Fa24+Assignment+2 +sol+part+1 / Module+3+lesson+plan+Fa24

Module 3 Transport Layer

A transport protocol is said to be reliable if (the service abstraction it provides to the upper-layer is) "no transferred data bits are corrupted (flipped from 0 to 1 or vice versa) or lost, and all are delivered in the order in which they were sent".

- Acknowledgments are needed from receiver to sender. Why?
- Disregarded NAK, and we kept using ACK. Why? How can we tell the sender that a packet is corrupted?
- Question 1: b) In the rdt protocols, why did we need to introduce timers?
  - Timers are needed to handle losses in the channel. If the ACK for a transmitted packet is not received within the duration of the
- What is a stop-and-wait protocol? Do you think it is widely implemented in network design?
- Discussion 31 The sender Utilization of a stop-and-wait protocol is defined as the fraction of time the sender is actually busy sending bits into the channel.
- Question 2
- Go-Back-N (GBN) protocol,
  - Protocol Used: Go-Back-N. Sliding Window size = 4. Timeout interval = 2RTT. Assume transmission time of a packet = 1/3 RTT. Assume Ack1 is lost, then Packet 2 is lost. As mentioned in Figure 3.20. The sender starts a timer when it transmits each packet. When a new (not duplicated) ACK is received, the base (and window) slides to the number of ACK+1.
  - Go-Back-N Solution
- Selective Repeat
  - a. Have the source send five packets, and then pause the animation before any of the five packets reach the destination. Then kill the first packet and resume the animation. Describe what happens.
    - Answer: Packet 0 is lost. Packets 1, 2, 3, 4 are stored in receiver, and ACK 1, 2, 3, 4 are sent to sender. Sender marks them as received.
    - After timeout: Pkt 0 is sent again, then when pkt 0 arrives at receiver, pkt 0, 1, 2, 3, 4 are delivered, and ACK 0 is sent to sender.
  - b. Repeat the experiment, but now let the first packet reach the destination and kill the first acknowledgment. Describe again what happens.
    - pkt 0,1,2,3,4 are received at receiver. ACK 1, 2, 3, 4, are sent to sender. After time out, pkt 0 is sent to receiver, then ACK 0 is sent back to sender, and receiver delivers all packets in order to upper layer.
    - c. Finally, try sending six packets. What happens? Not allowed
- Disadvantage of Go back N:
  - In GO-Back-N, out-of-order packets are discarded. When a packet was lost, GO-Back-N retransmitted all the packets in the sliding window, whereas Selective Repeat retransmitted the lost packet only.
- Advantage of GBN

Fa24+Assignment+2 +sol+part+2

- Problem 1
- Problem 2
- Problem 3

Module+4+lesson+plan+Fa24

Network programming TCP

- The ISO OSI Model
  - Encapsulation
    - Socket: end point of communication
- Types of communications
  - What defines a unique connection?
  - Does TCP listen on one port and talk on another port?
  - What is the maximum number of concurrent TCP connections that a server can handle?
  - Concurrent connection request vs Concurrent open connection
    - What are active and passive socket?
- Client-server communication Stream sockets: connection-oriented
  - Stream socket communication
  - Generic example for a stream socket
  - Client-server communication Datagram sockets: connectionless
- Client-server communication Datagram sockets: connectionless
- Question: At this time, the TCP client creates a socket and sends a connection establishment request. What are the statements to do that?
- Question: Server should always be up and running and listening to the incoming connections. Once it receives the client connection request, what should it do?
- Connection-Oriented Transport: TCP
- Flow Control of TCP

Congestion Control of TCP

- TCP slow start
- TCP congestion-control algorithm
- when should this exponential growth end?
- Example:
  - Congestion Avoidance state
    - Leaving Congestion Avoidance state
- Fast Recovery
  - Fast-recovery state
  - Leaving fast-recovery state
- True or false?
- Checksum
  - At the receiver
  - At the receiver: in case of single-bit error
  - Checksum

Module3+Transport+Layer+lesson+plan+Fa24

Module 3 Transport Layer

1

- Acknowledgments are needed from receiver to sender. Why?

2

- isregarded NAK, and we kept using ACK. Why? How can w
ell the sender that a packet is corrupted?

5

Question 1:

b) In the *rdt* protocols, why did we need to introduce timers?

5

- Timers are needed to handle losses in the channel. If the ACK for a transmitted packet is not received within the duration of the

6

- What is a stop-and-wait protocol? Do you think it is widely implemented in network design?

6

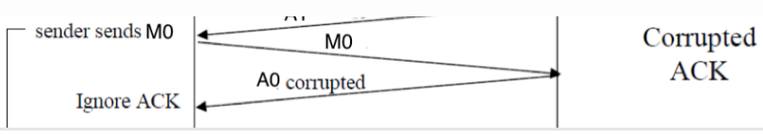
is defined as the fraction of time the sender is actually busy sending bits into the channel:

6

-

7

8

- 

8

14

Go-Back-N (GBN) protocol 1

- 

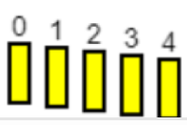
15

acknowledgment

16

- number n-1), the receiver **sends an ACK** for packet *n* and **delivers the data** portion of the packet to the upper layer.

17



yellow) are sent from sender to receiver

18

19

b) Packet 0 is selected then killed

20

c) packet 0 is lost

20

d) receiver discarded all packets.

Nothing is kept in receiver

21

e) After timeout, sender sends all 5 packets: the lost packet and all

22

23

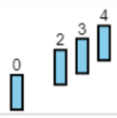
g) the 5 ACK are received by sender. The window slides 5 positions to the

24

a) ACK0 is lost

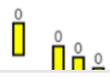
25

sender, all packets 0 to 4 are marked as acknowledged, and window slides to right.
:An ACK of packet i when received by



26

A) packet 1 is lost



27

B) All acknowledgements have the sequence # 0 in them (0 is the last packet received correctly at receiver)

28

C) Only packet0 is acknowledged. Sliding window advances one position only to the right.

What happens if packet 5 is sent?

(why packets 1, 2, 3, 4 are not

29

29

F) After timout, all 5 unacknowledged packets (1 to 5) are retransmitted

30

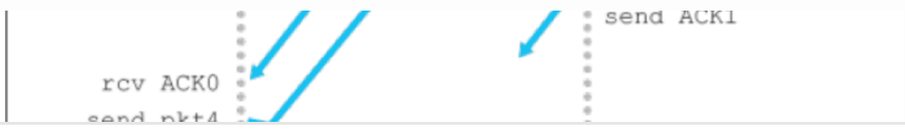
31

G) packets 1 to 5 are acknowledged and the window moves to 6 to 10

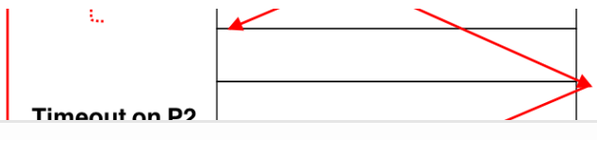
32

Assume Ack1 is lost, then Packet 2 is lost.

As mentioned in Figure 3.20: The sender starts a timer when it



33



34

Selective Repeat

35

operation of multiple logical timers [Varghese 1997].

3. *ACK received*. If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window. If the packet's sequence

35

eiver buffers

1st

37

animation before any of the five packets reach the destination.

38

Then kill the first packet and resume the animation.

Describe what happens.

38

39

destination and kill the first acknowledgment. Describe again what happens.

39

upper layer.

39

Disadvantage of Go back N:

39

at the receiver.

This individual, as-needed, retransmission will require that the **receiver acknowledges correctly received packets**, whether or

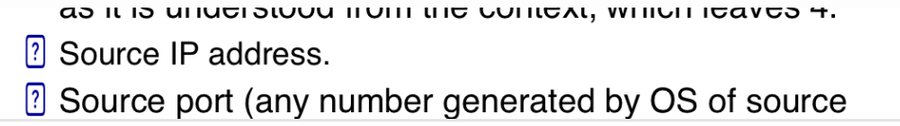
40

packet was lost, GO-Back-N retransmitted all the packets in the sliding window. whereas Selective Repeat retransmitted the lost

40

selective repeat sent a duplicate ACK and as **GO-Back-N used cumulative acknowledgment**, so that duplicate ACK was unnecessary

43

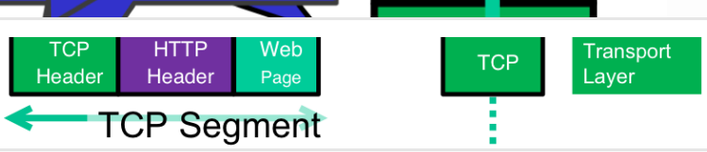


Module+4+lesson+plan+Fa24

(1) 1

TCP

(1) 4



(1) 14

These processes communicate with each other by reading from, and writing to, sockets.

(1) 18

Treats data as a continuous Stream of bytes

Reliable

Connection-oriented (need to establish connection

(1) 21

as it is understood from the context, which leaves 4:

Source IP address.

Source port (any number generated by OS of source

(1) 22

If clients make multiple TCP connection to server, It's the client OS that must generate **different random source ports**, so that server can see them

(1) 23

same — only the source port varies to differentiate the different connections.

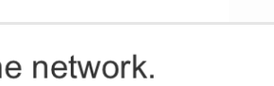
Theoretically, Ports are 16-bit numbers, therefore the

(about 5–10), and this size limits the number of **concurrent connection requests**.

A passive socket is not con-nect-ed, but rather awaits an in-com-ing con-nec-tion (it is a listening

Client-server communication
Stream sockets: connection-oriented

When the server "hears" the knocking, it creates a new door, more precisely, a ***new* socket**, that is **dedicated to that connection of this particular client**.



30

Client-server communication
Datagram sockets: **connectionless**

At this time, the TCP client **creates a socket** and **sends a connection establishment request**: What

listening to the incoming connections. Once it receives the client connection request, what

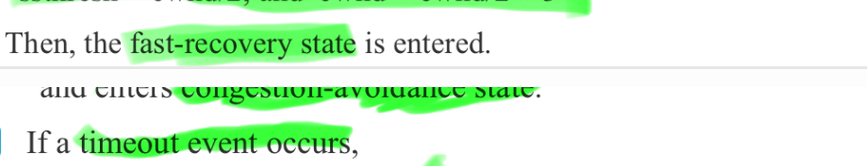
rwnd=RcvBuffer–[LastByteRcvd–LastByteRead]

Because the spare room changes with time, *rwnd* is dynamic.

Congestion Control of TCP

rate at which a TCP sender can send traffic into the network.

MSS (maximum segment size) is the maximum amount of application-layer data in the segment



when should this exponential growth end?

Data[3] got lost, which is why we are stuck on ACK[2]

TCP sender increases *cwnd* by (MSS/ *cwnd*) * MSS whenever a new acknowledgment arrives.

Example: if MSS = 1,460 bytes and *cwnd* is 14,600 bytes, then 10

Then, Congestion Avoidance transitions to the slow-start state

When a triple duplicate ACK is received,

The idea is to use the arriving dupACKs to pace retransmission. We assume that each arriving dupACK indicates that some data packet following the lost packet

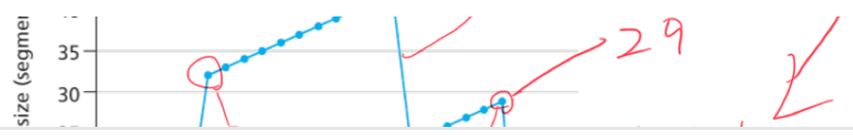
ssthresh = cwnd/2; and cwnd = cwnd/2 + 3

Then, the fast-recovery state is entered.

and enters congestion-avoidance state.

If a timeout event occurs,

ssthresh = cwnd/2 (cwnd is the value when the loss event occurred)



Answer:
False, it is set to half of the current value of the congestion window: ssthresh = cwnd/2

(1) 86

1 + 1 + 1 = 1 1

Figure 7.6 Rules of binary addition.

1 1 1
0 1 1

(1) 88

1 1 1 1 1
0 1 0 1 0 0 1 1
1 0 1 1 1 0 0 1

(1) 90

1 0 1 1 1 0 0 1
1 0 0 1 0 1 1 0 1
0 0 1 0 1 1 0 1
\+ 1
0 0 1 0 1 1 1 0

(1) 91

checksum
1 0 1 1 1 1 1 1 1 : a 0 in answer => error

(1) 93

All one-bit errors will be detected, but two-bit errors can be undetected (e.g., if the last digit of the first word is converted to a 0 and the last

- Fa24+Assignment+2+sol+part+1

Fa24+Assignment+2+sol+part+2

(3) 1

propagation delay).
The data frames carry a useful payload of size 10,000 bits. Assume that both ACK and data frames have 400 bits of header information, and that ACK frames carry no data.

(3) 1

The cycle time for stop and wait operation is T_{frame} + Tp+ T_{ACK} + Tp = 10,400bits/400,000bps+3sec+400bits/400,000bps+3 = 6.027 sec

(3) 1

The cycle time for sliding window is T_{frame} + Tp+ T_{ACK} + Tp = 10,400/400,000+3+400/400,000 +3 = 6.027sec

(3) 2

Annotate fully your graph to show the above requirements.

Solution:
Important points that should be present in the diagram:

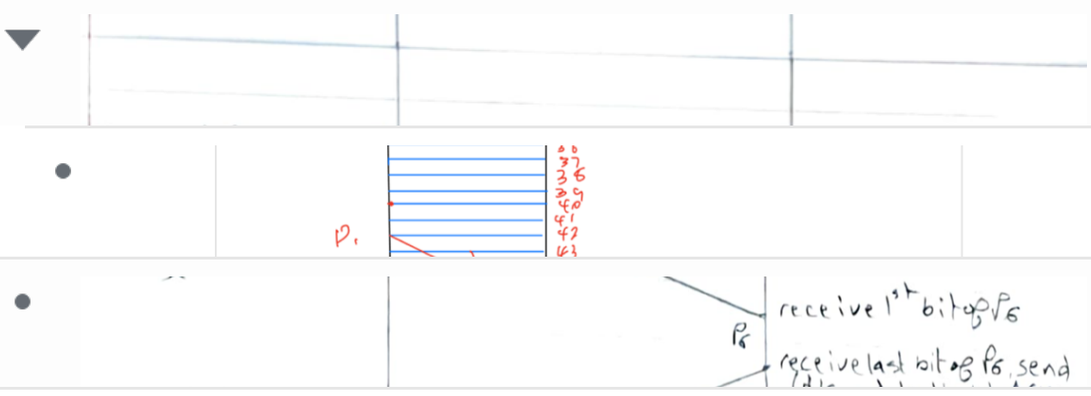
(2) 1

(4) 2

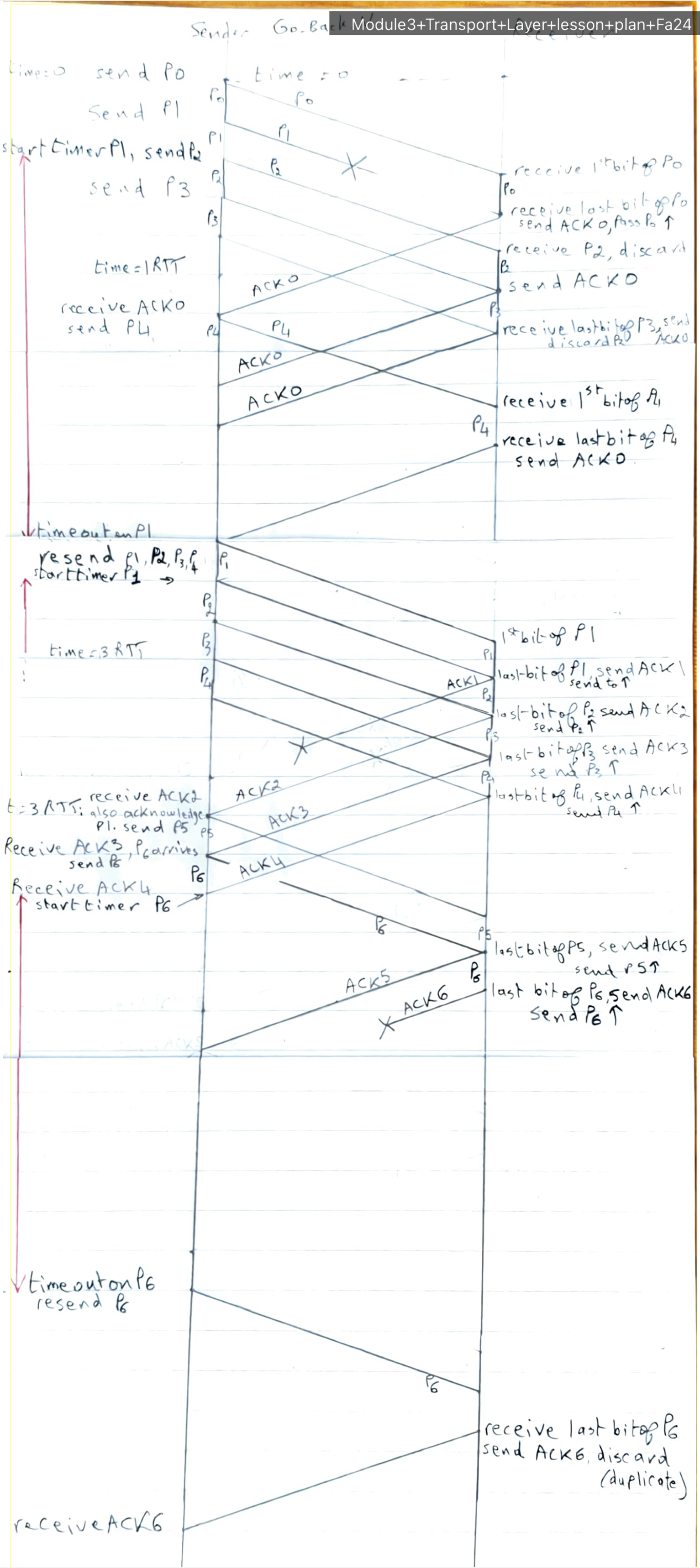
(3) 3

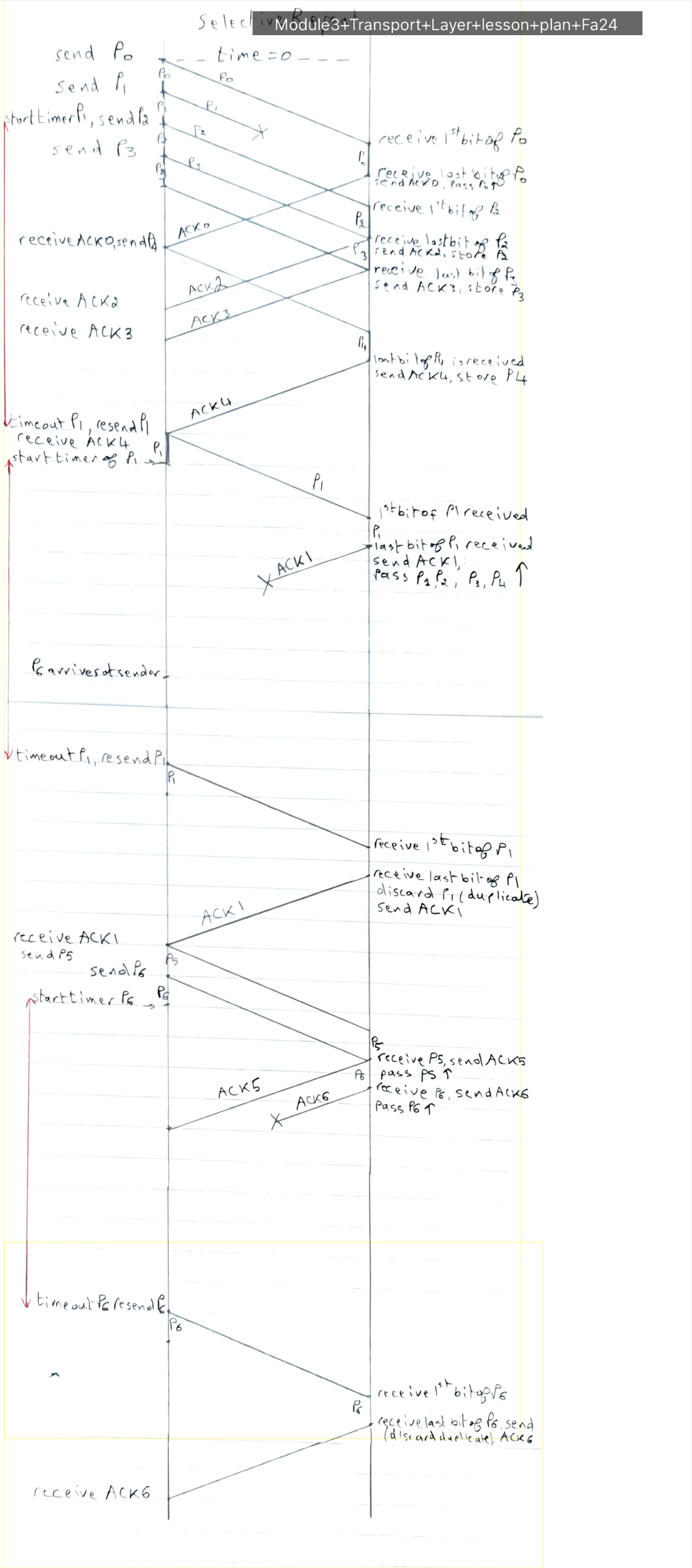
sends ACK for each one of these packets
- A timer is started for P1 when P1 is transmitted.
- On timeout of P1 timer, only P1 is retransmitted.

(4) 3



- E7668CFE-8171-4D2B-81EC-DFF53266DC7B


Sender Go-Back-N
Receiver
time=0 send P0
Send P1
start timer P1, send P2
send P3
time=1 RTT
receive ACK0
send P4
time out on P1
resend P1, P2, P3, P4
start timer P1
time=3 RTT
t=3 RTT: receive ACK2
also acknowledge P1. send P5
Receive ACK3, P6 arrives
send P6
Receive ACK4
start timer P6
time out on P6
resend P6
receive ACK6
time=0
receive 1st bit of P0
receive last bit of P0
send ACK0, Pass P0 ↑
receive P2, discard
send ACK0
receive last bit P3, send ACK0
discard P2
receive 1st bit of P1
receive last bit of P4
send ACK0
1st bit of P1
last bit of P1, send ACK1
send to ↑
last bit of P2 send ACK2
send P2 ↑
last bit of P3 send ACK3
send P3 ↑
last bit of P4, send ACK4
send P4 ↑
last bit of P5, send ACK5
send P5 ↑
last bit of P6, send ACK6
send P6 ↑
receive last bit of P6
send ACK6, discard
(duplicate)
ACK0
ACK0
ACK0
ACK1
ACK2
ACK3
ACK4
ACK5
ACK6
P0
P1
P2
P3
P4
P5
P6


Selective Repeat
send P_0
send P_1
start timer P_1, send P_2
send P_3
receive ACK0, send P_4
receive ACK2
receive ACK3
timeout P_1, resend P_1
receive ACK4
start timer of P_1
P_6 arrives at sender
timeout P_1, resend P_1
receive ACK1
send P_5
send P_6
start timer P_6
timeout P_6 resend P_6
receive ACK6
time=0
P_0
P_1
P_2
P_3
ACK0
ACK2
ACK3
P_4
ACK4
P_1
ACK1
P_5
ACK5
P_6
ACK6
P_1
receive 1st bit of P_0
P_0
receive last bit of P_0
send ACK0, pass P_0 ↑
receive 1st bit of P_2
P_2
receive last bit of P_2
send ACK2, store P_2
P_3
receive last bit of P_3
send ACK3, store P_3
last bit of P_4 is received
send ACK4, store P_4
1st bit of P_1 received
P_1
last bit of P_1 received
send ACK1,
pass P_1, P_2, P_3, P_4 ↑
receive 1st bit of P_1
receive last bit of P_1
discard P_1 (duplicate)
send ACK1
P_5
receive P_5, send ACK5
pass P_5 ↑
P_6
receive P_6, send ACK6
pass P_6 ↑
receive 1st bit of P_6
P_6
receive last bit of P_6, send
(discard duplicate) ACK6

Northeastern University
Khoury College of Computer Sciences
CS 5700 – Fundamentals of Computer Networking
Problem Set 2 Solution
Due Monday October 7, 2024 at 11:59 pm on Canvas.

Problem 1:
Consider a full-duplex 400 Kbps satellite link with a 3 second end-to-end delay (i.e. one way propagation delay).
The data frames carry a useful payload of size 10,000 bits. Assume that both ACK and data frames have 400 bits of header information, and that ACK frames carry no data.
What is the effective data throughput:
a) When using Stop and Wait?

The propagation delay Tpr= 3sec
The cycle time for stop and wait operation
*is T_{frame} + Tp+ T_{ACK} + Tp = 10,400*bits*/400,000*bps+3sec+400bits/400,000bps+3 = 6.027 sec*
effective data throughput is data bits per frame /cycle time = 10,000/6.027 = 1659.2 bps = 1.659 kbps

b) When using sliding windows with a sender window size of *W = 5* data frames?

The propagation delay Tpr= 3sec for W=5 packets.
The cycle time for sliding window
is T_{frame} + Tp+ T_{ACK} + Tp = 10,400/400,000+3+400/400,000 +3 = 6.027sec

effective data throughput is data bits in W frame /cycle time of W frames
*= 5*10,000/6.027 = 8.15kbps=8149.96 bps*

Note: that the **effective data throughput** in reliable data transfer is defined to be the ratio of the amount of useful data sent by the sender divided by the total time elapsed from the moment the first bit of the frame containing this data is transmitted to the moment the last bit of corresponding ACK is received by the sender.

Problem 2:

Draw a space-time diagram similar to Figure 3.22 of textbook for the following problem: Sender is sending 6 packets to receiver (**P0 to P5**) that arrived at time =0. Then packet P6 arrived at time 4RTT. No other packets were received by sender after packet P6. Draw the graph to scale, for example, you can choose RTT to be 5cm or 10cm, and draw everything relative to this scale. Show the transmission time and propagation time in terms of RTT. Show when the first bit and last bit of a packet are received by receiver. Show when a packet is discarded and when it is passed to the upper layer. Show when the sender receives the acknowledgment of each packet and when it starts the timer of each packet.

Protocol Used: Go-Back-N

Assume the following:

- Sliding Window size = 4
- Timeout interval =2RTT
- A timer is started when the last bit of a packet is transmitted.
- Transmission time of a packet = 1/5 RTT
- Propagation delay from sender to receiver is equal to the propagation delay from receiver to sender = RTT/2.
- The acknowledgement of packet Pi (i=0 to 6) is called ACKi.
- Packet P1 is lost, then ACK1 is lost, and then ACK6 is lost.
- The receiver transmits an ACK for a packet after the last bit of this packet arrives at that receiver.
- Neglect the transmission time of ACK.

Annotate fully your graph to show the above requirements.

Solution:

Important points that should be present in the diagram:

- P1 will not arrive to receiver. so ACK1 is not sent.
- A timer is started after the last bit of each packet is transmitted.
- Packets received by receiver after packet P1 will be discarded, so receiver sends ACK0 for all those packets.
- After ACK0 is received, P4 ius sent
- On timeout of P1 timer, P1 and all packages already sent after P1 are retransmitted.
- Since Window size = 4, the graph should have a maximum of 4 unacknowledged packets at the sender.
- When ACK1 is lost, no need to retransmit P1 since ACK2 also acknowledges P1 (cumulative acknowledgments of Go-Back-N)

- When ACK6 is lost, sender retransmits P6 on timeout of

Problem 3:

Repeat problem 3 for the Selective Repeat.

