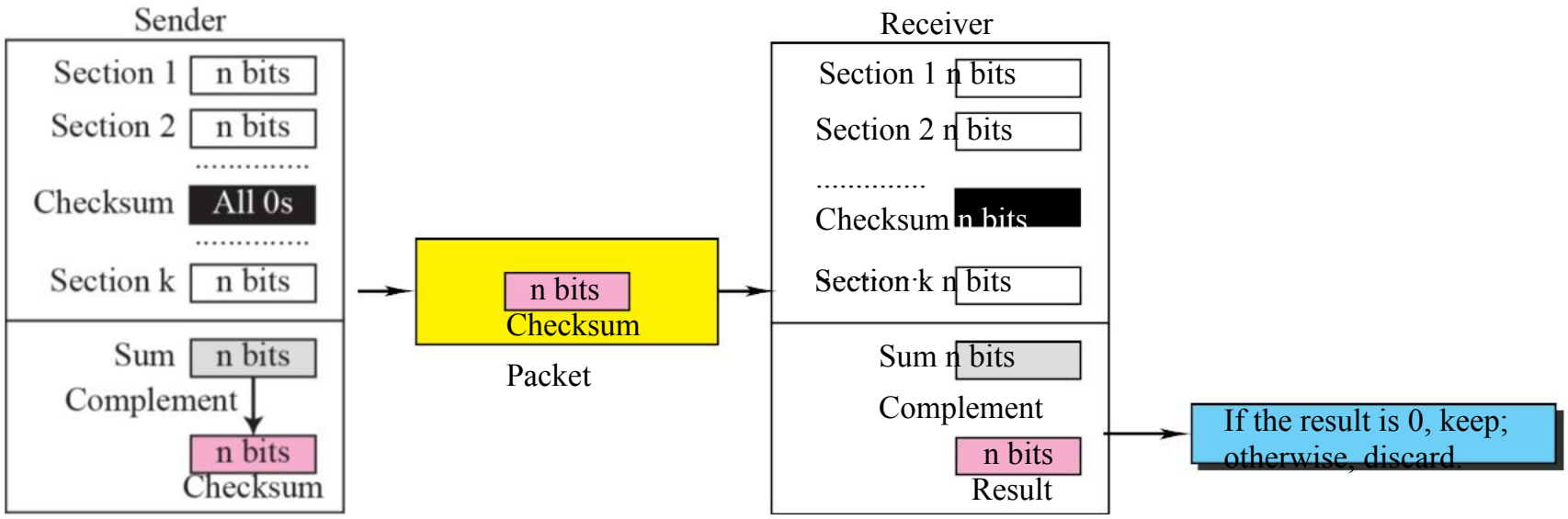
- treat segment contents as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:

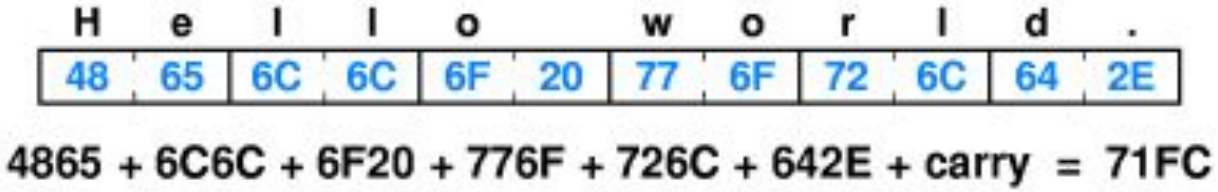
- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?* More later

....

Checksum Concept



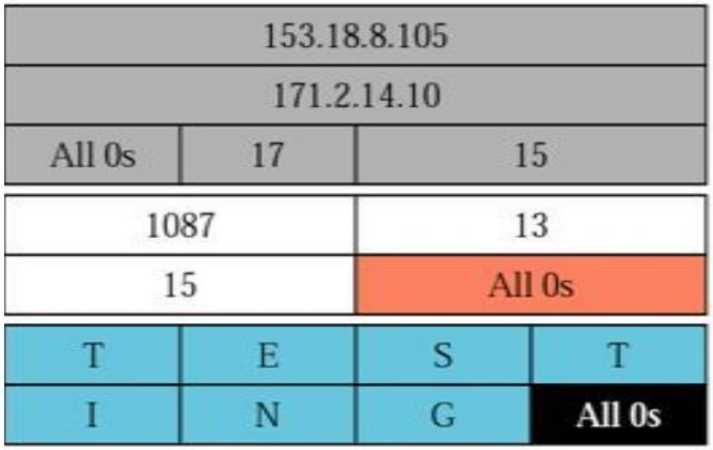
Checksum Example



- to compute the checksum, the sender treats the data as a sequence of binary integers and computes their sum, as illustrated above
- each pair of characters is treated as a **16-bit integer**
- **if the sum overflows 16 bits, the carry bits are added to the total**
- the advantage of such checksums is their size and ease of computation
- addition requires very little computation and the cost of sending an additional 16-bits is negligible

Checksum calculation at sender side for 7 bytes of data

shows the checksum calculation for a very small user datagram with only 7 bytes of data. Because the number of bytes of data is odd, padding is added for checksum calculation. The pseudoheader as well as the padding will be dropped when the user datagram is delivered to IP



10011001 00010010	153.18
00001000 01101001	8.105
10101011 00000010	171.2
00001110 00001010	14.10
00000000 00010001	0 and 17
00000000 00001111	15
00000100 00111111	1087
00000000 00001101	13
00000000 00001111	15
00000000 00000000	0 (checksum)
01010100 01000101	T and E
01010011 01010100	S and T
01001001 01001110	I and N
01000111 00000000	G and 0 (padding)
10010110 11101011	Sum
01101001 00010100	Checksum

Checksum calculation at receiver side

Sender sends the UDP segment along with checksum filled

153.18.8.105			
171.2.14.10			
All 0s	17	15	
1087		13	
15		01101001 00010100	
T	E	S	T
I	N	G	Pad

10011001 00010010 ----------> 153.18
00001000 01101001 ----------> 8.105
10101011 00000010 ----------> 171.2
00001110 00001010 ----------> 14.1
00000000 00010001 ----------> 0 and 17
00000000 00001111 ----------> 15
00000100 00111111 ----------> 1087
00000000 00001101 ----------> 13
00000000 00001111 ----------> 15
01101001 00010100 ----------> Checksum
01010100 01000101 ----------> T and E
01010011 01010100 ----------> S and T
01001001 01001110 ----------> I and N
01000111 00000000 ----------> G and 0(padding)

Receiver receives the UDP segment correctly

153.18.8.105			
171.2.14.10			
All 0s	17	15	
1087		13	
15		01101001 00010100	
T	E	S	T
I	N	G	Pad

10011001 00010010 ----------> 153.18
00001000 01101001 ----------> 8.105
10101011 00000010 ----------> 171.2
00001110 00001010 ----------> 14.1
00000000 00010001 ----------> 0 and 17
00000000 00001111 ----------> 15
00000100 00111111 ----------> 1087
00000000 00001101 ----------> 13
00000000 00001111 ----------> 15
01101001 00010100 ----------> Checksum
01010100 01000101 ----------> T and E
01010011 01010100 ----------> S and T
01001001 01001110 ----------> I and N
01000111 00000000 ----------> G and 0(padding)

11111111 11111111 Sum
00000000 00000000 CheckSum

What value is sent for the checksum in one of the following hypothetical situations?

a. The sender decides not to include the checksum.

b. The sender decides to include the checksum, but the value of the sum is all 1s.

c. The sender decides to include the checksum, but the value of the sum is all 0s.

Solution

a. The value sent for the checksum field is all 0s to show that the checksum is not calculated.

b. When the sender complements the sum, the result is all 0s; the sender complements the result again before sending. The value sent for the checksum is all 1s. The second complement operation is needed to avoid confusion with the case in part a.

c. This situation never happens because it implies that the value of every term included in the calculation of the sum is all 0s, which is impossible; some fields in the pseudoheader have nonzero values (see Appendix D).

Questions..

A client uses UDP to send data to a server. The data are 16 bytes. Calculate the efficiency of this transmission at the UDP level (ratio of useful bytes to total bytes).

. Data are 16 bytes, length of UDP header is 8 bytes, so the ratio is $\frac{16}{16+8} = \frac{2}{3}$.

UDP and TCP use 1s complement for their checksums. Suppose you have the following three 8-bit bytes: 01010011, 01100110, 01110100. What is the 1s complement of the sum of these 8-bit bytes? (Note that although UDP and TCP use 16-bit words in computing the checksum, for this problem you are being asked to consider 8-bit sums.) Show all work. Why is it that UDP takes the 1s complement of the sum; that is, why not just use the sum?

The following is a dump of a UDP header in hexadecimal format.

0632000DOO lCE217

a. What is the source port number?
b. What is the destination port number?
c. What is the total length of the user datagram?
d. What is the length of the data?

(a) Source port number $= 0632_{16} = 1586$

(b) Destination port number $= 000D_{16} = 13$

(c) Total length $= 001C_{16} = 28$ bytes

(d) Since the header is 8 bytes the data length is $28 - 8 = 20$ bytes.

1. In cases where reliability is not of primary importance, UDP would make a good transport protocol. Give examples of specific cases.
2. Are both UDP and IP unreliable to the same degree? Why or why not?
3. Do port addresses need to be unique? Why or why not? Why are port addresses shorter than IP addresses?
4. What is the dictionary definition of the word *ephemeral?* How does it apply to the concept of the ephemeral port number?

TCP

TCP: Overview

- connection-oriented
- point-to-point:
  - one sender, one receiver
  - No multicasting
- reliable, in-order *byte steam:*
  - no "message boundaries"
  - Becoz of acks
- *Stream Delivery*
- *send & receive buffers*

- full duplex communication:
- flow controlled:
  - sender will not overwhelm receiver
- Congestion control

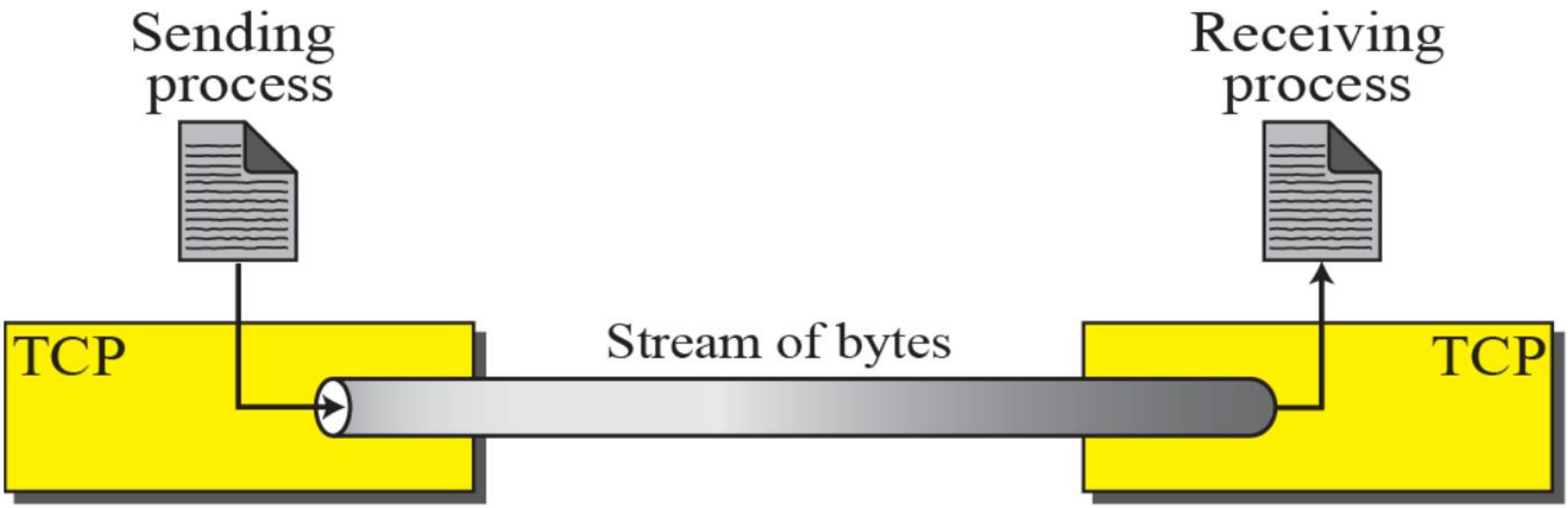
Transmission Control Protocol (TCP)

- the *Transmission Control Protocol* (TCP) is the transport level protocol that provides *reliability* in the TCP/IP protocol suite from an application program's perspective, TCP offers:
  - *connection-oriented*: an application requests a connection, and then uses it for data transfer
  - *point-to-point communication*: each TCP connection has exactly two end points
  - *reliability*: TCP guarantees that the data sent across the connection will be delivered exactly as sent, **without missing or duplicate data**
  - *full-duplex connection*: a TCP connection allows data to flow in both directions at any time
  - *stream interface*: TCP allows an application to send a continuous stream of bytes across the connection
  - *reliable startup*: TCP requires that two applications must agree to the new connection before it is established
  - *graceful shutdown*: TCP guarantees to deliver all the data reliably before closing the connection

Table 15.1 *Well-known Ports used by TCP*

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20 and 21	FTP	File Transfer Protocol (Data and Control)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol

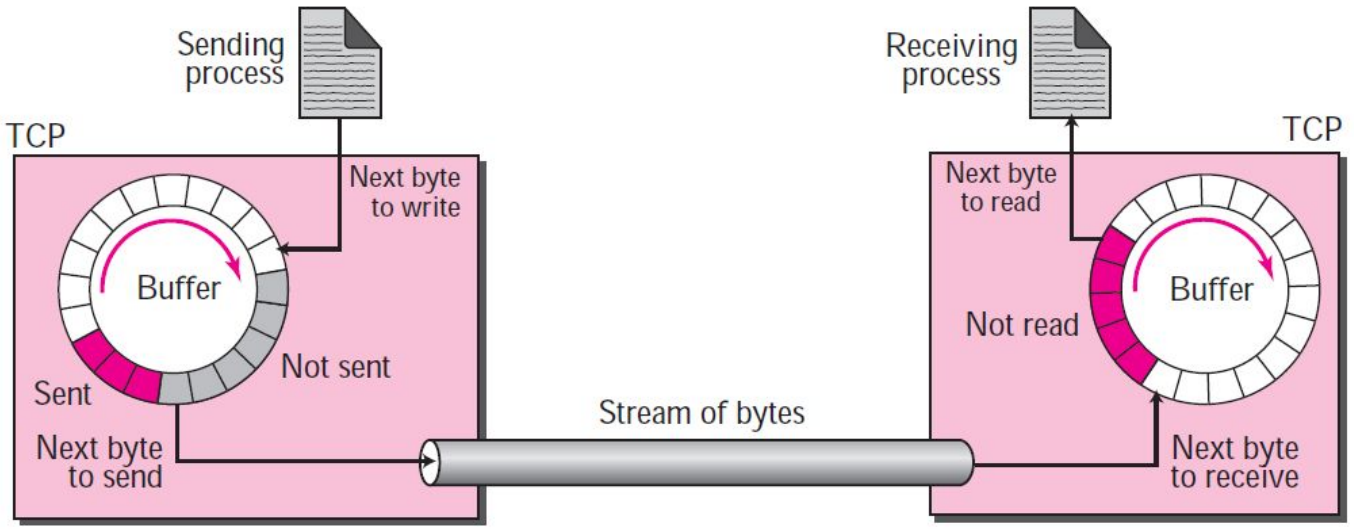
Stream delivery



TCP allows the sending process to deliver data as a stream of bytes and allows the receiving process obtain data as a stream of bytes.

TCP creates an environment in which the two processes seem to be connected by an imaginary "tube that carries their bytes across the Internet

Sending and Receiving Buffers



- Because the sending and the receiving processes may not necessarily write or read data at the same rate, TCP needs buffers for storage.
- There are two buffers, the sending buffer and the receiving buffer, one for each direction.
- These buffers are also necessary for flow- and error-control mechanisms used by TCP. One way to implement a buffer is to use a circular array of 1-byte locations.
- The TCP sender keeps these bytes in the buffer until it receives an acknowledgment.

Reliable Data Transfer

- TCP is a *reliable* data transfer protocol implemented on top of an *unreliable* network layer (IP)
- some problems:
  - bits in a packet may be *corrupted*
  - packets can be *lost* by the underlying network
- some solutions:
  - *acknowledgements* (ACKs) can be used to indicate packet received correctly
  - a *countdown timer* can be used to detect packet loss
  - packet *retransmission* can be used for lost packets

Numbering System:

- Byte Number
- Sequence number
- Acknowledgement number

The bytes of data being transferred in each connection are numbered by TCP. The numbering starts with an arbitrarily generated number.

The value in the sequence number field of a segment defines the number assigned to the first data byte contained in that segment.

The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive.

The acknowledgment number is cumulative.

Example

Suppose a TCP connection is transferring a file of 5,000 bytes.

The first byte is numbered 10,001. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1,000 bytes?

Solution

The following shows the sequence number for each segment:

Segment 1	→	**Sequence Number:**	**10,001**	**Range:**	**10,001**	to	**11,000**
Segment 2	→	**Sequence Number:**	**11,001**	**Range:**	**11,001**	to	**12,000**
Segment 3	→	**Sequence Number:**	**12,001**	**Range:**	**12,001**	to	**13,000**
Segment 4	→	**Sequence Number:**	**13,001**	**Range:**	**13,001**	to	**14,000**
Segment 5	→	**Sequence Number:**	**14,001**	**Range:**	**14,001**	to	**15,000**

SEGMENT

Before discussing TCP in more detail, let us discuss the TCP packets themselves. A packet in TCP is called a segment.

MSS: Maximum Segment size

- The MSS value counts only data octets
- The maximum segment size to **avoid fragmentation** is equal to the largest datagram size that any host is required to be able to reassemble minus the IP header size and TCP header sizes.

IPv4 hosts are required to be able to handle an MSS of 536 octets (= 576- 20 - 20)

IPv6 hosts are required to be able to handle an MSS of 1220 octets (= 1280 - 40 - 20).

