

COMP 333 I/933 I: Computer Networks and Applications

Week 2

Transport Layer (Continued)

Reading Guide: Chapter 3, Sections: 3.4, 3.5

Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

rdt3.0: channels with errors *and* loss

new assumption:

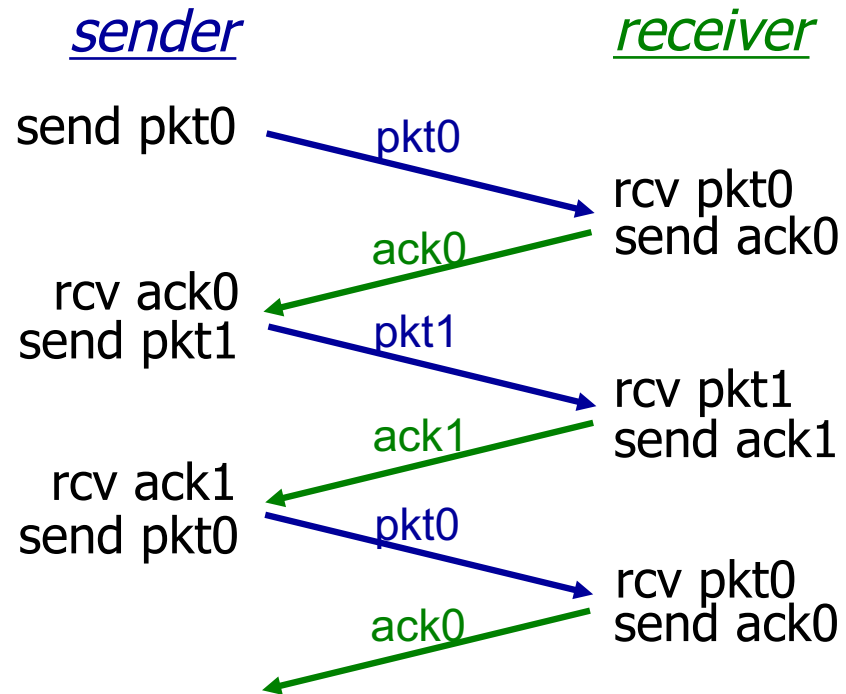
underlying channel can also lose packets (data, ACKs)

- checksum, seq. #, ACKs, retransmissions will be of help ... but not enough

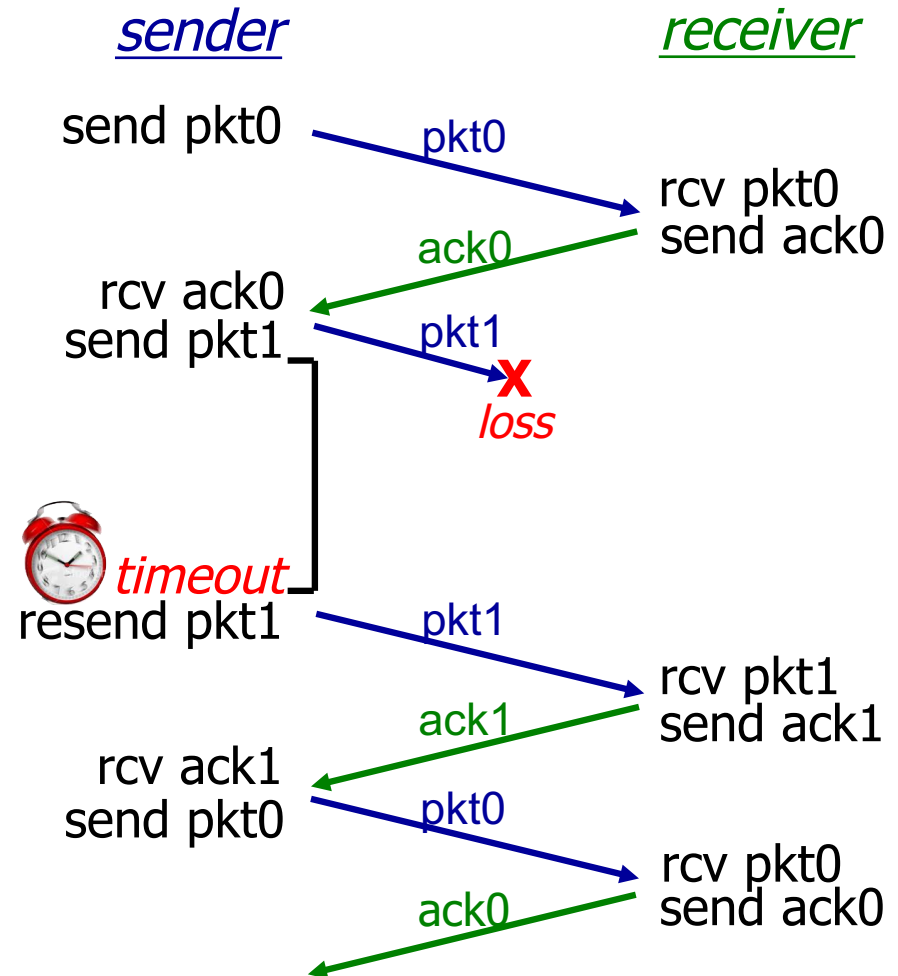
approach: sender waits “reasonable” amount of time for ACK

- ❖ retransmits if no ACK received in this time
- ❖ if pkt (or ACK) just delayed (not lost):
 - retransmission will be duplicate, but seq. #'s already handles this
 - receiver must specify seq # of pkt being ACKed
- ❖ requires countdown timer