Important points that should be present in the diagram:

- P1 will not arrive to receiver. so ACK1 is not sent.
- A timer is started after the last bit of each packet is transmitted.
- Packets received by receiver after packet 1 will be accepted, and buffered and receiver sends ACK for each one of these packets
- A timer is started for P1 when P1 is transmitted.
- On timeout of P1 timer, only P1 is retransmitted.
- Since Window size = 4, the graph should have a maximum of 4 unacknowledged packets at the sender.
- When ACK1 is lost, sender retransmits P1 on timeout of P1
- When ACK6 is lost, sender retransmits P6 on timeout of P6

P2 is received,
discarded, send A2

Timeout

Module 3 Lesson Plan

Module 3 Transport Layer

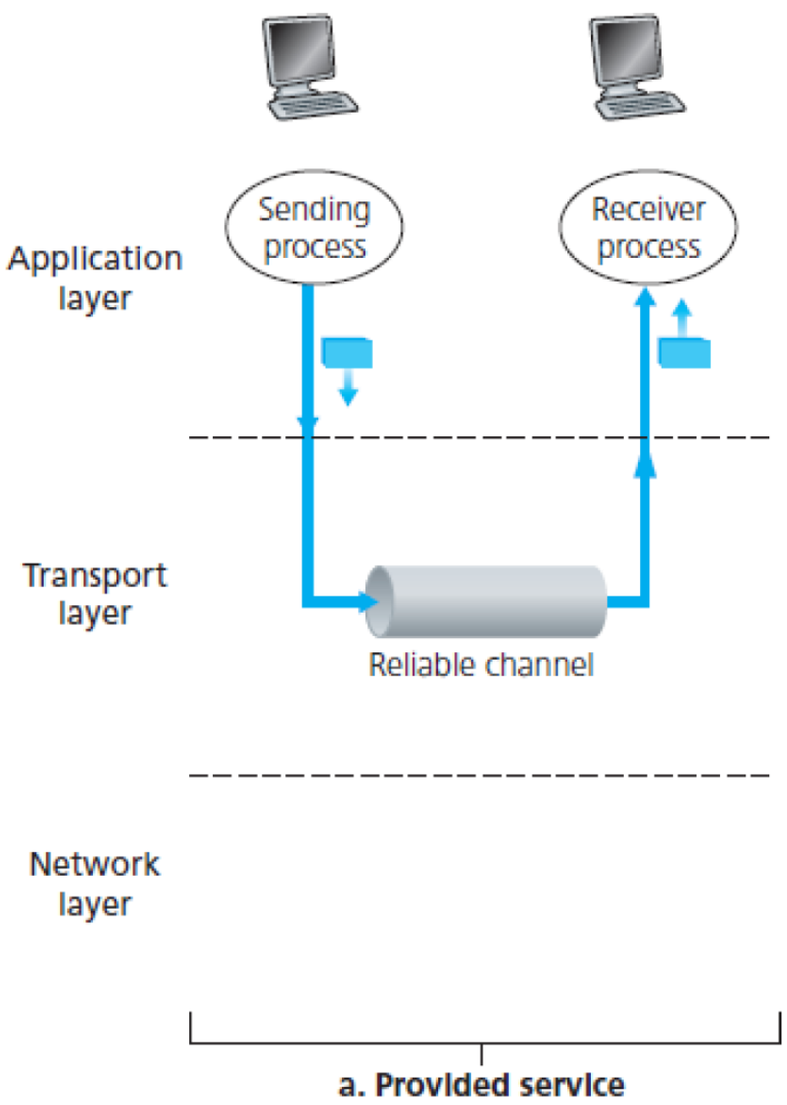
We read in chapter 2, that **TCP provides a reliable, connection-oriented service** to the invoking application.

Let us find out what is an **rdt (reliable data transfer) protocol**, and why and how we can improve it to make it a reliable protocol.

A transport protocol is said to be **reliable** if *(the service abstraction it provides to the upper-layer* is "**no transferred data bits are corrupted** (flipped from 0 to 1, or vice versa) **or lost**, and **all are delivered in the order in which they were sent"** .

A transport protocol is said to be **reliable** if *(the service abstraction it provides to the upper-layer* is "**no transferred data bits are corrupted** (flipped from 0 to or vice versa) **or lost**, and **all are delivered in the order in which they were sent"** .

Please note that this is a **service abstraction** since the layer below the reliable data transfer protocol may be unreliable.



In section 3.4, we started building a transport protocol from scratch, and several attempts were made to improve that protocol.

Discussion 7

We reached the conclusion that Acknowledgment/Negative Acknowledgments are needed from receiver to sender. Why?

Discussion 7 Answer:
These control messages allow the receiver to let the sender know what has been received correctly, and what has been received in error and thus requires repeating.

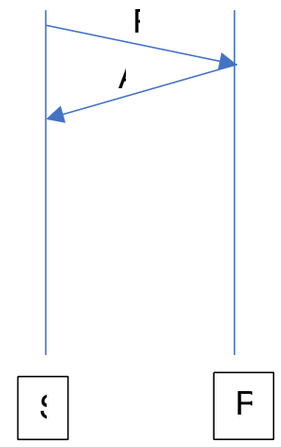
Question 1:

a) In the rdt protocols, why did we need to introduce sequence numbers?

Solution:

The problems with duplicate packets is that the receiver doesn't know whether the ACK or NAK it last sent was received correctly at the sender. Thus, it cannot know a priori whether an arriving packet contains new data or is a retransmission!

Receive ACK, send new Packet, or didn't receive ACK, resend old Packet



Sequence numbers are required for a receiver to find out whether an arriving packet contains new data or is a retransmission.

Discussion 8

In our preliminary design of a reliable data transfer protocol, we introduced the concept of ACK and NAK. But we then disregarded NAK, and we kept using ACK. Why? How can we tell the sender that a packet is corrupted?

Discussion 8 Answer:

We can accomplish the same effect as a NAK if, instead of sending a NAK, we send an ACK for the last correctly received packet. A sender that receives two ACKs for the same packet (that is, receives duplicate ACKs) knows that the receiver did not correctly receive the packet following the packet that is being ACKed twice.

Question 1:

b) In the *rdt* protocols, why did we need to introduce timers?

Answer:

Timers are needed to handle losses in the channel. If the ACK for a transmitted packet is not received within the duration of the timer for the packet, the packet (or its ACK) is assumed to have been lost. Hence, the packet is retransmitted.

Discussion 9

What is a stop-and-wait protocol? Do you think it is widely implemented in network design?

Discussion 9 Answer:

In a stop-and-wait protocol, the sender will **not send a new piece** of data **until** it is sure that the receiver has **correctly received** the current packet.
Stop-and-wait protocols result in a **very low utilization** of the bandwidth and the network resources.

Discussion 10 The sender Utilization of a stop-and wait protocol is defined as the fraction of time the sender is actually busy sending bits into the channel:

$U_{sender} = T_{tr}/(T_{tr} + W$ wait time before another packet can be transmitted)

Consider two hosts 1000Km apart. Host A is sending a message of size 1000Bytes to host B. A can transmit at a rate of 2Mbps. Suppose bits travel at the speed of 200,000,000 m/sec.

Find the utilization of the sender (host A):

Discussion 10 Solution:

Ttr = Length of packet/ Bit rate = 1000 Bytes/2,000,000 bps = 1000*8 bits/2,000,000 bps=4 msec

Wait time = RTT for packet to arrive to receiver, and ACK to go from Receiver to Sender

Wait time = RTT= 2* Tpropagation (since transmission time of ACK is negligible) = 2* distance/speed

= 2*1,000,000 m / 200,000,000 (m/sec) = 0.01 sec = 10 msec

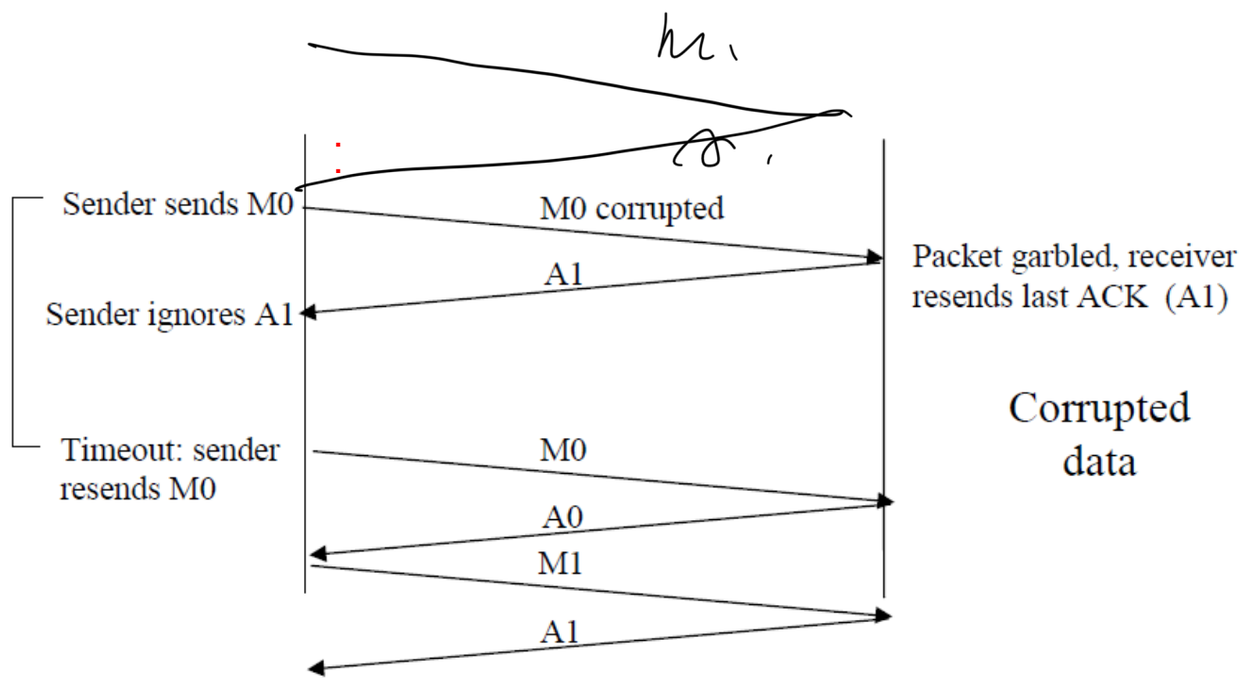
U_{sender} = Ttr / (T_{tr} + Wwait time before another packet can be ransmitted) =
4/(4+10) = 28.6% => the sender is idle most of the time

Question 2:

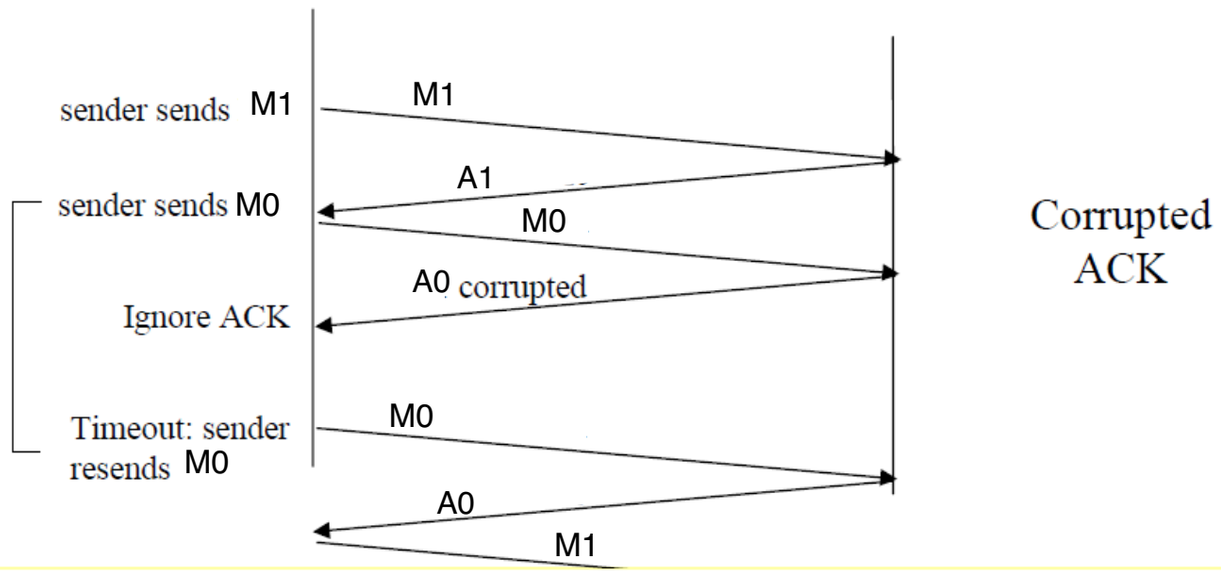
Give a trace of the operation of protocol rdt3.0 similar to that used in Figure 3.16 in the following cases:

a) packets M0 is corrupted. rdt 3.0 is a stop-and-wait protocol

M0 ? A0 ? M1 ? A1 , then **M0(corrupted)**



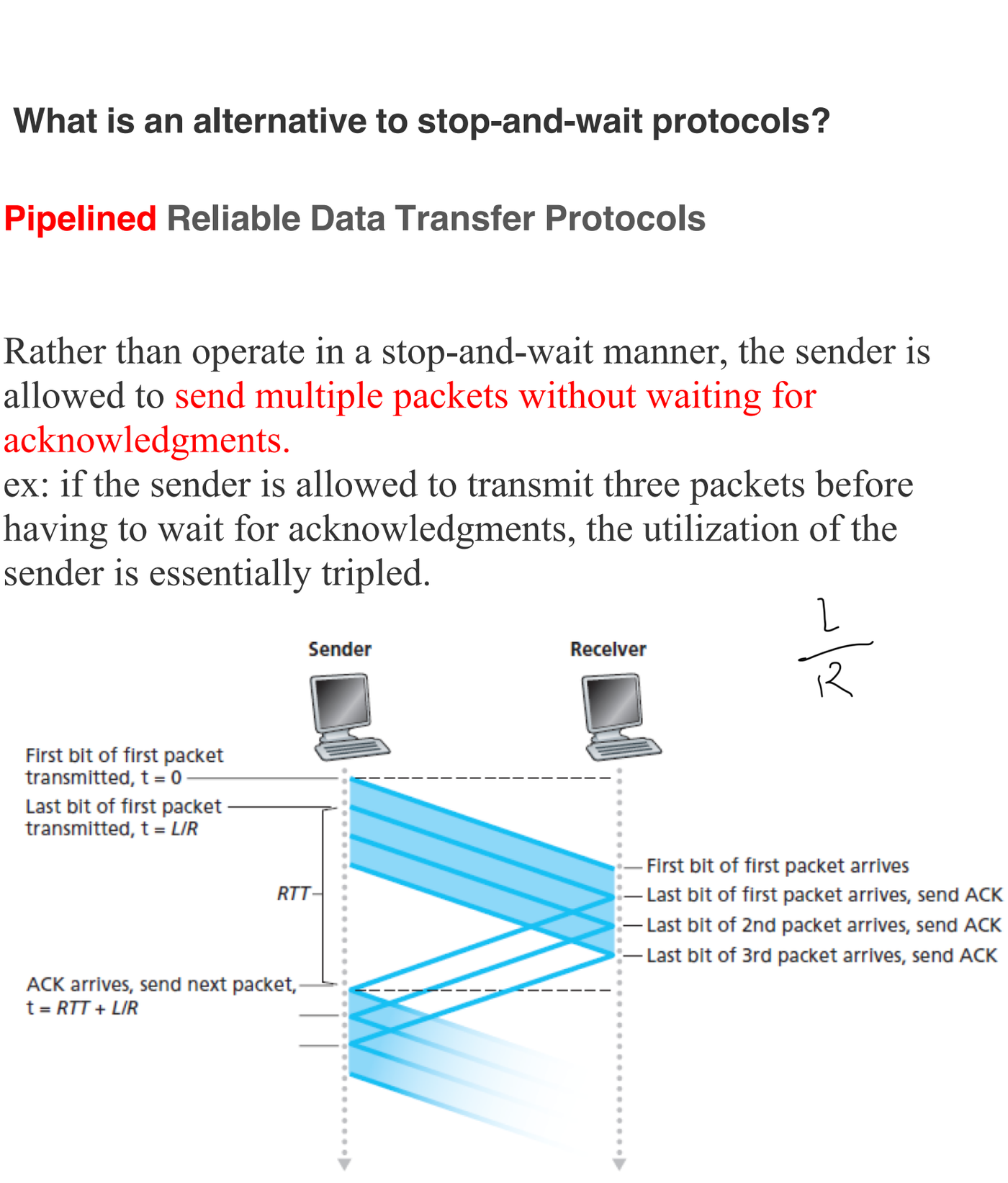
b) Acknowledgement 0 is corrupted.



What is an alternative to stop-and-wait protocols?

Pipelined Reliable Data Transfer Protocols

Rather than operate in a stop-and-wait manner, the sender is allowed to send multiple packets without waiting for acknowledgments.
ex: if the sender is allowed to transmit three packets before having to wait for acknowledgments, the utilization of the sender is essentially tripled.

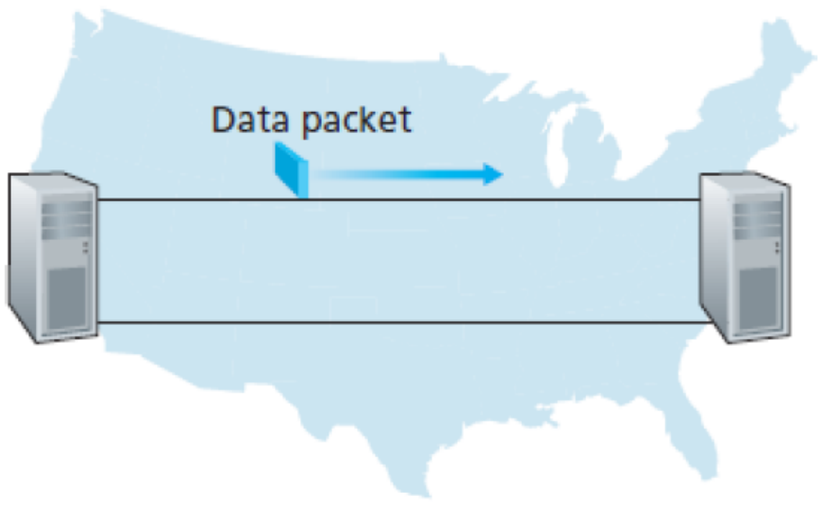


$\frac{L}{R}$

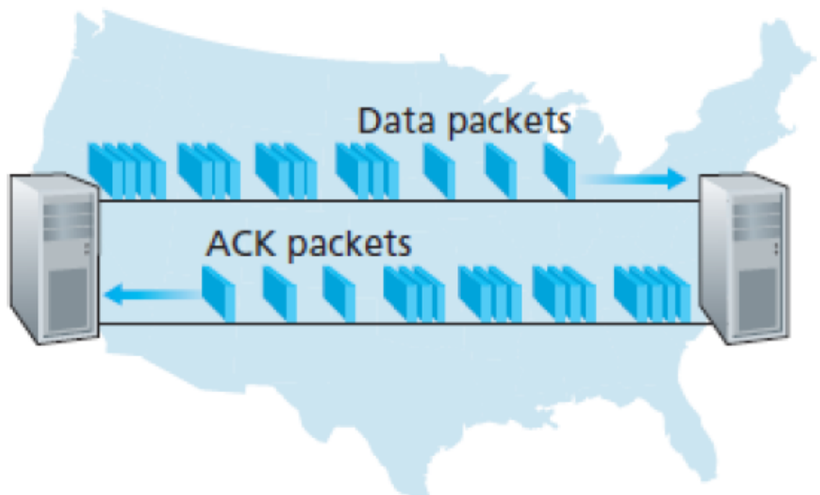
b. Pipelined operation

Since the many in-transit sender-to-receiver packets can be visualized as filling a pipeline, this technique is known as **pipelining**.

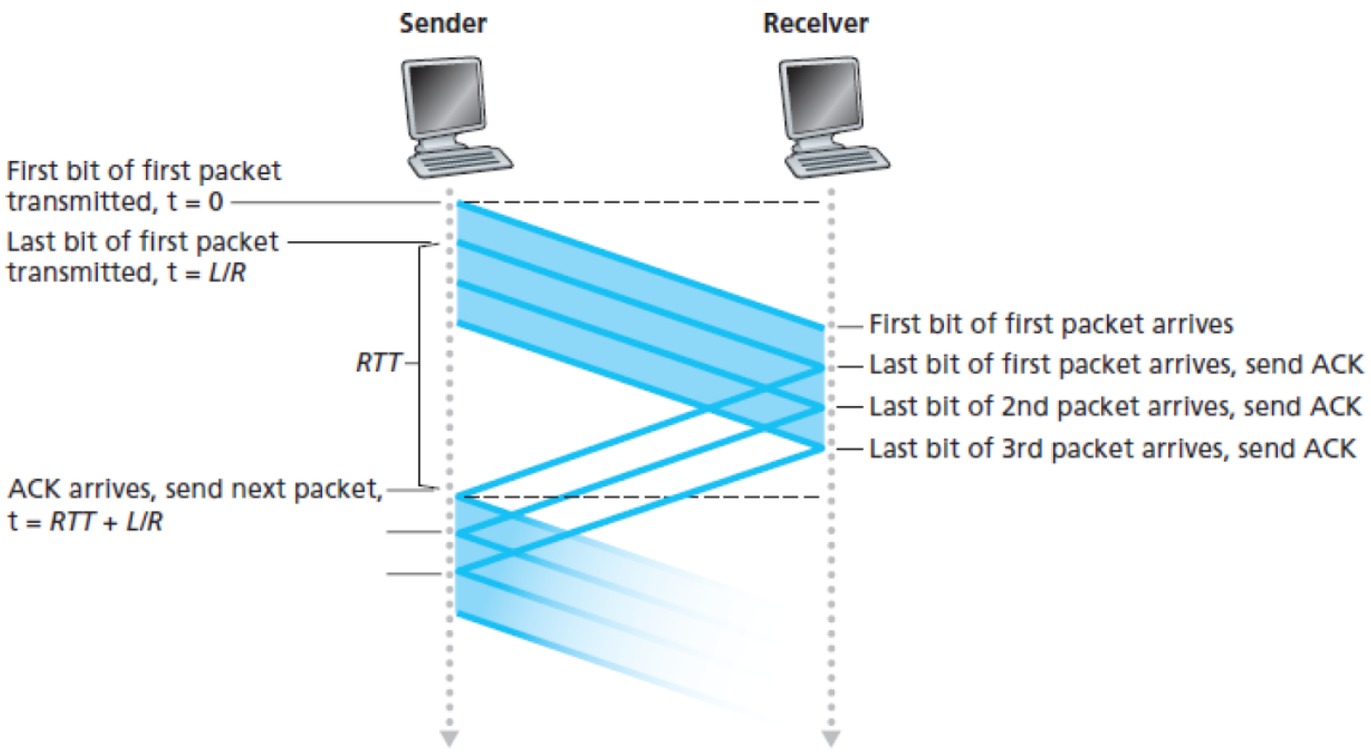
the sequence number space must be at least twice as large as the window size, $k \geq 2w$ to avoid confusing retransmission packets with newly transmitted packets with same sequence number.



a. A stop-and-wait protocol in operation



b. A pipelined protocol in operation



b. Pipelined operation

Pipelining has the following consequences for reliable data transfer protocols:

- The range of sequence numbers must be increased.
- The sender and/or receiver sides of the protocols may have to buffer more than one packet. (For ex. the sender will have to buffer packets that have been transmitted but not yet acknowledged).
- Buffering of correctly received packets may also be needed at the receiver.
- The range of sequence numbers needed and the buffering requirements will depend on the manner in which a data transfer protocol responds to lost, corrupted, and overly delayed packets.

- Two basic approaches toward pipelined error recovery can be identified: **Go-Back-N** and **selective repeat**.

In a **Go-Back-N (GBN) protocol**, the sender is allowed to transmit multiple packets (when available) without waiting for an acknowledgment, but is constrained to have **no more than some maximum allowable number, *N*, of unacknowledged packets in the pipeline.** **Why?**

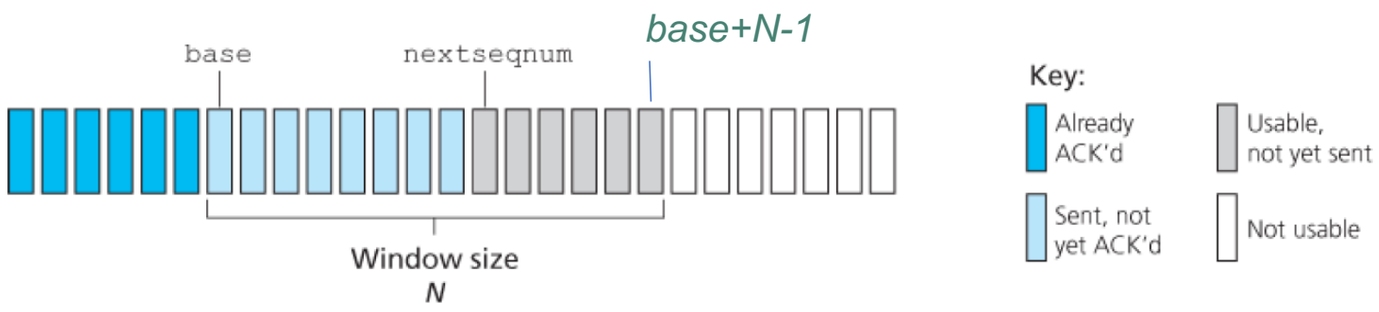


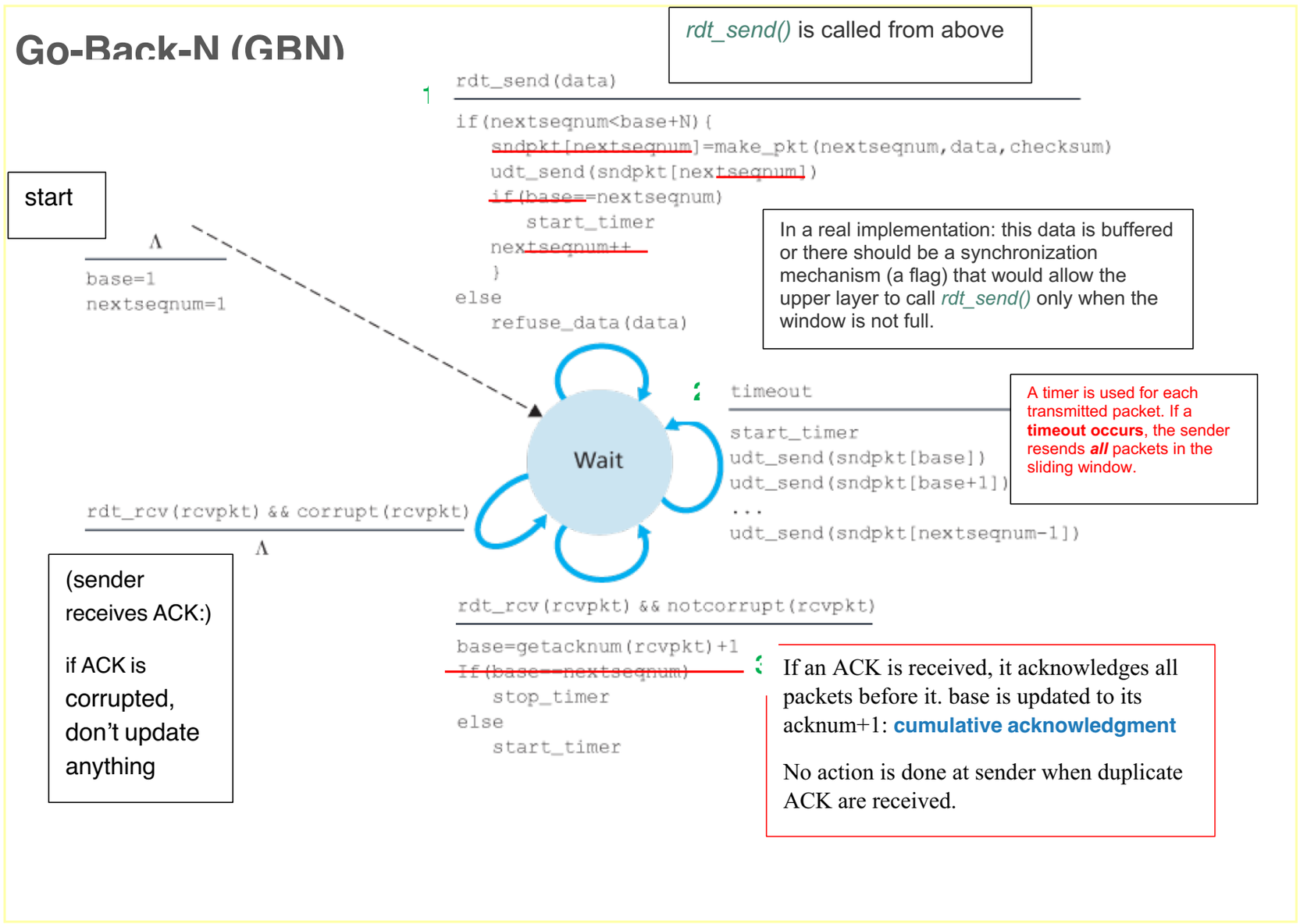
Figure 3.19 Sender's view of sequence numbers in Go-Back-N

base: the sequence number of the oldest unacknowledged.