TCP segment structure

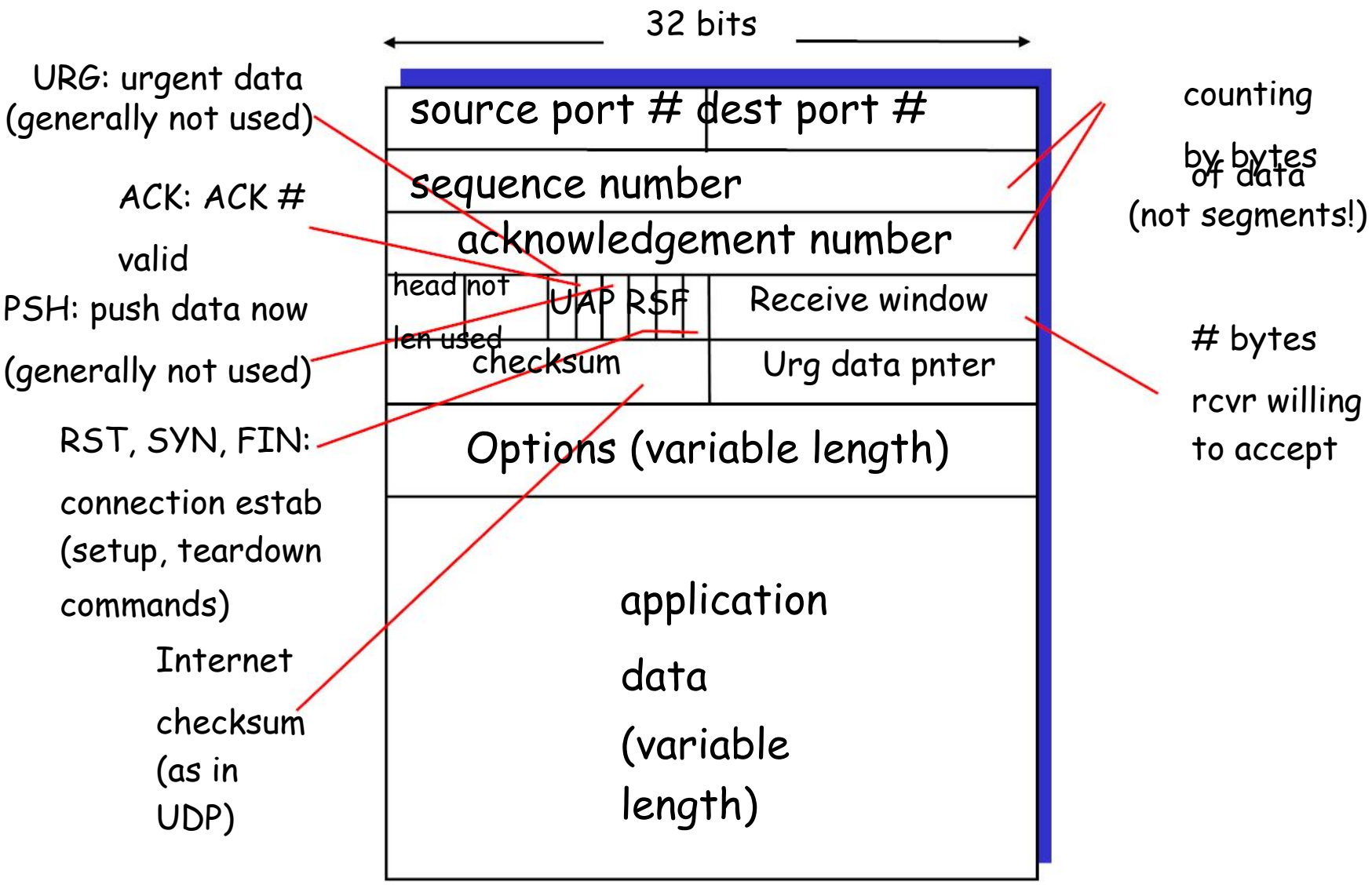
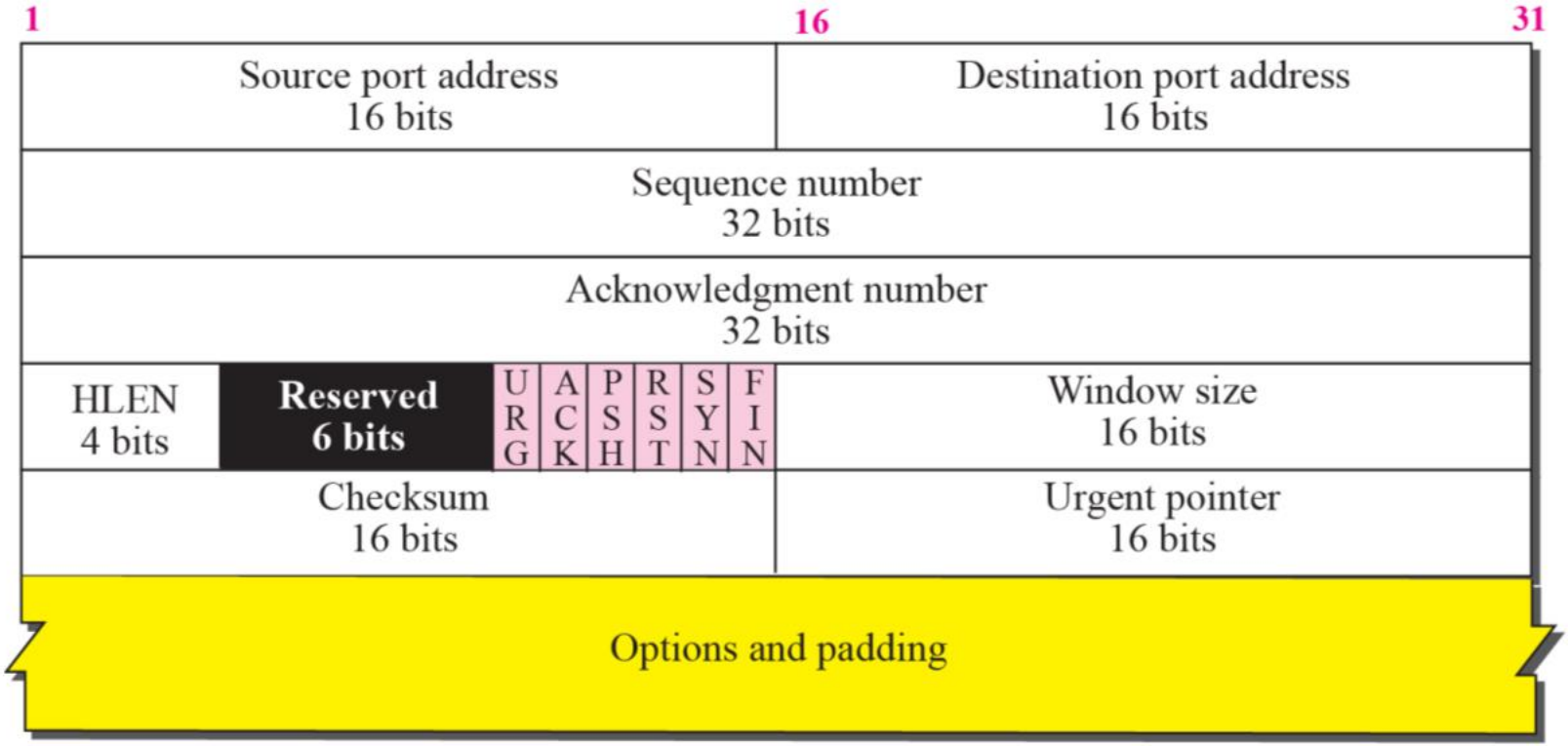


Figure 15.5 *TCP segment format*

a. Segment



b. Header

Sequence Number:

- Sequence number is 32 bits long.
- So the range of SeqNo is

  $0 <= SeqNo <= 2^{32}-1$ » 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

• If the SYN flag is set (1), then this is the initial sequence number. The sequence number of the actual first data byte and the acknowledged Number in the corresponding ACK are then this sequence number plus 1.

• If the SYN flag is clear (0), then this is the accumulated sequence number of the first data byte of this segment for the current session.

Acknowledgement Number :

- Acknowledgements are **piggybacked,** i.e. a segment from A ->B can contain an acknowledgement for a data sent in the B ->A direction

  - A hosts uses the AckNo field to send acknowledgements. (If a
    host sends an AckNo in a segment it sets the “ACK flag”)

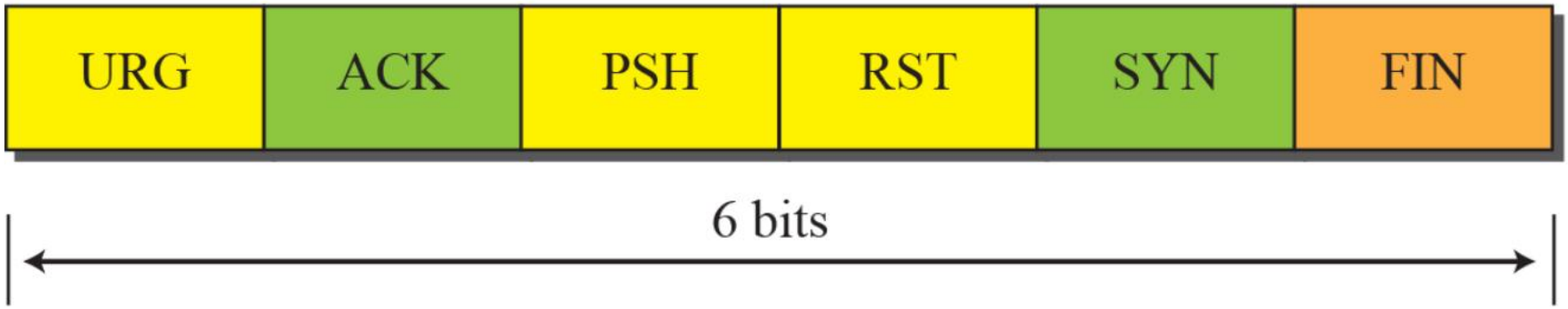
Header Length (4bits):

- Length of header in 32-bit words
- It indicates the number of 4-byte words.
- The minimum size header is 5 words and the maximum is 15 words thus giving the minimum size of 20 bytes and maximum of 60 bytes
- Q. Why minimum is 20 and maximum is 60.?
- Reserve bits are stored for future use.

Note that TCP header has variable length (with minimum 20 bytes and maximum 60 bytes.)

Control field

URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection



Window Size:

- Each side of the connection advertises the window size
- Window size (the size of the *receive window*)is the maximum number of bytes that a receiver can accept.
- Maximum window size is = 65535 bytes

TCP windows

The throughput of a communication is limited by two windows:

1. congestion window (for congestion control)
2. receive window (for flow control)

Each TCP segment contains the current value of the receive window.

For example a sender receives an ack which acknowledges byte 4000 and specifies a receive window of 10000 (bytes), the sender will not send packets after byte 14000, even if the congestion window allows it.

TCP Checksum:

- TCP checksum covers over both TCP header and TCP data (also covers some parts of the IP header)

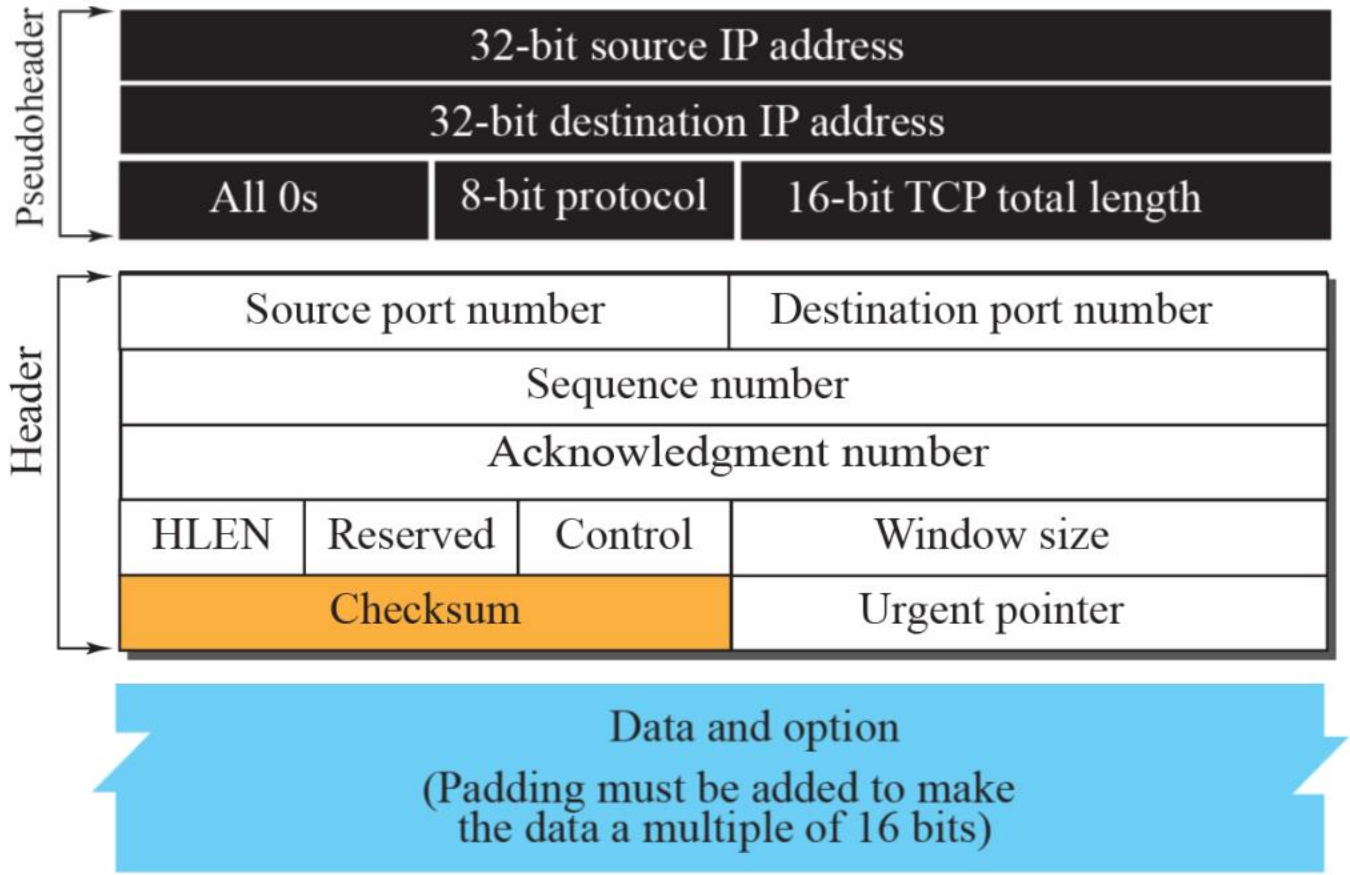
Urgent Pointer:

- Only valid if URG flag is set
- Used when segment contains some urgent data.

Options:

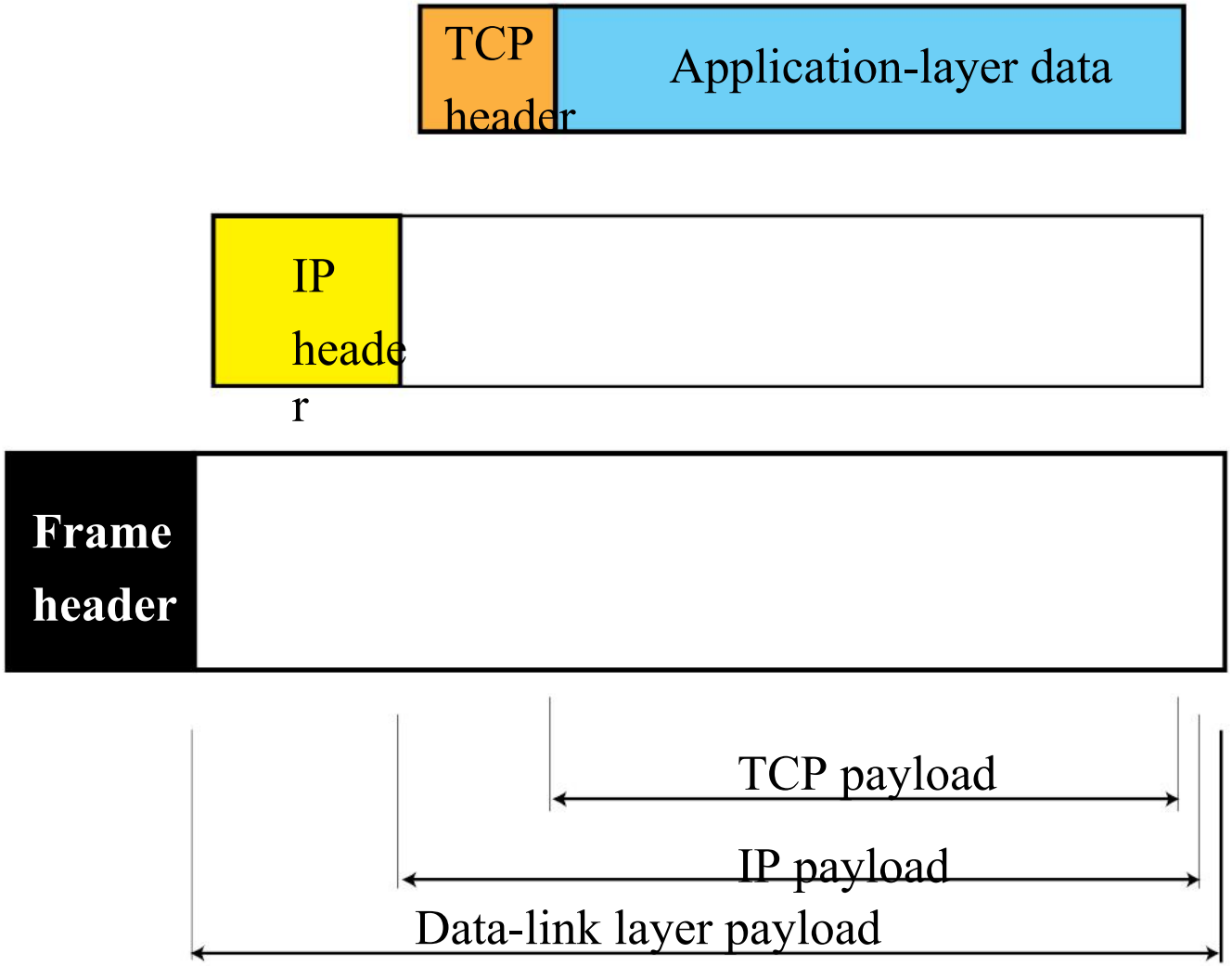
- Additional information in header.

Pseudoheader added to the TCP segment



Note: The use of the checksum in TCP is mandatory.