rdt3.0 in action

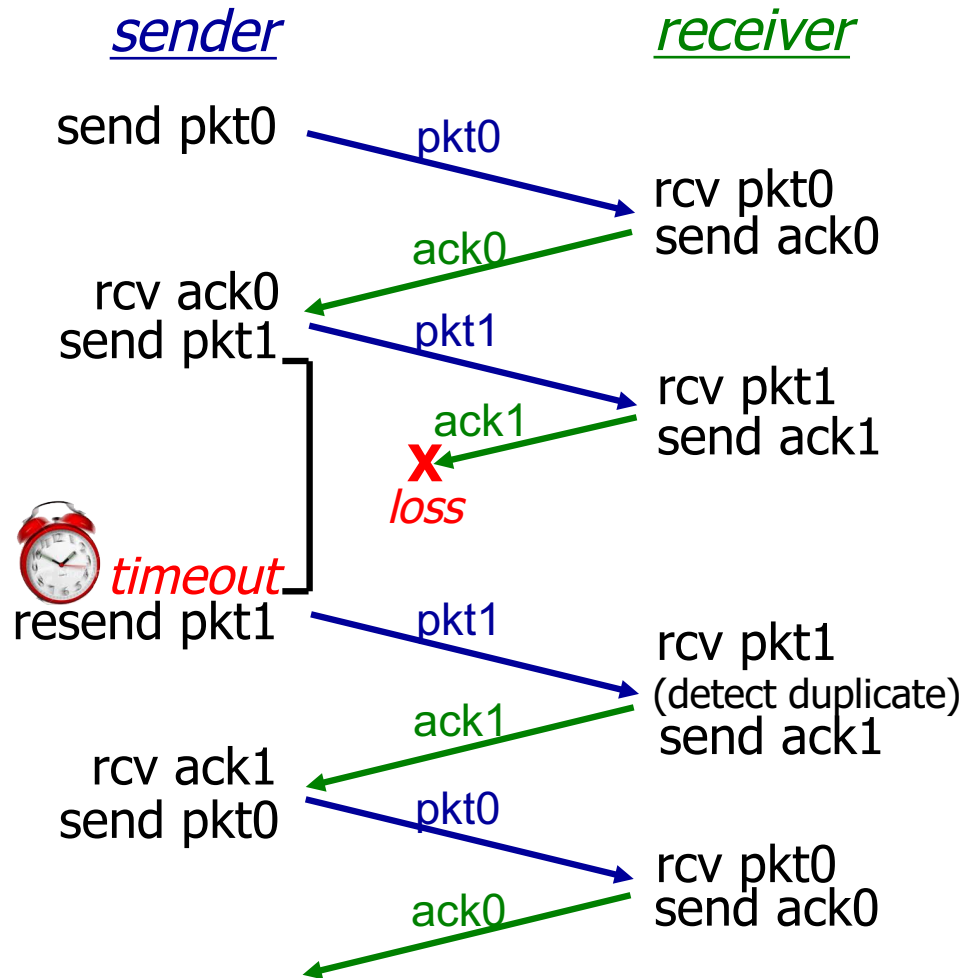


(a) no loss

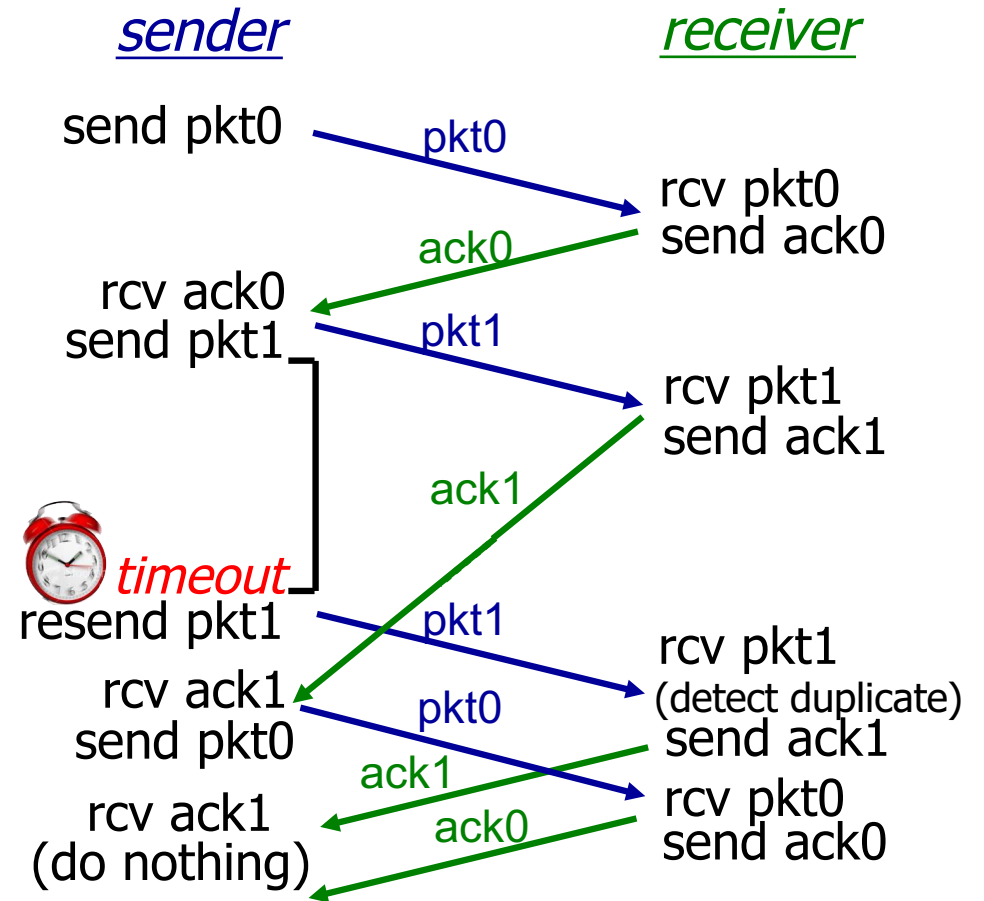


(b) packet loss

rdt3.0 in action

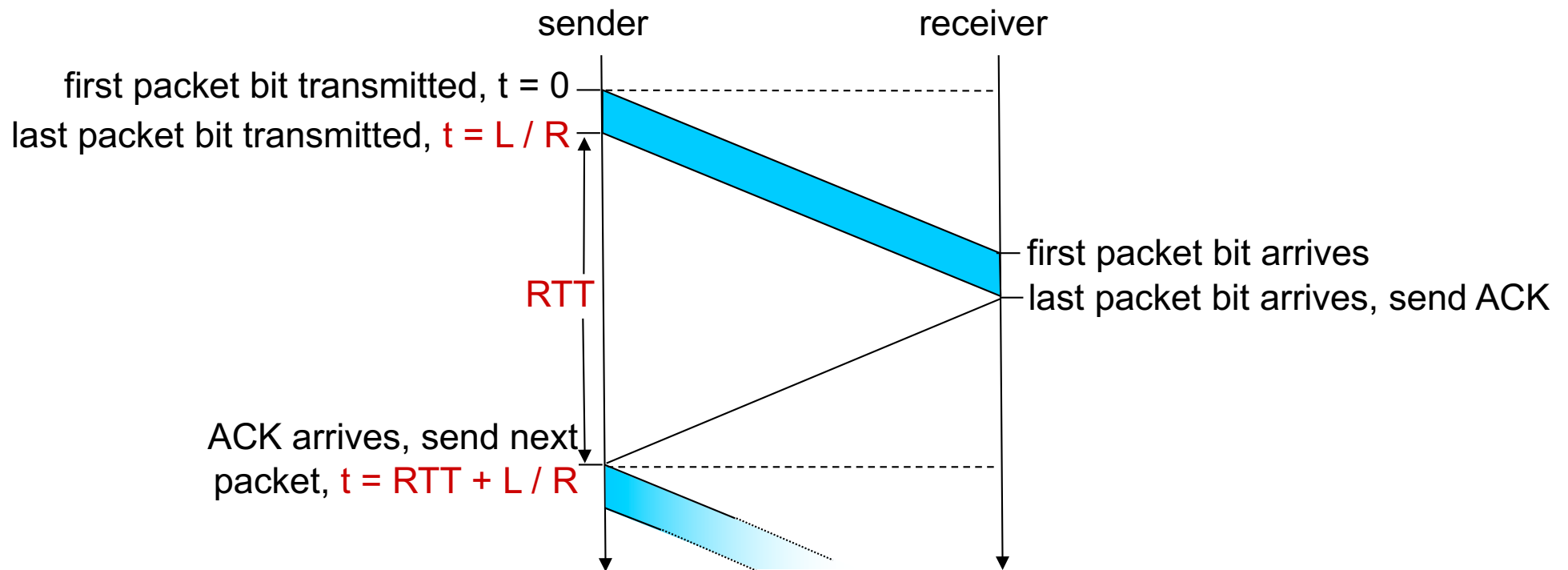


(c) ACK loss



(d) premature timeout/ delayed ACK

rdt3.0: stop-and-wait operation



$$U_{\text{sender}} = \frac{L/R}{RTT + L/R}$$

Performance of rdt3.0

- rdt3.0 is correct, but performance stinks
- e.g.: 1 Gbps link, 8000 bit packet and 30msec RTT:

$$D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microseconds}$$

- U_{sender} : **utilization** – fraction of time sender busy sending

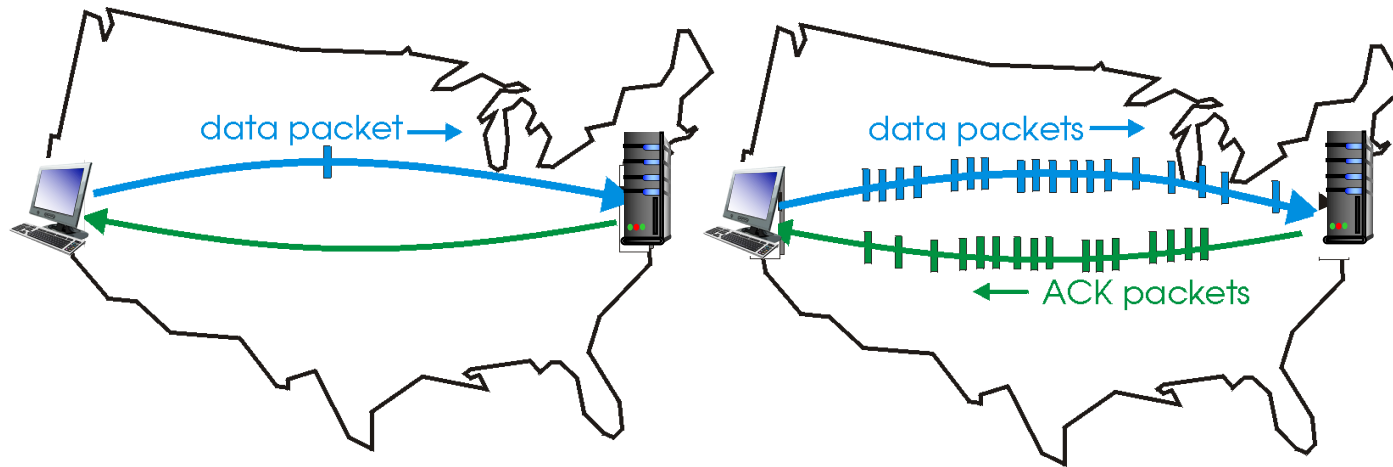
$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

- RTT=30 msec, 1KB pkt every 30.008 msec: 33kB/sec thruput over 1 Gbps link
- Network protocol limits use of physical resources!