As the protocol operates and more packets are acknowledged, this window slides forward over the sequence number space.

GBN protocol is a **sliding-window protocol**. A limited N is needed for flow control and congestion control.

Figure 3.20 Extended FSM description of the GBN sender



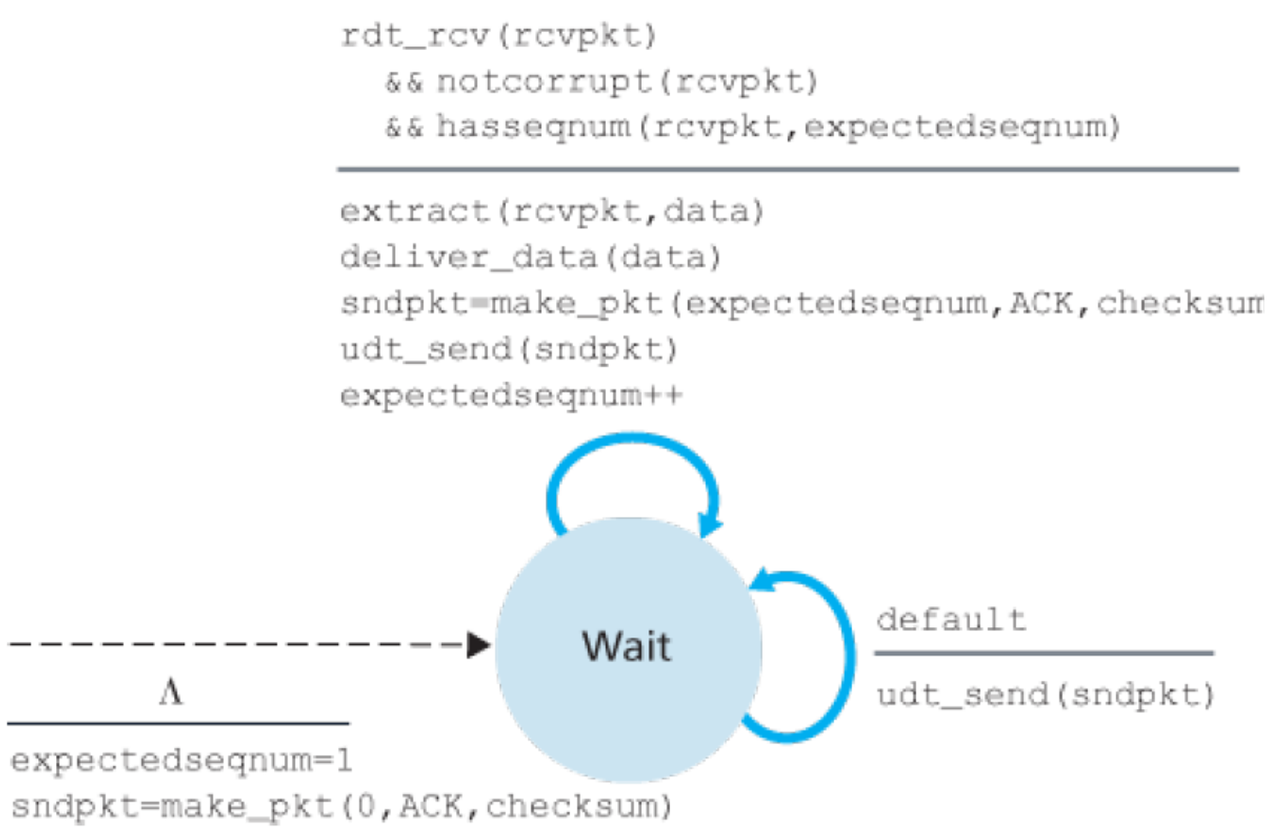


Figure 3.21 Extended FSM description of the GBN receiver

The GBN sender's actions:

- (if the sliding window is not full), data is accepted from the application layer, a sequence number is added to it (plus additional bits), and the packet is sent
- A timer is set for each transmitted packet. If a timeout occurs, the sender resends all packets in the sliding window.
- If an ACK is received, it acknowledges all packets before it. base is updated to its acknum+1: cumulative acknowledgment

- No action is done at sender when duplicate ACK are received.

The **receiver's actions** in **GBN** are simple.

- If a packet with sequence number *n* is **received correctly and is in order** (i.e. after delivering a packet with sequence number n-1), the receiver **sends an ACK** for packet *n* and **delivers the data** portion of the packet to the upper layer.
- In all other cases, the **receiver discards** the packet and **resends an ACK** for the **most recently received in-order** packet.

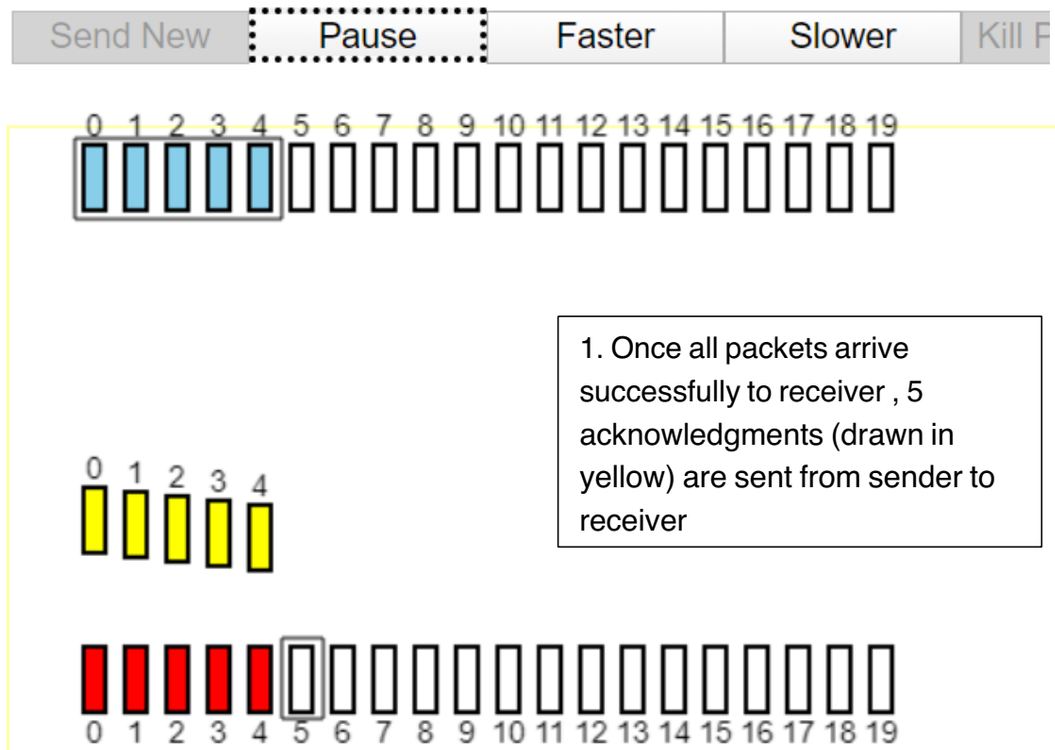
Question 2:

Visit the **Go-Back-N** Java applet at the companion Web site:
https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/#interactiveanimations
(**choose interactive animations, then chapter 3 Go-back-N**)

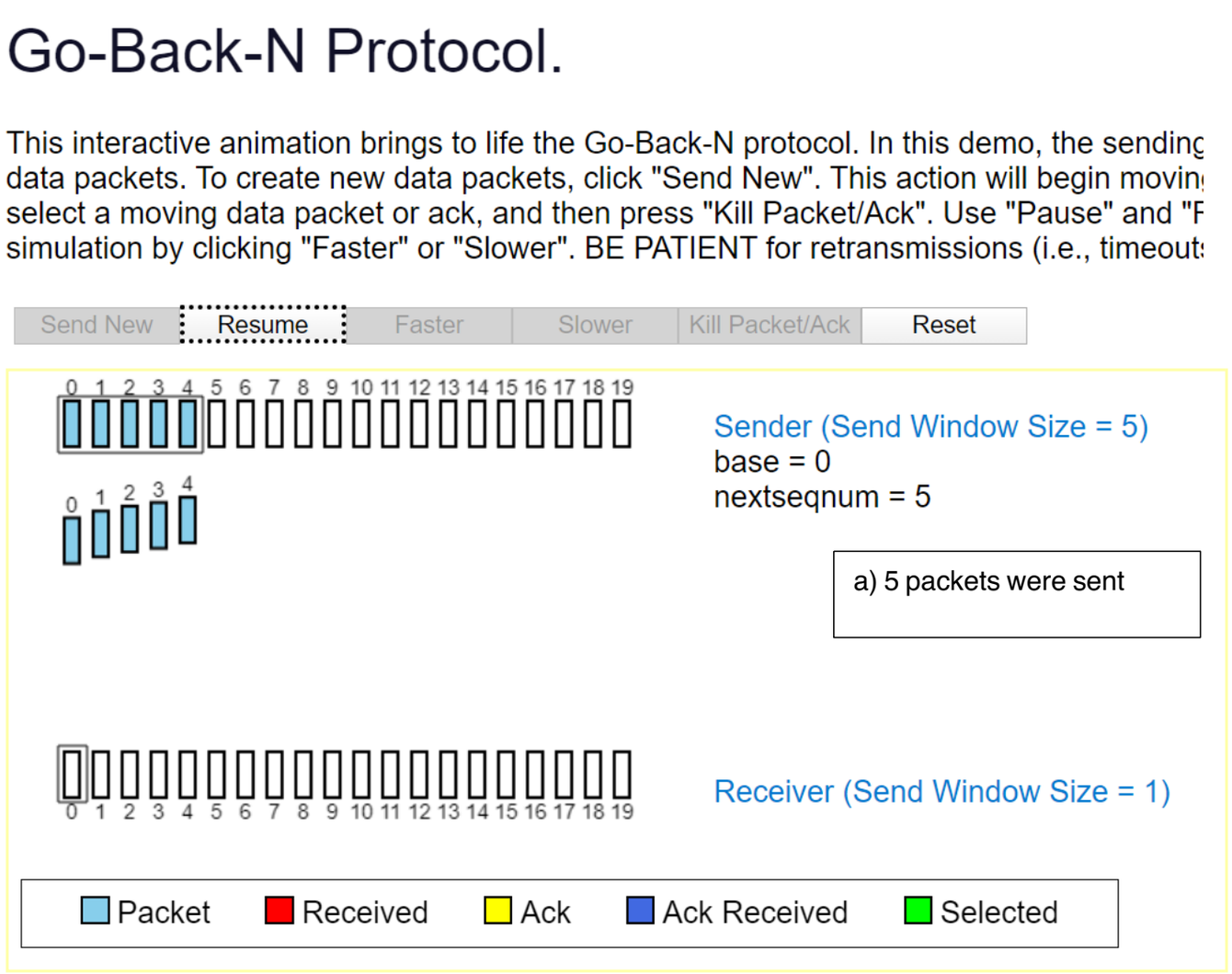
1. **Choose "faster" option**. Have the source send five packets (**click quickly 5 times on Send new**).
Describe what happens.

Go-Back-N Protocol.

This interactive animation brings to life the Go-Back-N
sending window limits the sender to a maximum of 5 o
packets. To create new data packets, click "Send New'
moving data packets between sender and receiver. To
moving data packet or ack, and then press "Kill Packet
"Resume" to make selecting easier. Speed up or slow
clicking "Faster" or "Slower". BE PATIENT for retransm



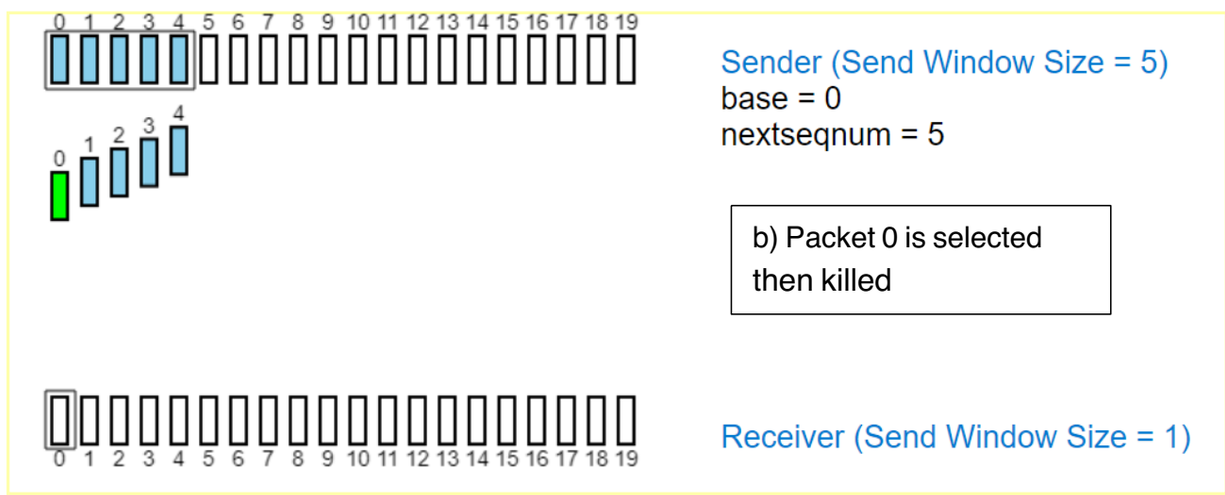
2. **Choose “faster” option**. Have the source send five packets (**click quickly 5 times on Send new**), and then **pause** the animation before any of the five packets reach the destination. Then kill the first packet: **click on packet 0** and select **kill packet**. Then **resume** the animation.
Describe what happens.



Go-Back-N Protocol.

This interactive animation brings to life the Go-Back-N protocol. In this demo, the sending v
data packets. To create new data packets, click "Send New". This action will begin moving
select a moving data packet or ack, and then press "Kill Packet/Ack". Use "Pause" and "Re
simulation by clicking "Faster" or "Slower". BE PATIENT for retransmissions (i.e., timeouts)

Send New | Resume | Faster | Slower | Kill Packet/Ack | Reset



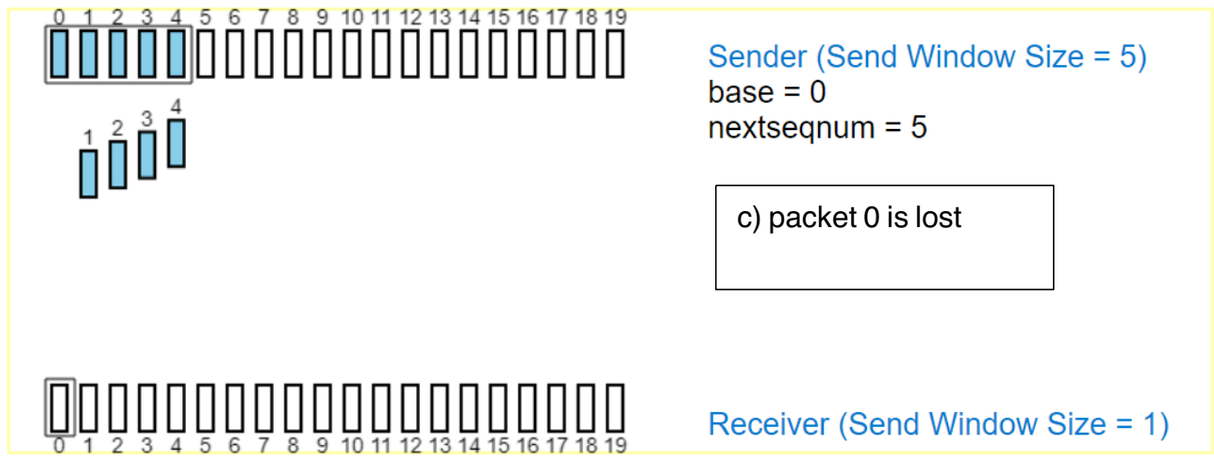
Packet | Received | Ack | Ack Received | Selected

(S) Action at Sender (R) Action at Receiver

Go-Back-N Protocol.

This interactive animation brings to life the Go-Back-N protocol. In this demo, the sending v
data packets. To create new data packets, click "Send New". This action will begin moving
select a moving data packet or ack, and then press "Kill Packet/Ack". Use "Pause" and "Re
simulation by clicking "Faster" or "Slower". BE PATIENT for retransmissions (i.e., timeouts)

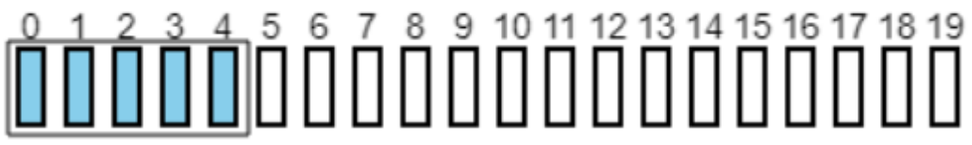
Send New | Resume | Faster | Slower | Kill Packet/Ack | Reset



Go-Back-N Protocol.

This interactive animation brings to life the Go-Back
sending window limits the sender to a maximum of
packets. To create new data packets, click "Send N
moving data packets between sender and receiver.
moving data packet or ack, and then press "Kill Pac
"Resume" to make selecting easier. Speed up or sl
clicking "Faster" or "Slower". BE PATIENT for retrar

Send New | Pause | Faster | Slower



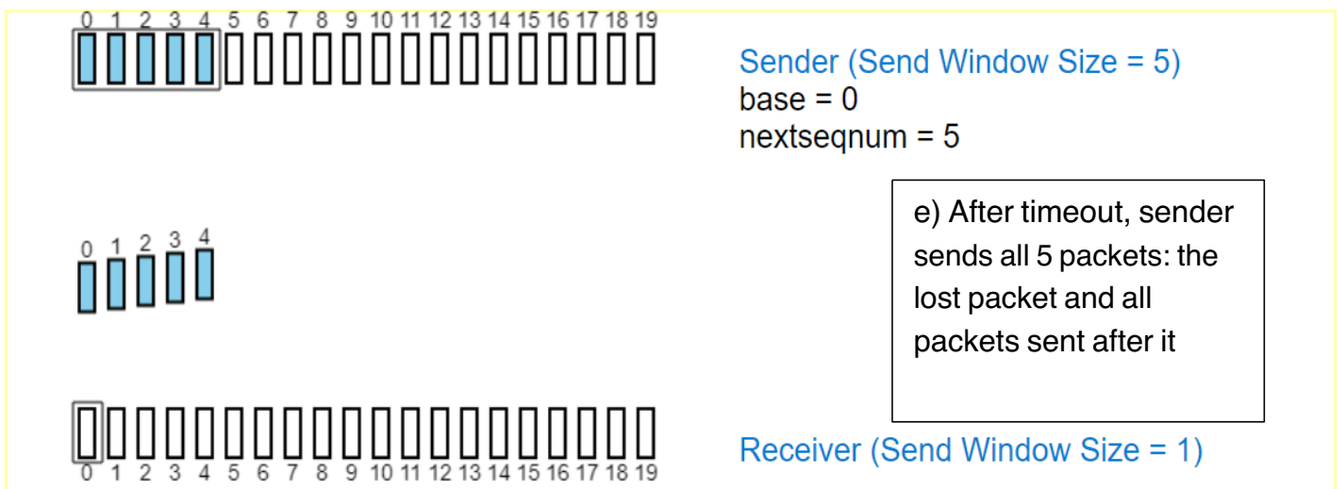
d) receiver discarded all packets.

Nothing is kept in receiver

Go-Back-N Protocol.

This interactive animation brings to life the Go-Back-N protocol. In this demo, the sending win
data packets. To create new data packets, click "Send New". This action will begin moving dat
select a moving data packet or ack, and then press "Kill Packet/Ack". Use "Pause" and "Resu
simulation by clicking "Faster" or "Slower". BE PATIENT for retransmissions (i.e., timeouts).

Send New | Resume | Faster | Slower | Kill Packet/Ack | Reset



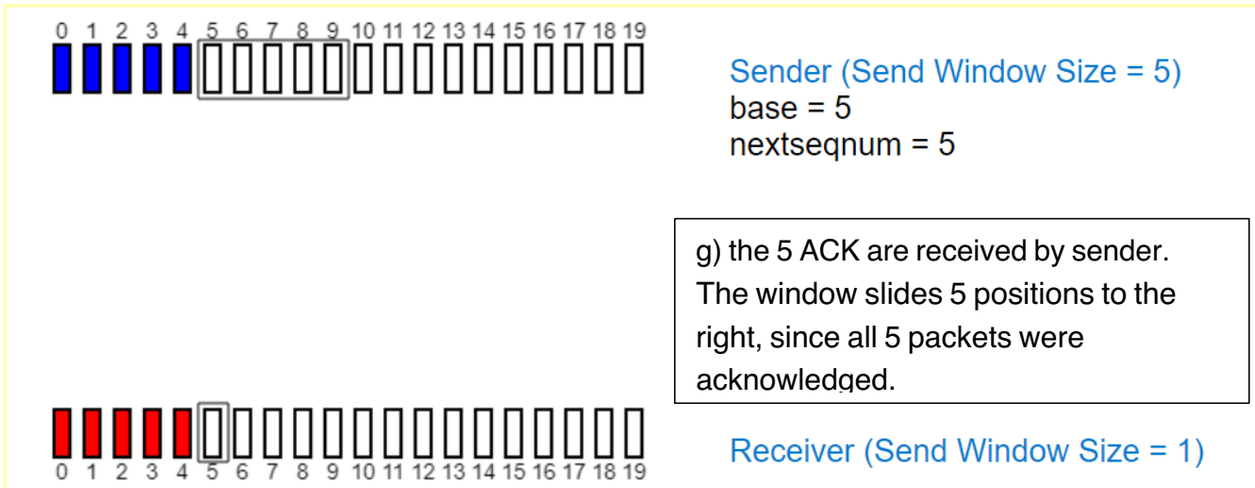
Packet | Received | Ack | Ack Received | Selected

(S) Action at Sender (R) Action at Receiver

Go-Back-N Protocol.

This interactive animation brings to life the Go-Back-N protocol. In this demo, the sending wi
data packets. To create new data packets, click "Send New". This action will begin moving da
select a moving data packet or ack, and then press "Kill Packet/Ack". Use "Pause" and "Resu
simulation by clicking "Faster" or "Slower". BE PATIENT for retransmissions (i.e., timeouts).

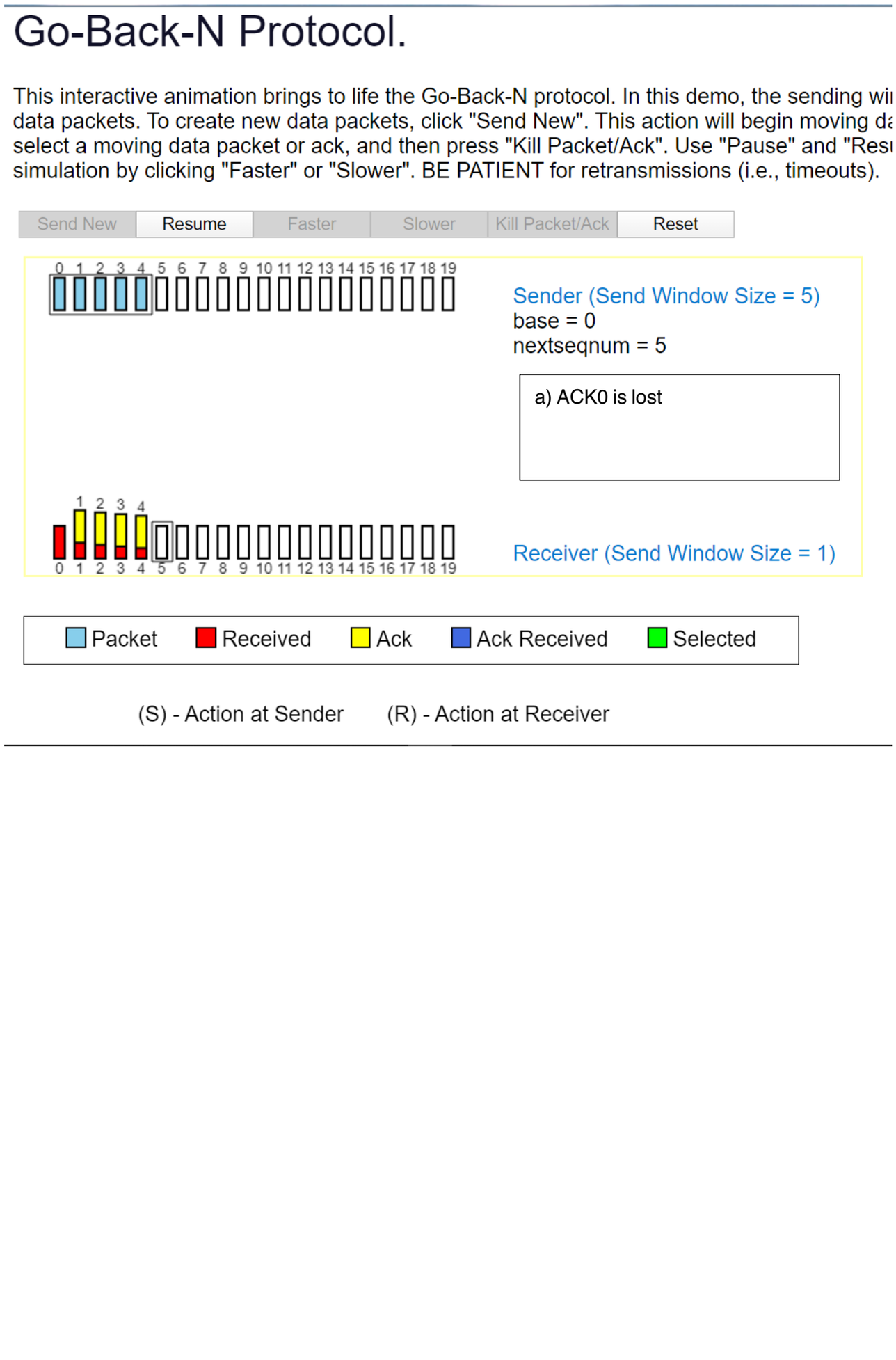
Send New | Pause | Faster | Slower | Kill Packet/Ack | Reset



Packet | Received | Ack | Ack Received | Selected

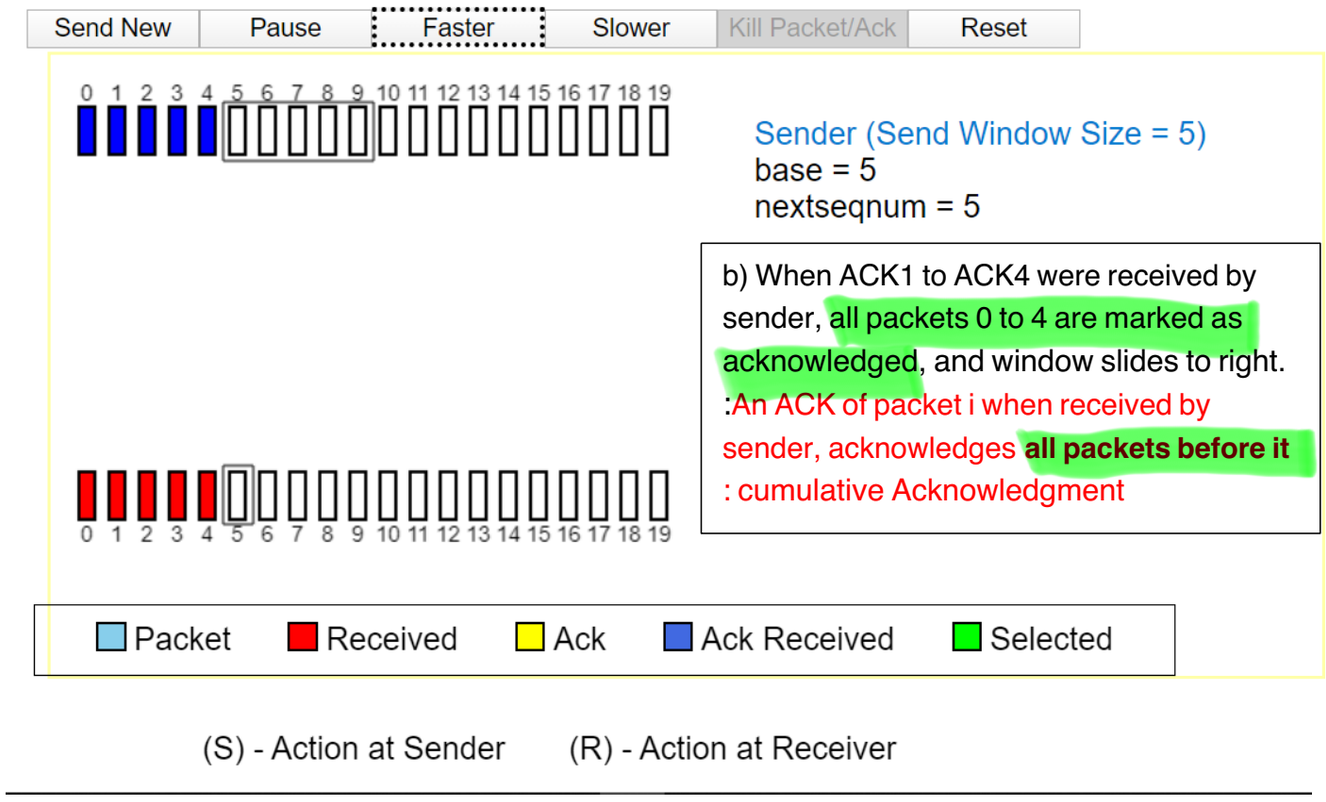
(S) - Action at Sender (R) - Action at Receiver

3. Repeat the experiment, but now let the first packet reach the destination and kill the first acknowledgment. Describe again what happens.