Encapsulation



The following is a dump of a TCP header in hexadecimal format.

$$(05320017\ 00000001\ 00000000\ 500207FF\ 00000000)_{16}$$

a. What is the source port number?
b. What is the destination port number?
c. What the sequence number?
d. What is the acknowledgment number?
e. What is the length of the header?
f. What is the type of the segment?
g. What is the window size?

a.1330

b. 23

c. 1

d. 0

e. 5===5*4=20 bytes

f. 2===SYN

g. 2047

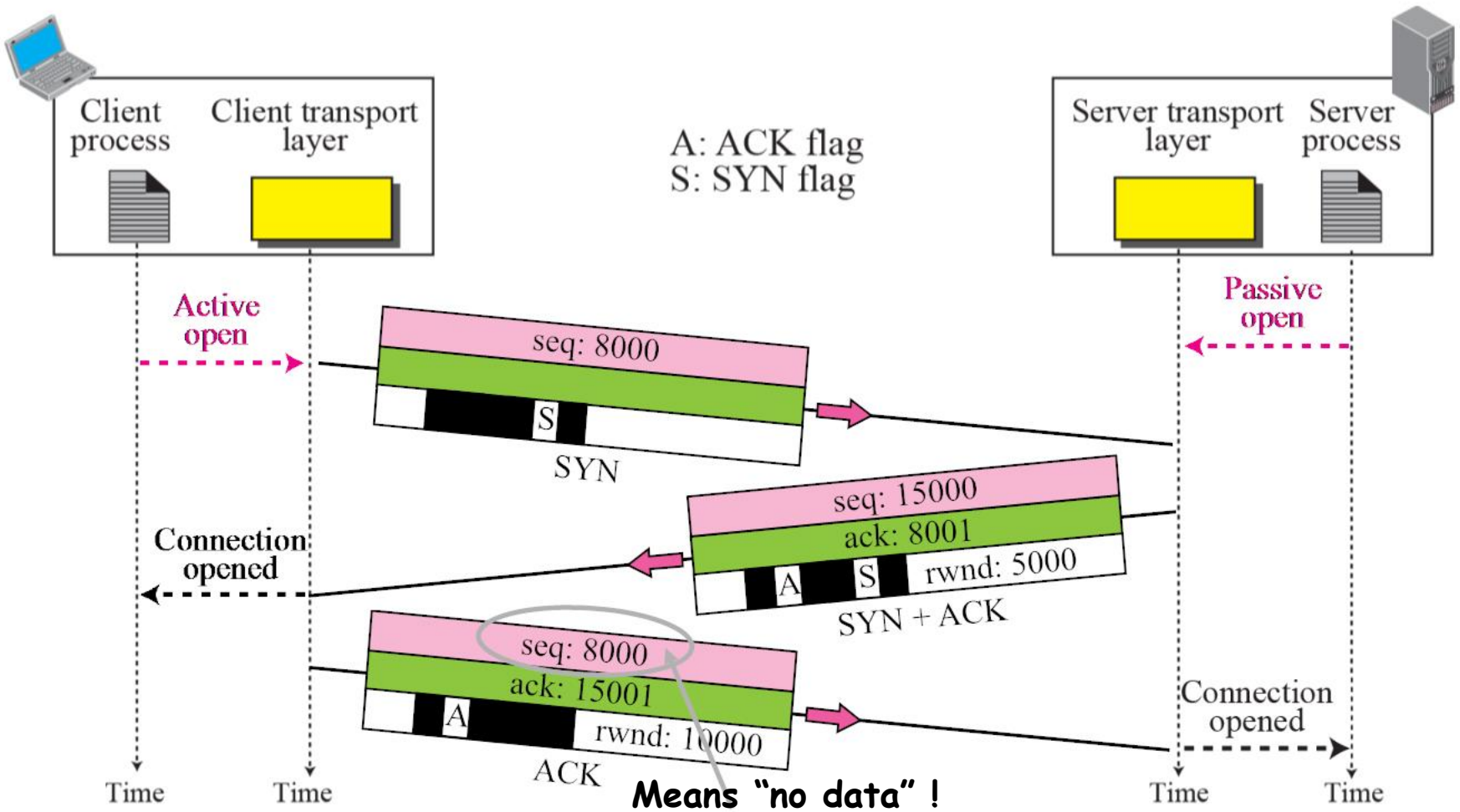
A TCP CONNECTION

TCP is connection-oriented. It establishes a virtual path between the source and destination. All of the segments belonging to a message are then sent over this virtual path. You may wonder how TCP, which uses the services of IP, a connectionless protocol, can be connection-oriented. The point is that a TCP connection is virtual, not physical. TCP operates at a higher level. TCP uses the services of IP to deliver individual segments to the receiver, but it controls the connection itself. If a segment is lost or corrupted, it is

In TCP, connection-oriented transmission requires three phases:

- ✓ **Connection Establishment**
- ✓ **Data Transfer**
- ✓ **Connection Termination**
- ✓ **Connection Reset**

Initial sequence number (ISN)



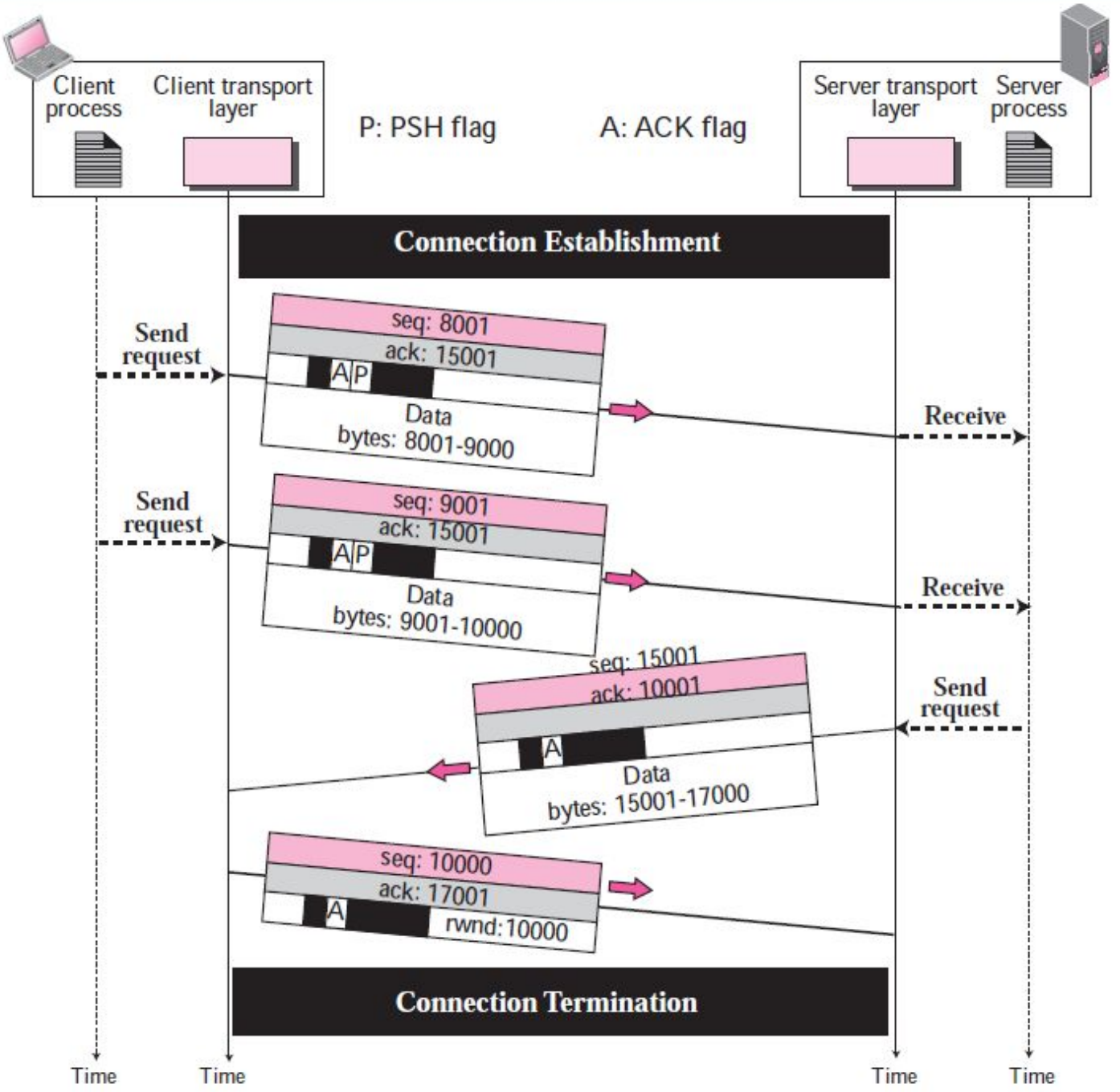
Note

A SYN segment cannot carry data, but it consumes one sequence number

A SYN + ACK segment cannot carry data, but does consume one sequence number.

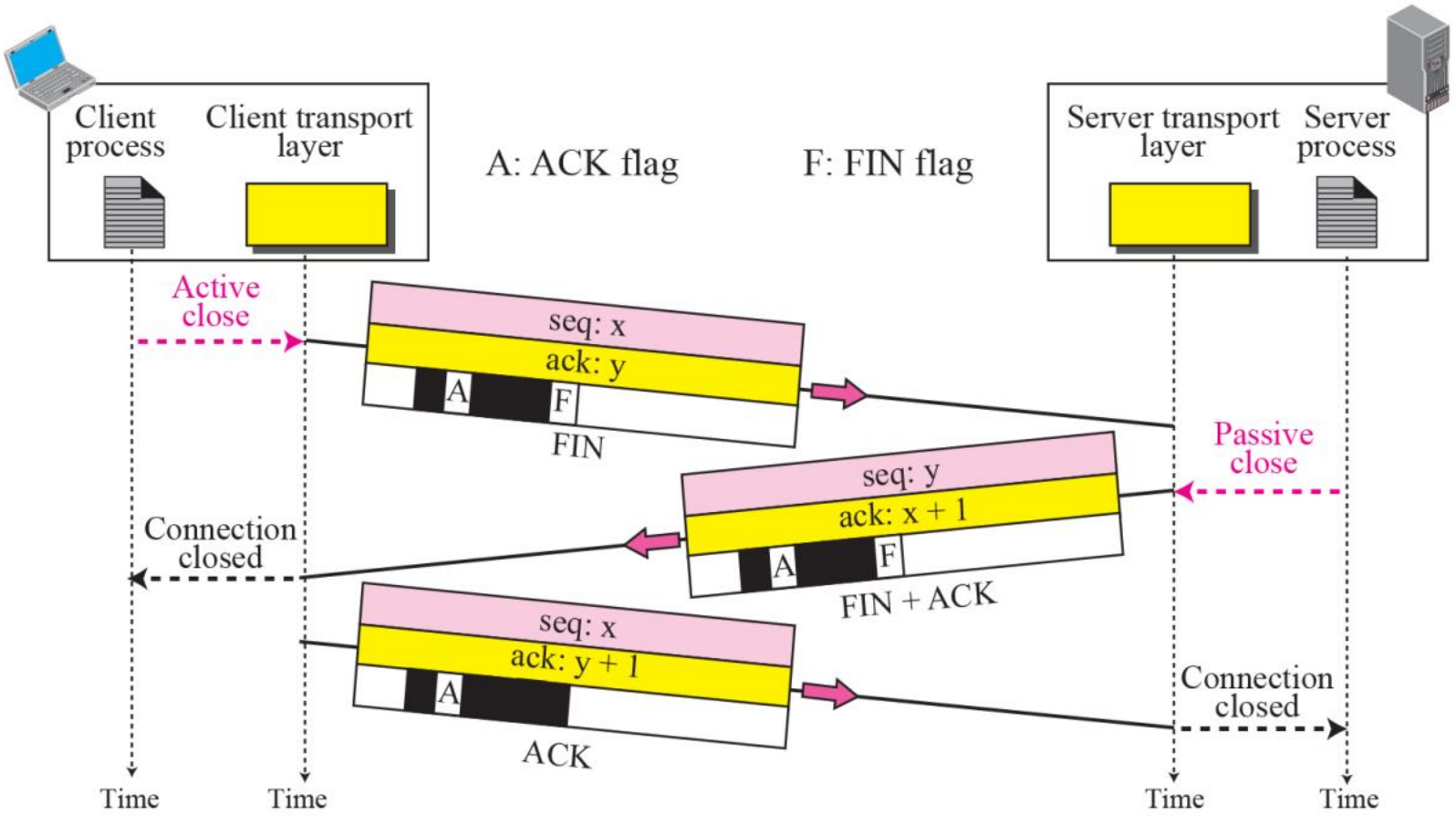
An ACK segment, if carrying no data, consumes no sequence number

Data Transfer



- Most implementations today allow two options for connection termination: **three-way handshaking and four-way handshaking with a half-close option.**

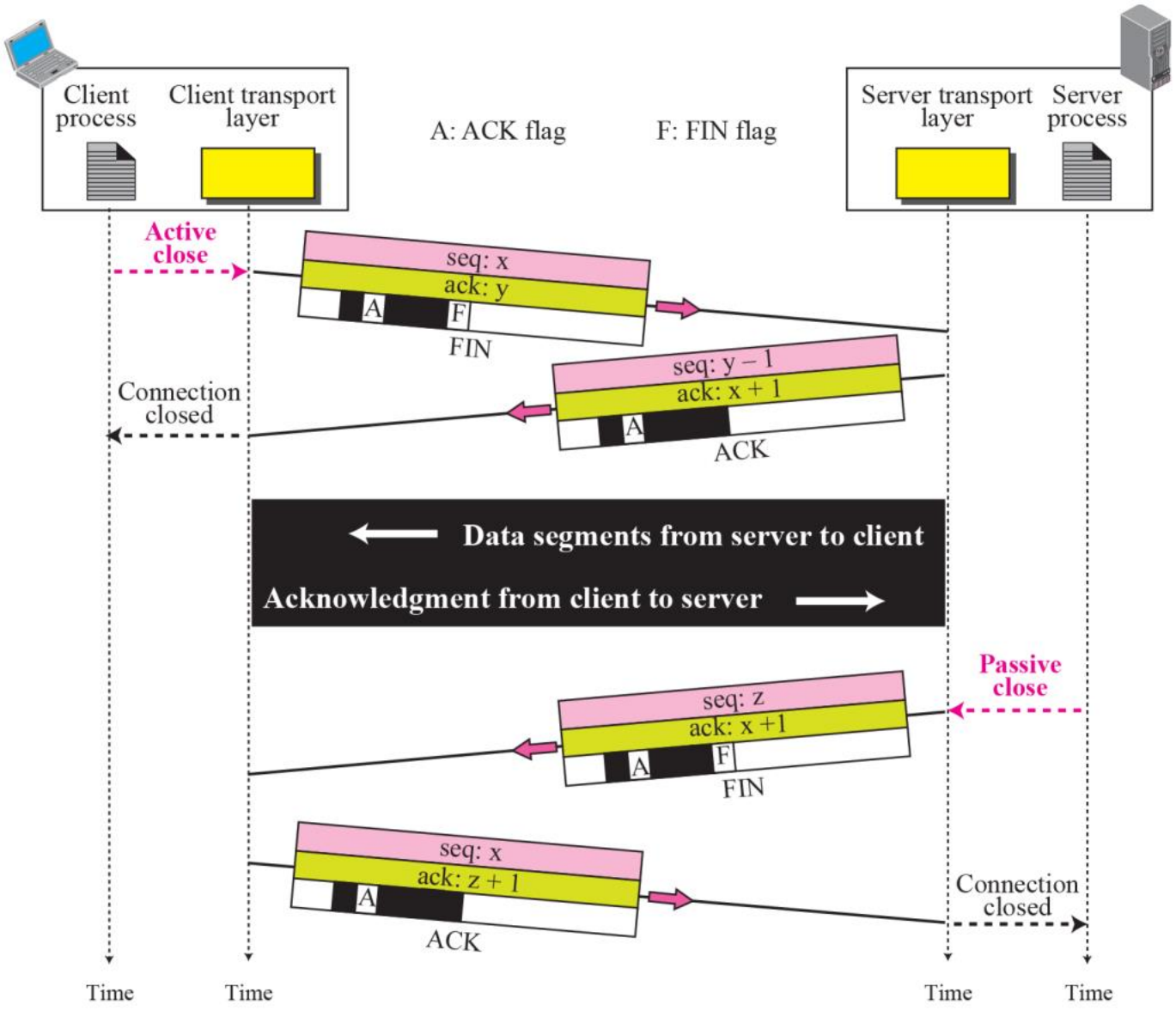
Connection termination using three-way handshake



The FIN segment consumes one sequence number if it does not carry data

The FIN + ACK segment consumes one sequence number if it does not carry data

Half-Close


Client process
Client transport layer
A: ACK flag
F: FIN flag
Server transport layer
Server process
Active close
seq: x
ack: y
A
F
FIN
seq: y – 1
ack: x + 1
A
ACK
Connection closed
Data segments from server to client
Acknowledgment from client to server
Passive close
seq: z
ack: x +1
A
F
FIN
seq: x
ack: z + 1
A
ACK
Connection closed
Time
Time
Time
Time

Question:

In a TCP connection, the initial sequence number at a client site is 2171. The client opens the connection, sends only one segment carrying 1000 bytes of data, and closes the connection. What is the value of the sequence number in each of the following segments sent by the client?

(a) The SYN segment.
(b) The data segment
(c) The FIN segment

Solution

(a) 2171

(b) 2172

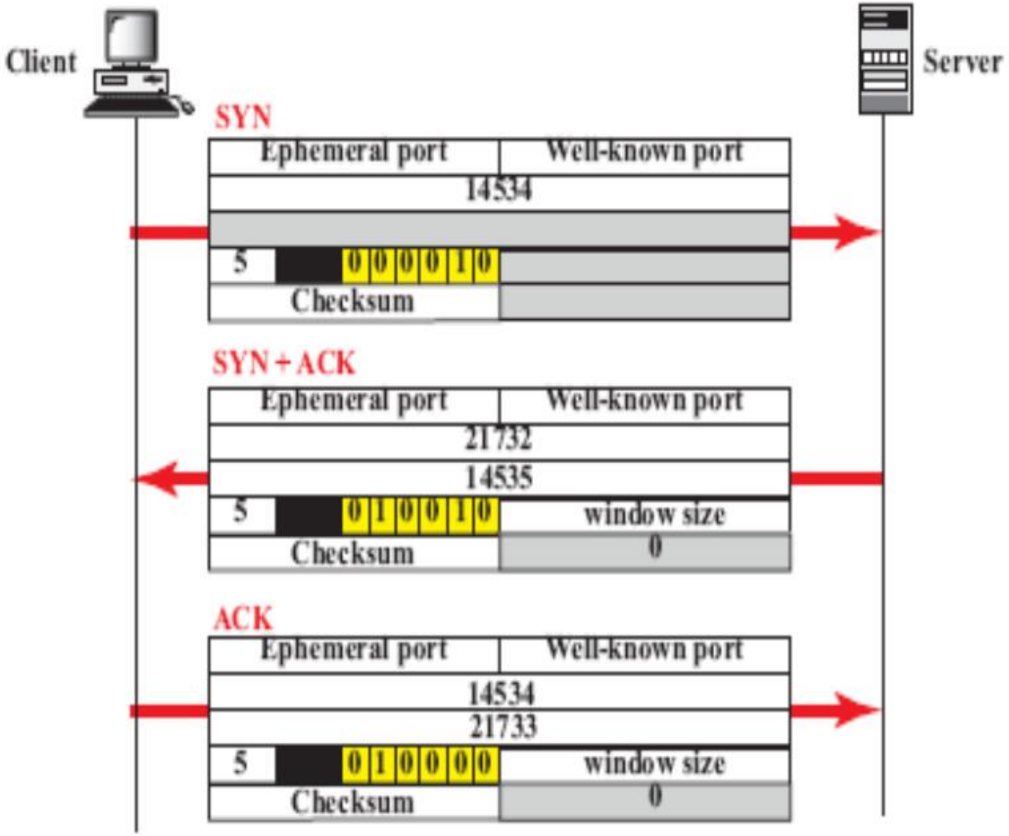
(c) 3172 (2172-3171 data)+3172 finish

- TCP provides reliable transfer through a mixture of sequence number, receiver buffer, cumulative acknowledgement, and fast retransmission. Answer the following True or False problems. If it's False, explain why.
- a) Host A is sending host B a large file over a TCP connection. Assume host B has no data to send to A. Host B will not send acknowledgements to host A because host B cannot piggyback the acknowledgement on data.
  - False. Piggyback is only for efficiency. If there's no data packet to be piggybacked to, then B will just send the acknowledgement packet.
- b) The size of the TCP advertised window (RcvWindow) never changes throughout the duration of the connection.
  - False. It is the size of the receiver's buffer that's never changed. RcvWindow is the part of the receiver's buffer that's changing all the time depending on the processing capability at the receiver's side and the network traffic.
- c) Suppose host A is sending host B a large file over a TCP connection. The number of unacknowledged bytes that A sends cannot exceed the size of the receiver's buffer.
  - False and True. The number of unacknowleged bytes that A sends cannot exceed the size of the receiver's window. But if it can't exceed the receiver's window, then it surely has no way to exceed the receiver's buffer as the window size is always less than or equal to the buffer size. On the other hand, for urgent messages, the sender CAN send it in even though the receiver's buffer is full.
- d) Suppose host A is sending host B a large file over a TCP connection. If the sequence number for a segment of this connection is *m*, then the sequence number for the subsequent segment will necessarily be *m+1*.
  - False. The sequence number of the subsequent segment depends on the number of 8-byte characters in the current segment.