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

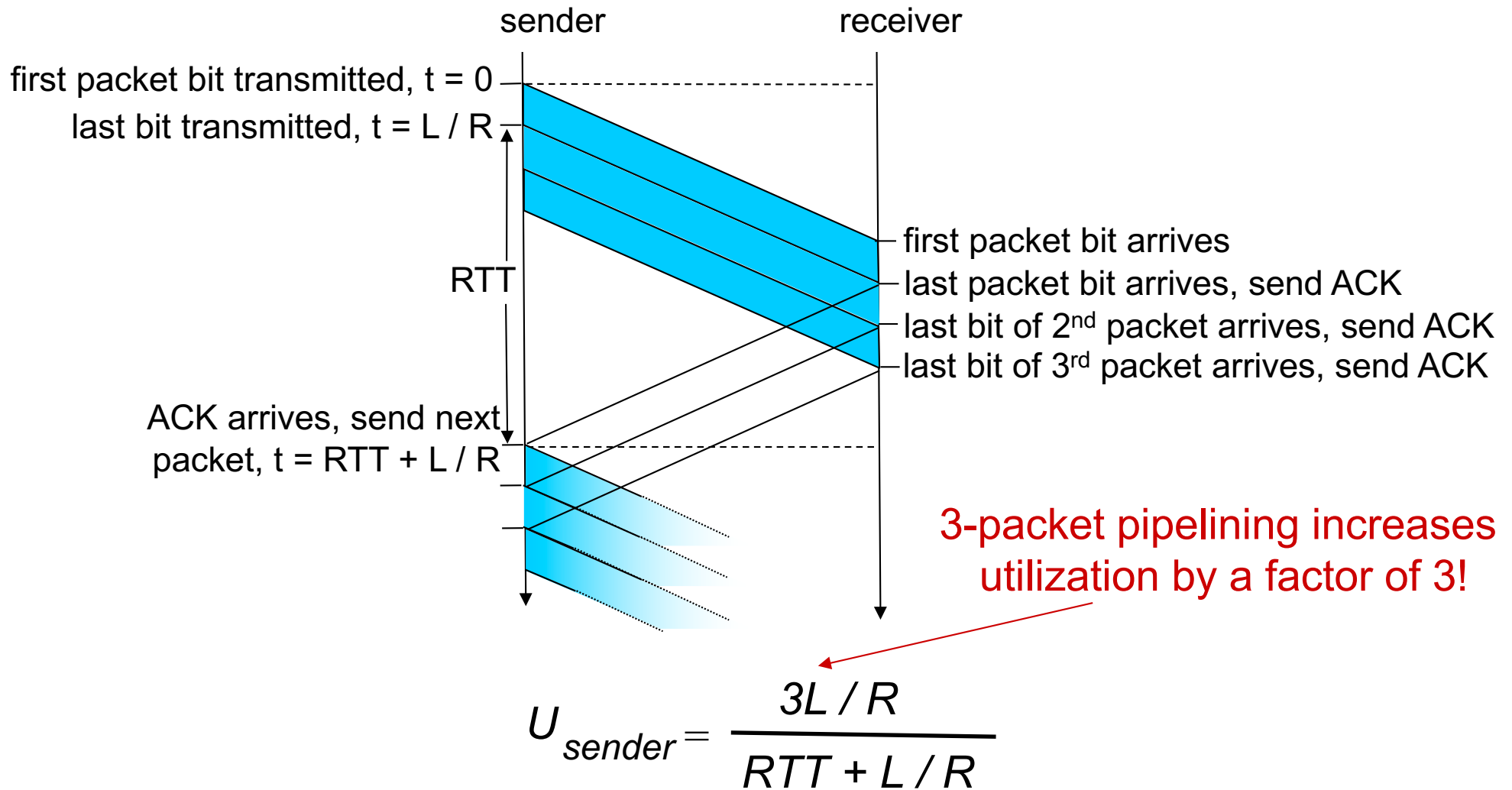


(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

- ❖ two generic forms of pipelined (sliding window) protocols: *go-Back-N*, *selective repeat*

Pipelining: increased utilization



Pipelined protocols: overview

Go-Back-N:

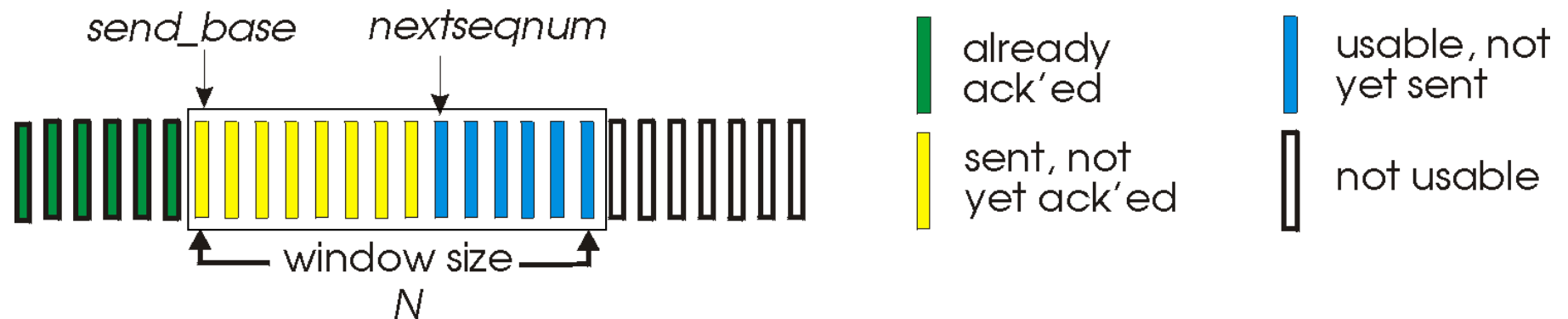
- Sender can have up to N unacked packets in pipeline
- Sender has **single timer** for oldest unacked packet, when timer expires, retransmit *all* unacked packets
- There is no buffer available at Receiver, out of order packets are discarded
- Receiver only sends **cumulative ack**, doesn't ack new packet if there's a gap

Selective Repeat:

- Sender can have up to N unacked packets in pipeline
- Sender maintains timer for each unacked packet, when timer expires, retransmit only that unacked packet
- Receiver has buffer, can accept out of order packets
- Receiver sends **individual ack** for each packet