Go-Back-N Protocol.

This interactive animation brings to life the Go-Back-N protocol. In this demo, the sending wi
data packets. To create new data packets, click "Send New". This action will begin moving da
select a moving data packet or ack, and then press "Kill Packet/Ack". Use "Pause" and "Resu
simulation by clicking "Faster" or "Slower". BE PATIENT for retransmissions (i.e., timeouts).



3. Finally, try sending six packets. What happens?

The 6th packet is not sent since the window is of size 5.

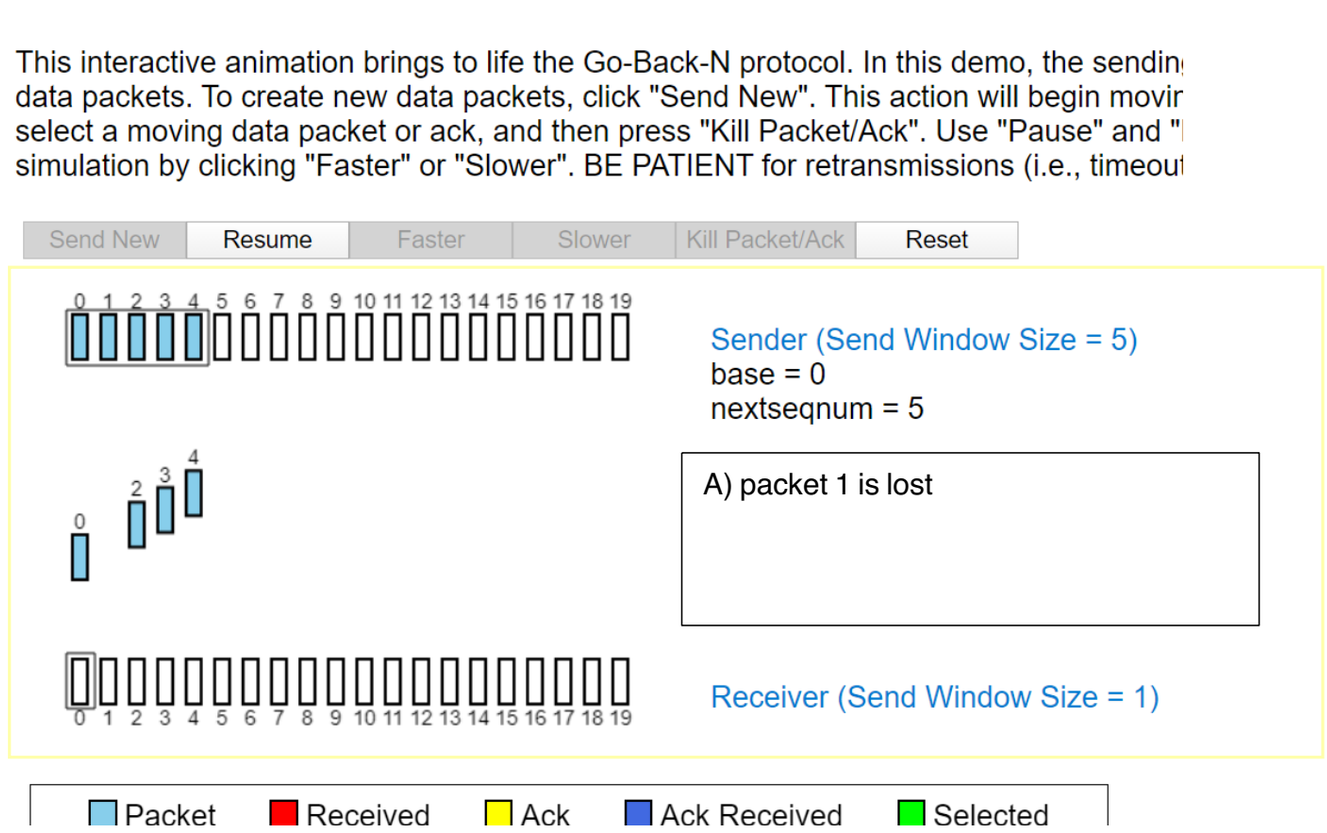
Discussion 11: Repeat the Go-Back-N animation: send 5 packets (0 to 4), then pause animation, and kill packet 1. Describe what happens:

Are acknowledgments sent?

If packet #5 arrives to sender can it be sent?

What happens next?

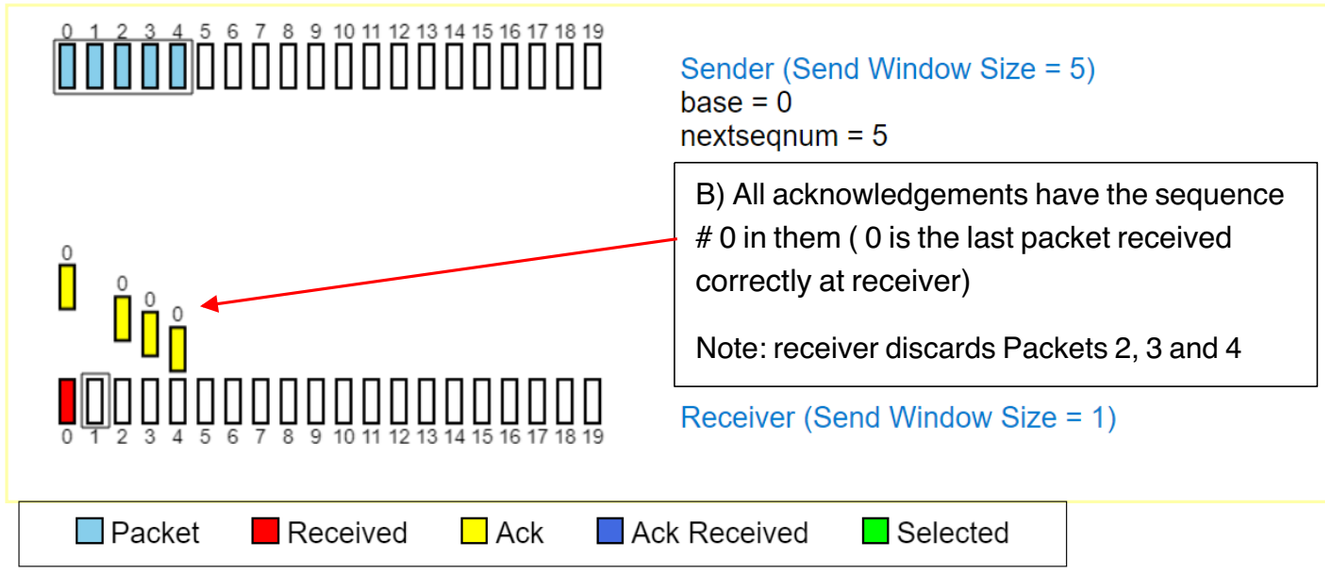
Answer



Go-Back-N Protocol.

This interactive animation brings to life the Go-Back-N protocol. In this demo, the sending windc
data packets. To create new data packets, click "Send New". This action will begin moving data
select a moving data packet or ack, and then press "Kill Packet/Ack". Use "Pause" and "Resum
simulation by clicking "Faster" or "Slower". BE PATIENT for retransmissions (i.e., timeouts).

Send New | Resume | Faster | Slower | Kill Packet/Ack | Reset

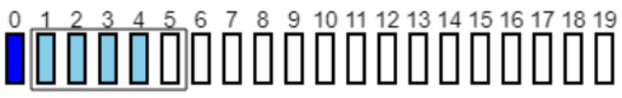


How is the sliding window changed?

Go-Back-N Protocol.

This interactive animation brings to life the Go-Back-N protocol. In this demo, the sendin
data packets. To create new data packets, click "Send New". This action will begin movin
select a moving data packet or ack, and then press "Kill Packet/Ack". Use "Pause" and "I
simulation by clicking "Faster" or "Slower". BE PATIENT for retransmissions (i.e., timeout

Send New | Resume | Faster | Slower | Kill Packet/Ack | Reset



Sender (Send Window Size = 5)
base = 1
nextseqnum = 5

C) Only packet0 is acknowledged. Sliding window advances one position only to the right.

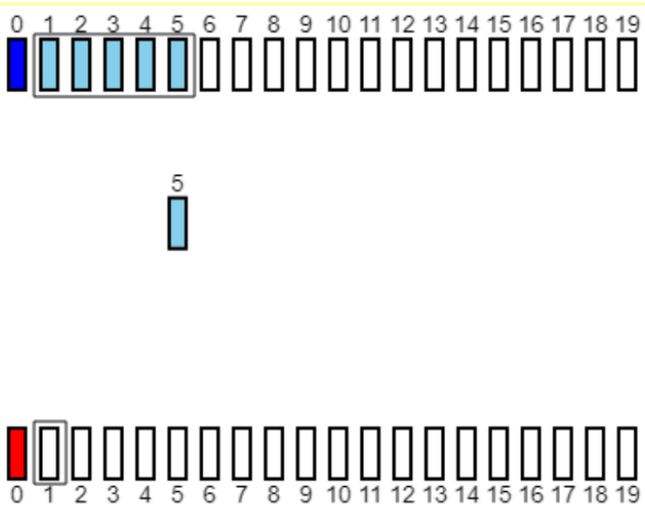
Receiver (Send Window Size = 1)

(S) - Action at Sender (R) - Action at Receiver

Go-Back-N Protocol.

This interactive animation brings to life the Go-Back-N protocol. In this demo, the sending window limits the sender to a maximum of 5 outstanding, unacked data packets. To create new data packets, click "Send New". This action will begin moving data packets between sender and receiver. To simulate loss, select a moving data packet or ack, and then press "Kill Packet/Ack". Use "Pause" and "Resume" to make selecting easier. Speed up or slow down the simulation by clicking "Faster" or "Slower". BE PATIENT for retransmissions (i.e., timeouts).

Send New | Resume | Faster | Slower | Kill Packet/Ack | Reset



Sender (Send Window Size = 5)
base = 1
nextseqnum = 6

What happens if packet 5 is sent?

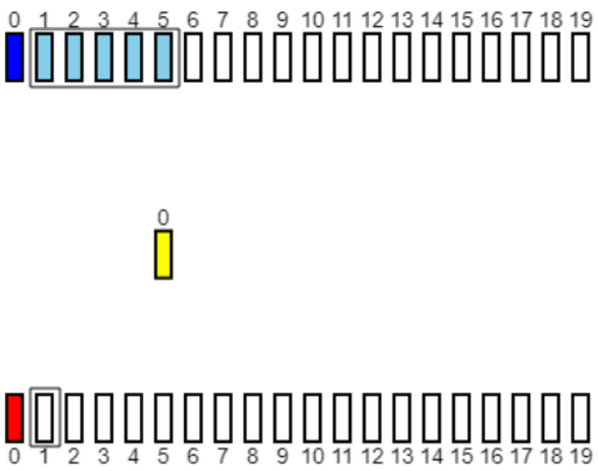
(why packets 1, 2, 3, 4 are not resent??)

Receiver (Send Window Size = 1)

Go-Back-N Protocol.

This interactive animation brings to life the Go-Back-N protocol. In this demo, the sending
data packets. To create new data packets, click "Send New". This action will begin moving
select a moving data packet or ack, and then press "Kill Packet/Ack". Use "Pause" and "R
simulation by clicking "Faster" or "Slower". BE PATIENT for retransmissions (i.e., timeouts

Send New | Resume | Faster | Slower | Kill Packet/Ack | Reset



Sender (Send Window Size = 5)
base = 1
nextseqnum = 6

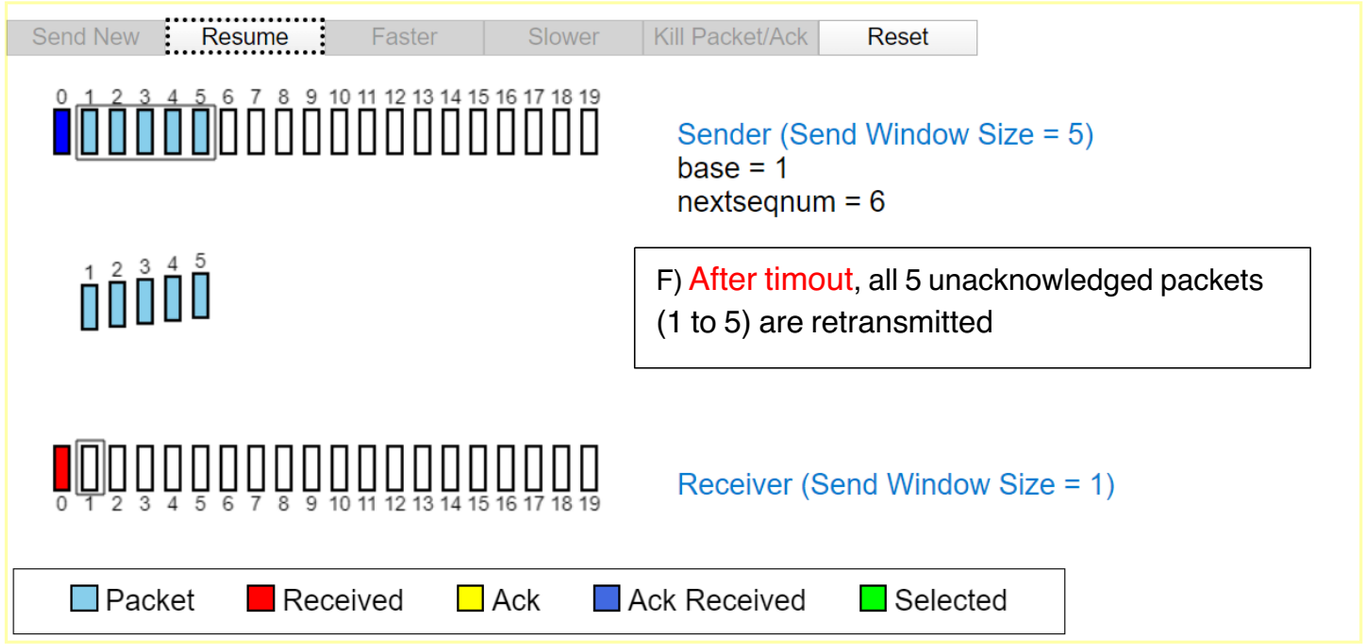
Receiver (Send Window Size = 1)

Packet | Received | Ack | Ack Received | Selected

(S) - Action at Sender (R) - Action at Receiver

Go-Back-N Protocol.

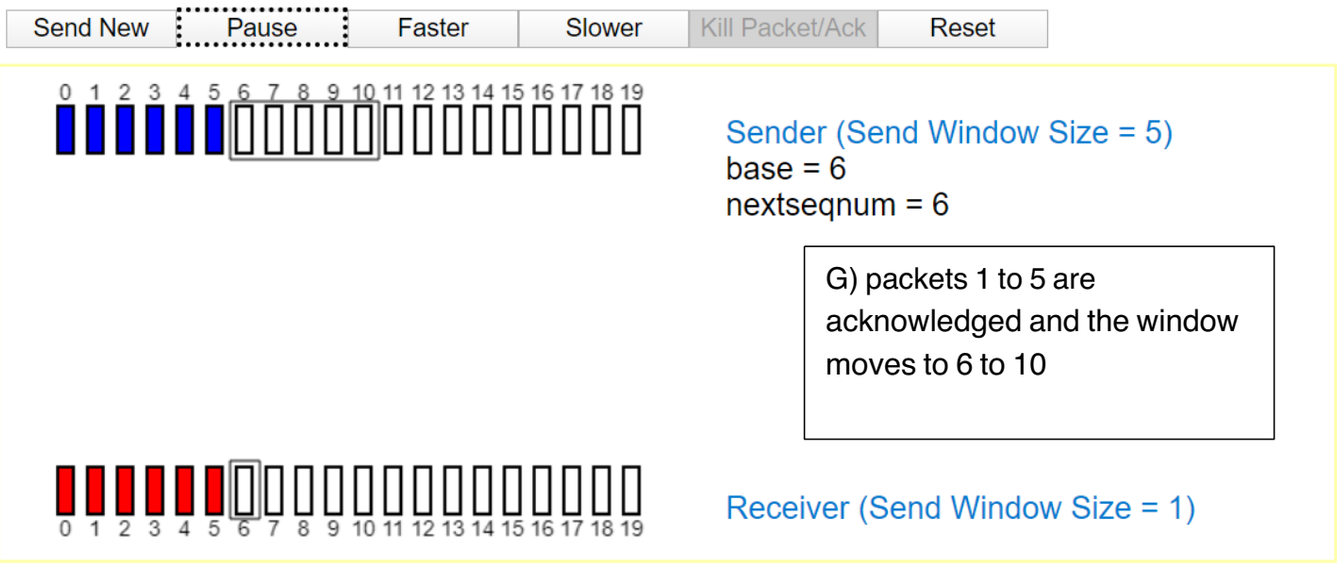
This interactive animation brings to life the Go-Back-N protocol. In this demo, the sending win
data packets. To create new data packets, click "Send New". This action will begin moving dat
select a moving data packet or ack, and then press "Kill Packet/Ack". Use "Pause" and "Resu
simulation by clicking "Faster" or "Slower". BE PATIENT for retransmissions (i.e., timeouts).



(S) - Action at Sender (R) - Action at Receiver

Go-Back-N Protocol.

This interactive animation brings to life the Go-Back-N protocol. In this demo, the sending windo
data packets. To create new data packets, click "Send New". This action will begin moving data
select a moving data packet or ack, and then press "Kill Packet/Ack". Use "Pause" and "Resume
simulation by clicking "Faster" or "Slower". BE PATIENT for retransmissions (i.e., timeouts).



(S) - Action at Sender (R) - Action at Receiver

Discussion Work in groups to draw a diagram similar to Figure 3.22 of textbook (repeated here for your convenience) for the following problem: (**Stop after 5 packets are accepted by receiver**)

Protocol Used: Go-Back-N

Sliding Window size = 4

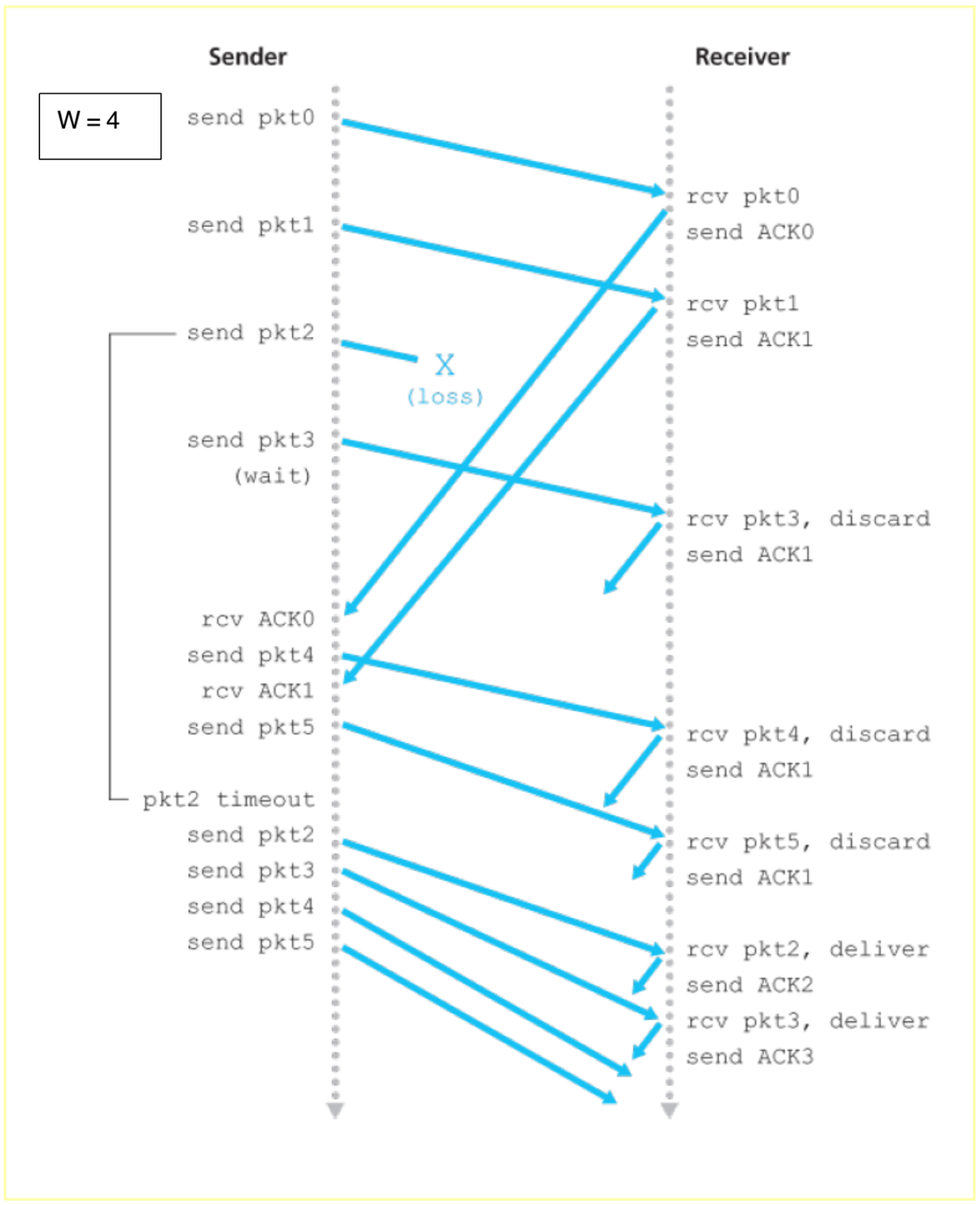
Timeout interval = 2RTT

Assume transmission time of a packet = 1/3 RTT

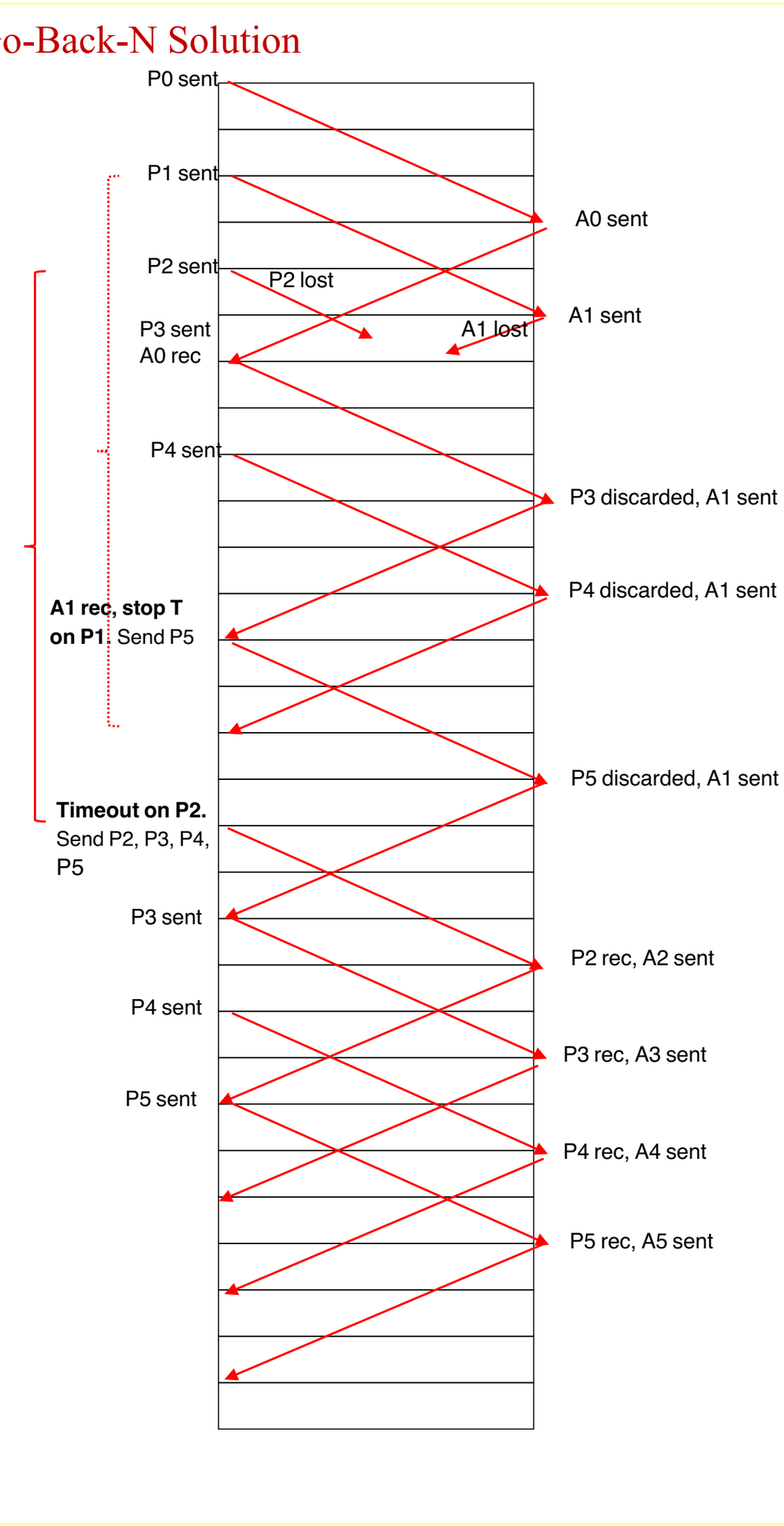
Assume Ack1 is lost, then Packet 2 is lost.