- e) Suppose host A sends host B one segment with sequence number 38 and 4 bytes of data. Then in the same segment the acknowledgement number is necessarily 42.
  - False. The acknowledgement number has nothing to do with the sequence number. The ack. number indicates the next sequence number A is expecting from B.
- f) Suppose that the last sampleRTT in a TCP connection is equal to 1 second. Then timeout for the connection will necessarily be set to a value >= 1 second.
  - False. Next_RTT = alpha * last_estimated_RTT + (1-alpha)*newly_collected_RTT_sample. In this case even though the last sampleRTT which is the newly_collected_RTT_sample is 1sec, the next_RTT still depends on alpha and last_estimated_RTT. Therefore, the next_RTT is not necessarily greater than 1sec.
- g) With the selective repeat protocol, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.
  - False. SR uses selective acknowlegement. The ack. number has no way to fall outside the current window.
- h) With the Go-Back-N, it is possible for the sender to receive an ACK for a packet that falls outside of its current window.
  - True. GBN uses cumulative acknowlegement. Imagine a scenario where ACK1 arrives AFTER ACK2. Once the sender receives ACK2, it would know that both packet1 and 2 were received correctly. So it can remove packet1 and 2 from its window. Now if ACK1 arrives, then ACK1 actually falls outside the current window.

Question:

TCP opens a connection using initial sequence number (ISN) of 14,534. The other party opens the connection with an ISN 21732. Show the three TCP segments during the connection establishment.



Flow Control

Flow Control

- Flow Control
- Sender variables
- Receiver variables
- Sliding Window
- Sliding Window Protocols

Flow Control

Flow control defines the amount of data a source can send before receiving an acknowledgement from receiver.

What are we looking for?

- The flow control protocol must not be too slow (can't let sender send 1 byte and wait for acknowledgement).
- The flow control protocol must make sure that receiver does not get overwhelmed with data (can't let sender send all of its data without worrying about acknowledgements).

How do we do that?

- TCP uses a *sliding window* protocol.
- For each TCP duplex connection the sending and receiving TCP peer use this window to control the flow.

Sliding Windows

What is a Sliding Window?

- Receiver: *offered window* - acknowledges data sent and what it is prepared to receive
  - receiver can send an ACK, but with an offered window of 0
  - later, the receiver sends a *window update* with a non-zero offered window size
- Sender: *usable window* - how much data it is prepared to send immediately

Sender Variables

- Each segement is assigned a sequence number - SeqNum
- The sender maintains three variables: send window size (SWS),

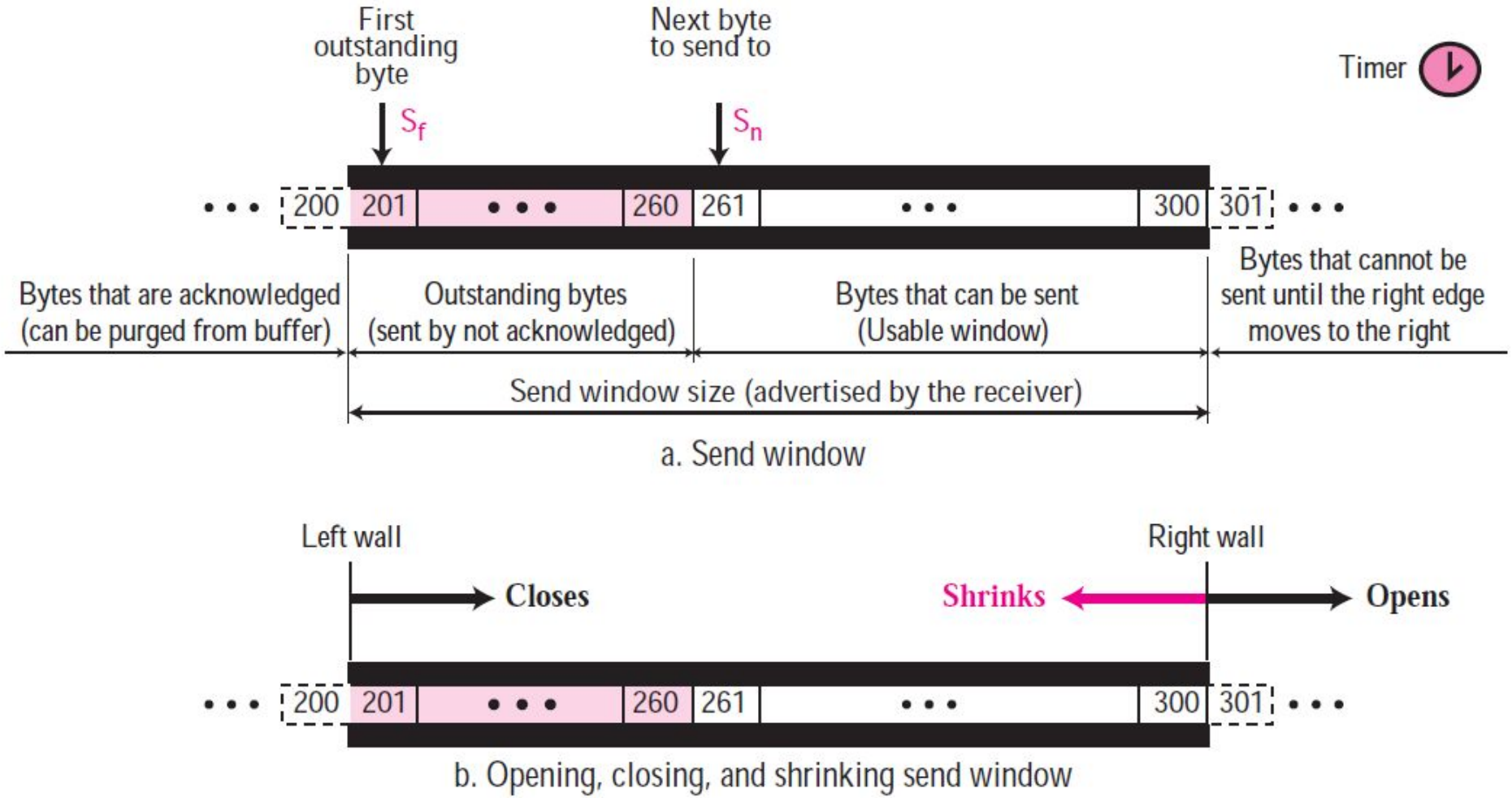
  last ACK received (LAR),

  and last Frame sent (LFS)

Sender Variables

- SWS:: the upper bound on the number of outstanding frames (not ACKed) the sender can transmit
- LAR :: the sequence number of the last ACK received.
- LFS :: the sequence number of the last frame sent.

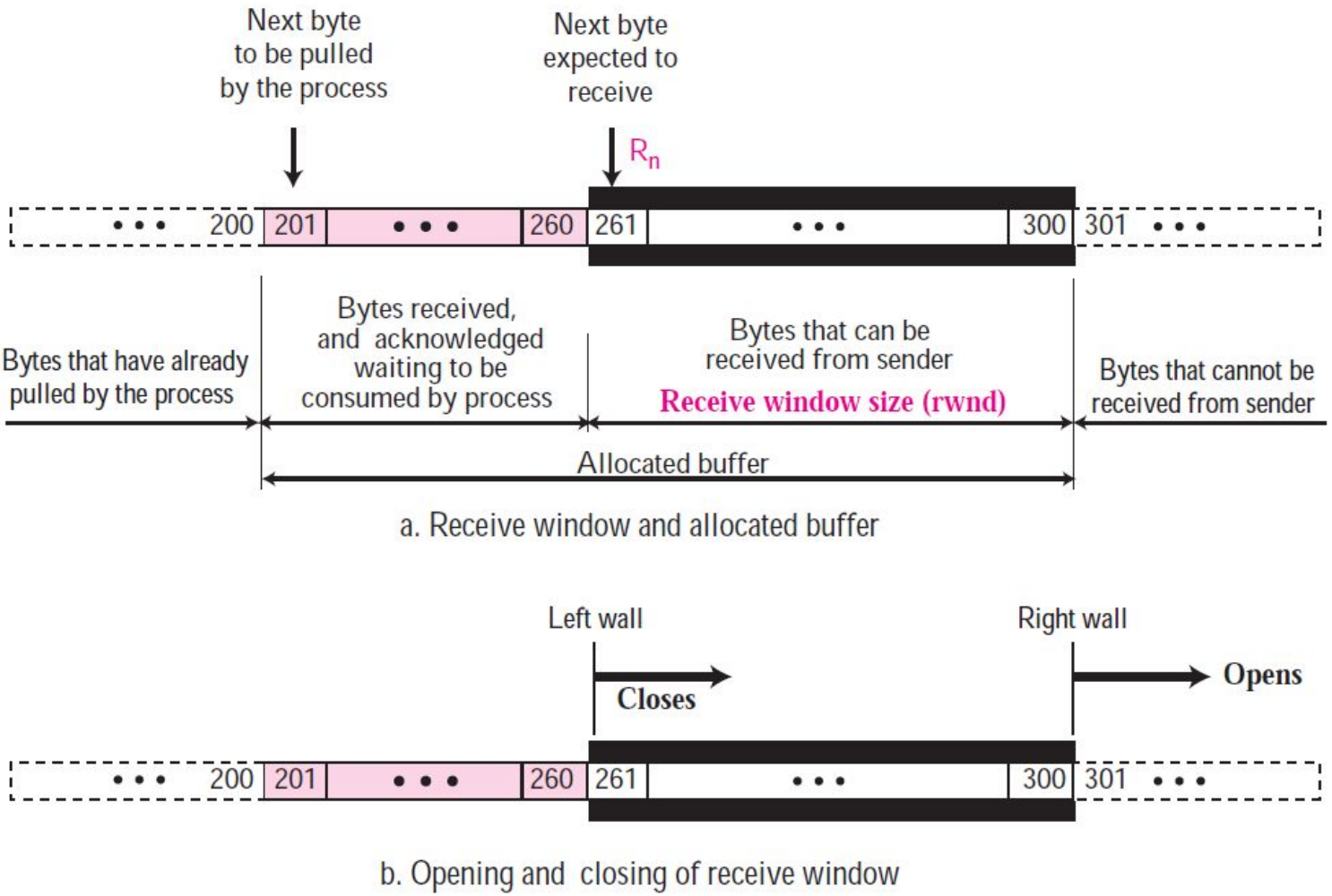
Figure 15.22 *Send window in TCP*



Receiver variables

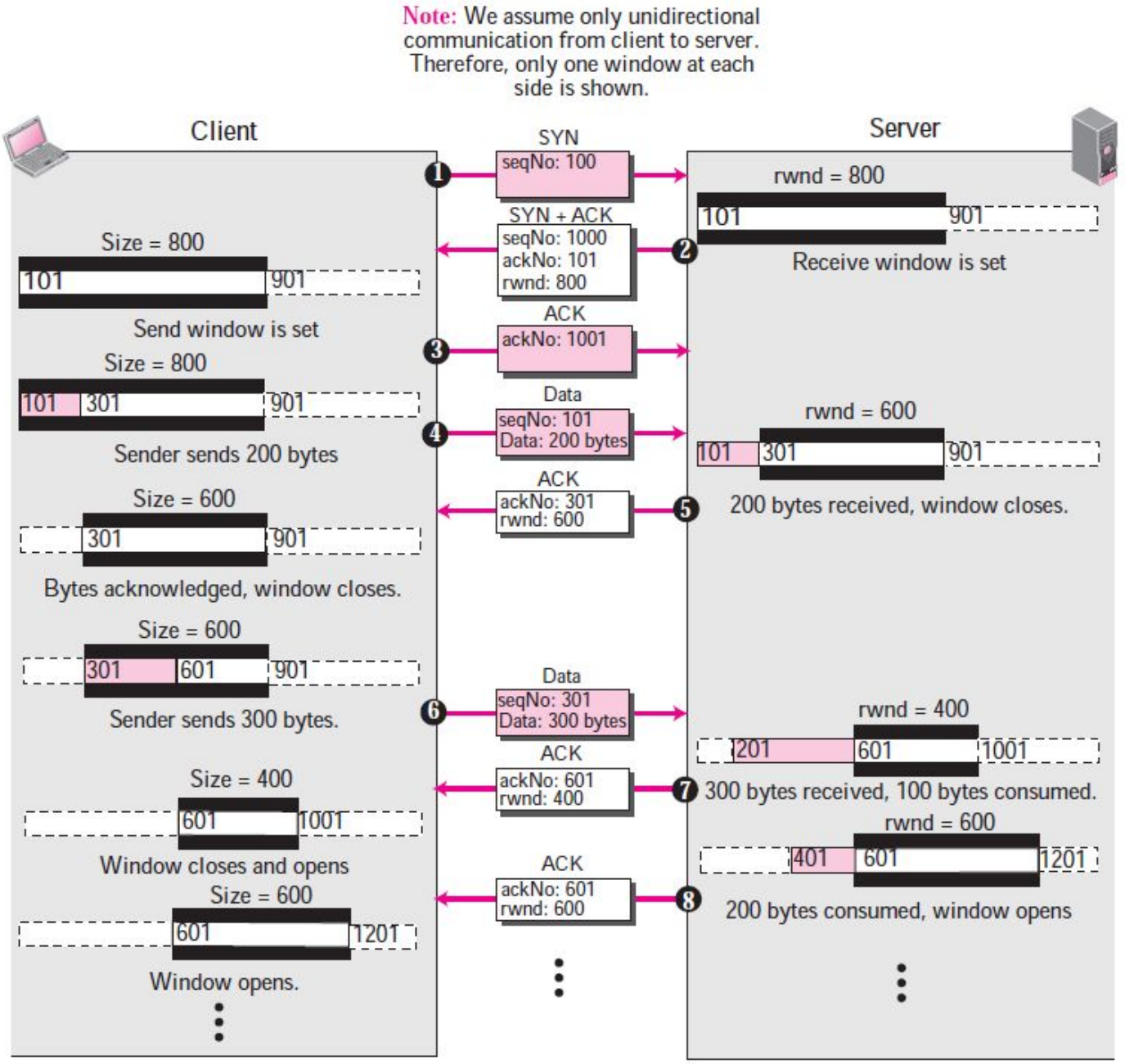
- Receiver window size (RWS) :: the upper bound on the number of out-of-order frames the receiver is willing to accept.
- Largest acceptable frame (LAF) :: the sequence number of the largest acceptable frame.
- Last frame received (LFR) :: the sequence number of the last frame received.

Figure 15.23 *Receive window in TCP*



rwnd = buffer size - number of waiting bytes to be pulled

Figure 15.25 *An example of flow control*

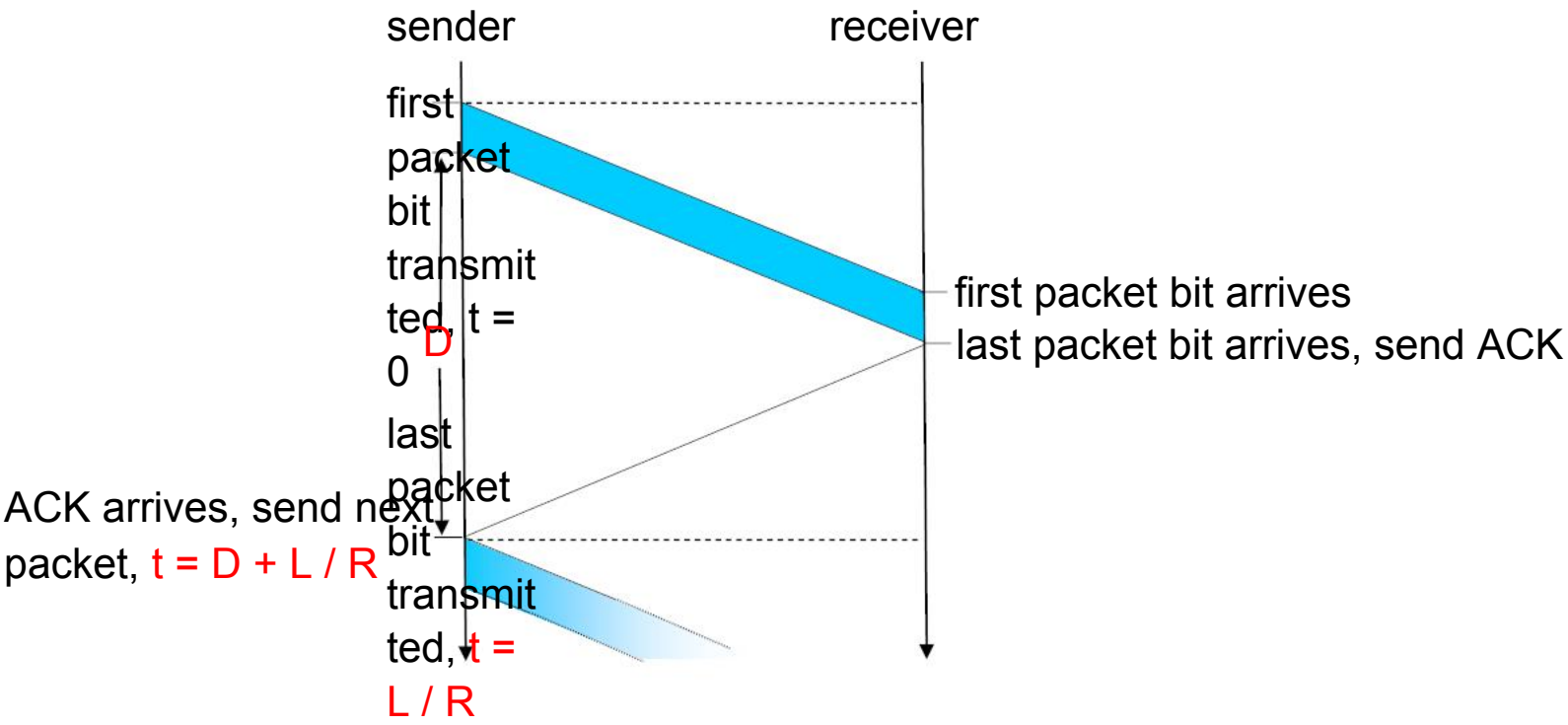


Flow Control Mechanisms

1. One possibility is send 1 data and wait for ack.
2. Or send all the data without worrying about ack's. For this reason window concept came.