Go-Back-N: sender

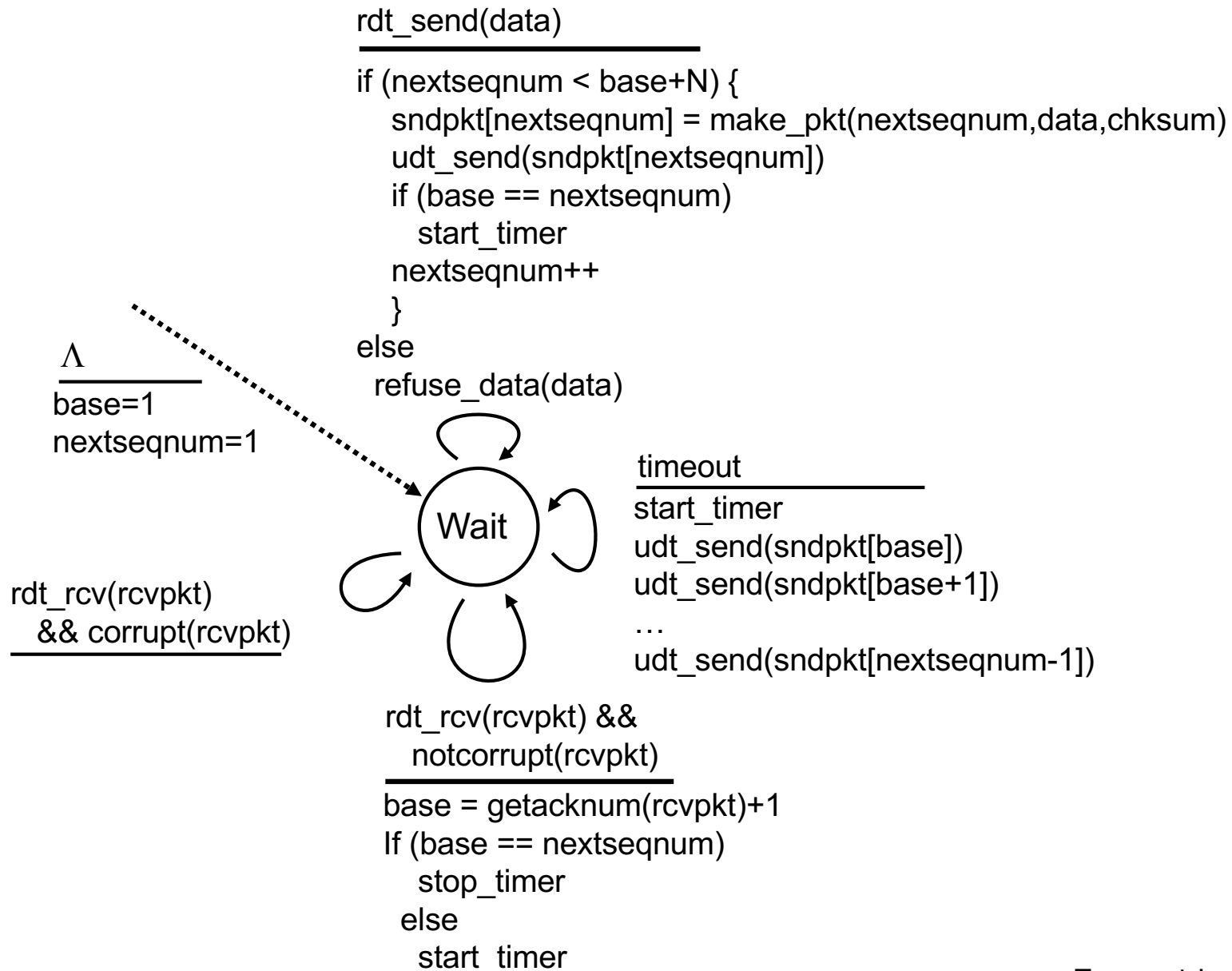
- ❖ k-bit seq # in pkt header
- ❖ “window” of up to N, consecutive unack’ed pkts allowed



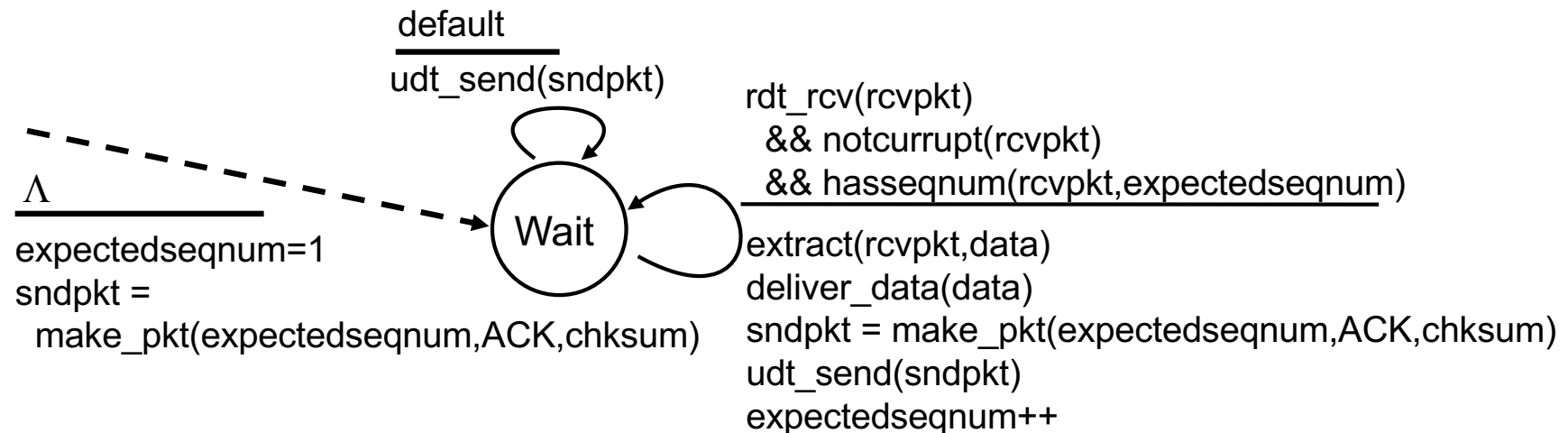
- ❖ ACK(n): ACKs all pkts up to, including seq # n - “cumulative ACK”
 - may receive duplicate ACKs (see receiver)
- ❖ timer for oldest in-flight pkt
- ❖ *timeout(n)*: retransmit packet n and all higher seq # pkts in window

Applets: http://media.pearsoncmg.com/aw/aw_kurose_network_2/applets/go-back-n/go-back-n.html
http://www.ccs-labs.org/teaching/rn/animations/gbn_sr/

GBN: sender extended FSM



GBN: receiver extended FSM



ACK-only: always send ACK for correctly-received pkt with highest *in-order* seq #

- may generate duplicate ACKs
- need only remember **expectedseqnum**

❖ out-of-order pkt:

- discard (don't buffer): *no receiver buffering!*
- re-ACK pkt with highest in-order seq #

GBN in action

sender window (N=4)

0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8
 0 1 2 3 4 5 6 7 8

sender

send pkt0
 send pkt1
 send pkt2
 send pkt3
 (wait)

rcv ack0, send pkt4
 rcv ack1, send pkt5

ignore duplicate ACK



pkt 2 timeout

send pkt2
 send pkt3
 send pkt4
 send pkt5

receiver

receive pkt0, send ack0
 receive pkt1, send ack1

receive pkt3, discard,
 (re)send ack1

receive pkt4, discard,
 (re)send ack1

receive pkt5, discard,
 (re)send ack1

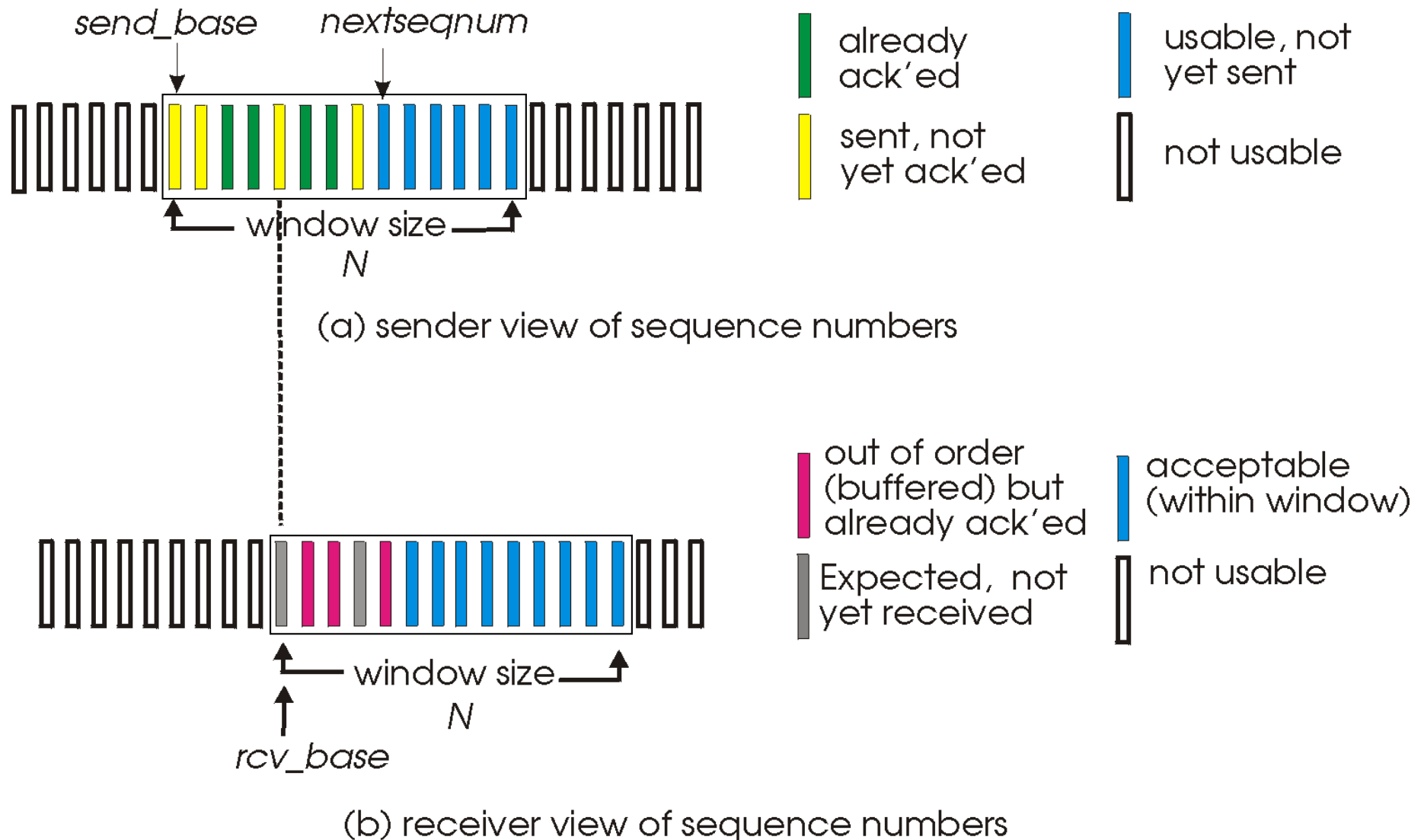
rcv pkt2, deliver, send ack2
 rcv pkt3, deliver, send ack3
 rcv pkt4, deliver, send ack4
 rcv pkt5, deliver, send ack5