As mentioned in Figure 3.20: The sender starts a timer when it transmits each packet.

When a new (not duplicated) ACK is received, the base (and window) slides to the number of ACK+1.


Sender
Receiver
W = 4
send pkt0
send pkt1
send pkt2
X
(loss)
send pkt3
(wait)
rcv ACK0
send pkt4
rcv ACK1
send pkt5
pkt2 timeout
send pkt2
send pkt3
send pkt4
send pkt5
rcv pkt0
send ACK0
rcv pkt1
send ACK1
rcv pkt3, discard
send ACK1
rcv pkt4, discard
send ACK1
rcv pkt5, discard
send ACK1
rcv pkt2, deliver
send ACK2
rcv pkt3, deliver
send ACK3

Go-Back-N Solution



Selective Repeat

Sender

1. *Data received from above.* When data is received from above, the SR sender checks the next available sequence number for the packet. If the sequence number is within the sender's window, the data is packetized and sent; otherwise it is either buffered or returned to the upper layer for later transmission, as in GBN.
2. *Timeout.* Timers are again used to protect against lost packets. However, each packet must now have its own logical timer, since only a single packet will be transmitted on timeout. A single hardware timer can be used to mimic the operation of multiple logical timers [Varghese 1997].
3. *ACK received.* If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window. If the packet's sequence number is equal to `send_base`, the window base is moved forward to the unacknowledged packet with the smallest sequence number. If the window moves and there are untransmitted packets with sequence numbers that now fall within the window, these packets are transmitted.

Receiver buffers correctly received packets if out of order

1st unack packet

Figure 3.24 SR sender events and actions

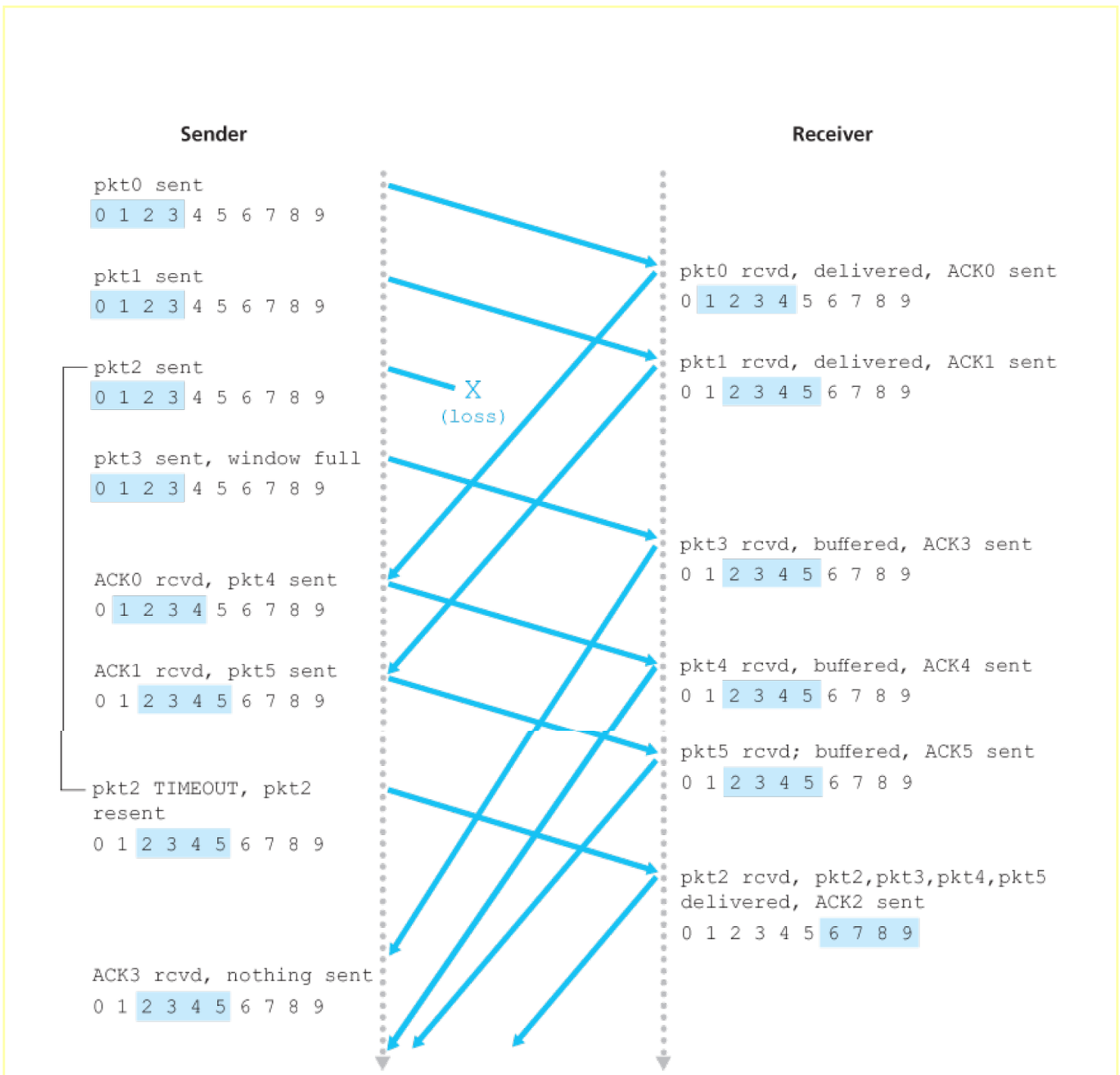


Figure 3.26 SR operation

Question 3:

Repeat Question 2, but now with the Selective Repeat Java applet. How are Selective Repeat and Go-Back-N different?

Visit the Selective Repeat Java applet at the companion Web site:
https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/#interactiveanimations

a. Have the source send five packets, and then pause the animation before any of the five packets reach the destination. Then kill the first packet and resume the animation. Describe what happens.

Answer
Packet 0 is lost. Packets 1, 2, 3, 4 are stored in receiver, and ACK 1, 2, 3,4 are sent to sender. **Sender marks them as received.**

After timeout, Pkt 0 is sent again, then when pkt 0 arrives at receiver, pkt 0, 1, 2, 3, 4, are delivered, and ACK0 is sent to sender.

b. Repeat the experiment, but now let the first packet reach the destination and kill the first acknowledgment. Describe again what happens.

pkt 0,1,2,3,4 are received at receiver. ACK 1, 2, 3, 4, are sent to sender. After time out, pkt 0 is sent to receiver, then ACK0 is sent back to sender. and receiver delivers all packets in order to upper layer.

c. Finally, try sending six packets. What happens? Not allowed

d. How are Selective Repeat and Go-Back-N different?

Disadvantage of Go back N:

When the window size is large, many packets can be in the pipeline. **A single packet error can thus cause GBN to retransmit a large number of packets**, many unnecessarily.

Selective-repeat protocols avoid unnecessary retransmissions by having the **sender retransmit only those packets that it**

suspects were received in error (that is, were lost or corrupted) at the receiver.
This individual, as-needed, retransmission will require that the **receiver acknowledges correctly received packets** whether or not they are in order. GBN sends to sender ACK of last correctly received packet.

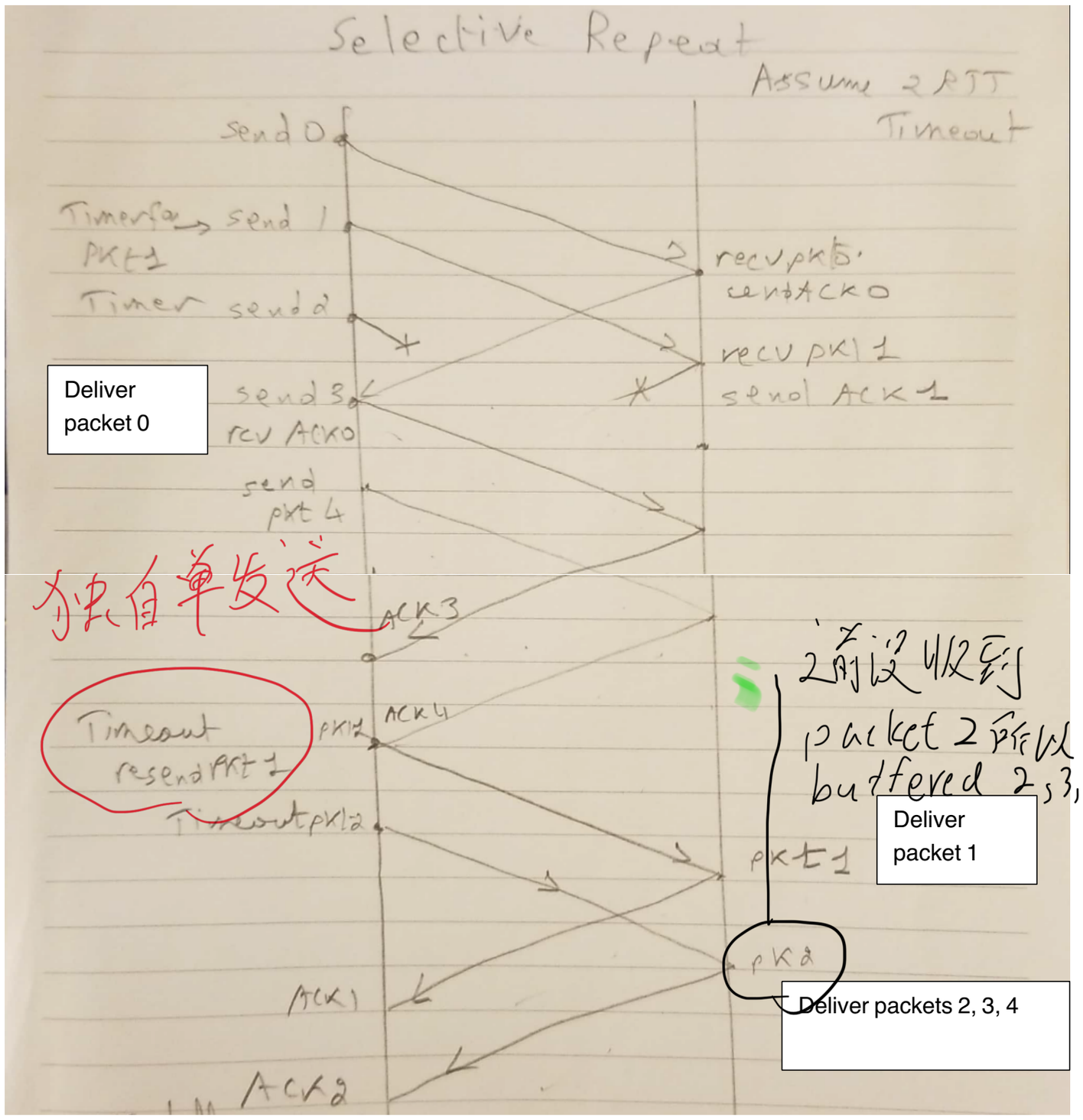
SR, like GBN, uses a window size of *N* to limit the number of outstanding, unacknowledged packets in the pipeline. However, unlike GBN, the **sender will have already received ACKs** for some of the packets in the window.

in SR, **Out-of-order packets are buffered** until any missing packets (that is, packets with lower sequence numbers) are received, at which point a batch of packets can be delivered in order to the upper layer.

in GO-Back-N, **out-of-order packets are discarded**. When a packet was lost, GO-Back-N retransmitted all the packets in the sliding window. whereas Selective Repeat retransmitted the lost packet only.

Advantage of GBN: In case of **lost acknowledgement**, **selective repeat sent a duplicate ACK** and as **GO-Back-N used cumulative acknowledgment**, so that duplicate ACK was unnecessary.

0 1 2 3 4

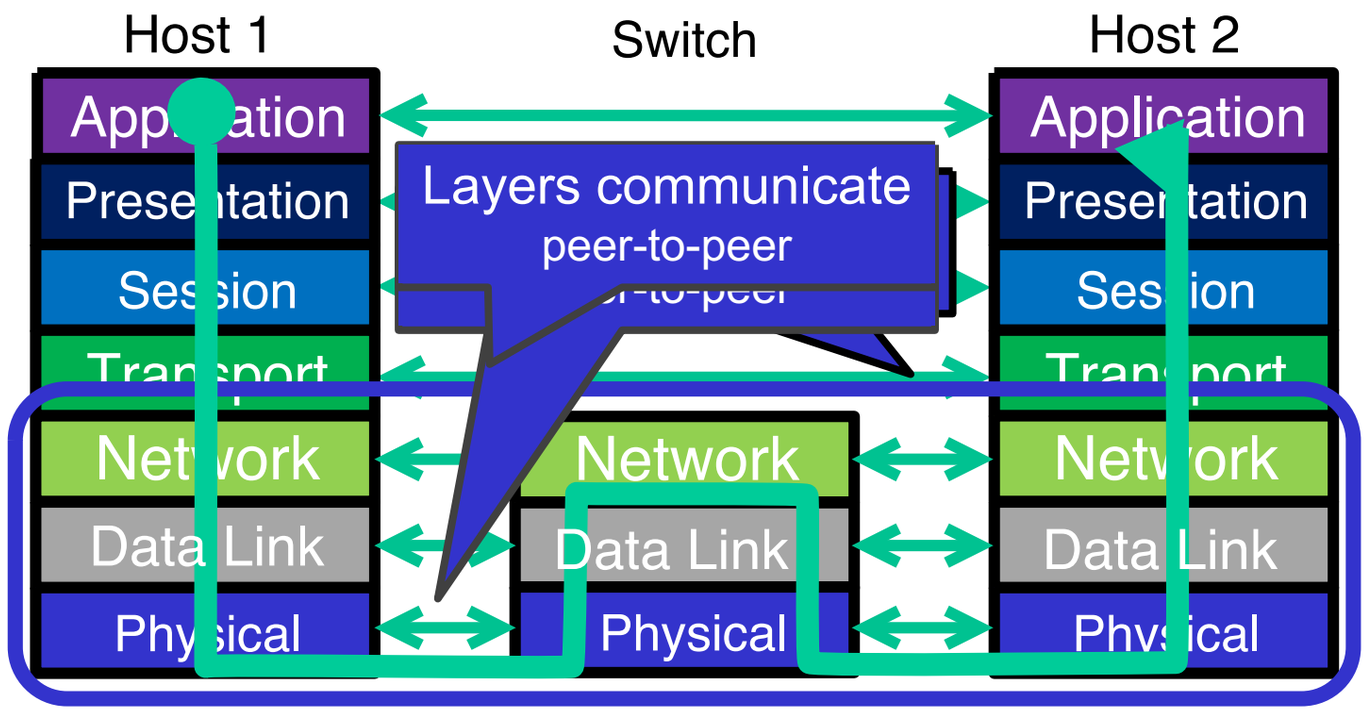

Selective Repeat
Assume 2 RTT
Timeout
send 0
Timer for pkt 1
send 1
Timer
send 2
send 3
rcv ACK0
send pkt 4
recv pkt 0. send ACK 0
recv pkt 1
send ACK 1
Deliver packet 0
快速自单发送
ACK 3
Timeout
resend pkt 2
pkt 2
ACK 4
Timeout pkt 2
pkt 1
Deliver packet 1
2前没收到
packet 2 开始
buffered 2, 3, 4
pkt 2
Deliver packets 2, 3, 4
ACK 1
ACK 2

Module 4 Lesson Plan

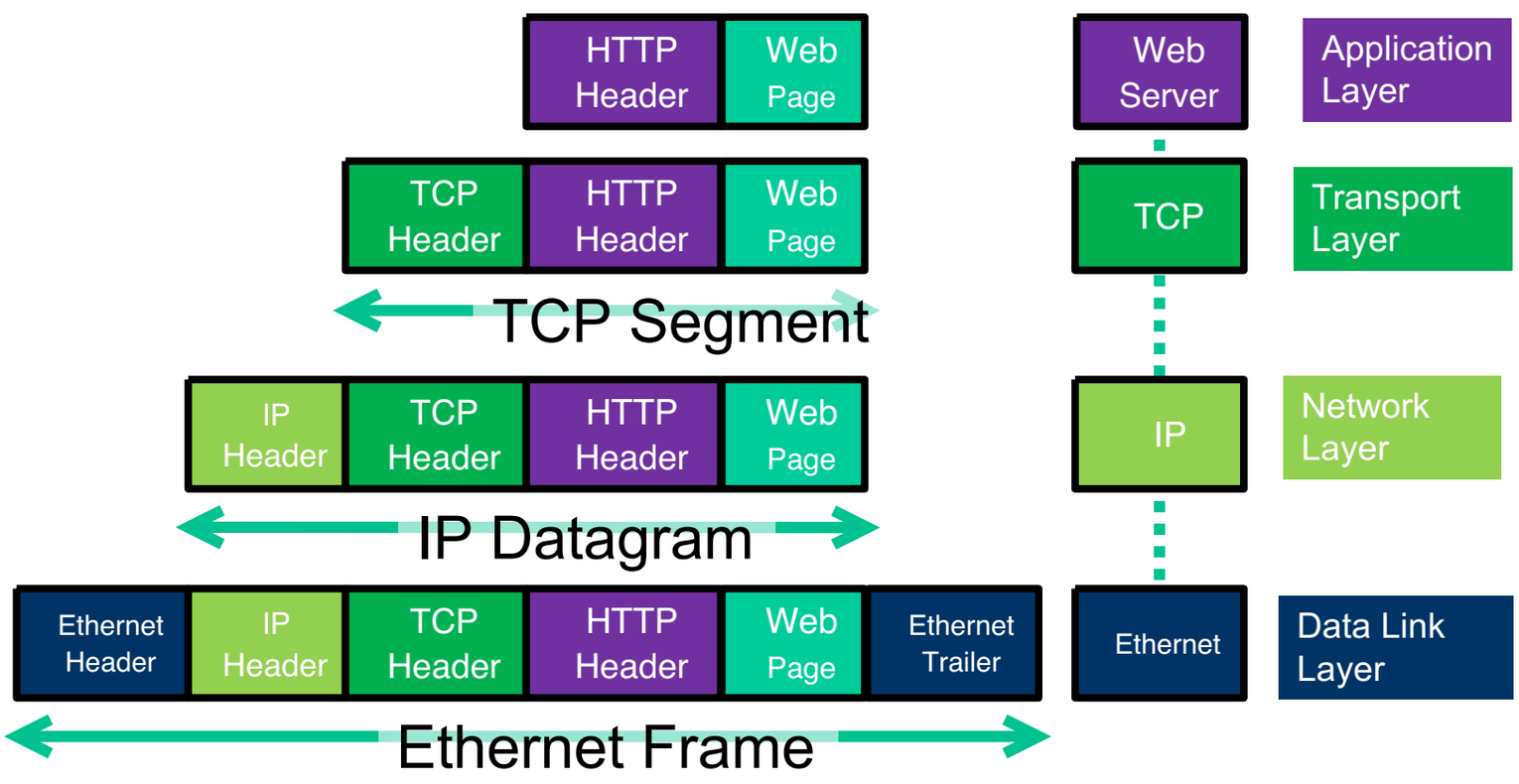
Network programming TCP

Some material in the following slides is based on material from the course COS 461 at Princeton University

The ISO OSI Model

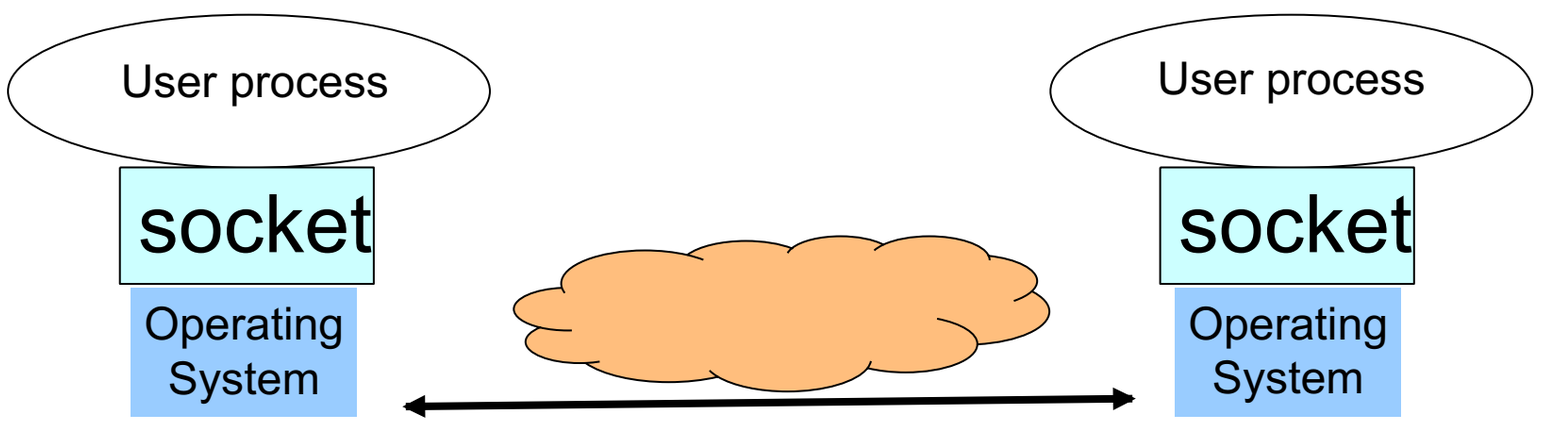


Encapsulation



Socket: end point of communication

- A typical network application consists of a pair of programs—a **client program** and a **server program**—residing in two different end systems.
- When these two programs are executed, a **client process** and a **server process** are created.
- To send a message from one process to another, the message must traverse the **underlying network**(s).
- These processes communicate with each other by reading from, and writing to, sockets.



18

- Resources on sockets and socket programming:
- https://medium.com/fantageek/understanding-socket-and-port-in-tcp-2213dc2e9b0c#:~:text=What%20is%20the%20maximum%20number,more%20than%20one%20connection%20simultaneously.
- https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm
- Commented tutorial on sockets
- https://www.youtube.com/watch?v=_iHMMo7SDfQ&list=PLm556dMNleHc1MWN5BX9B2XkwkNE2Djiu&index=14

Types of communications

- Sockets support the following types of communications

- Stream socket — corresponds to TCP
  - Treats data as a continuous Stream of bytes
  - Reliable
  - Connection-oriented (need to establish connection first)

- Datagram socket — corresponds to UDP
  - Sends and receives data in the form of individual packets, each with its own destination address.
  - Best effort (no guarantee of delivery)
  - Connectionless

21

Ports, connections and sockets

- What defines a unique connection ?
- There are 5 elements that identify a connection. They call them 5-tuple
  - Protocol (Stream or Datagram). This is often omitted as it is understood from the context, which leaves 4.
  - Source IP address.
  - Source port (any number generated by OS of source host.
  - Target IP address.
  - Target port.
- A socket is the endpoint of a connection

Read 23-25

- Does TCP listen on one port and talk on another port ?
- No. TCP listens on 1 port and talk on that same port.
- If clients make multiple TCP connection to server, It's the client OS that must generate **different random source ports**, so that server can see them as unique connections

- What is the maximum number of concurrent TCP connections that a server can handle?
- A single listening port can accept more than one connection simultaneously.
- If a client has many connections to the same port on the same destination, then three of those fields will be the same — only the source port varies to differentiate the different connections.
- Theoretically, Ports are 16-bit numbers, therefore the maximum number of connections **any given client** can have to any given host port is **64K**.
- The **real limit is file descriptors**. Each individual socket connection is given a file descriptor, so the limit is really the number of file descriptors that the system has been configured to allow and resources to handle.