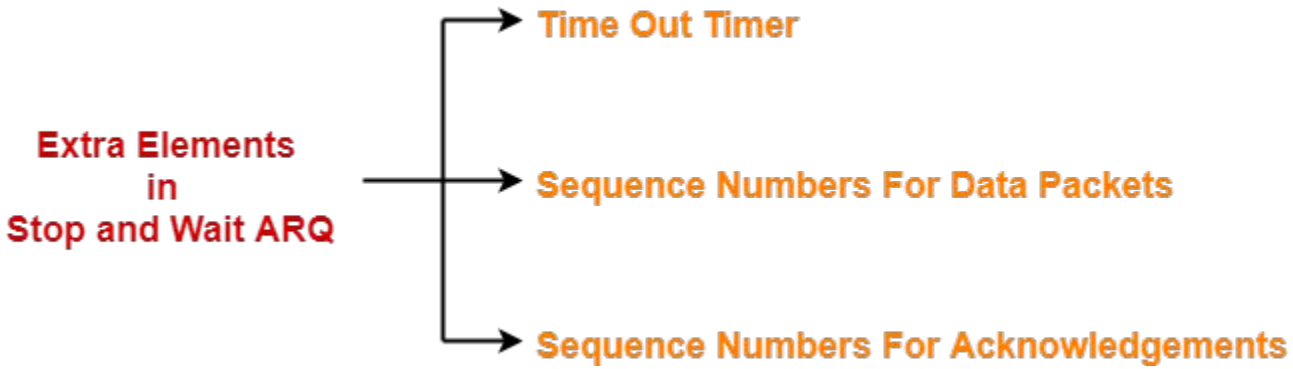
- Stop and Wait
  - The main problem faced by the Stop and Wait protocol is the occurrence of deadlock due to-
    - Loss of data packet
    - Loss of acknowledgement
- Stop and wait ARQ
- Go back N

1. stop-and-wait operation



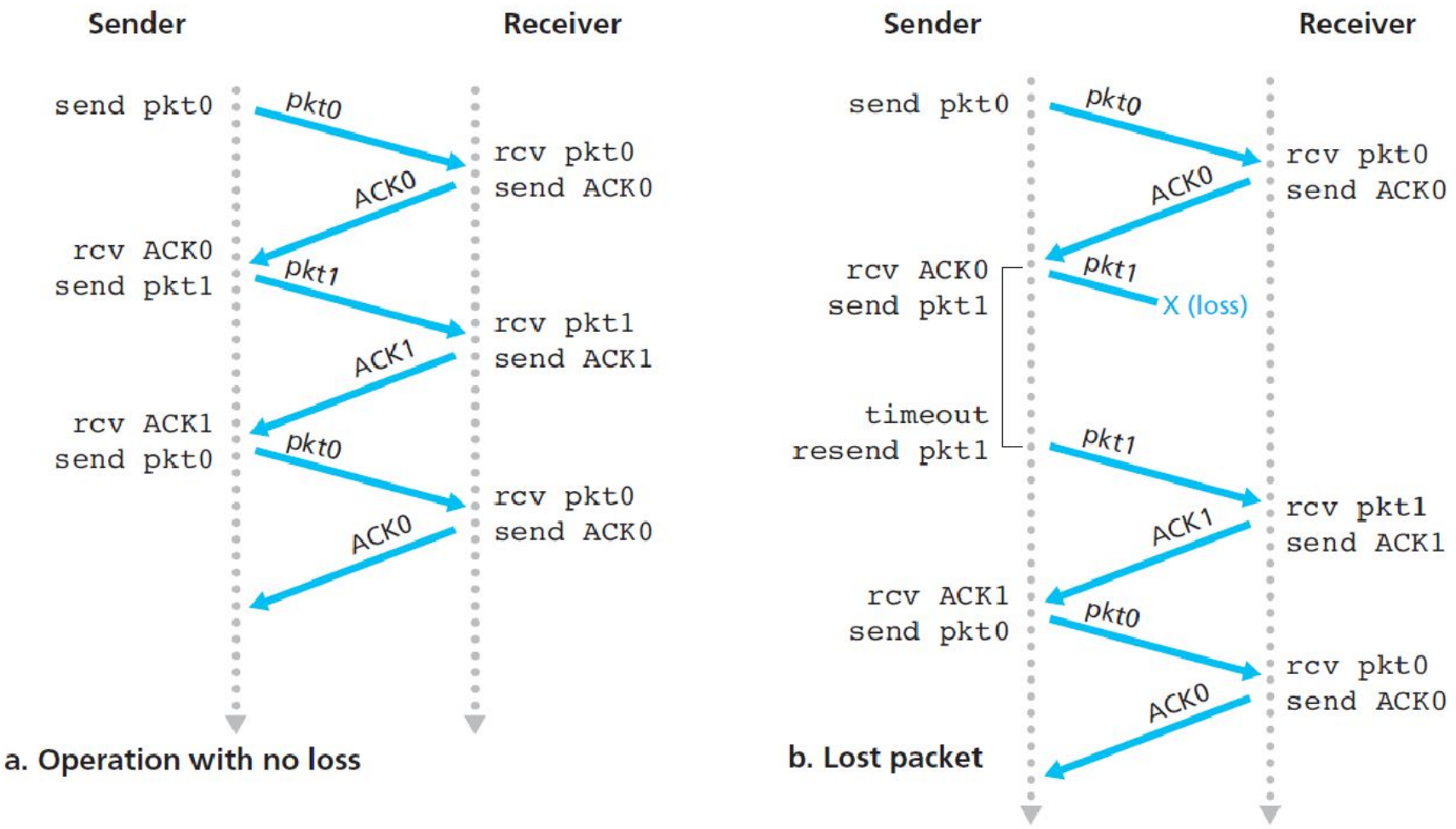
Stop and wait ARQ(**Automatic Repeat Request.)**

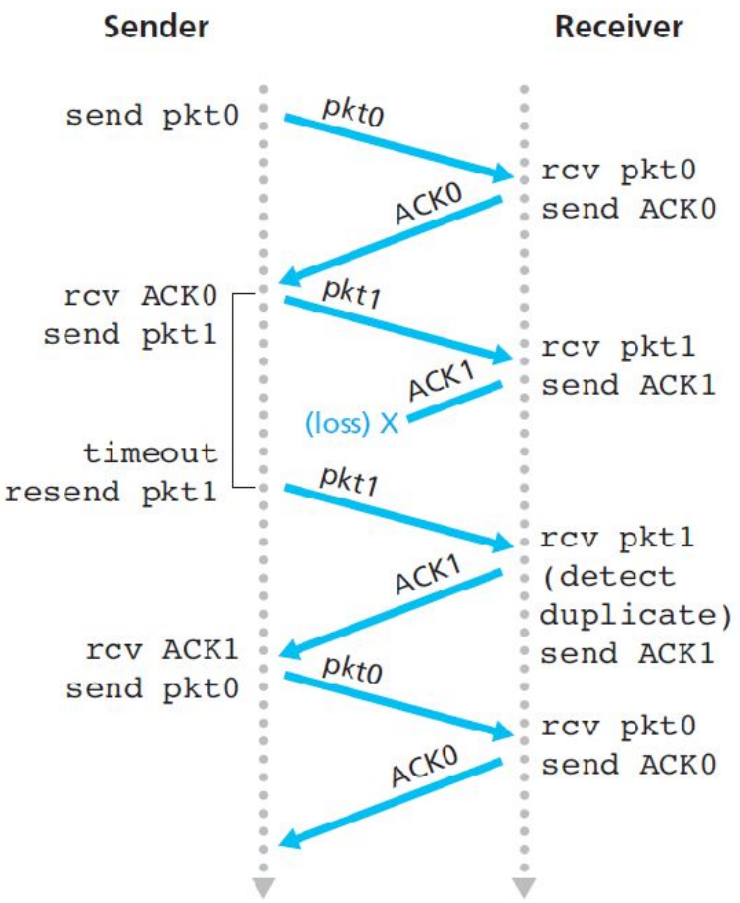
- Stop and wait ARQ works similar to stop and wait protocol.
- It provides a solution to all the limitations of stop and wait protocol.
- Stop and wait ARQ includes the following three extra elements.



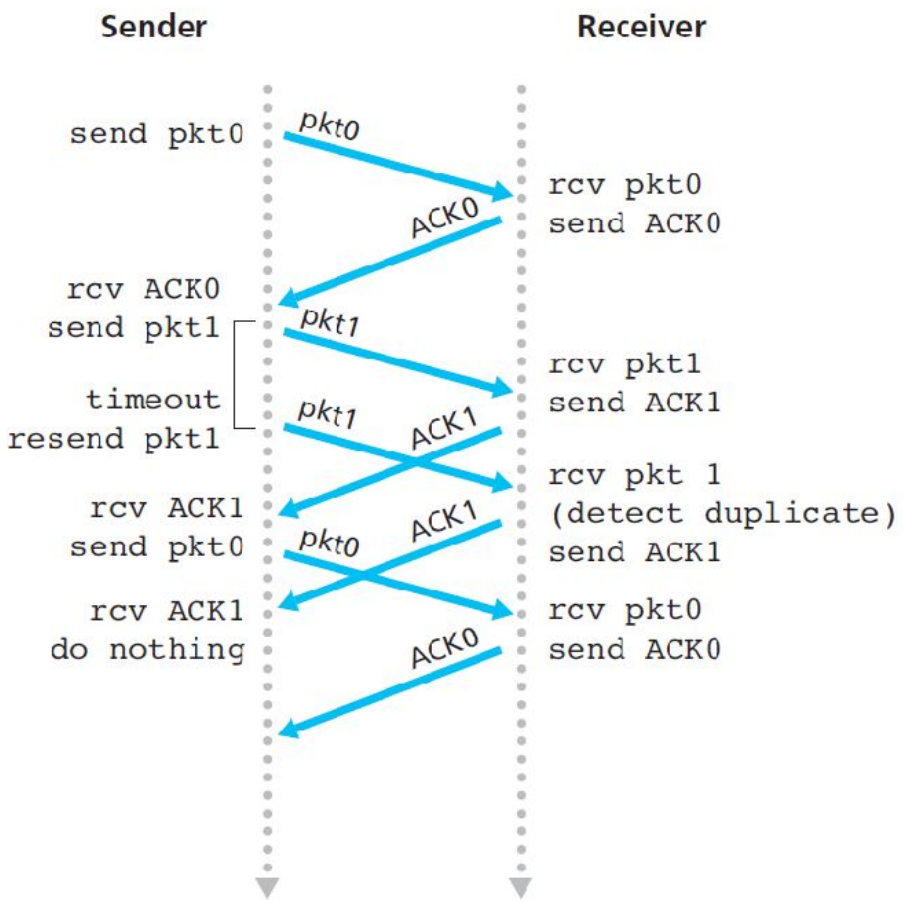
- Stop and wait ARQ is a one bit sliding window protocol where-
- Sender window size = 1
- Receiver window size = 1

- Thus, in stop and wait ARQ,
  - Minimum number of sequence numbers required = Sender Window Size + Receiver Window Size =1+1=2
  - 0,1


Sender
Receiver
send pkt0
pkt0
rcv pkt0
send ACK0
ACK0
rcv ACK0
send pkt1
pkt1
rcv pkt1
send ACK1
ACK1
rcv ACK1
send pkt0
pkt0
rcv pkt0
send ACK0
ACK0
a. Operation with no loss
Sender
Receiver
send pkt0
pkt0
rcv pkt0
send ACK0
ACK0
rcv ACK0
send pkt1
pkt1
X (loss)
timeout
resend pkt1
pkt1
rcv pkt1
send ACK1
ACK1
rcv ACK1
send pkt0
pkt0
rcv pkt0
send ACK0
ACK0
b. Lost packet

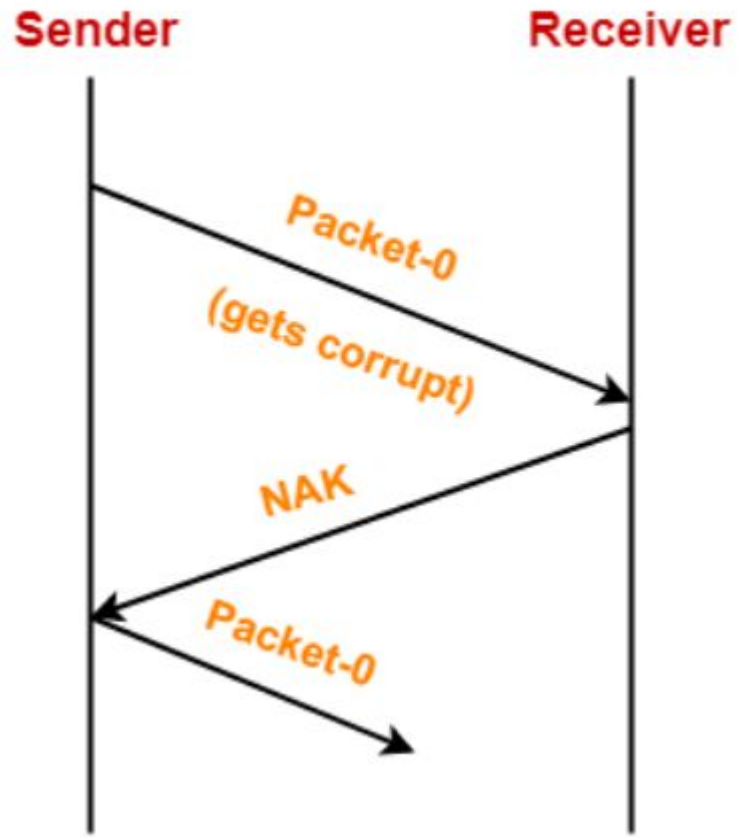


c. Lost ACK



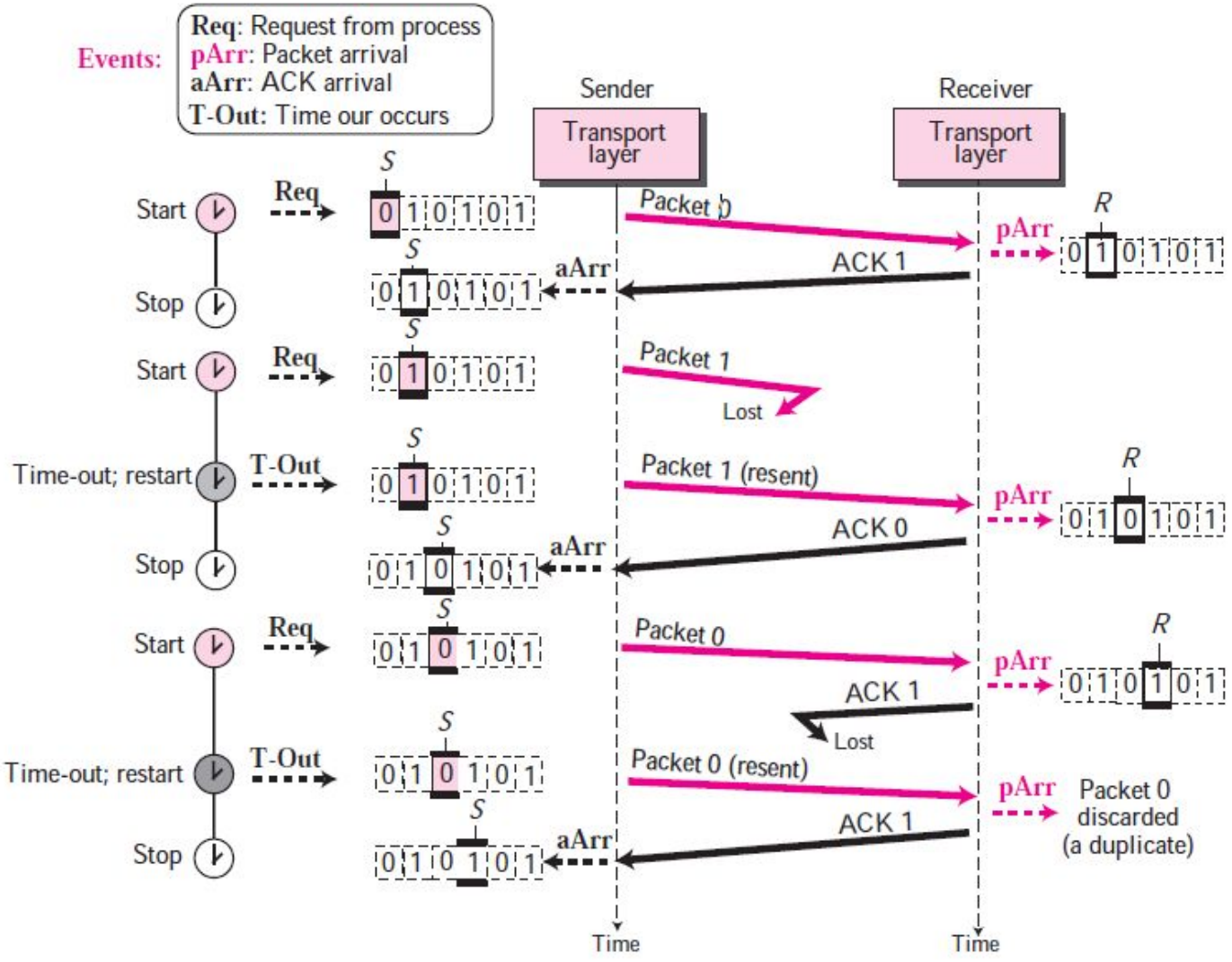
d. Premature timeout

- stop-and-wait : Damaged Frame
- stop-and-wait : Lost Frame
- stop-and-wait : Lost ACK and NAK (Negative ACK)



Window size

Figure 13.21 *Flow diagram for Example 13.4*



Limitation of Stop and Wait ARQ-

- The major limitation of Stop and Wait ARQ is its very less efficiency. **(Bandwidth)**
- To increase the efficiency, protocols like Go back N and Selective Repeat are used.

sender link utilization (Stop and wait ARQ)

Stop-and-Wait Link Efficiency [1]

The time to send 1 frame and receive an acknowledgement

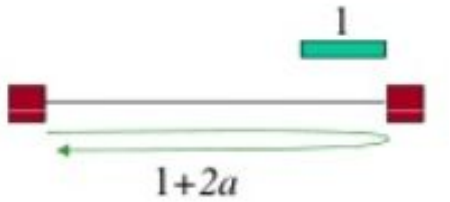
$$T_F = t_{frame} + t_{prop} + t_{proc} + t_{ack} + t_{prop} + t_{proc}$$

Assume t_{proc} and t_{ack} are negligible, the time to send a message containing n frames is

$$T = n(2\,t_{prop} + t_{frame})$$

Only $n \times t_{frame}$ is actually spent transmitting data. Hence, maximum possible utilization (or efficiency) of the link is

$$u = \frac{n \times t_{frame}}{n(2t_{prop} + t_{frame})} = \frac{t_{frame}}{2t_{prop} + t_{frame}}$$



Define $a = \frac{t_{prop}}{t_{frame}}$, then $u = \frac{1}{1+2a}$; smaller a higher link efficiency

U = (L / R) / (2 * P + L / R)

OR

U=(L/R)/(RTT+L/R)

Stop-and-Wait Link Utilizatio

- Source sends a single frame and waits for

$$T_{fram} + 2T_{prop} + T_{ack} = 2(T_{trans} + T_{prop})$$

- T_{prop} is large relative to T_{trans}
- Propagation delay is long relative to transmission time
- Transmission of only **one frame** at a time
- Waiting for a long time to receive **ACK**
- **Link is mostly idle**
- Stop-and-Wait Protocol **Reduces** Link Utilization

In a Stop-and-Wait ARQ system, the bandwidth of the line is 1 Mbps, and 1 bit takes 20 ms to make a round trip. What is the bandwidth-delay product? If the system data frames are 1000 bits in length, what is the utilization percentage of the link?