Selective repeat

- ❖ receiver *individually* acknowledges all correctly received pkts
 - buffers pkts, as needed, for eventual in-order delivery to upper layer
- ❖ sender only resends pkts for which ACK not received
 - sender timer for each unACKed pkt
- ❖ sender window
 - N consecutive seq #'s
 - limits seq #s of sent, unACKed pkts

Applet: http://media.pearsoncmg.com/aw/aw_kurose_network_3/applets/SelectRepeat/SR.html

Selective repeat: sender, receiver windows



Selective repeat

— sender —

data from above:

- ❖ if next available seq # in window, send pkt

timeout(n):

- ❖ resend pkt n, restart timer

ACK(n) in [sendbase, sendbase+N]:

- ❖ mark pkt n as received
- ❖ if n smallest unACKed pkt, advance window base to next unACKed seq #

— receiver —

pkt n in [rcvbase, rcvbase+N-1]

- ❖ send ACK(n)
- ❖ out-of-order: buffer
- ❖ in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

pkt n in [rcvbase-N, rcvbase-1]

- ❖ ACK(n)

otherwise:

- ❖ ignore

Selective repeat in action

sender window (N=4)

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8

sender

send pkt0

send pkt1

send pkt2

send pkt3

(wait)

rcv ack0, send pkt4

rcv ack1, send pkt5

record ack3 arrived



pkt 2 timeout

send pkt2

record ack4 arrived

record ack5 arrived

receiver

receive pkt0, send ack0

receive pkt1, send ack1

receive pkt3, buffer,
send ack3

receive pkt4, buffer,
send ack4

receive pkt5, buffer,
send ack5

rcv pkt2; deliver pkt2,
pkt3, pkt4, pkt5; send ack2

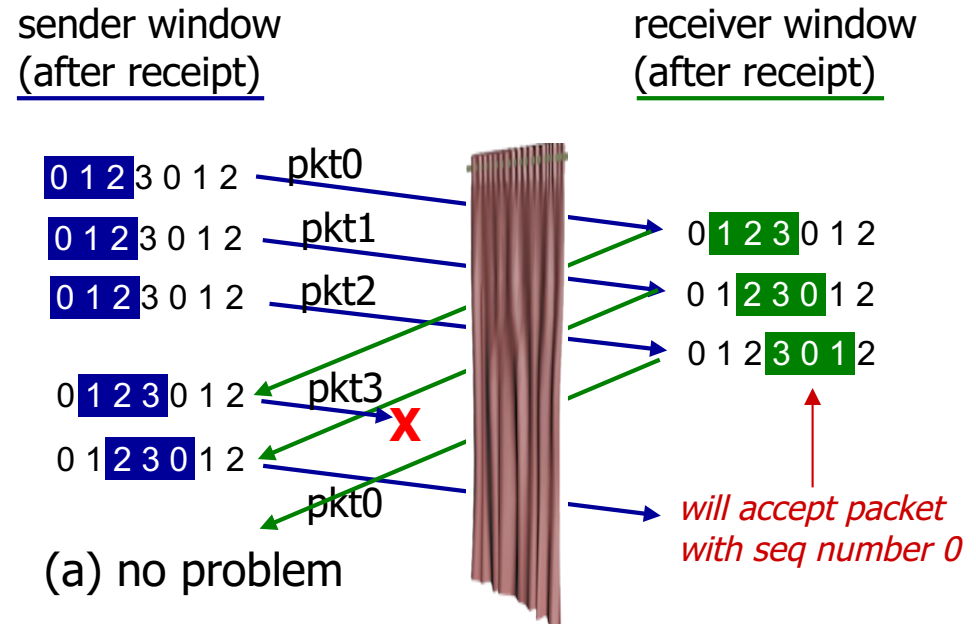
Q: what happens when ack2 arrives?

Selective repeat: dilemma

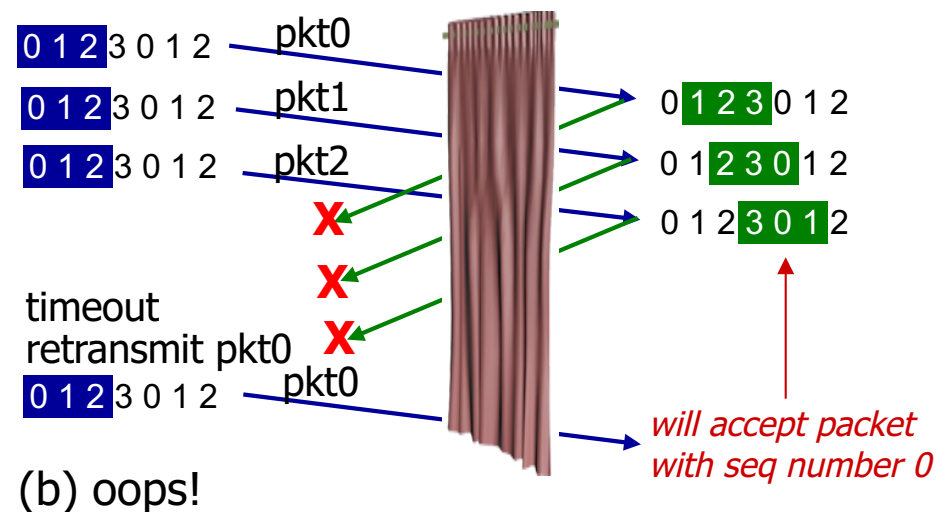
example:

- ❖ seq #'s: 0, 1, 2, 3
- ❖ window size=3
- ❖ receiver sees no difference in two scenarios!
- ❖ duplicate data accepted as new in (b)

Q: what relationship between seq # size and window size to avoid problem in (b)?



*receiver can't see sender side.
receiver behavior identical in both cases!
something's (very) wrong!*



Recap: components of a solution

- ❖ Checksums (for error detection)
- ❖ Timers (for loss detection)
- ❖ Acknowledgments
 - cumulative
 - selective
- ❖ Sequence numbers (duplicates, windows)
- ❖ Sliding Windows (for efficiency)
- ❖ Reliability protocols use the above to decide when and what to retransmit or acknowledge

Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

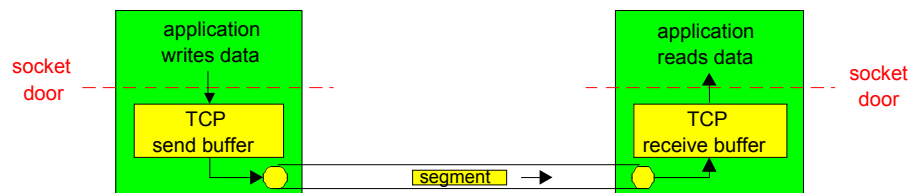
Practical Reliability Questions

- ❖ How do the sender and receiver keep track of outstanding pipelined segments?
- ❖ How many segments should be pipelined?
- ❖ How do we choose sequence numbers?
- ❖ What does connection establishment and teardown look like?
- ❖ How should we choose timeout values?