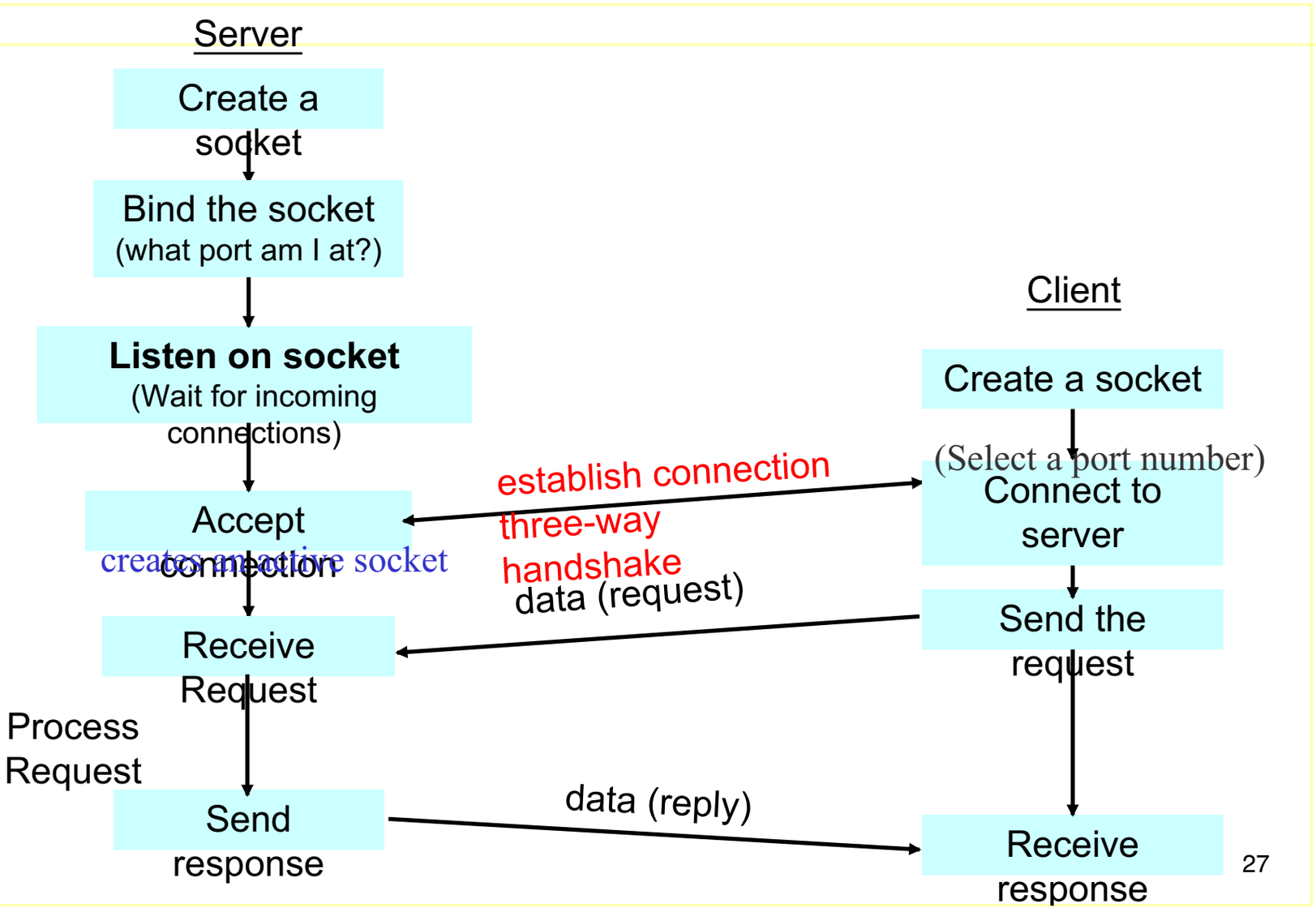
Concurrent connection request vs Concurrent open connection

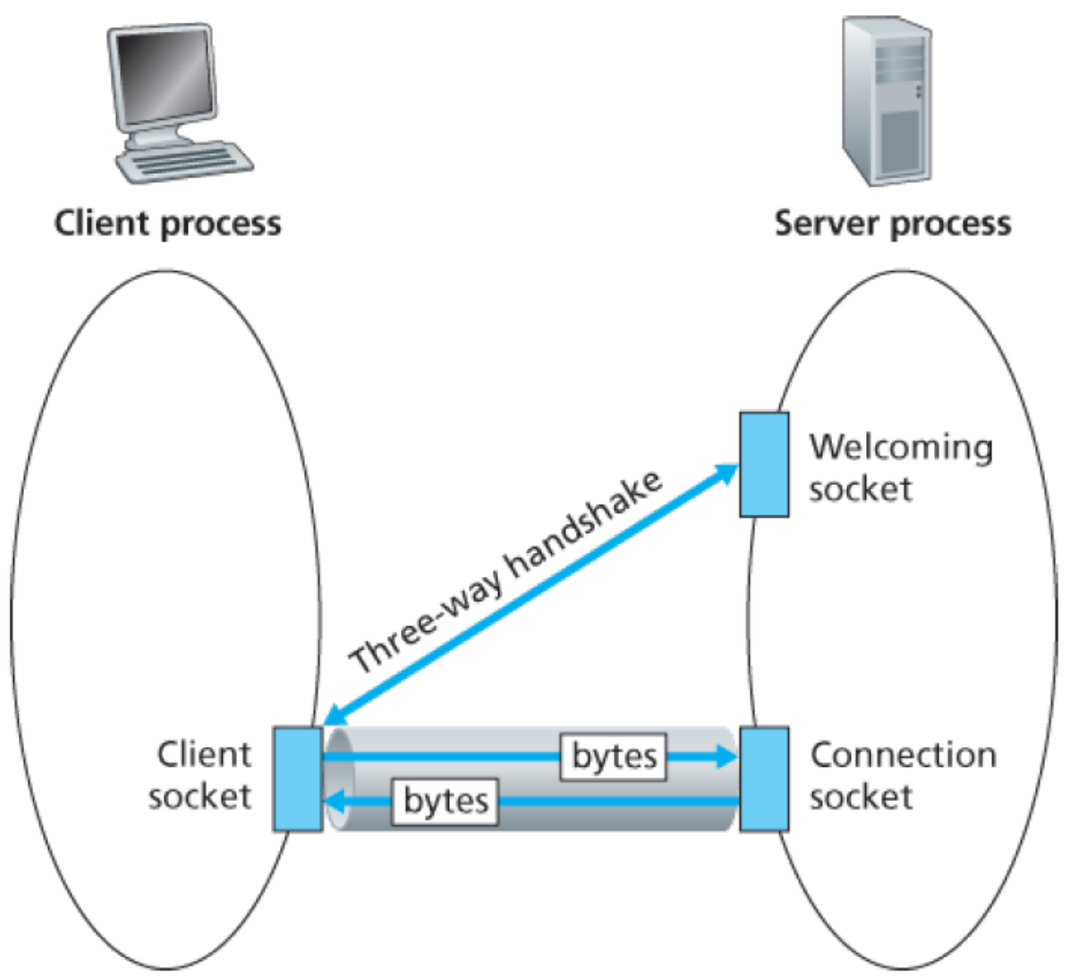
- When clients want to make TCP connection with server, this request will be queued in server ‘s backlog queue. This backlog queue size is small (about 5–10), and this size limits the number of **concurrent connection requests**.
- However, server quickly pick connection request from that queue and accept it. **Connection request which is accepted are called open connection** . The number of concurrent open connections is limited by server ‘s resources allocated for **file descriptor**.

What are active and passive socket ?

- Sockets come in two primary flavors.
- An active socket is con-nect-ed to a remote active socket via an **open data con-nec-tion.**
- A passive socket is not con-nect-ed, but rather awaits an in-com-ing con-nec-tion (it is a listening socket), which will spawn a new active socket once a con-nec-tion is es-tab-lished.
- **Each port** can have a **single passive socket** bound to it, await-ing in-com-ing con-nec-tions, and mul-ti-ple active sockets, each cor-re-spond-ing to an open con-nec-tion on the port.

Client-server communication
Stream sockets: connection-oriented





The TCP Server has two types of sockets

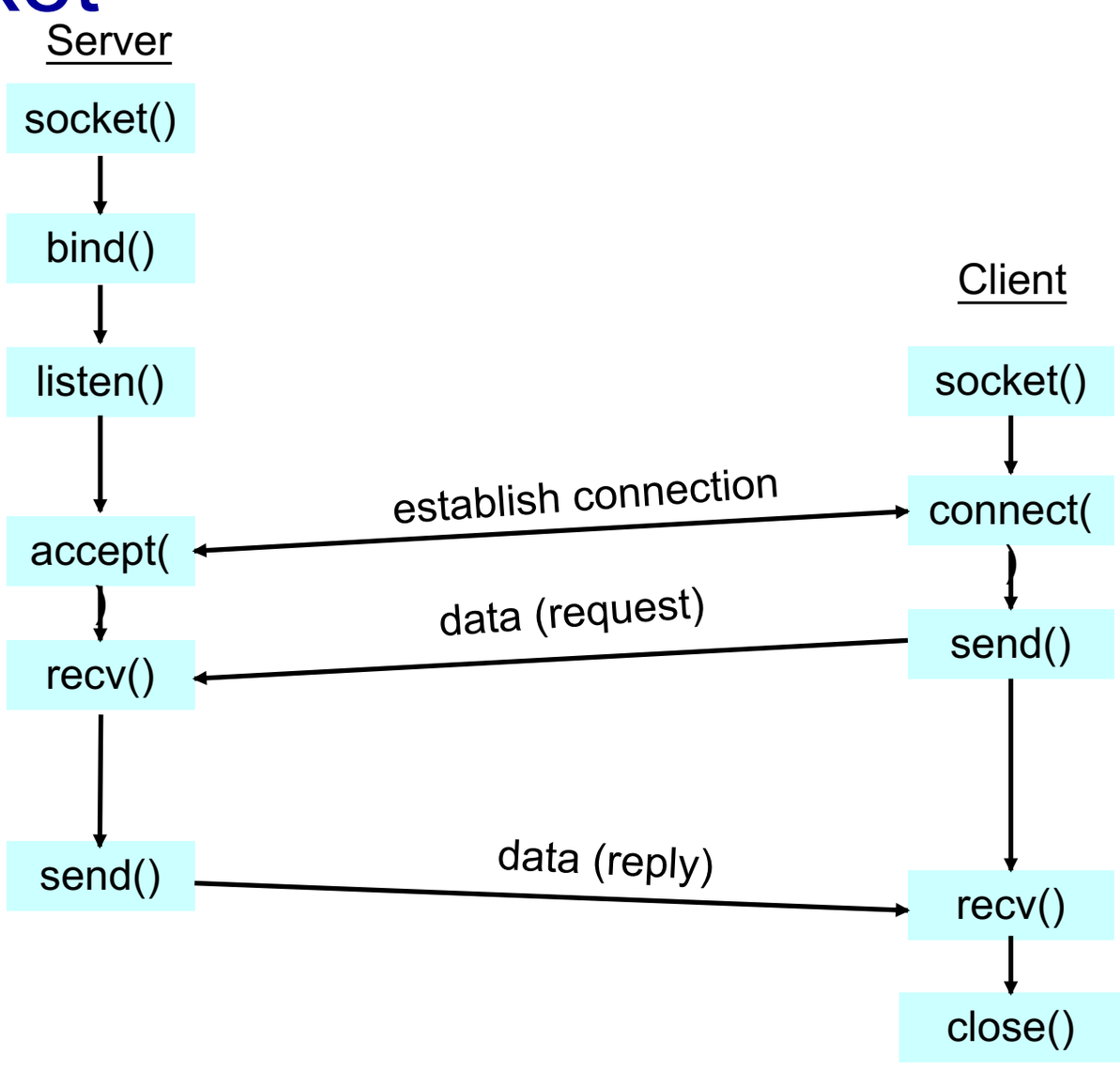
- A listening socket
- Connection sockets (one instance for each connection)

28

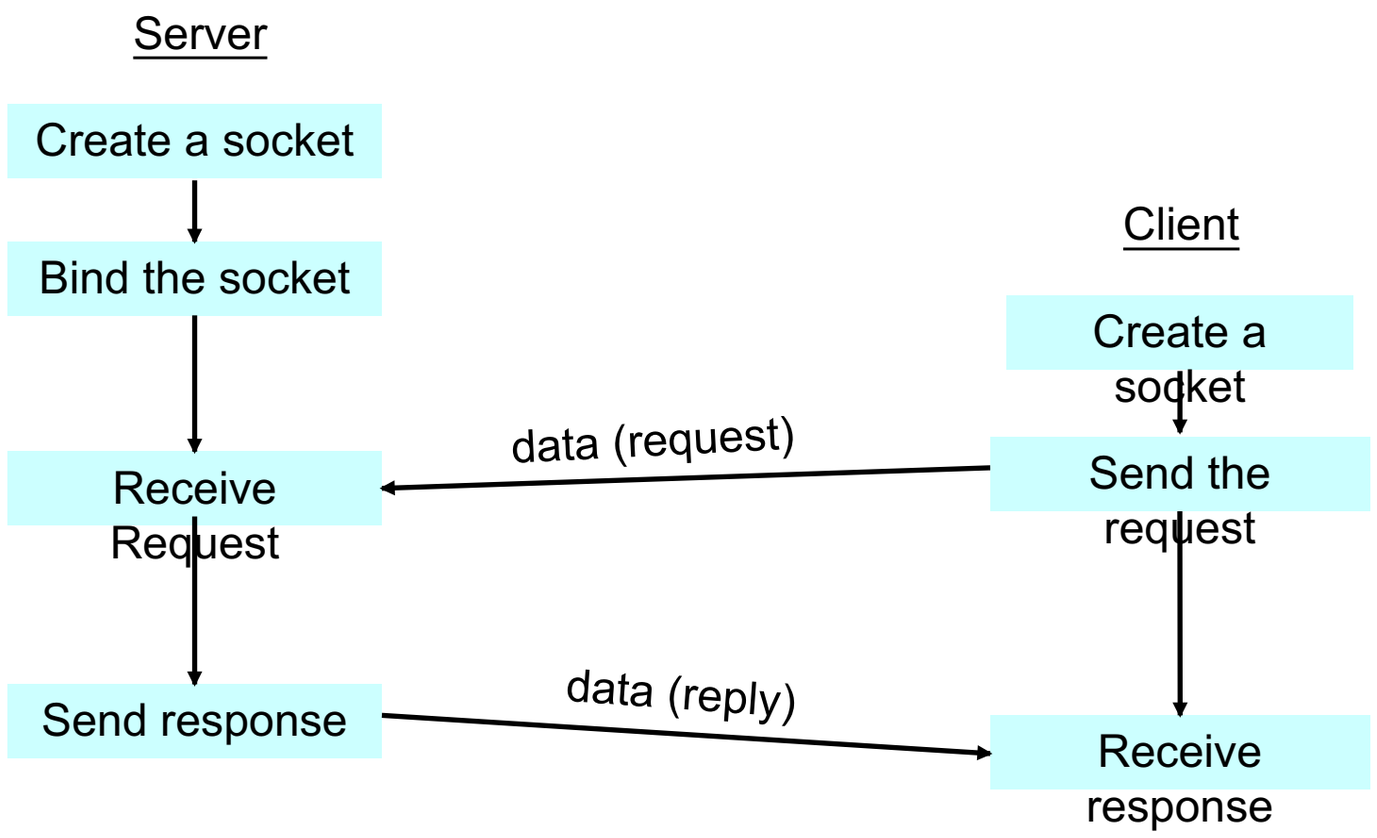
Stream socket communication

- During the **three-way handshake**, the client process knocks on the welcoming door of the server process.
- When the server “hears” the knocking, it creates a new door, more precisely, a ***new* socket**, that is **dedicated to that connection of this particular client**.
- For example: the **welcoming door** is a TCP socket object that we can call ***serverSocket***
- The **new**ly created **socket** dedicated to the client making the connection can be called ***connectionSocket***.

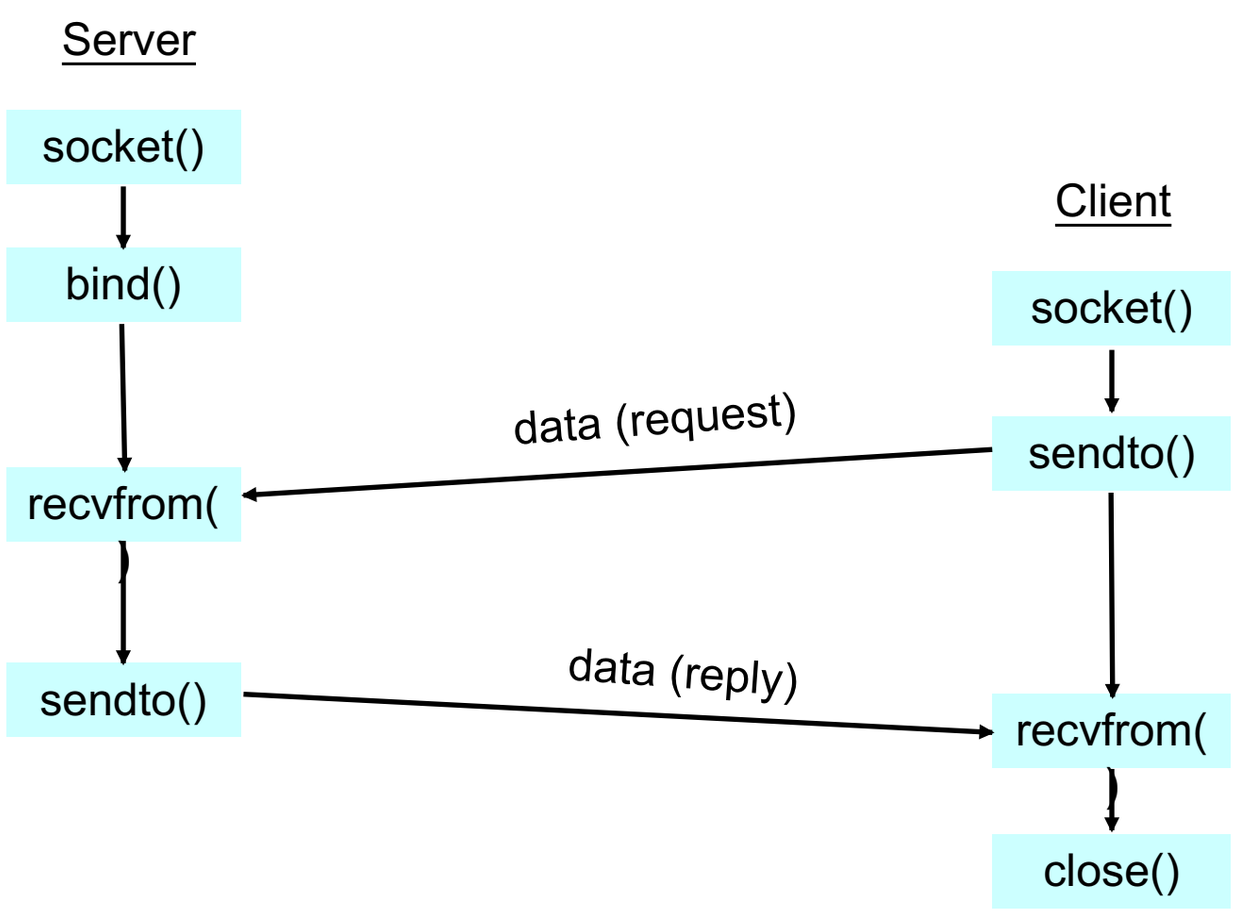
Generic example for a stream socket



Client-server communication
Datagram sockets: **connectionless**



Generic example for a datagram socket



32

Question

At this time, the TCP client **creates a socket** and **sends a connection establishment request**: What are the statements to do that?

Answer

At this time, the TCP client creates a socket and sends a connection establishment request:

clientSocket = socket(AF_INET, SOCK_STREAM)
serverName = ‘www.xxxxx.org’
serverPort = 6789
clientSocket.connect((**serverName**, **serverPort**))

Question

Server should **always** be up and running and listening to the incoming connections. Once it receives the client connection request, what should it do?

Answer

Server should **always** be up and running and listening to the incoming connections. Once it receives the client connection request, what should it do? create a connection socket

while True:

print('The server is ready to receive')

Set up a new "**connection**" **socket** for the client

connectionSocket, addr = serverSocket.accept()

The newly created connection socket is identified by these four values; all subsequently arriving segments whose source port, source IP address, destination port, and destination IP address match these four values will be demultiplexed to this socket.

the web browser's "client" socket and the web server's "connection" socket are peers. This is a "peer to peer" conversation.

As the **designer**, you will have to decide what the rules of etiquette are for a conversation. Normally, the connecting socket starts the conversation, by sending in a request.

example: send a string: sentence from client to server

Client code: encode converts a string to bytes

clientSocket.send(sentence.encode())

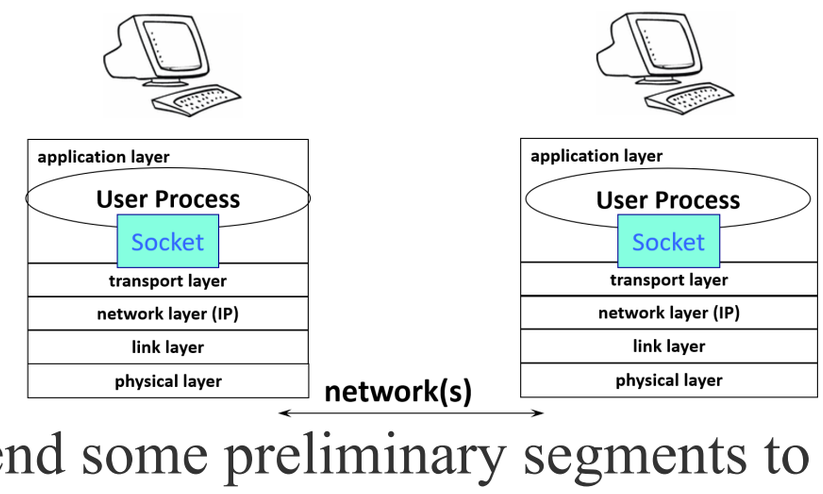
Server:

sentence = connectionSocket.recv(1024).decode()

Server receives the request message from the client, sets size of buffer receiving the message to 1024 bytes and use **decode()** method to convert the message from bytes to string.

Connection-Oriented Transport: TCP

- Before one application process can begin to send data to another, the two processes must first "handshake" with each other



- they must send some preliminary segments to each other to establish the parameters of the data transfer

46

TCP Segment's Header

A TCP segment header includes:

- the source and destination port numbers, which are used for multiplexing/demultiplexing data from/to upper-layer applications.
- A checksum field which is used to detect errors in the segment.
- A sequence number field
- An acknowledgment number field
- A receive window field used for flow control
- in addition to other fields

Sequence Numbers and Acknowledgment Numbers

- TCP views data as an ordered, stream of bytes
- The **sequence number for a segment** is therefore the byte-stream **number of the first byte** in the segment.
- TCP is full-duplex, so that a host may be receiving data from another host while it sends data to it (as part of the same TCP connection).
- The **acknowledgment number** that Host A puts in its segment is the **sequence number of the next byte** Host A is **expect**ing from Host B.

48

Cumulative Acknowledgments

- Example: Host A has received one segment from Host B containing bytes 0 through 535 and another segment containing bytes 900 through 1,000.
- For some reason Host A has not yet received bytes 536 through 899.
- Thus, A's next segment**s** to B will contain **536 in the acknowledgment number field** (similar to Go-Back-N). These acknowledgements are called **duplicate ACK**.
- Host A **keeps the out-of-order bytes** and waits for the missing bytes to fill in the gaps (similar to Selective Repeat)

Example: Suppose Host A sends two TCP segments back to back to Host B over a TCP connection.

The first segment has sequence number 90; the second has sequence number 110.

a. How much data is in the first segment?

The sequence number for a segment is the number of the first byte in the segment.

Example: Suppose Host A sends two TCP segments back to back to Host B over a TCP connection.

The first segment has sequence number 90; the second has sequence number 110.

a. How much data is in the first segment?

The sequence number for a segment is the number of the first byte in the segment.

Answer: 110 – 90 = 20 bytes

b. Suppose that the **first segment is lost** but the second segment arrives at B. In the acknowledgment that Host B sends to Host A, what will be the acknowledgment number?

The acknowledgment number that Host B sends to A is the sequence number of the next byte Host B is expecting from Host A.

Answer: byte 90 (since the packet with sequence 90 was lost)

P27. Host A and B are communicating over a TCP connection, and Host B has already received from A all bytes **up through byte 126**. Suppose Host A then sends two segments to Host B back to back. The first and second segments contain 80 and 40 bytes of data, respectively. In the **first** segment, the sequence number is **127**, the source port number is 302, and the destination port number is 80. Host B sends an acknowledgment whenever it receives a segment from Host A.

a. In the second segment sent from Host A to B, what are the sequence number, source port number, and destination port number?

In the **second** segment from Host A to B, the sequence number is 127+ 80 = 207, source port number is 302 and destination port number is 80.

b. If the first segment arrives before the second segment, in the acknowledgment of the first arriving segment, what is the acknowledgment number, the source port number, and the destination port number?

Reminder: first segment: the sequence number is 127, second segment: sequence number is 207

If the first segment arrives before the second, in the acknowledgement of the first arriving segment, the acknowledgement number is 207 (**the sequence number of the first byte of the second segment**), the source port number is 80 (since **B is the source of this ACK**) and the destination port number is 302 (**A is the destination of this ACK**).

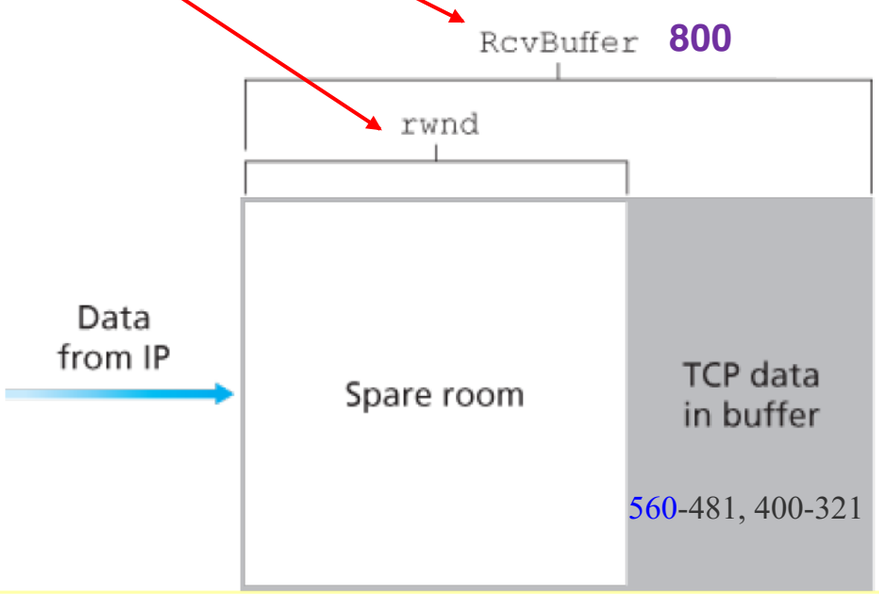
c. If the second segment (let us call it X) arrives before the first segment, in the acknowledgment of the first arriving segment (X), what is the acknowledgment number?

Reminder: first segment: the sequence number is 127,
second segment: sequence number is 207

If the second segment arrives before the first segment, in the acknowledgement of the first arriving segment (X), the acknowledgement number is 127, indicating that it is still waiting for bytes 127 and onwards.

Flow Control of TCP

- TCP provides flow control by having the *sender* maintain a variable called the **receive window** *rwnd*.
- the receive window is used to give the sender an **idea** of how much free buffer space is available at the receiver.
- The receive window, denoted *rwnd* is set to the amount of spare room in the buffer:
- *rwnd=RcvBuffer–[LastByteRcvd–LastByteRead]*
- Because the spare room changes with time, *rwnd* is dynamic.



Receiving out of order packets does not affect rwnd, space should be available to these packets since they may arrive at any time

60

How is rwnd used to provide **flow-control**?

MSS — Maximum segement size

Host B (the **receiver**) places its current value of *rwnd* in the receive window field of **every segment it sends to A** .
Initially, Host B sets *rwnd = RcvBuffer* .

Host A in turn keeps track of two variables, *LastByteSent* and *LastByteAcked*.

Note that the difference between these two variables, *LastByteSent – LastByteAcked* , is the amount of unacknowledged data that A has sent into the connection.

A controls the flow to the receiver by keeping
LastByteSent–LastByteAcked ≤ rwnd

800-721, 720-641, 640-561, 560-481, 400-321 320…

If Host B's receive buffer becomes full it sets its *rwnd* = 0.

To keep receiving ACK from B with updated *rwnd* , Host A keeps sending segments with **one** data byte even if B's receive window is zero.

When B updates *rwnd* , A can resume sending segments to B.

Congestion Control of TCP

- The TCP **congestion-control** mechanism operating at the sender keeps track of an additional variable, the **congestion window**.

- The congestion window, denoted *cwnd* , imposes a constraint on the rate at which a TCP sender can send traffic into the network.

- **Amount of unacknowledged data at a sender ≤ min(*cwnd* ,*rwnd*)**

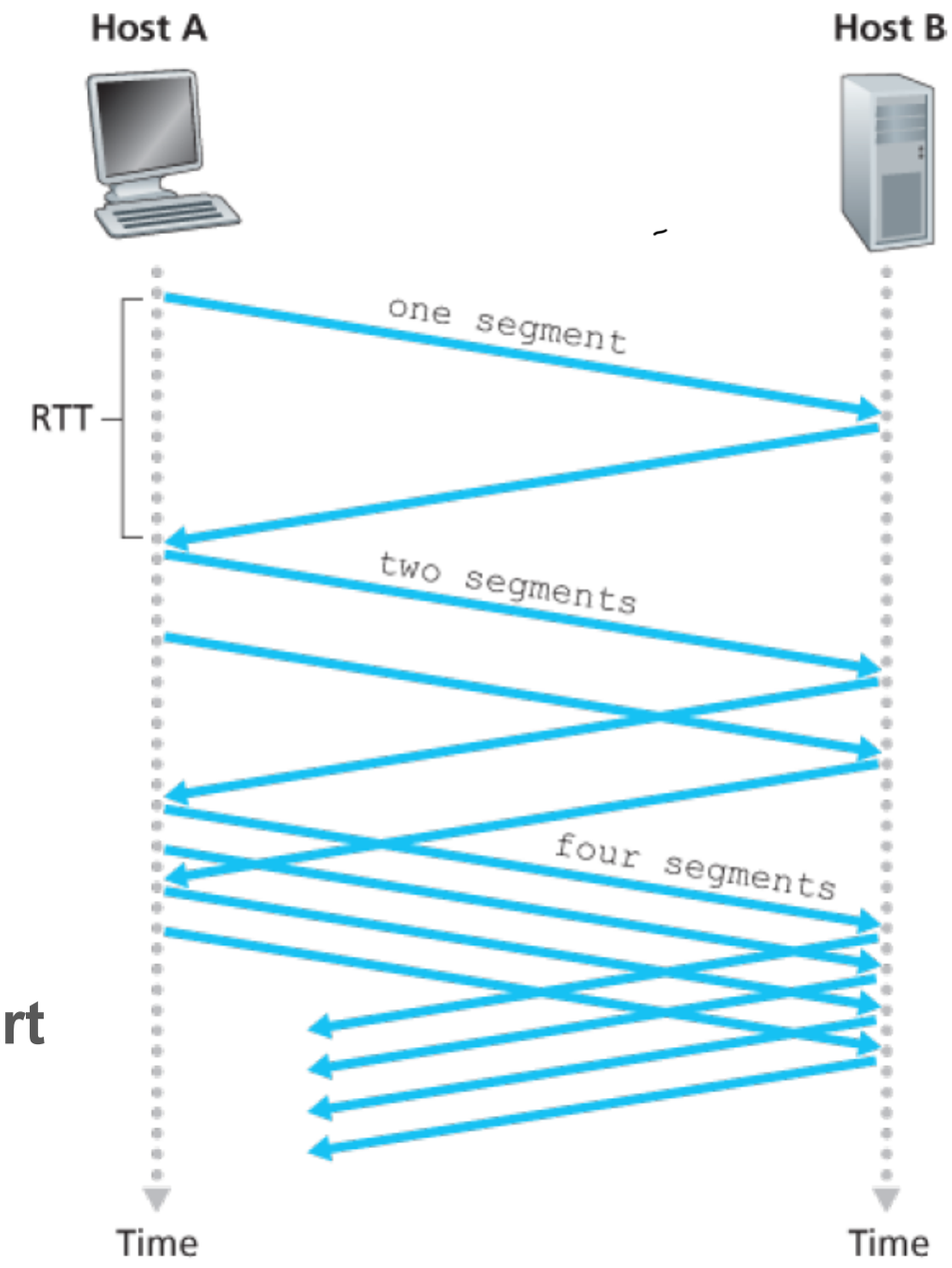
- How is cwnd updated?