Solution

The bandwidth-delay product is

$$1 \times 10^{6} \times 20 \times 10^{-3} = 20{,}000 \text{ bits}$$

The system can send 20,000 bits during the time it takes for the data to go from the sender to the receiver and then back again. However, the system sends only 1000 bits. We can say that the link utilization is only 1000/20,000, or 5%. For this reason, for a link with high bandwidth or long delay, use of Stop-and-Wait ARQ wastes the capacity of the link.

Example

- L=8000bits per packet
- Bandwidth=1Gbps
- RTT=15ms
- Time to enter last bit into channel=L/R=8ms
- Last bit coming to receiver side=RTT/2+L/R
- Last ack emerges back to sender=RTT+L/R
- Find efficiency of sender

$$U_{\text{sender}} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

That is, the sender was busy only 2.7 hundredths of one percent of the time! Viewed another way, the sender was able to send only 1,000 bytes in 30.008 milliseconds, an effective throughput of only 267 kbps—even though a 1 Gbps link was available! Imagine the unhappy network manager who just paid a fortune for a gigabit capacity link but manages to get a throughput of only 267 kilobits per second!

U = (L / R) / (2 * P + L / R)

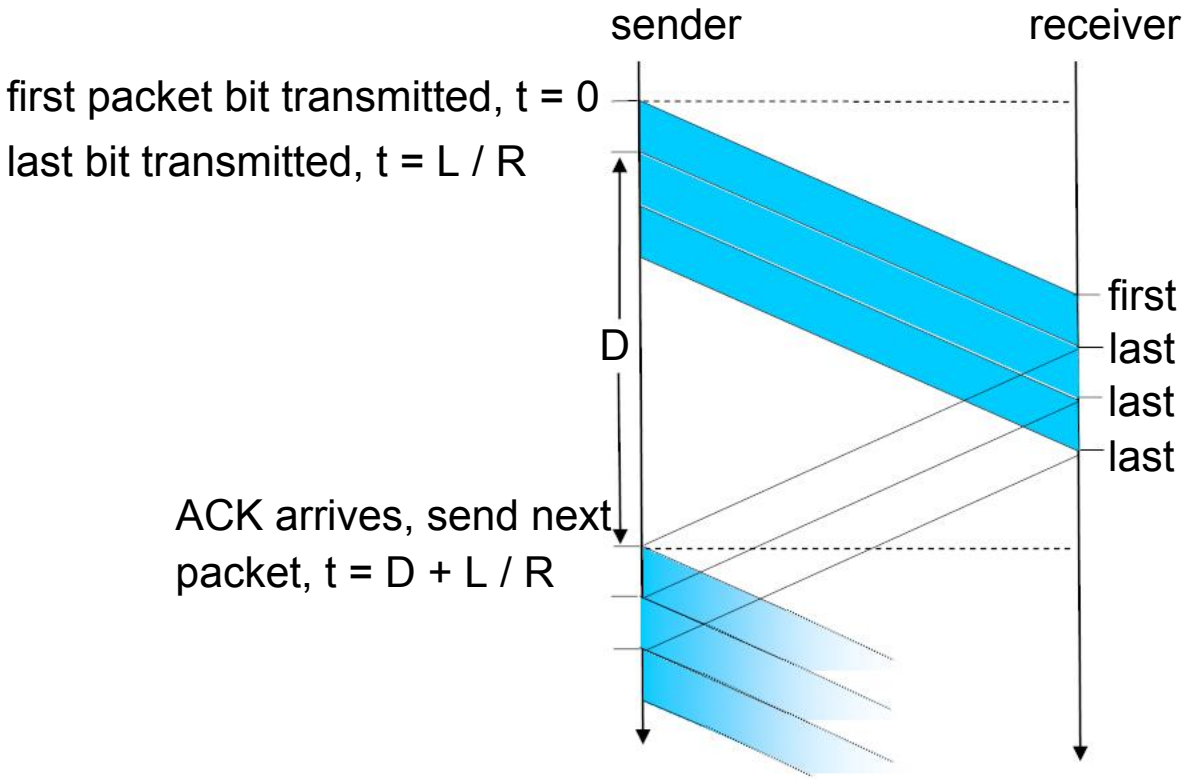
OR

U=(L/R)/(RTT+L/R)

Pipelined protocols

- Pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged pkts
  - range of sequence numbers must be increased
  - buffering at sender and/or receiver
- Two generic forms of pipelined protocols
  - *go-Back-N*
  - *selective repeat*

Pipelining: increased utilization



L/R=8ms

$$U_{sender} = \frac{3 * L / R}{D + L / R} = \frac{24ms}{100ms}$$

Increase utilization by a factor of 3!

0.24

=

- How to calculate window size?

2. Go-Back-N

- In Go-Back-N if one frame is lost or damaged all frames sent since the last frame acknowledged are retransmitted.
- Sending device keeps copies of all transmitted frames until they have been acknowledged.
- Sending device is equipped with the timer.
- Both ACK and NAK must be numbered for identification.
- On timeout, send all packets previously sent but not yet ACKed.
- Uses a single timer – represents the oldest transmitted, but not yet ACKed pkt
- **NAK does not indicate whether the frame has been lost or damaged, just that it needs to be resent.**

Window size

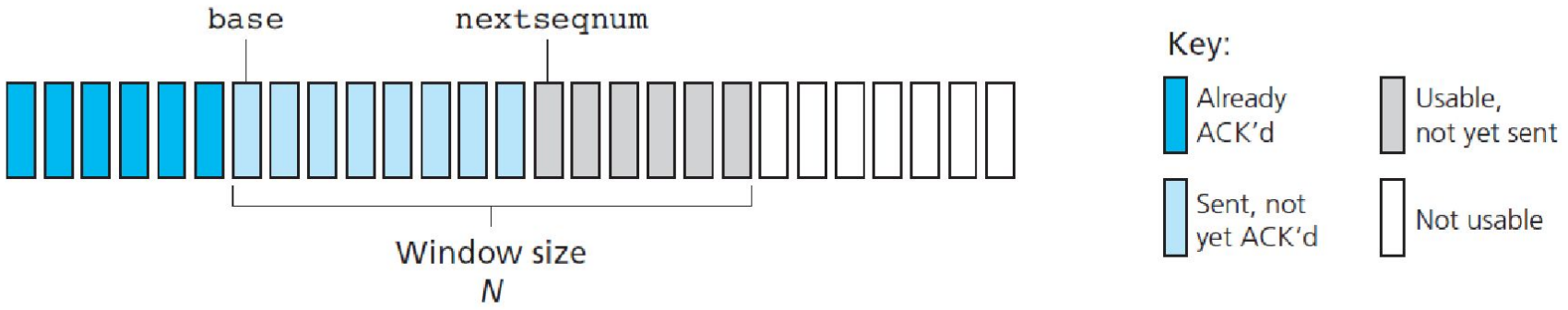


Figure 3.19 ♦ Sender's view of sequence numbers in Go-Back-N

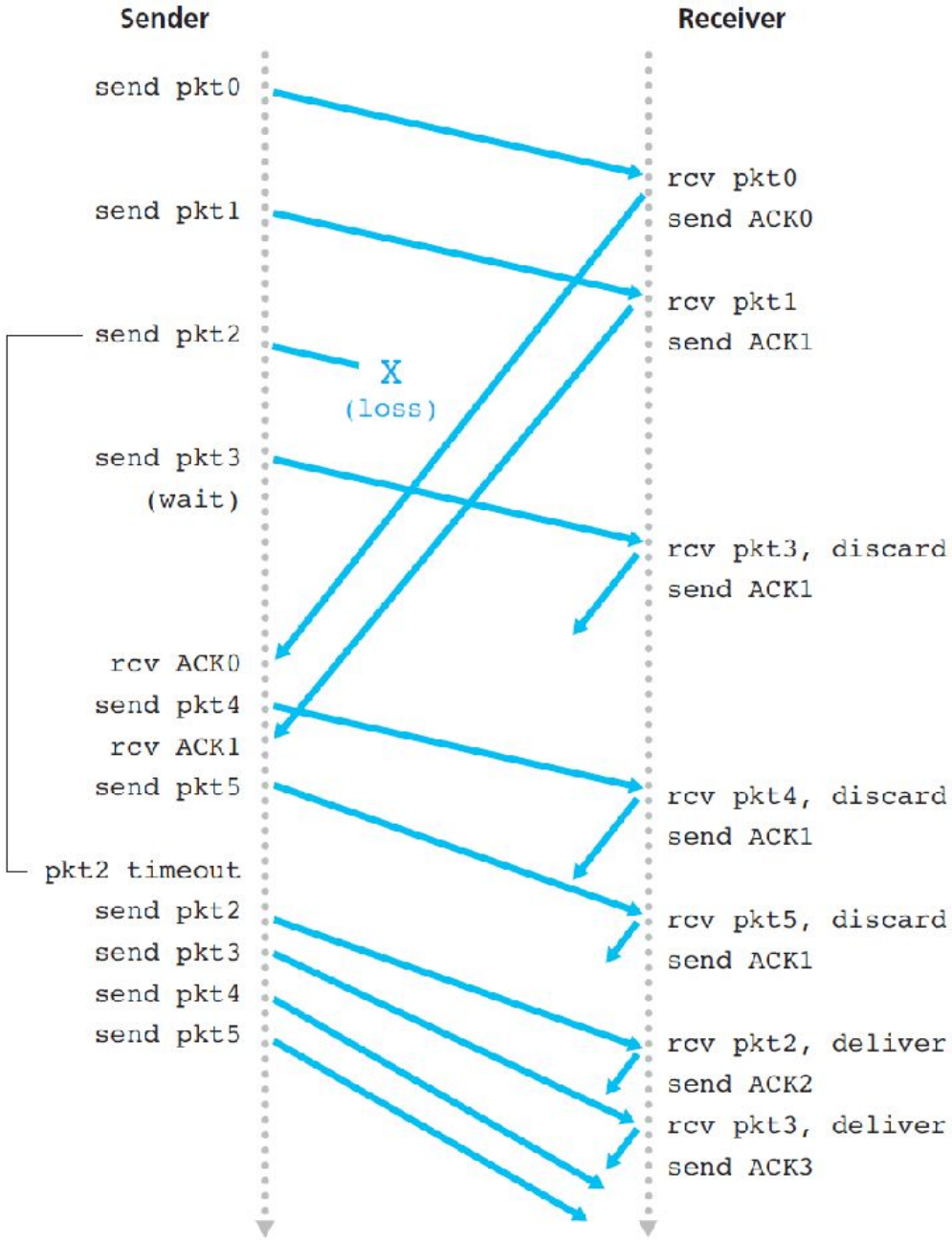
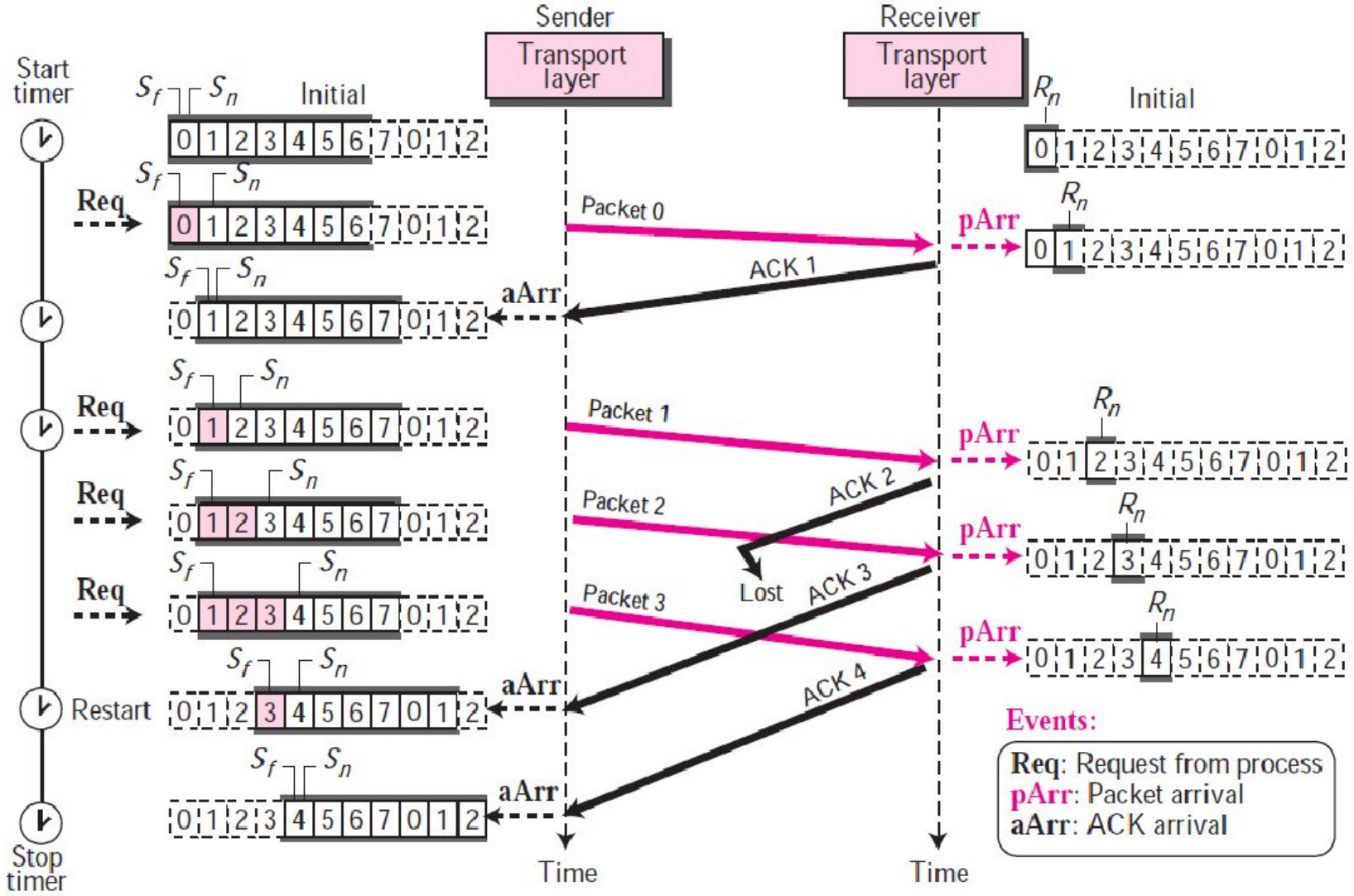
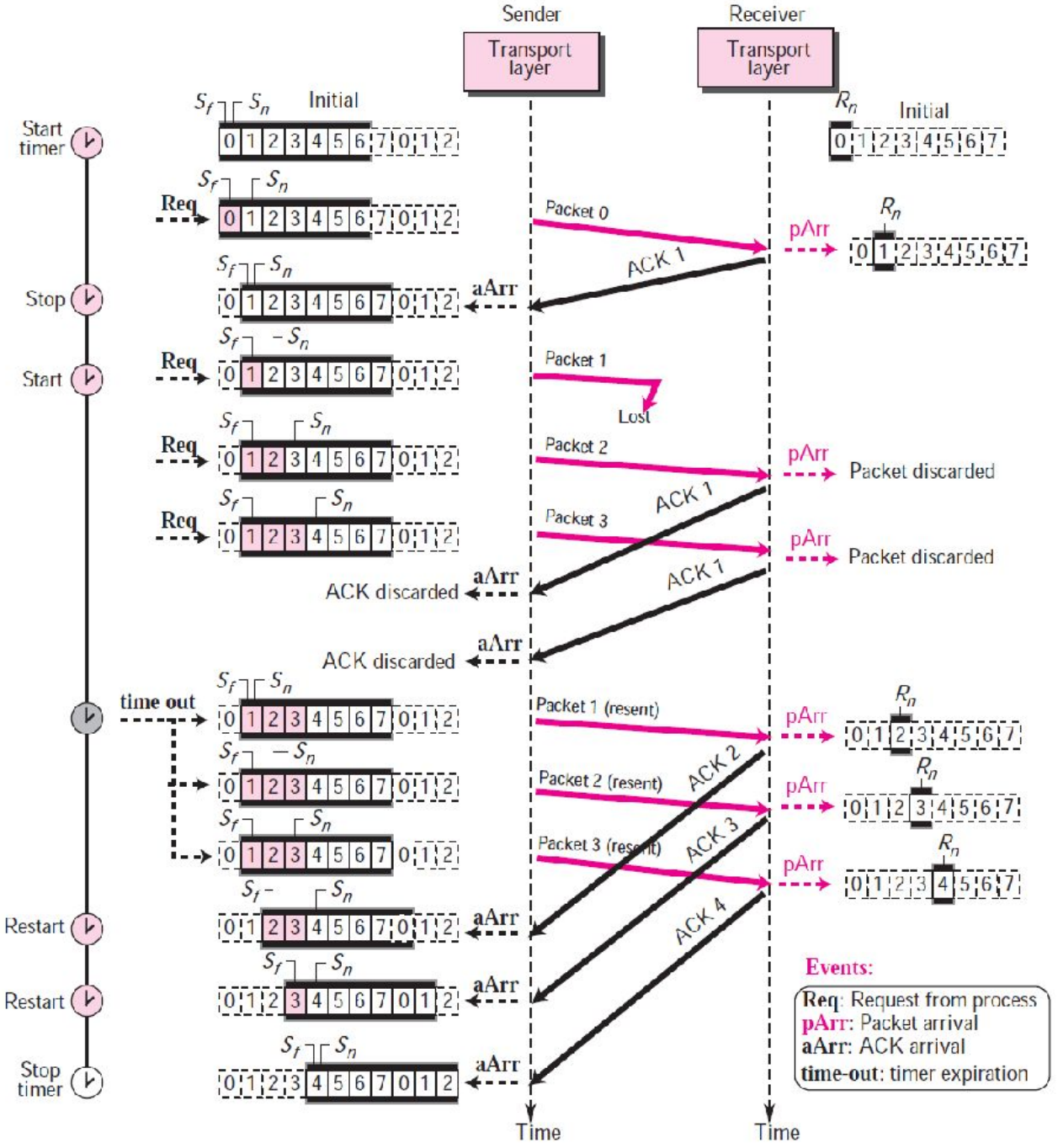