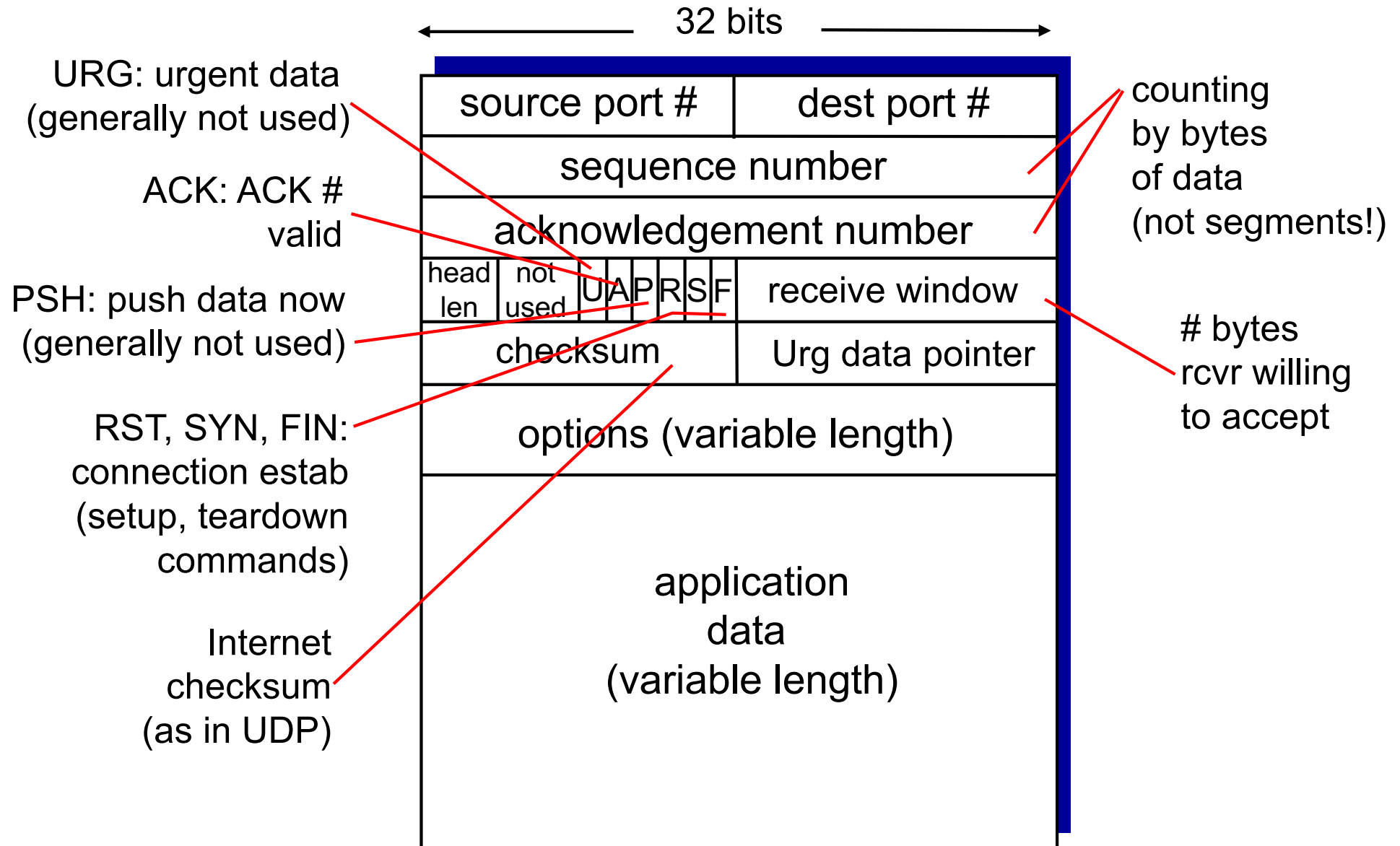
TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581

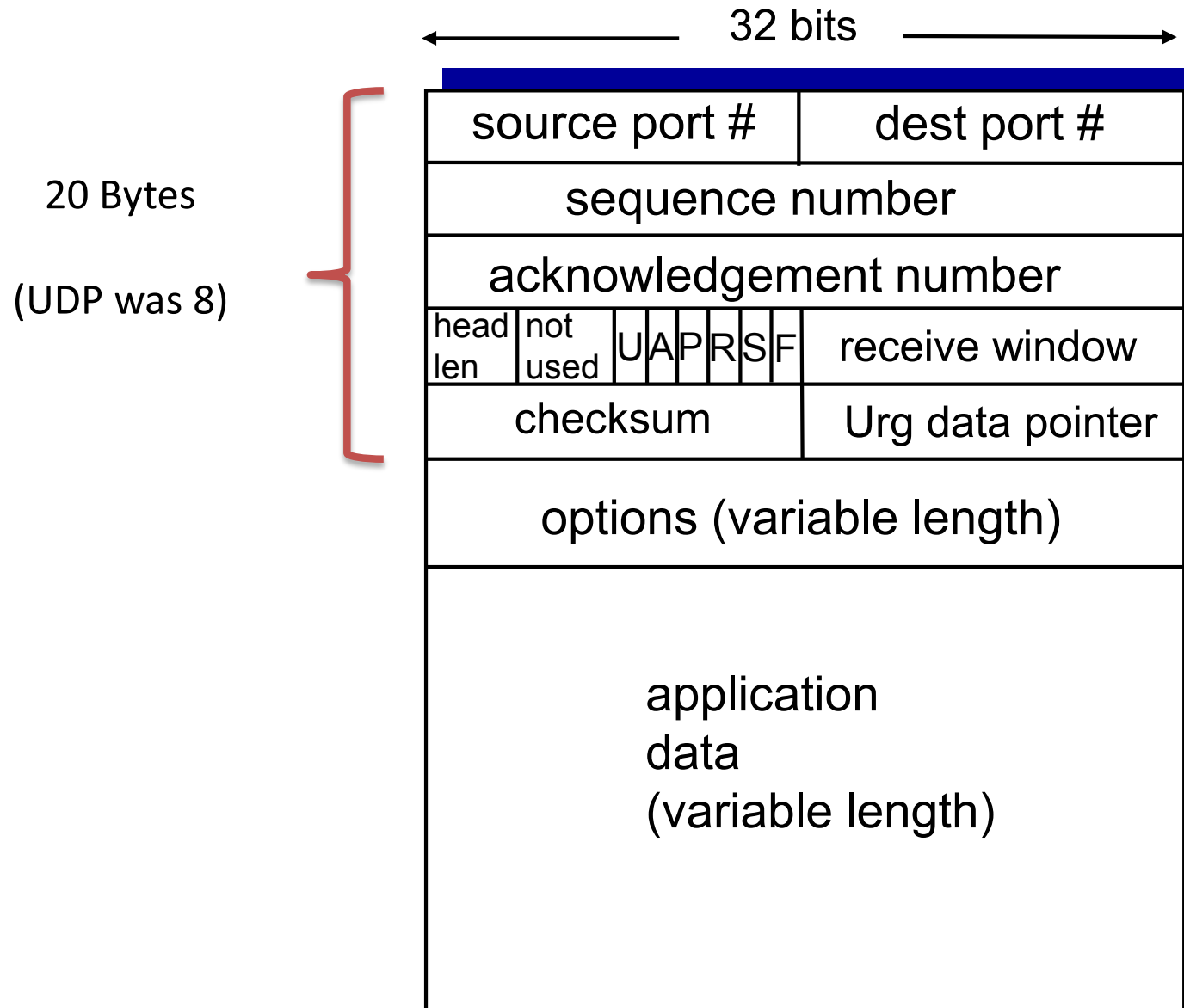
- ❖ **point-to-point:**
 - one sender, one receiver
- ❖ **reliable, in-order *byte stream*:**
 - no “message boundaries”
- ❖ **pipelined:**
 - TCP congestion and flow control set window size
- ❖ **send and receive buffers**
- ❖ **full duplex data:**
 - bi-directional data flow in same connection
 - MSS: maximum segment size
- ❖ **connection-oriented:**
 - handshaking (exchange of control msgs) initializes sender, receiver state before data exchange
- ❖ **flow controlled:**
 - sender will not overwhelm receiver