TCP congestion-control algorithm has three major components: (1) **slow start**, (2) **congestion avoidance**, and (3) **fast recovery**.

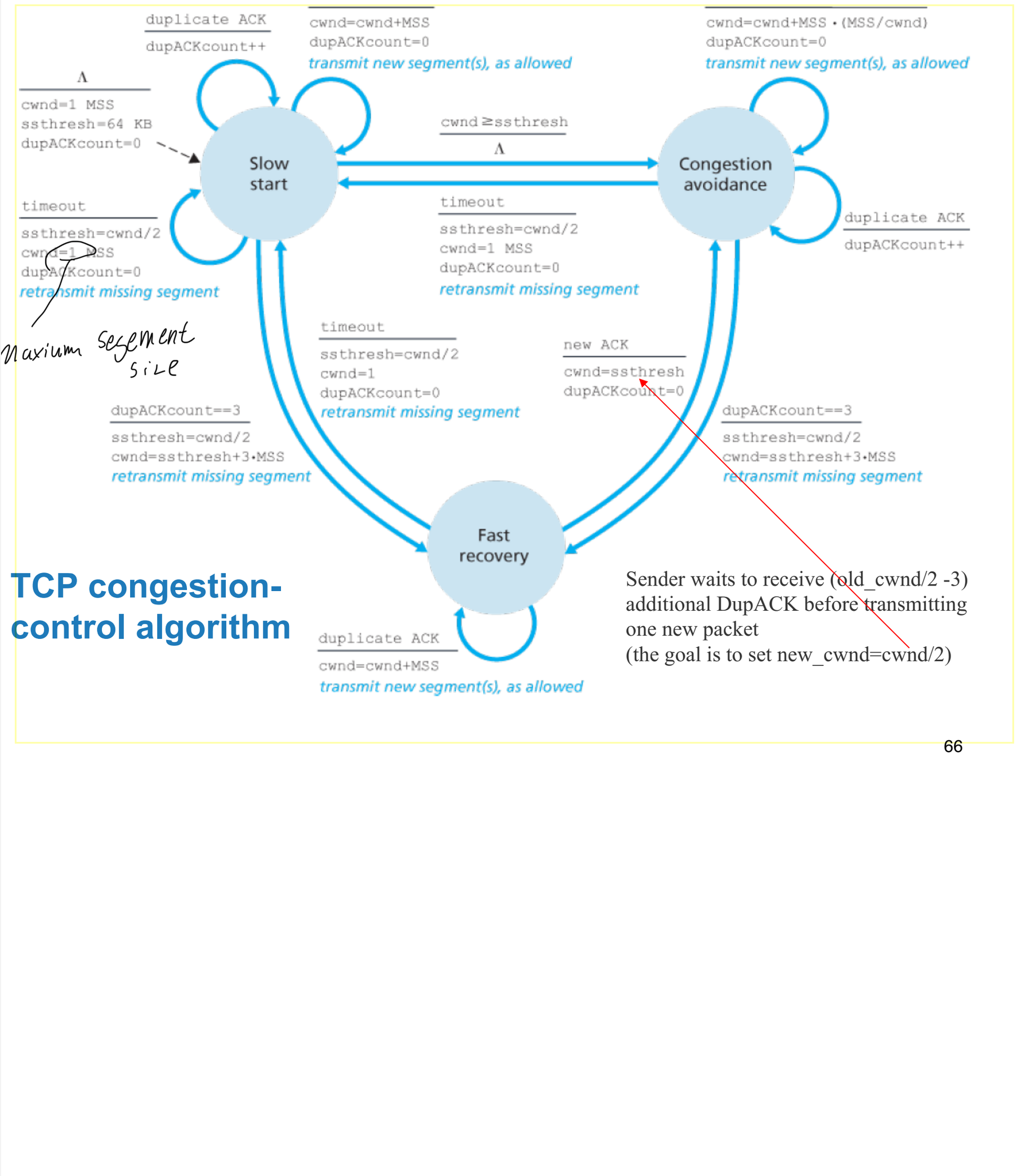
Slow start:

- When a TCP connection begins, the value of *cwnd* is typically initialized to a small value of 1 MSS
- MSS (maximum segment size) is the maximum amount of application-layer data in the segment
- the value of *cwnd* begins at 1 MSS and increases by 1 MSS every time a transmitted segment is first acknowledged
- This process results in a **doubling** of the sending rate every RTT. Thus, **the TCP send rate starts slow ~~but~~ grows exponentially during the slow start phase**.

TCP slow start



65



TCP congestion-control algorithm

Sender waits to receive (old_cwnd/2 -3) additional DupACK before transmitting one new packet
(the goal is to set new_cwnd=cwnd/2)

when should this exponential growth end?

1. If there is a **loss event** (i.e., congestion) indicated by a **timeout,** the TCP sender begins the **slow start** process again.
2. It sets the value of a second state variable, ***ssthresh*** ("slow start threshold") to ***cwnd/2*** *(*half of the value of the congestion window value when congestion was detected.) and **sets the value of** ***cwnd***
3. When the value of *cwnd* equals *ssthresh*, slow start ends and TCP transitions into **congestion avoidance mode**.
4. The final way in which slow start can end is if **three duplicate ACKs** are detected, in which case TCP performs a **fast retransmit** and enters the **fast recovery mode**
5. What does 3 duplicate ACK mean?

Example:

- From the sender's perspective, if we send packets 1,2,3,4,5,6 and get back ACK[1], ACK[2], ACK[2], ACK[2], ACK[2], we can infer two things:

- Data[3] got lost, which is why we are stuck on ACK[2] (retransmitting missing segment means retransmit Data[3])
- Data 4,5 and 6 *probably* did make it through, and triggered the three duplicate ACK[2]s (the three ACK[2]s following the first ACK[2]).

Congestion Avoidance 避免

- On entry to the congestion-avoidance state, the value of *cwnd* is approximately half its value when congestion was last encountered.
- TCP adopts a more conservative approach and increases the value of *cwnd* by just a single MSS every RTT (i.e. when all segments of *cwnd* have been acknowledged)
- TCP sender increases *cwnd* by (MSS/ *cwnd*) * MSS whenever a new acknowledgment arrives.
- Example: if MSS = 1,460 bytes and *cwnd* is 14,600 bytes, then 10 segments are being sent within an RTT.
- Each arriving ACK increases the congestion window size by 1/10 *MSS, and thus, total increase of the congestion window = one MSS after ACKs of all 10 segments have been received.

Leaving Congestion Avoidance state

- If a timeout event occurs,

  ssthresh = cwnd/2, (cwnd is the value when the loss event occurred)

  and cwnd = 1

  Then, Congestion Avoidance transitions to the slow-start state

- When a triple duplicate ACK is received,

  ssthresh = cwnd/2; and cwnd = cwnd/2 + 3

  Then, Congestion Avoidance transitions to the Fast recovery state

Fast Recovery

- When a packet is lost, the goal is to set new_cwnd=oldcwnd/2, and go to Congestion avoidance. How this is done?
- The idea is to use the arriving dupACKs to pace retransmission. We assume that each arriving dupACK indicates that some data packet following the lost packet has been delivered successfully, decreasing the number of packets in the network.
- Sender waits to receive (old_cwnd/2 -3) additional DupACK before transmitting one new packet, Why?

fast-recovery state

- When a triple duplicate ACK is received, TCP updates ssthresh to be half the value of cwnd and halves the value of cwnd (adding in 3 MSS for good measure to account for the triple duplicate ACKs received):

  ssthresh = cwnd/2; and cwnd = cwnd/2 + 3

- Then, the fast-recovery state is entered.
- In this state, the value of cwnd is increased by 1 MSS for every duplicate ACK received **for the missing segment** that caused TCP to enter the fast-recovery state.

Leaving fast-recovery state

- when an ACK arrives for the missing segment, TCP sets:

  cwnd = ssthresh (which is the old cwnd/2)

  and enters congestion-avoidance state.

- If a timeout event occurs,

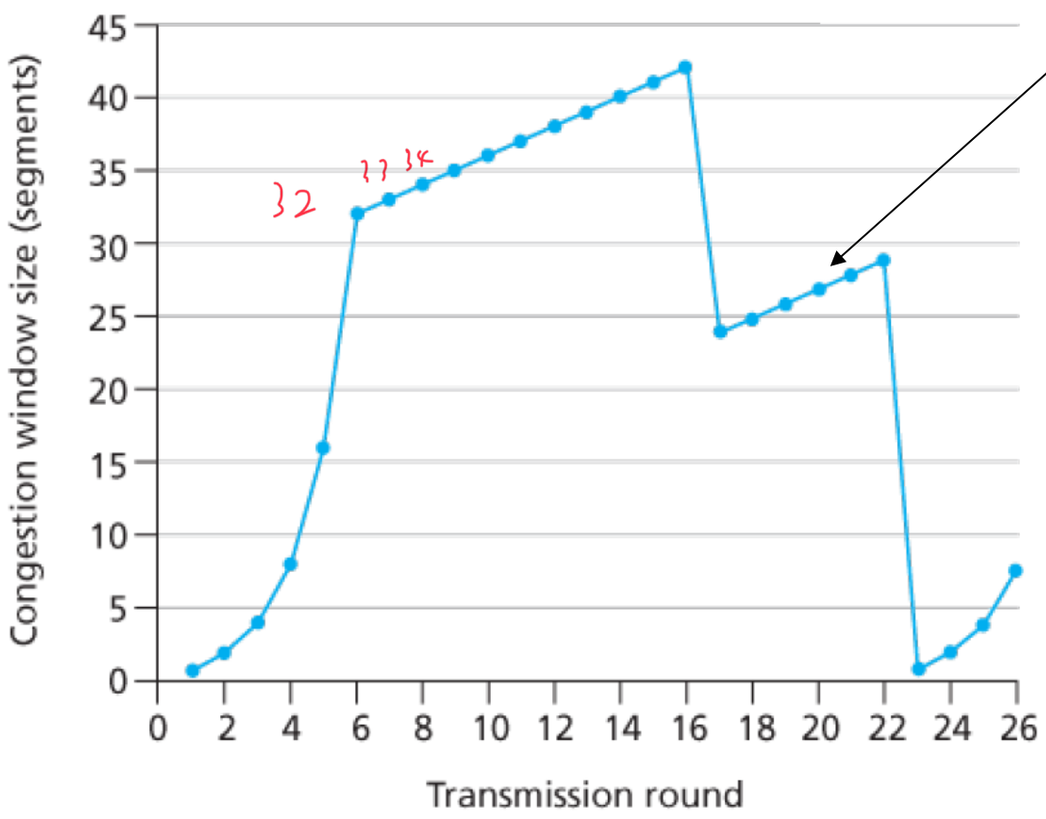
  ssthresh = cwnd/2, (cwnd is the value when the loss event occurred)

  and cwnd = 1

- Then, fast recovery transitions to the slow-start state

Assuming TCP Reno is the protocol experiencing the behavior shown below, answer the following questions:
a. Identify the intervals of time when TCP slow start is operating.
b. Identify the intervals of time when TCP congestion avoidance is operating.

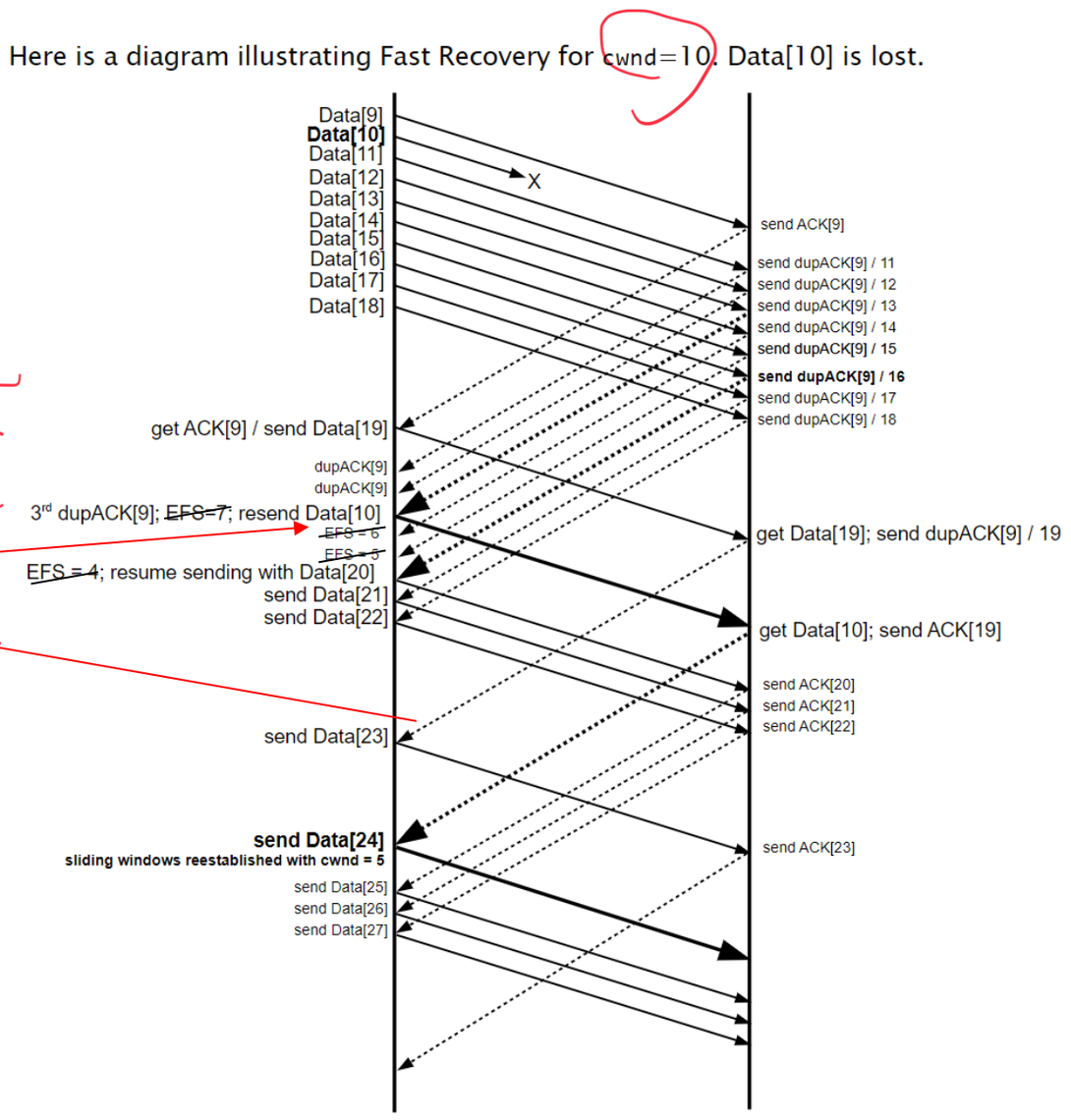
Discussion 22



This is symbolic and not accurate. Sender waits to receive (old_large_cwnd – new_smaller_cwnd = 42-24= 18) additional DupACK before transmitting one new packet (the goal is to set cwnd=cwnd/2)

- cwnd = 10,
- after 3 duplicate ACK, new cwnd=8 cannot send new Packets
- **wait** to receive: 10-8=2 more DupACK (old cwnd-new cwnd), without sending new packets
- For each of remaining 10 -1-5 = 4 new DupACK, send a new packet
- When ACK of retransmitted packet is received, go to congestion avoidance state with cwnd = 10/2=5

- Note: packet nb is used instead of seq nb for simplicity



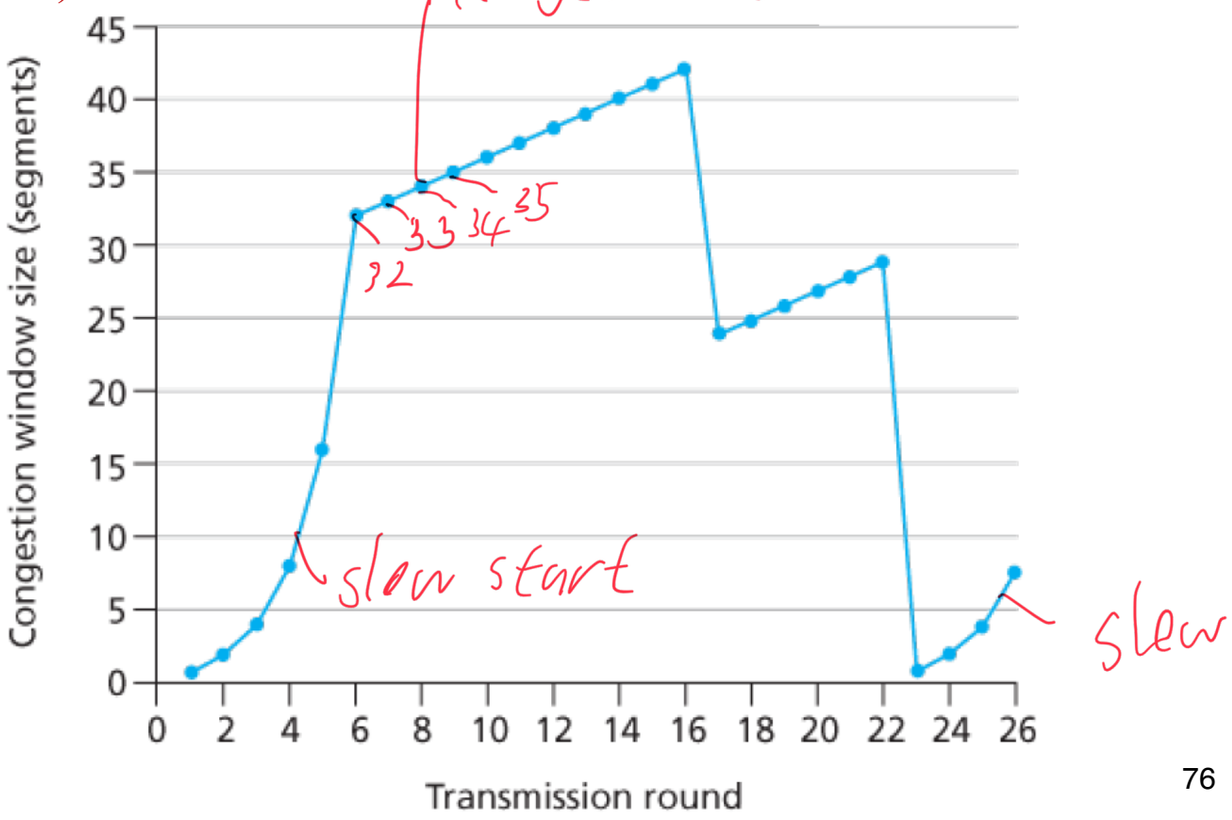
Assuming TCP Reno is the protocol experiencing the behavior shown below, answer the following questions:

a. Identify the intervals of time when TCP slow start is operating.

b. Identify the intervals of time when TCP congestion avoidance is operating.

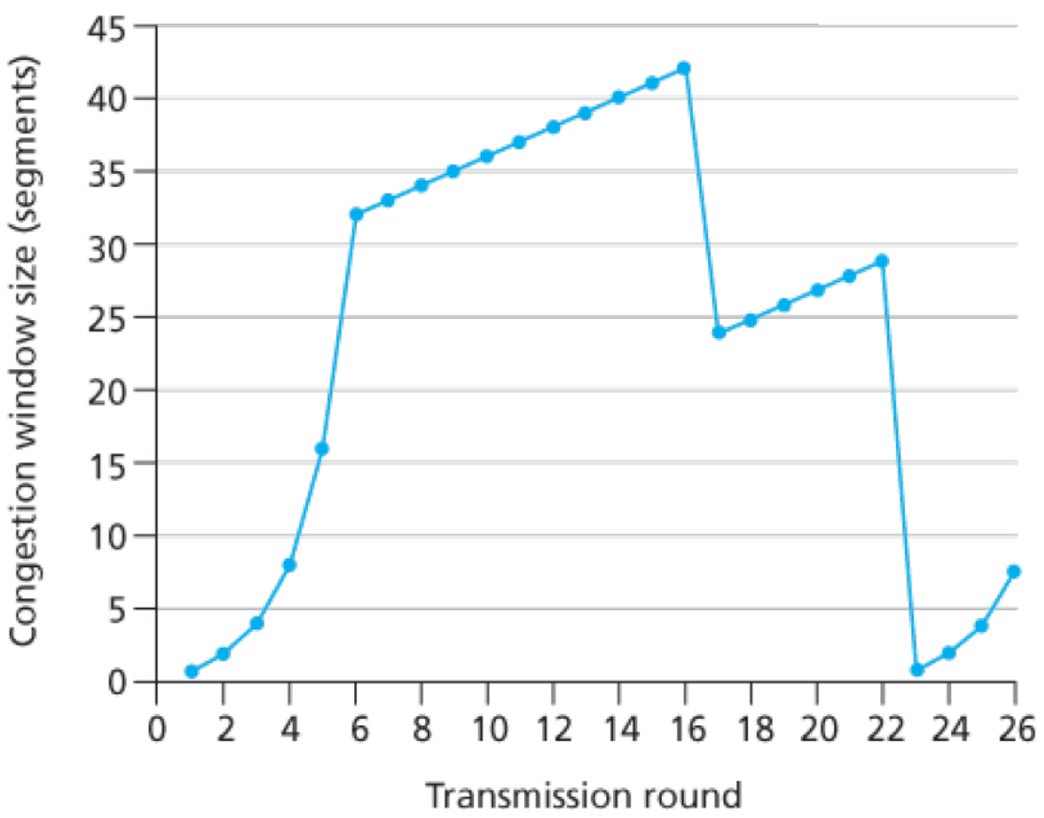
a) TCP **slowstart** is operating in the intervals [1,6] and [23,26] (exponential increase)

b) TCP **congestion avoidance** is operating in the intervals [6,16] (cwnd is not reduced, linear increase)

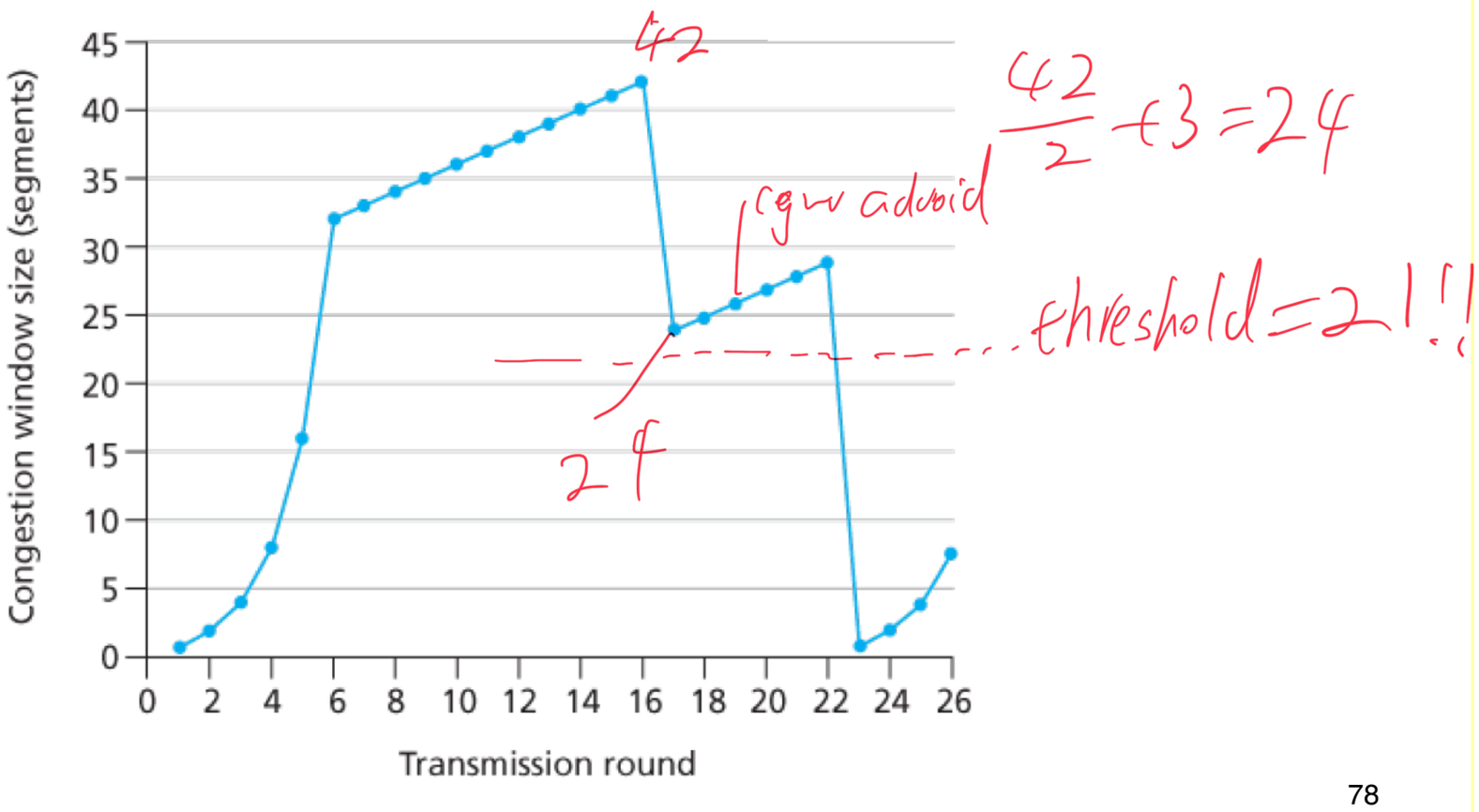


c. After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?
d. After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

Discussion 23

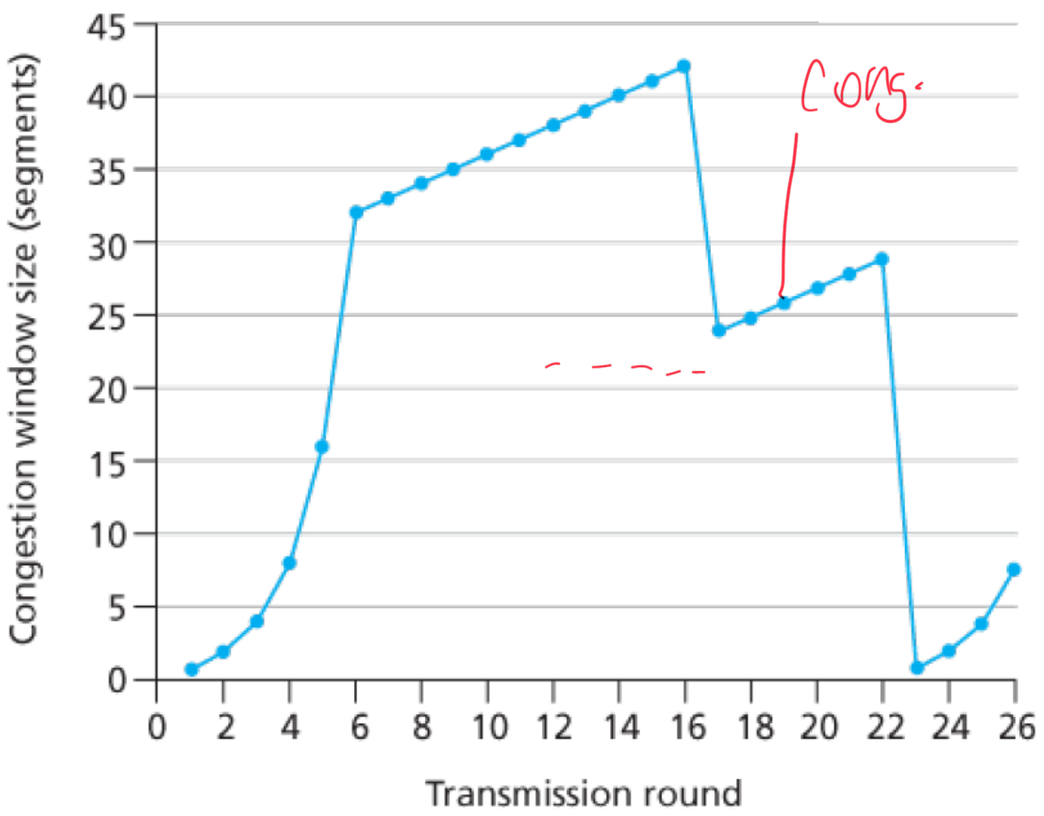


c. After the 16th transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?
d. After the 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?
c) packet loss is recognized by a triple duplicate ACK. cwnd = cwnd/2+3. If there was a timeout, the congestion window size would have dropped to 1.
d) segment loss is detected due to timeout, and hence the congestion window size is set to 1.