Figure 3.22 ♦ Go-Back-N in operation

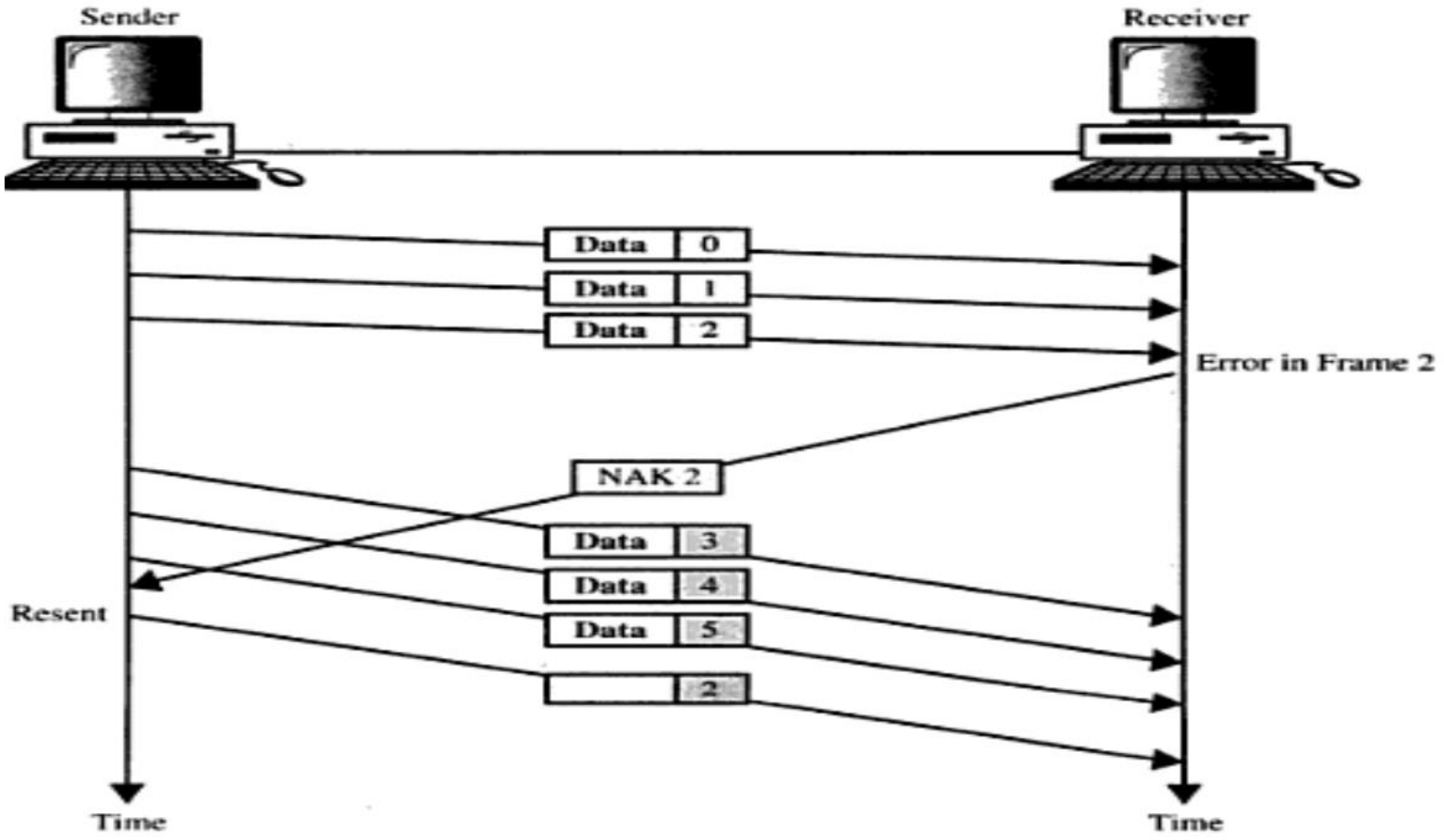

Sender
Transport layer
Receiver
Transport layer
Start timer
Initial
Req
Packet 0
pArr
ACK 1
aArr
Packet 1
ACK 2
Packet 2
Lost
ACK 3
Packet 3
Restart
ACK 4
Stop timer
Time
Time
Events:
Req: Request from process
pArr: Packet arrival
aArr: ACK arrival

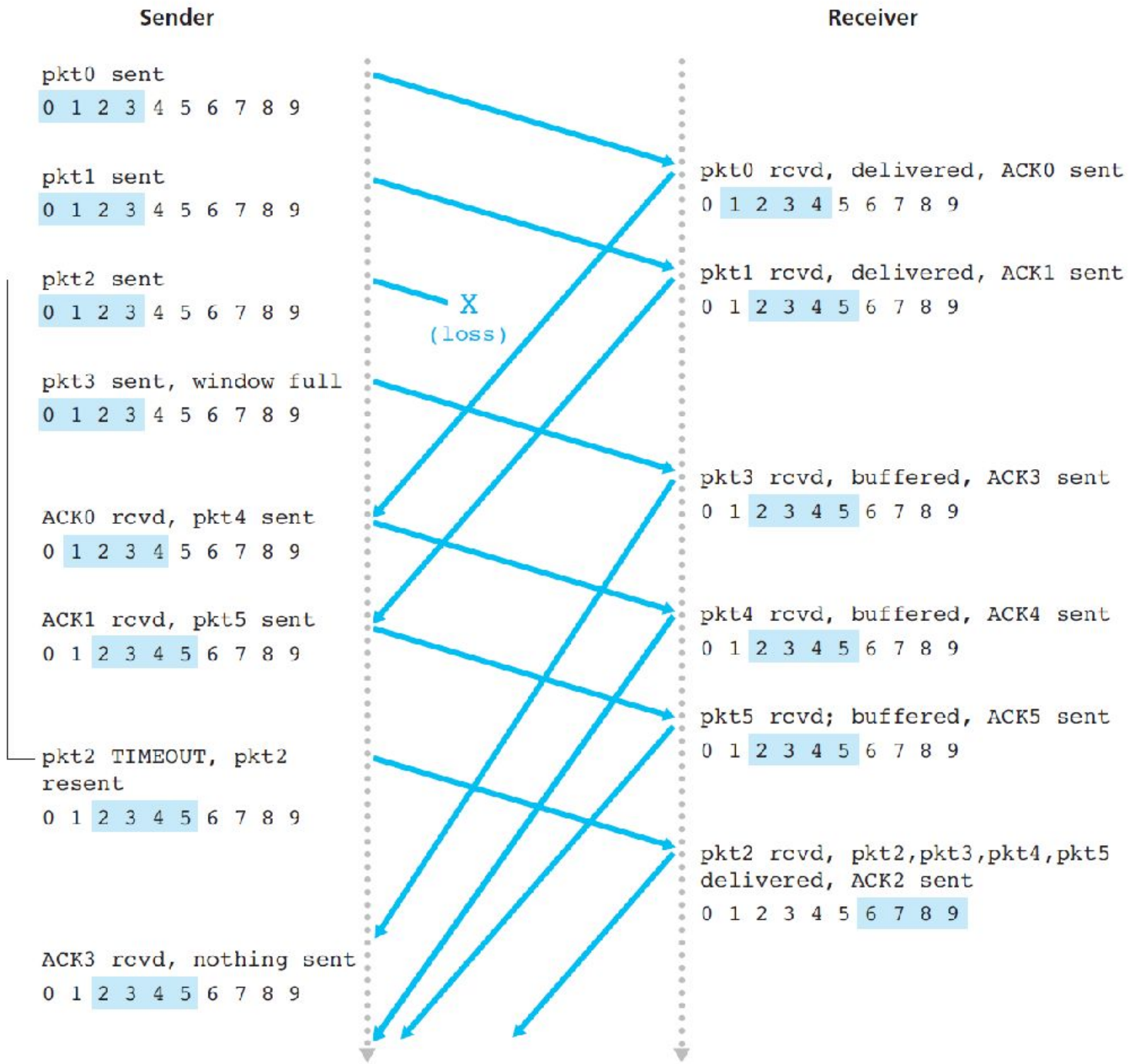

Sender
Receiver
Transport layer
Transport layer
Initial
Start timer
Stop
Start
Restart
Restart
Stop timer
Req
Packet 0
ACK 1
aArr
pArr
Packet 1
Lost
Packet 2
Packet discarded
Packet 3
ACK discarded
time out
Packet 1 (resent)
Packet 2 (resent)
Packet 3 (resent)
ACK 2
ACK 3
ACK 4
Events:
Req: Request from process
pArr: Packet arrival
aArr: ACK arrival
time-out: timer expiration
Time
Time

3. Selective Repeat

- In Selective Repeat, only the **specific damaged or lost** frame is **retransmitted**.
- Receiving device must contain sorting logic to enable it to reorder frames received out of sequence.
  - Sending device must contain a searching mechanism that allows it to find and select only requested frame for retransmission.

Selective reject, damaged data frame




Sender
Receiver
pkt0 sent
0 1 2 3 4 5 6 7 8 9
pkt1 sent
0 1 2 3 4 5 6 7 8 9
pkt2 sent
0 1 2 3 4 5 6 7 8 9
X
(loss)
pkt3 sent, window full
0 1 2 3 4 5 6 7 8 9
ACK0 rcvd, pkt4 sent
0 1 2 3 4 5 6 7 8 9
ACK1 rcvd, pkt5 sent
0 1 2 3 4 5 6 7 8 9
pkt2 TIMEOUT, pkt2
resent
0 1 2 3 4 5 6 7 8 9
ACK3 rcvd, nothing sent
0 1 2 3 4 5 6 7 8 9
pkt0 rcvd, delivered, ACK0 sent
0 1 2 3 4 5 6 7 8 9
pkt1 rcvd, delivered, ACK1 sent
0 1 2 3 4 5 6 7 8 9
pkt3 rcvd, buffered, ACK3 sent
0 1 2 3 4 5 6 7 8 9
pkt4 rcvd, buffered, ACK4 sent
0 1 2 3 4 5 6 7 8 9
pkt5 rcvd; buffered, ACK5 sent
0 1 2 3 4 5 6 7 8 9
pkt2 rcvd, pkt2,pkt3,pkt4,pkt5
delivered, ACK2 sent
0 1 2 3 4 5 6 7 8 9

In **Go-back-N ARQ**, the size of the sender window **must be less than 2^m**, where m is the number of bits used for the Representation of sequence numbers. **why the size of the sender window must be less than 2^m**

- the sequence numbers used modulo 2^*m*, a circle can represent the sequence number from 0 to $2^m - 1$

In the Stop-and-Wait protocol, we can use a 1-bit field to number the packets. The sequence numbers are based on modulo-2 arithmetic.

In the Go-Back-*N* Protocol, the sequence numbers are modulo 2^m, where *m* is the size of the sequence number field in bits.

In the Go-Back-*N* protocol, the acknowledgment number is cumulative and defines the sequence number of the next packet expected to arrive.

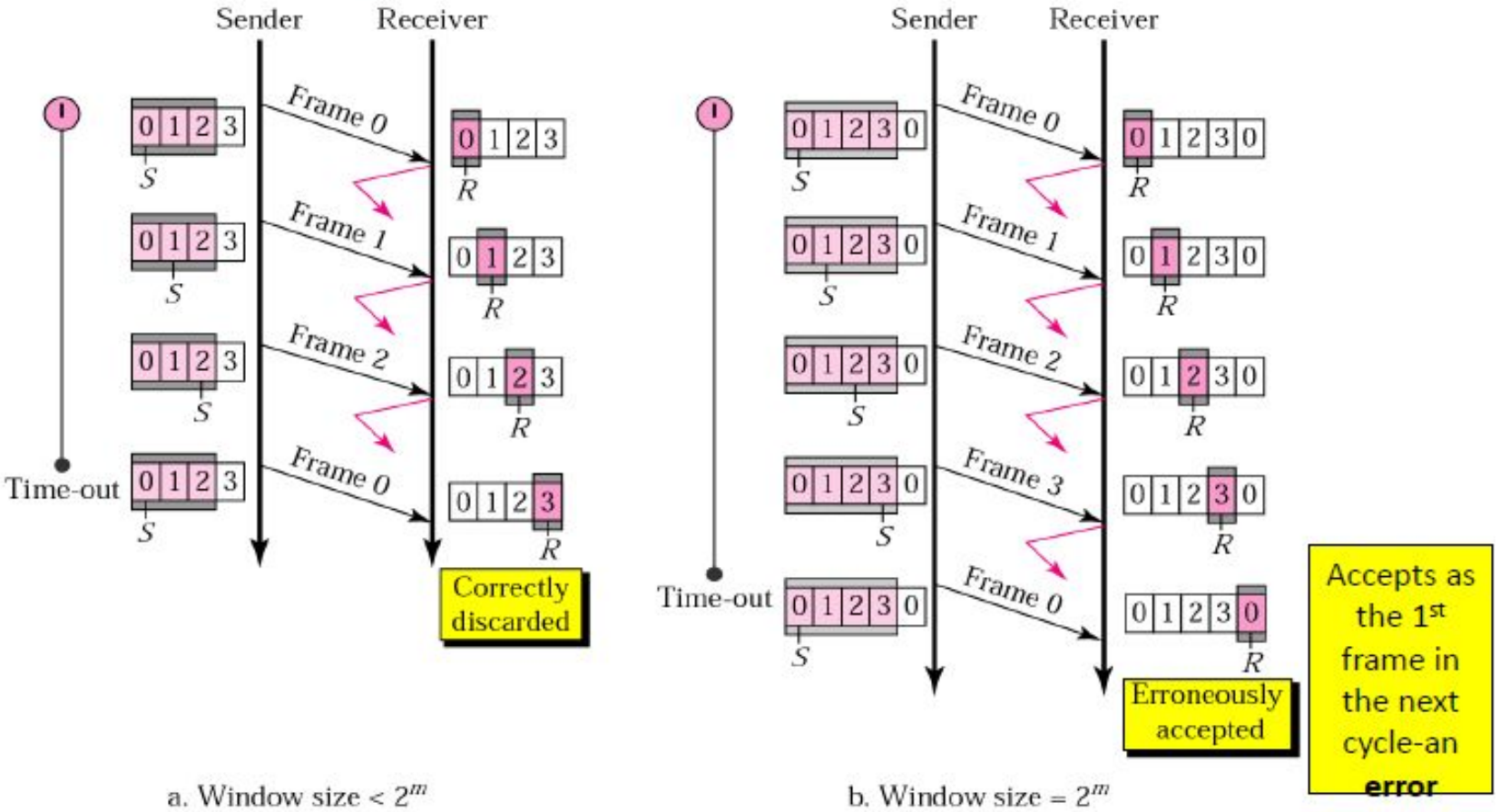
The send window is an abstract concept defining an imaginary box of maximum size = $2^m - 1$ with three variables: S_f, S_n, and S_{size}.

In the Go-Back-*N* protocol, the size of the send window must be less than 2^m; the size of the receive window is always 1.

In Selective-Repeat, the size of the sender and receiver window can be at most one-half of 2^m.

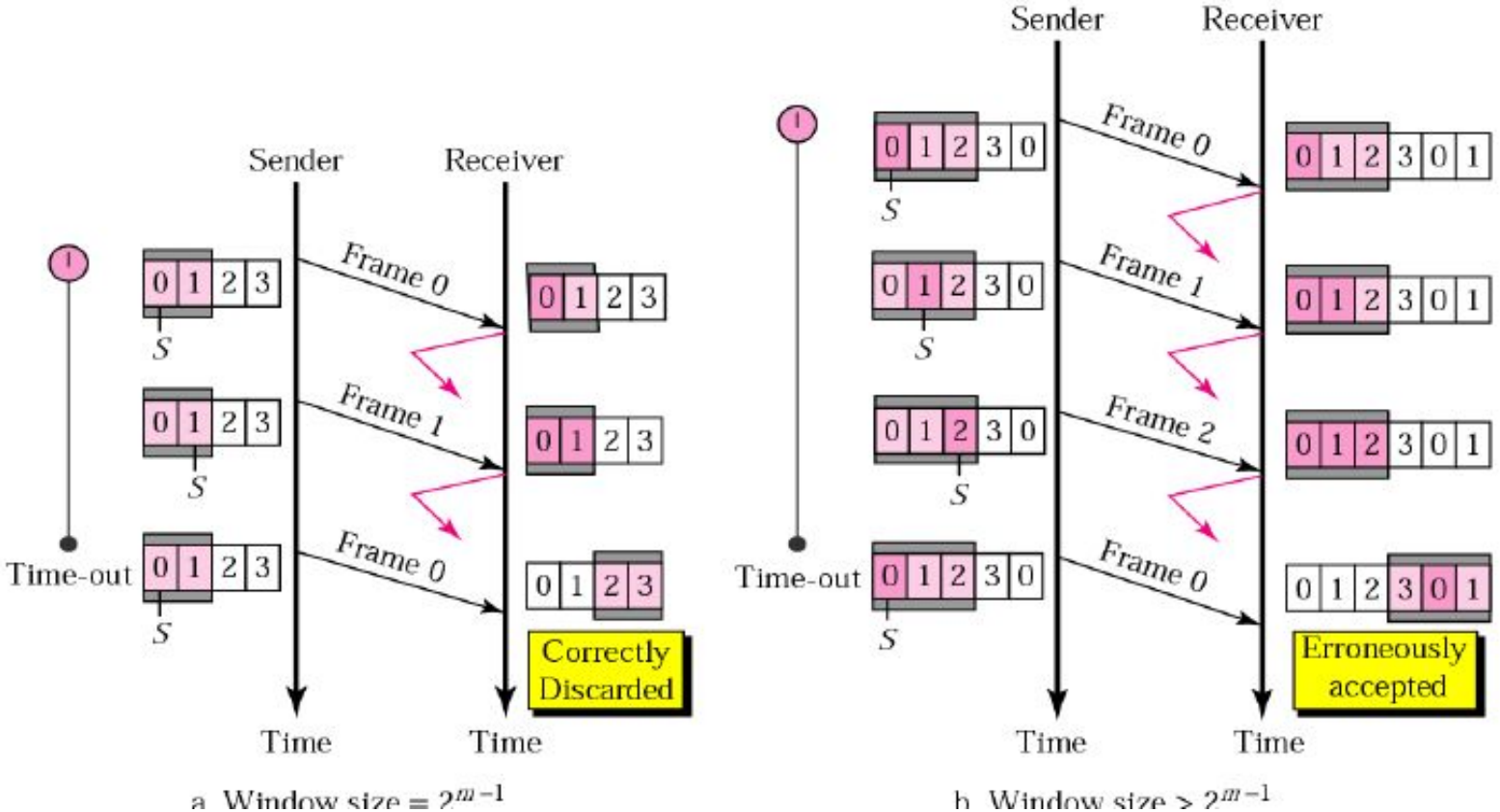
Go-Back-N ARQ, sender window size

- Size of the sender window must be less than 2^m. Size of the receiver is always 1. If m = 2, window size = $2^m - 1 = 3$.
- Fig compares a window size of 3 and 4.