TCP segment structure



TCP segment structure



Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

Recall: Components of a solution for reliable transport

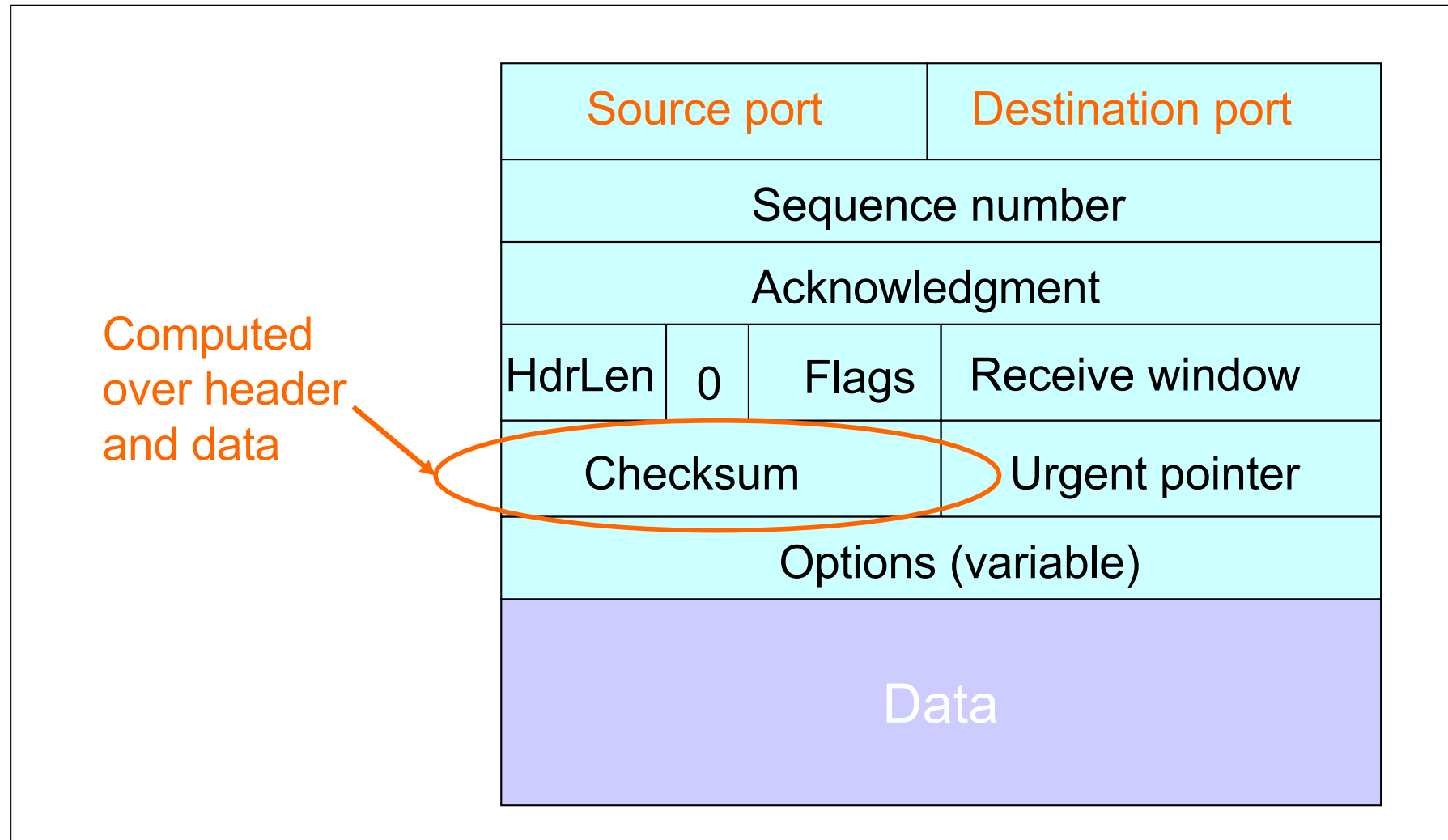
- ❖ Checksums (for error detection)
- ❖ Timers (for loss detection)
- ❖ Acknowledgments
 - cumulative
 - selective
- ❖ Sequence numbers (duplicates, windows)
- ❖ Sliding Windows (for efficiency)
 - Go-Back-N (GBN)
 - Selective Repeat (SR)

What does TCP do?

Many of our previous ideas, but some key differences

- ❖ Checksum

TCP Header



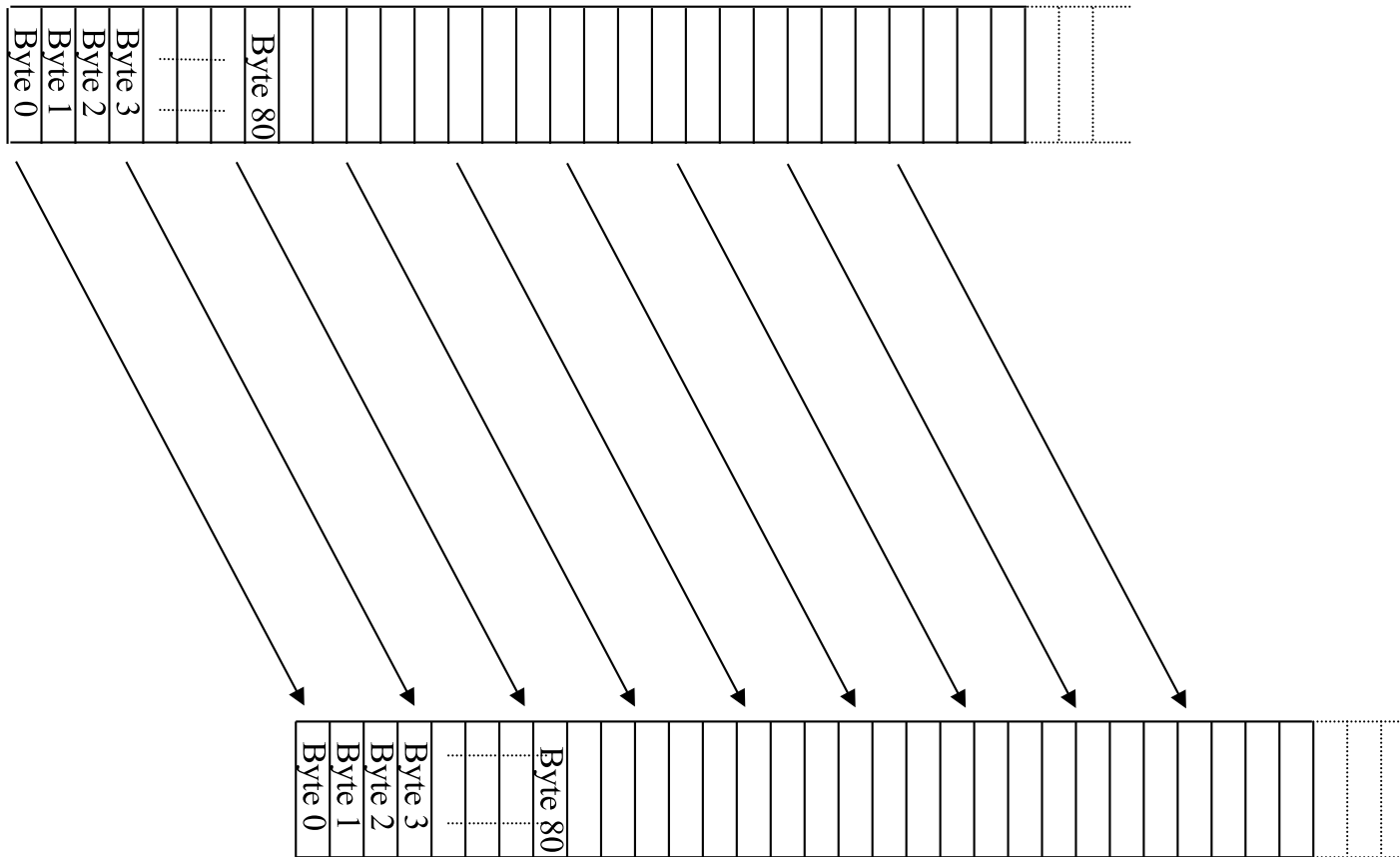
What does TCP do?