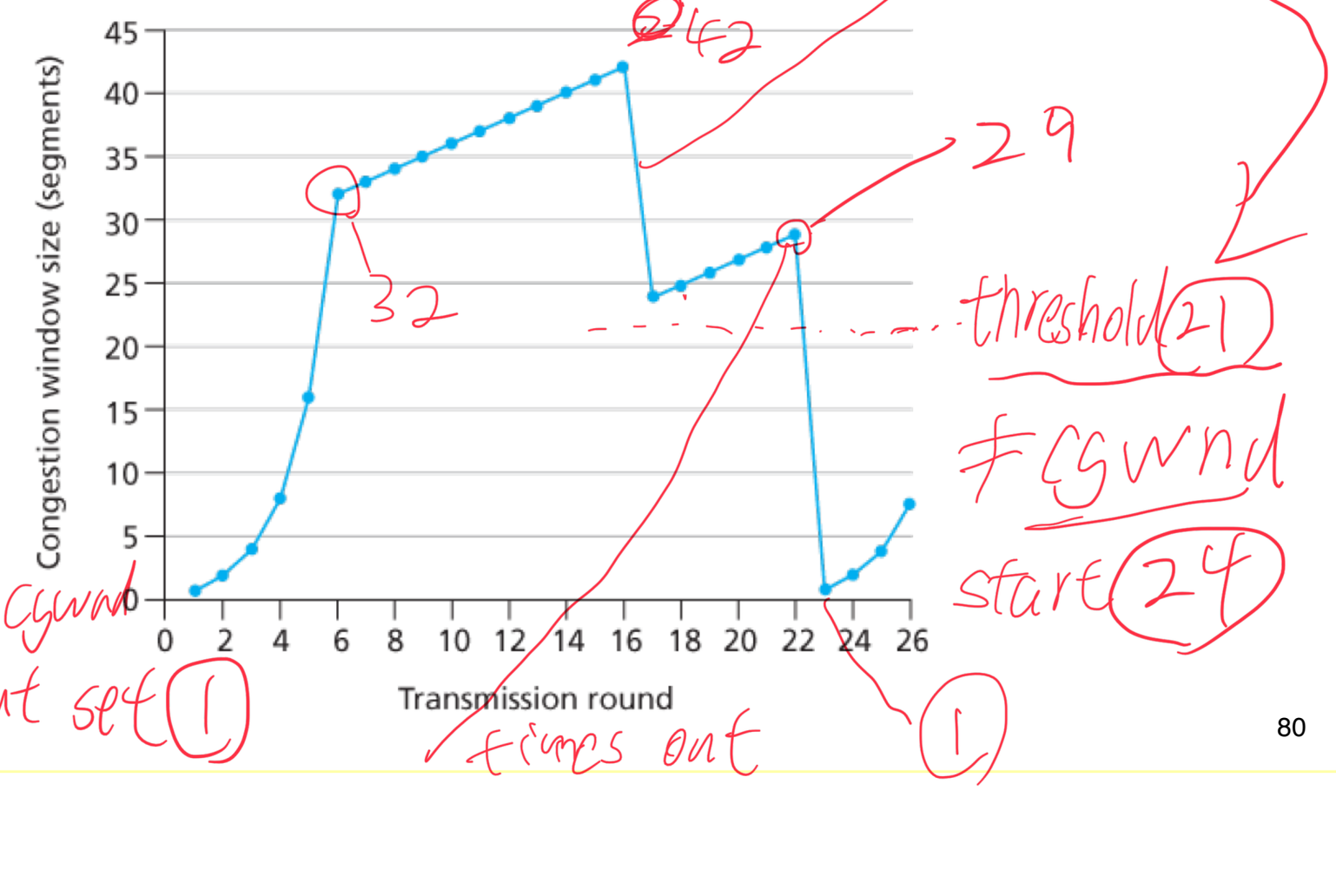
e. What is the initial value of ssthresh at the first transmission round?
f. What is the value of ssthresh at the 18th transmission round?
g. What is the value of ssthresh at the 24th transmission round?

Discussion 24



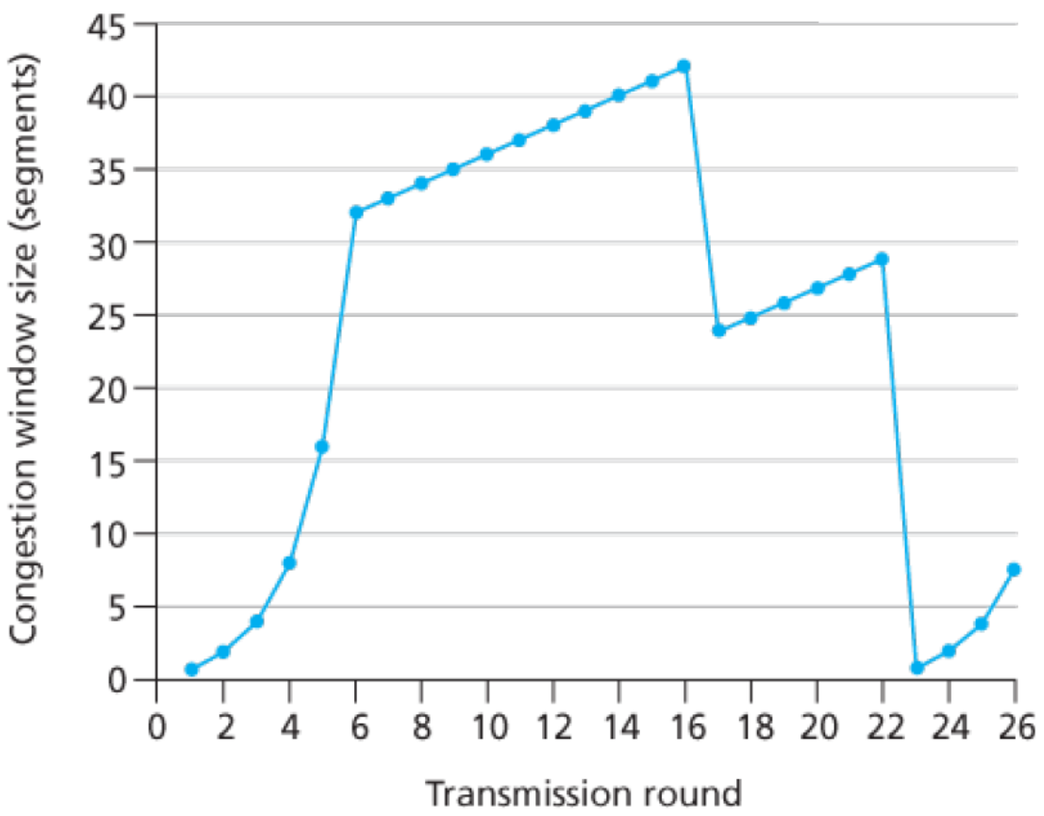
e. What is the initial value of ssthresh at the first transmission round?
f. What is the value of ssthresh at the 18th transmission round?
g. What is the value of ssthresh at the 24th transmission round?

e) The threshold is initially 32, since it is at this window size that slow start stops and congestion avoidance begins.
f) ssthresh=cwnd/2 when packet loss is detected = 42/2=21
g) ssthresh=cwnd/2 when packet loss is detected = floor(29/2) = 14



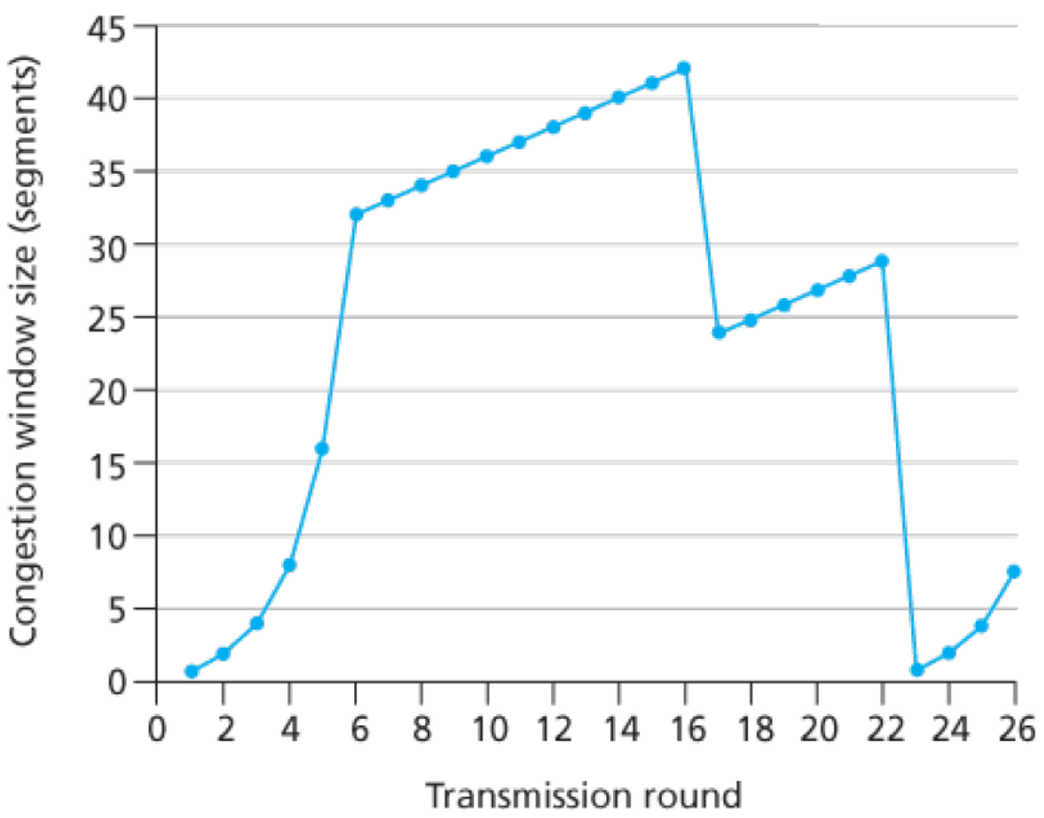
i) Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of ssthresh ?

Discussion 25



i) Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of ssthresh ?

i) ssthresh=cwnd/2 when packet loss is detected = 8/2=4
cwnd=ssthreshold + 3 = 4+3 = 7



82

Discussion 26

True or false?
Consider congestion control in TCP. When the timer expires at the sender, the value of ssthresh (slow start threshold) is set to one half of its previous value.

True or false?
Consider congestion control in TCP. When the timer expires at the sender, the value of ssthresh (slow start threshold) is set to one half of its previous value.

Answer:
False, it is set to half of the current value of the congestion window: ssthresh = cwnd/2

Checksum

To detect errors, UDP and TCP segments include a checksum in their header fields.

The bits of the checksum are found by adding the bits of all words in the segment (wrap the carry and add to result's rightmost bit), then finding their 1's complement.

Let us practice adding binary numbers (base 2):

Decimal digits: 0 1 2 3 4 5 6 7 8 9
Decimal addition: 2+1=3 7+1=8

		Carry	Sum
0 + 0	=	0	0
0 + 1	=	0	1
1 + 1	=	1	0
1 + 1 + 1	=	1	1

Figure 7.6 Rules of binary addition.

Binary digits : 0 1

$$\begin{array}{r} 0 \\ +1 \\ \hline 1 \end{array} \qquad \begin{array}{r} {\scriptstyle 1} \\ 1 \\ +\,1 \\ \hline 1\,0 \end{array} \qquad \begin{array}{r} {\scriptstyle 1} \\ 1 \\ 1 \\ +\,1 \\ \hline 1\,1 \end{array}$$

$$\begin{array}{r} 9 \\ +\,1 \\ \hline 1\,0 \end{array} \qquad \begin{array}{r} {\scriptstyle 1\;1\;1} \\ 0\,1\,1 \\ +\,1\,1\,1 \\ \hline 1\,0\,1\,0 \end{array}$$

Checksum

P3. Suppose the message that should be transmitted contains the following three bytes: 01010011, 01100110, 01110100.

Find the checksum of this message: i.e

- Add the three bytes
- In case of overflow, wrap the carry and add to result's rightmost bit
- Find the 1s complement of the final result

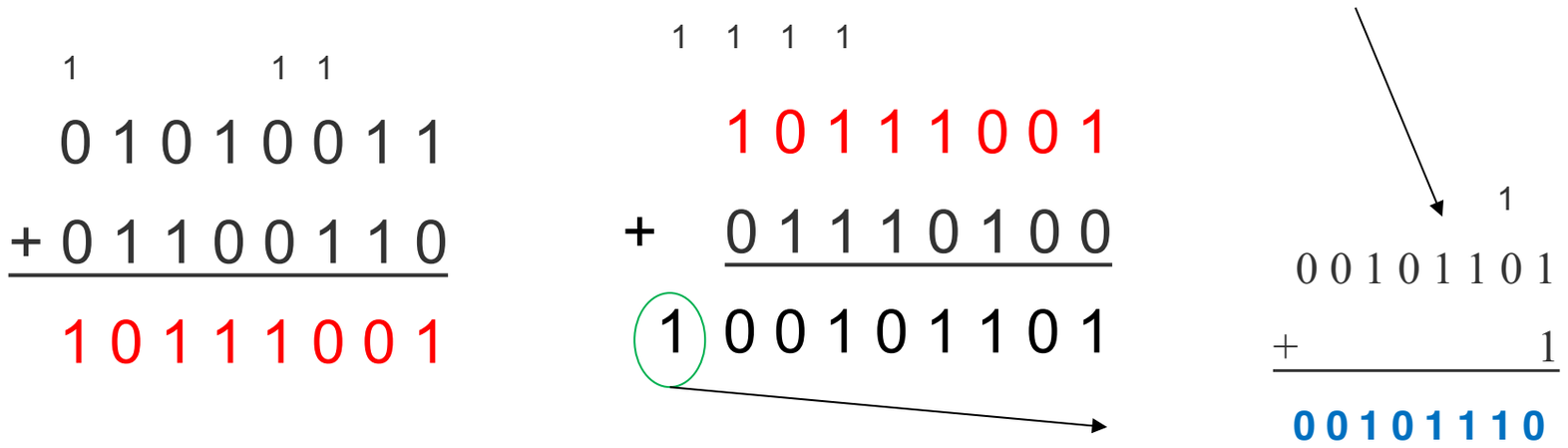
The 1's complement of a binary number is obtained by changing the 1s to 0s and the 0s to 1s.

(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.)

Checksum

Suppose the message that should be transmitted contains the following three 8-bit bytes: 01010011, 01100110, 01110100.

We start by adding the first 2 numbers, and then we add the third number to the result. (in case of overflow, wrap the carry and add to result)

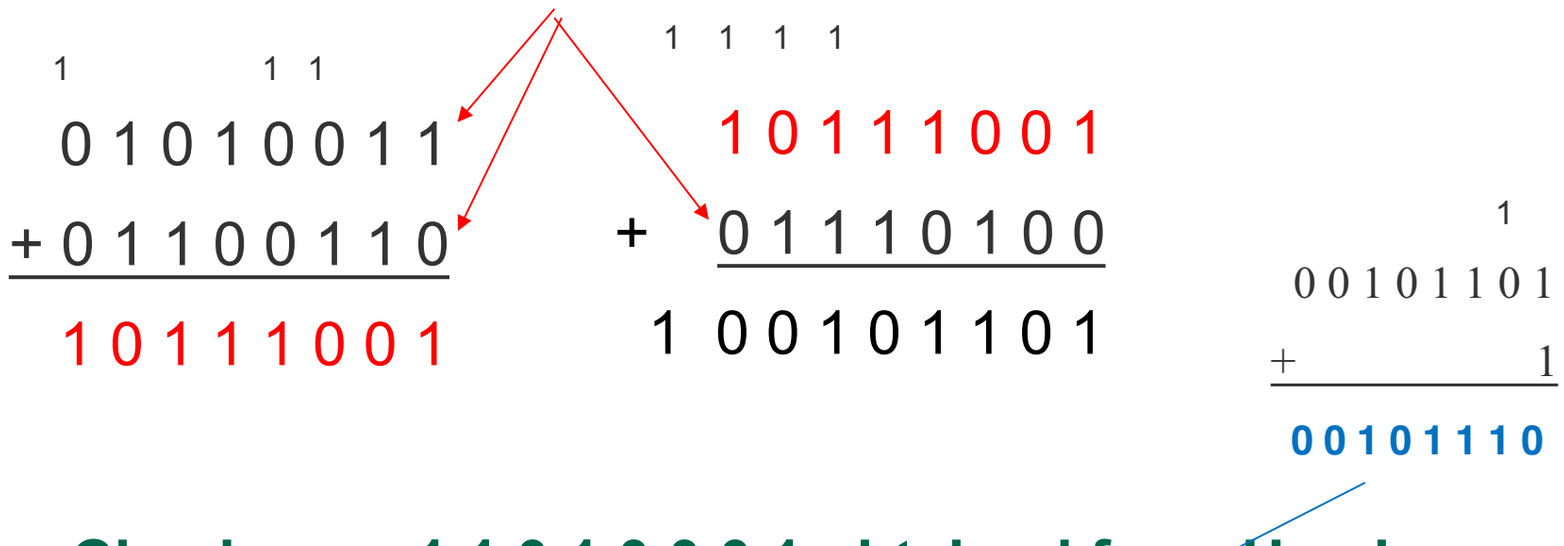


1's complement of result: flip 0s and 1s

=> Checksum is: 1 1 0 1 0 0 0 1

At the receiver

If the bytes received are as shown below; Receiver adds them to the checksum.



Checksum: 1 1 0 1 0 0 0 1 obtained from Header

90

The result obtained is: 0 0 1 0 1 1 1 0 sum of 3 bytes

+ 1 1 0 1 0 0 0 1 checksum

1 1 1 1 1 1 1 1 : all 1s=> no errors detected

At the receiver: in case of single-bit error

Suppose the received bytes are: 01010011, 01100110, 01110100. They are added to the checksum received: **1 1 0 1 0 0 0 1**

1

The result obtained is:

1 0 1 0 1 1 1 0 sum of 3 bytes

\+ 1 1 0 1 0 0 0 1 checksum

1 0 1 1 1 1 1 1 1 : a 0 in answer => error

1 1 1 1

0 1 0 1 0 0 1 1

\+ 0 1 1 0 0 1 1 0

1 0 1 1 1 0 0 1

1 0 1 1 1 0 0 1

\+ 0 1 1 1 0 1 0 0

1 1 0 0 1 0 1 1 0 1

1

1

1 0 1 0 1 1 0 1

\+ 1

1 0 1 0 1 1 1 0

91

Checksum

Is it possible that a 1-bit error will go undetected? How about a 2-bit error?

All one-bit errors will be detected, but two-bit errors can be undetected (e.g., if the last digit of the first word is converted to a 0 and the last digit of the second word is converted to a 1).

P5. Suppose that the UDP receiver computes the Internet checksum for the received UDP segment and finds that it matches the value carried in the checksum field. Can the receiver be absolutely certain that no bit errors have occurred?

No, the receiver cannot be absolutely certain that no bit errors have occurred. This is because of the manner in which the checksum for the packet is calculated.
If the corresponding bits (that would be added together) of two 16-bit words in the packet were 0 and 1 then even if these get flipped to 1 and 0 respectively, the sum still remains the same. Hence, the 1s complement the receiver calculates will also be the same. This means the checksum will verify even if there was transmission error.

$$T_{\text{trans\_data}} = \frac{10,400 \text{ bits}}{400,000 \text{ bps}} = 0.026 \text{ seconds } (26 \text{ ms})$$

2. **ACK Frame Transmission Time ((T_{\text{trans_ACK}})):**

$$T_{\text{trans\_ACK}} = \frac{400 \text{ bits}}{400,000 \text{ bps}} = 0.001 \text{ seconds } (1 \text{ ms})$$

Step 2: Total Time Per Frame

In the Stop and Wait protocol, the total time to send one frame and receive its acknowledgment is the sum of:

- **Data Frame Transmission Time**
- **Round-Trip Time (RTT)**
- **ACK Frame Transmission Time**

Thus,

$$\text{Total Time} = T_{\text{trans\_data}} + \text{RTT} + T_{\text{trans\_ACK}} = 0.026 \text{ s} + 6 \text{ s} + 0.001 \text{ s} = 6.027 \text{ seconds}$$

Step 3: Calculate Effective Throughput

Effective throughput is the ratio of useful data sent to the total time taken.

$$\text{Throughput} = \frac{\text{Useful Data}}{\text{Total Time}} = \frac{10,000 \text{ bits}}{6.027 \text{ s}} \approx 1,659 \text{ bps}$$

Final Answer:

- **Effective Data Throughput using Stop and Wait:** Approximately **1,659 bits per second (bps)**

(b) Sliding Window Protocol with (W = 5)

In a sliding window protocol, the sender can transmit (W) frames before needing ACKs.

1. **Total transmission time for (W) frames:**

$$T_{\text{tx}} = W \times T_d = 5 \times 0.026 = 0.13 \text{ seconds}$$

2. **Time when the last data frame is fully transmitted:**

$$T_{\text{last\_data\_sent}} = 0.13 \text{ seconds}$$

3. **Time when the last ACK is received:**

$$T_{\text{last\_ACK\_received}} = T_{\text{last\_data\_sent}} + 2T_p + T_a = 0.13 + 6 + 0.001 = 6.131 \text{ seconds}$$

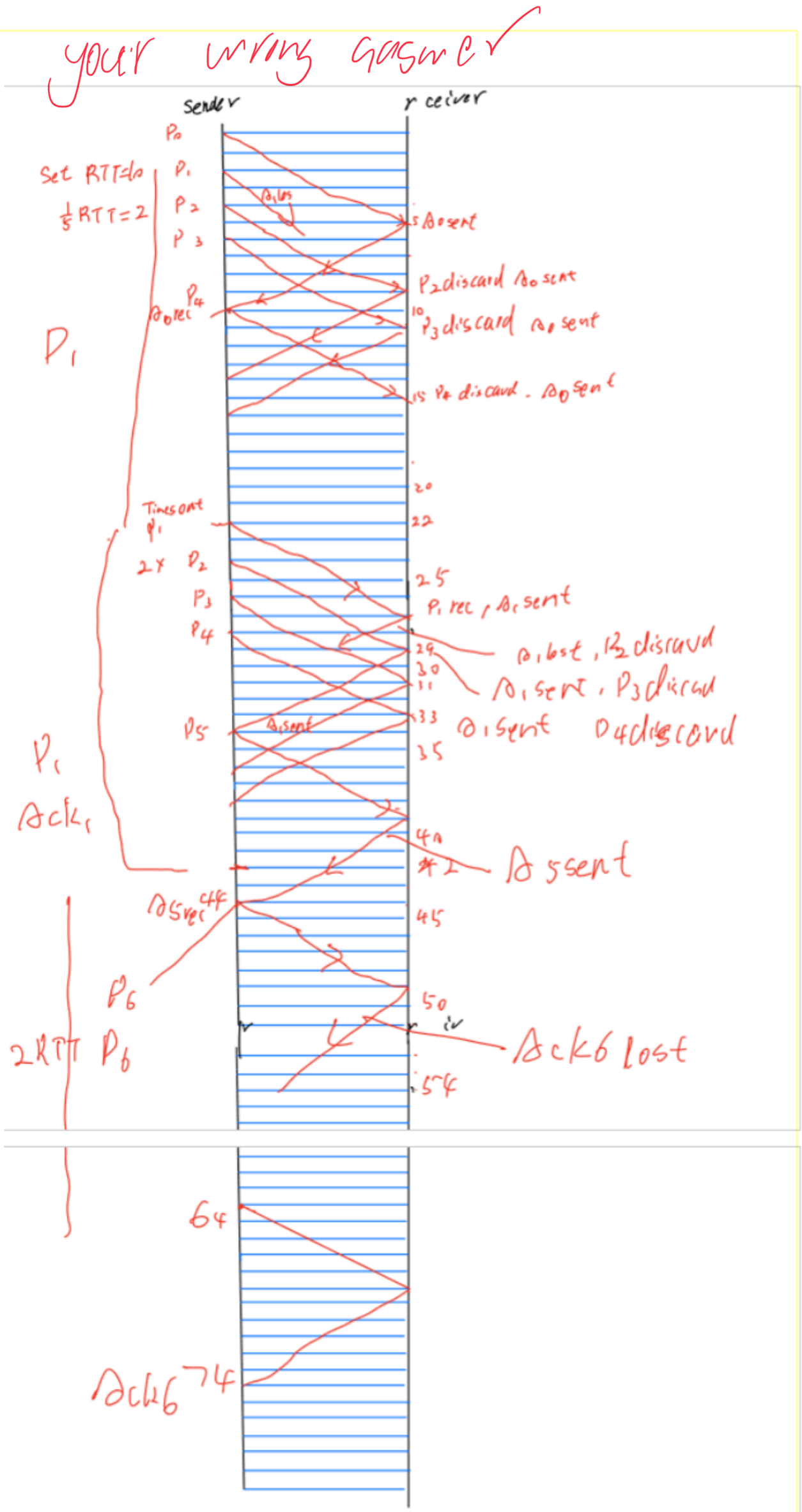
4. **Total useful data sent:**

$$\text{Total useful data} = W \times P = 5 \times 10,000 = 50,000 \text{ bits}$$

5. **Effective throughput (\eta):**

$$\eta = \frac{\text{Total useful data}}{\text{Total time}} = \frac{50,000 \text{ bits}}{6.131 \text{ seconds}} \approx 8155.28 \text{ bits per second}$$

E7668CFE-8171-4D2B-81EC-DFF53266DC7B



E7668CFE-8171-4D2B-81EC-DFF53266DC7B