Selective Repeat ARQ, sender window size

- Size of the sender and receiver windows must be at most one-half of 2^m. If m = 2, window size should be $2^m/2 = 2$. Fig compares a window size of 2 with a window size of 3. Window size is 3 and all ACKs are lost, sender sends duplicate of frame 0, window of the receiver expect to receive frame 0 (part of the window), so accepts frame 0, as the 1st frame of the next cycle – an **error**.



Sender and Receiver Window size

	Sender Window	Receiver Window
Stop and Wait	1	1
Go Back n	$2^n - 1$	1
Selective Repeat	2^{n-1}	2^{n-1}

Sliding Window?

- Window spans a portion of buffer containing bytes received from the process.
- The bytes inside the window are that can be in transit, they can be sent without worrying about ack.
- Windows has two walls: left and right
- Windows can be: opened, closed and shrunk
- Size of the window = min [rwnd, cwnd]
- Sliding window uses identification scheme based on size of window for keeping the track of transmitted and received segments. i.e. for modulus n 0 to n-1

Questions:

What is the value of the receiver window (rwnd) for host A if the receiver, host B, has a buffer size of 5,000 bytes and 1,000 bytes of received and unprocessed data?

Solution

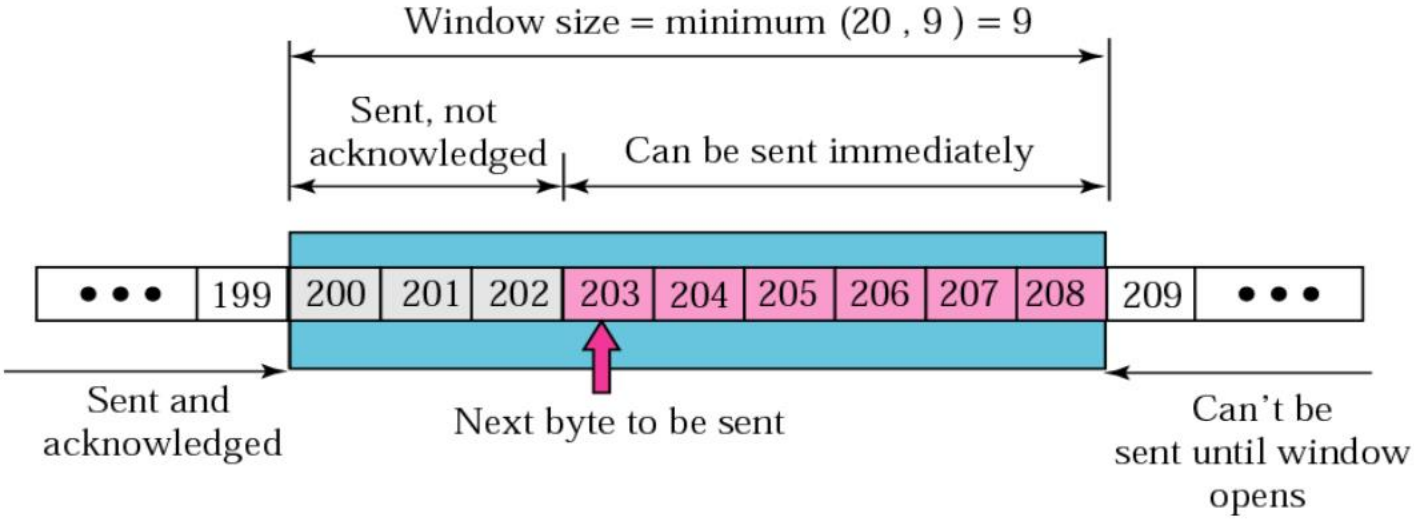
The value of rwnd = 5,000 − 1,000 = 4,000. Host B can receive only 4,000 bytes of data before overflowing its buffer. Host B advertises this value in its next segment to A.

What is the size of the window for host A if the value of rwnd is 3,000 bytes and the value of cwnd is 3,500 bytes?

Solution

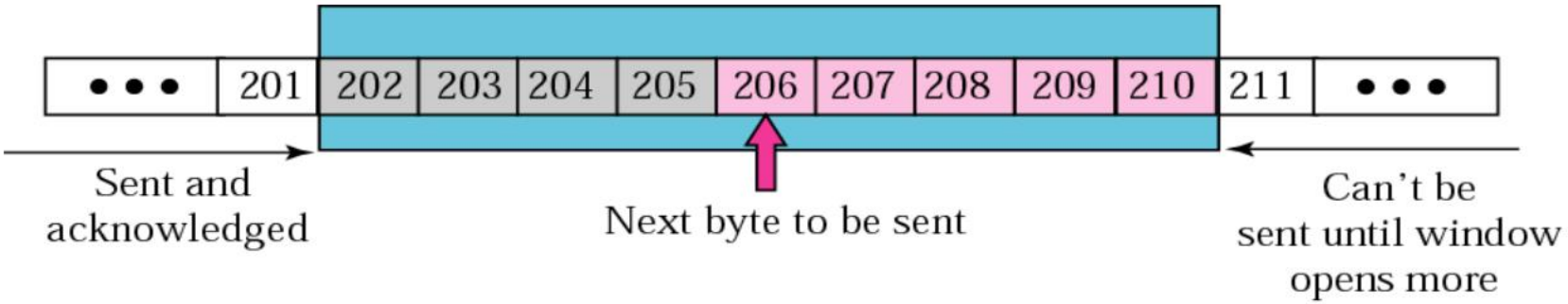
The size of the window is the smaller of rwnd and cwnd, which is 3,000 bytes.

Figure below shows an unrealistic example of a sliding window. The sender has sent bytes up to 202. We assume that cwnd is 20 (in reality this value is thousands of bytes). The receiver has sent an acknowledgment number of 200 with an rwnd of 9 bytes (in reality this value is thousands of bytes). The size of the sender window is the minimum of rwnd and cwnd or 9 bytes. Bytes 200 to 202 are sent, but not acknowledged. Bytes 203 to 208 can be sent without worrying about acknowledgment. Bytes 209 and above cannot be sent.



In previous Figure the server receives a packet with an acknowledgment value of 202 and an rwnd of 9. The host has already sent bytes 203, 204, and 205. The value of cwnd is still 20. Show the new window.

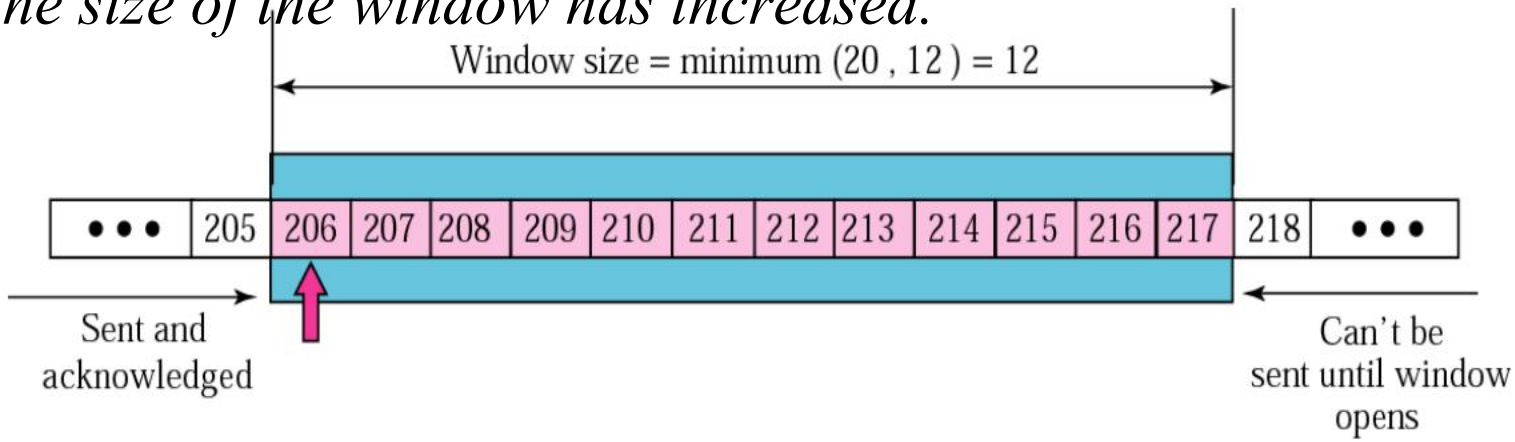
Solution:



Now In previous Figure the sender receives a packet with an acknowledgment value of 206 and an rwnd of 12. The host has not sent any new bytes. The value of cwnd is still 20. Show the new window.

Solution

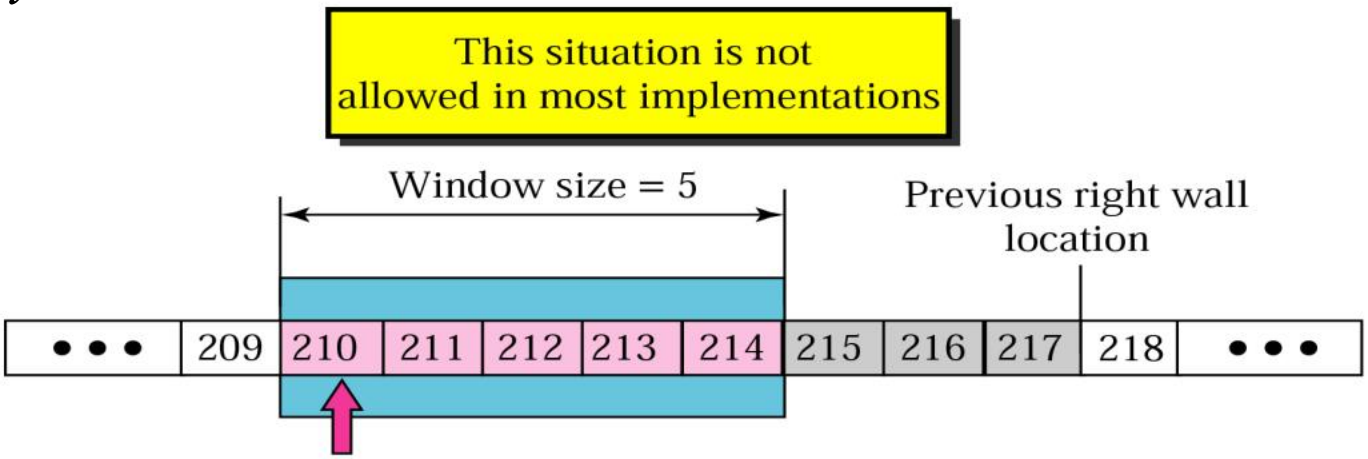
The value of rwnd is less than cwnd, so the size of the window is 12. Figure below shows the new window. Note that the window has been opened from the right by 7 and closed from the left by 4; the size of the window has increased.



In previous Figure the host receives a packet with an acknowledgment value of 210 and an rwnd of 5. The host has sent bytes 206, 207, 208, and 209. The value of cwnd is still 20. Show the new window.

Solution

The value of rwnd is less than cwnd, so the size of the window is 5. Figure below shows the situation. Note that this is a case not allowed by most implementations. Although the sender has not sent bytes 215 to 217, the receiver does not know this.



How can the receiver avoid shrinking the window in the previous example?

Solution

The receiver needs to keep track of the last acknowledgment number and the last rwnd. If we add the acknowledgment number to rwnd we get the byte number following the right wall. If we want to prevent the right wall from moving to the left (shrinking), we must always have the following relationship.

$$\textbf{new ack} + \textbf{new rwnd} \geq \textbf{last ack} + \textbf{last rwnd}$$

or

$$\textbf{new rwnd} \geq (\textbf{last ack} + \textbf{last rwnd}) - \textbf{new ack}$$

Note:

To avoid shrinking the sender window, the receiver must wait until more space is available in its buffer.

Note:

Some points about TCP's sliding windows:

- *The size of the window is the lesser of rwnd and cwnd.*
- *The source does not have to send a full window's worth of data.*
- *The window can be opened or closed by the receiver, but should not be shrunk.*
- *The destination can send an acknowledgment at any time as long as it does not result in a shrinking window.*
- *The receiver can temporarily shut down the window; the sender, however, can always send a segment of one byte after the window is shut down.*

Sliding window: more

- Window Shutdown
- Probing (to prevent deadlock)
- Silly Window syndrome
- Nagle's Algorithm
- Clark's solution

Sliding Window : Window Shutdown

- Window Shutdown
  - Receiver can temporarily shutdown the window by sending an rwnd of 0.
  - Sender can always send segment with one byte of data. (called probing, use to prevent dead lock)

Sliding Window : Silly Window Syndrome

- Sending application creates data slowly or receiving application consumes data slowly.
- Result : sending of data in very small segments.
- Example : 41 bytes with 1 byte of data
- SWS : When AdvertisedWindow < MSS
- How to avoid it?
  - not to introduce a small segment
  - receiver waits till MSS space is available before advertizing a window open from zero

Silly Window Syndrome – Sender Side

- Occurs at Sender Side
- Sending data is very small
- E.g., 41-byte datagram ➔ (20 bytes of TCP header, 20 bytes of IP header and 1byte of data)
  - overhead 41/1; using network capacity very inefficiently

Solution: Nagel's Algorithm

Nagle's Algorithm **Nagle's algorithm** is simple:

1. The sending TCP sends the first piece of data it receives from the sending application program even if it is only 1 byte.
2. After sending the first segment, the sending TCP accumulates data in the output buffer and waits until either the receiving TCP sends an acknowledgment or until enough data has accumulated to fill a maximum-size segment. At this time, the sending TCP can send the segment.
3. Step 2 is repeated for the rest of the transmission. Segment 3 is sent immediately if an acknowledgment is received for segment 2, or if enough data have accumulated to fill a maximum-size segment.

- If the application program is faster than the network the segments are larger(maximum size segment).
- If the application program is slower than the network the segments are smaller(less than the maximum size segment size).

Syndrome Created by the Receiver

- Receiver TCP → silly window syndrome → if it is serving an application program that consumes data slowly, E.g. 1 byte at a time.
- Clark's Solution:
  - Send an acknowledgment as soon as the data arrive, but to announce a window size of zero until either there is enough space to accommodate a segment of maximum size or until at least half of the receive buffer is empty.
- Delayed Acknowledgment:
  - Delay sending the acknowledgments; waits decent amount of space in its incoming buffer.
  - Disadvantages: sender unnecessarily retransmitting the unacknowledged segments.

Questions:

1. In stop-and-wait system, bandwidth of the line is 1 Mbps and 1 bit takes 20 ms to make a round trip. What is the bandwidth-delay product? If the system data frames are 1000bits in length what is the utilization percentage of the link?

2. What is the Utilization percentage of the link if the link uses go-back-n with a 15 frame sequence?

3. Consider transferring an enormous file of L bytes from Host A to Host B.

a) What is the maximum value of L such that TCP sequence numbers are not exhausted? Recall that the TCP sequence number field has 4 bytes.

b) Assume an MSS of 1460 bytes. For the L you obtain in (a), find how long it takes to transmit the file. Assume that a total of 66 bytes of transport, network, and data-link header are added to each segment before the resulting packet is sent out over a 10 Mbps link. Ignore flow control and congestion control so A can pump out the segments back.to back and continuously.

Note: MSS is the maximum size of user data carried by any segment in a TCP connection. In SYN segment, using TCP options header, each TCP entity informs to the other about which MSS wants to use. The smaller will be used.

(a) There are $2^{32} = 4,294,967,296$ possible sequence numbers.

The sequence number does not increment by one with each segment. Rather, it is the increment by the number of bytes of data sent. So the size of the MSS is irrelevant -- the maximum size file that can be sent from A to B is simply the number of bytes representable by 2^{32}. $2^{32} \approx 4.19$ Gbytes

b) The number of segments is $\left\lceil \frac{2^{32}}{1460} \right\rceil = \frac{\mathbf{4294967296}}{1460} = 2,941,758.$

66 bytes of header get added to each segment giving a total of 194,156,028 bytes of header. The total number of bytes transmitted is $2^{32} + 194,156,028 = 3,591 \times 10^7$ bits.

To transmit the file over a 10-Mbps link, it would take 3,591 seconds = 59 minutes

4. A system uses the Stop-and-Wait ARQ Protocol. If each packet carries 1000 bits of data, how long does it take to send 1 million bits of data if the distance between the sender and receiver is 5000 Km and the propagation speed is 2 x 108 m? Ignore waiting, and processing delays. We assume no data or control frame is lost or damaged.

16. Repeat above Exercise using the Go-back-N ARQ Protocol with window size of 7. Ignore the overhead due to the header and trailer.

17. Repeat above using the Selective-Repeat ARQ Protocol with a window size of 4. Ignore the overhead due to the header and the trailer.

We need to send 1000 frames. We ignore the overhead due to the header and trailer.

Data frame Transmission time = 1000 bits / 1,000,000 bits = 1 ms
Data frame trip time = 5000 km / 200,000 km = 25 ms
ACK transmission time = 0 (It is usually negligible)
ACK trip time = 5000 km / 200,000 km = 25 ms
Delay for 1 frame = 1 + 25 + 25 = 51 ms.
Total delay = 1000 × 51 = **51 s**

In the worst case, we send the a full window of size 7 and then wait for the acknowledgment of the whole window. We need to send 1000/7 ≈ 143 windows. We ignore the overhead due to the header and trailer.

Transmission time for one window = 7000 bits / 1,000,000 bits = 7 ms
Data frame trip time = 5000 km / 200,000 km = 25 ms
ACK transmission time = 0 (It is usually negligible)
ACK trip time = 5000 km / 200,000 km = 25 ms
Delay for 1 window = 7 + 25 + 25 = 57 ms.
Total delay = 143 × 57 ms = **8.151 s**

In the worst case, we send the a full window of size 4 and then wait for the acknowledgment of the whole window. We need to send 1000/4 = 250 windows. We ignore the overhead due to the header and trailer.

Transmission time for one window = 4000 bits / 1,000,000 bits = 4 ms
Data frame trip time = 5000 km / 200,000 km = 25 ms
ACK transmission time = 0 (It is usually negligible)
ACK trip time = 5000 km / 200,000 km = 25 ms
Delay for 1 window = 4 + 25 + 25 = 54 ms.
Total delay = 250 × 54 ms = **13.5 s**

Example-3: Bandwidth of channel is given as 1 GBps. How long can a packet stay in link without worrying about problem of having 2 packets with same sequence numbers?

Explanation –

Bandwidth = 1 GBps = 2^{30}

Sequence numbers = 2^{32}

So, Wrap around time:

= Sequence number/Bandwidth

=2^{32} / 2^{30}

=2^{2}

=4 seconds

https://www.geeksforgeeks.org/wrap-around-concept-and-tcp-sequence-number/

- https://www.geeksforgeeks.org/sliding-window-protocol-set-2-receiver-side/
- https://www.geeksforgeeks.org/sliding-window-protocol-set-3-selective-repeat/
- https://www.youtube.com/watch?v=glRgoKMYOKM

- https://slideplayer.com/slide/12881656/