Many of our previous ideas, but some key differences

- ❖ Checksum
- ❖ **Sequence numbers are byte offsets**

TCP “Stream of Bytes” Service ..

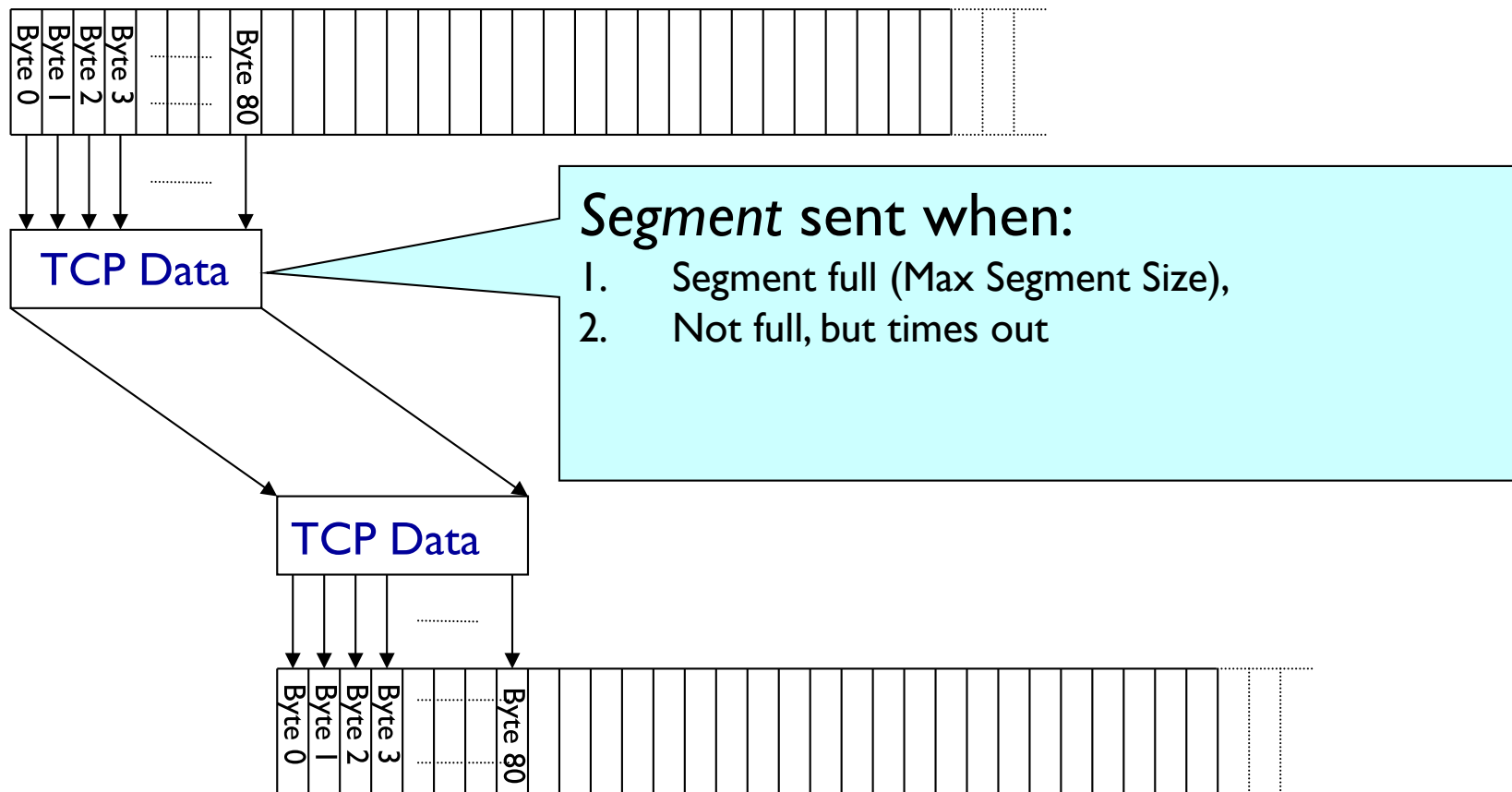
Application @ Host A



Application @ Host B

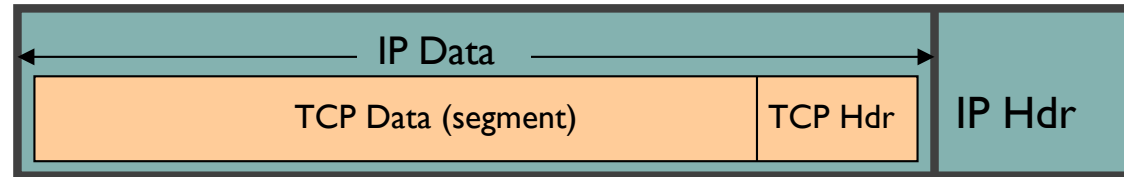
.. Provided Using TCP “Segments”

Host A



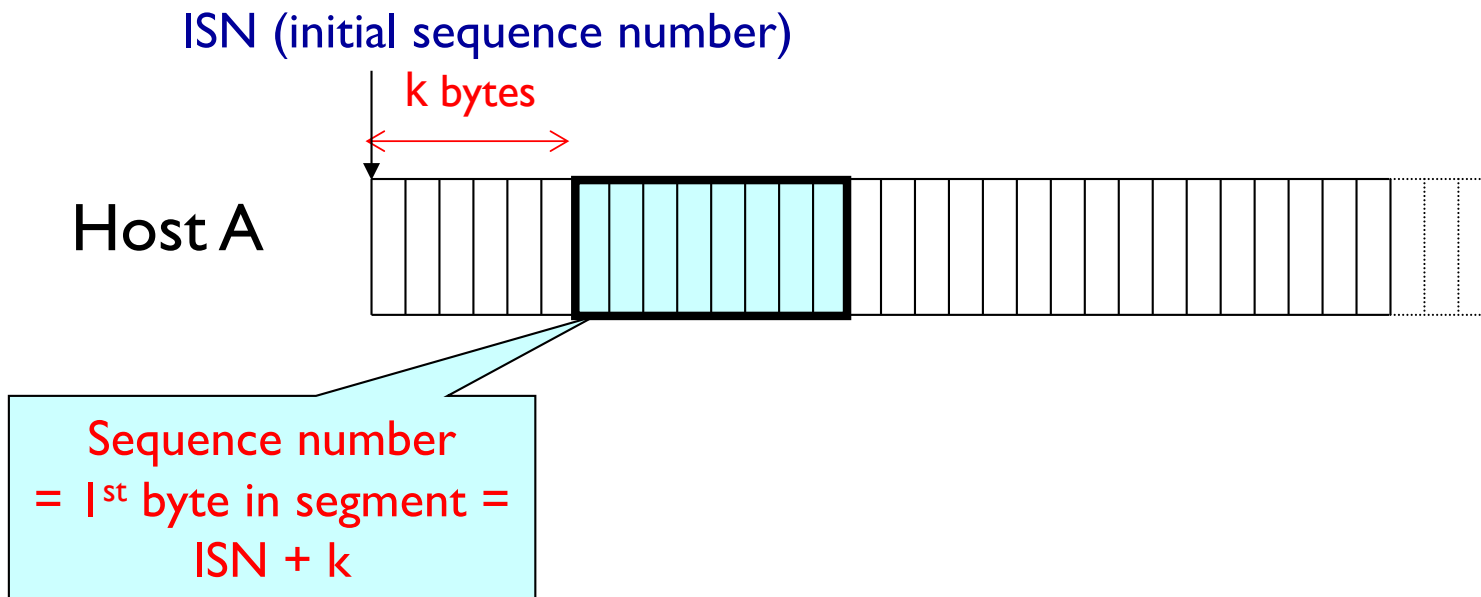
Host B

TCP Segment Size



- ❖ IP packet
 - No bigger than Maximum Transmission Unit (MTU)
 - E.g., up to 1500 bytes with Ethernet
- ❖ TCP packet
 - IP packet with a TCP header and data inside
 - TCP header ≥ 20 bytes long
- ❖ TCP segment
 - No more than Maximum Segment Size (MSS) bytes
 - E.g., up to 1460 consecutive bytes from the stream
 - $MSS = MTU - (IP \text{ header}) - (TCP \text{ header})$

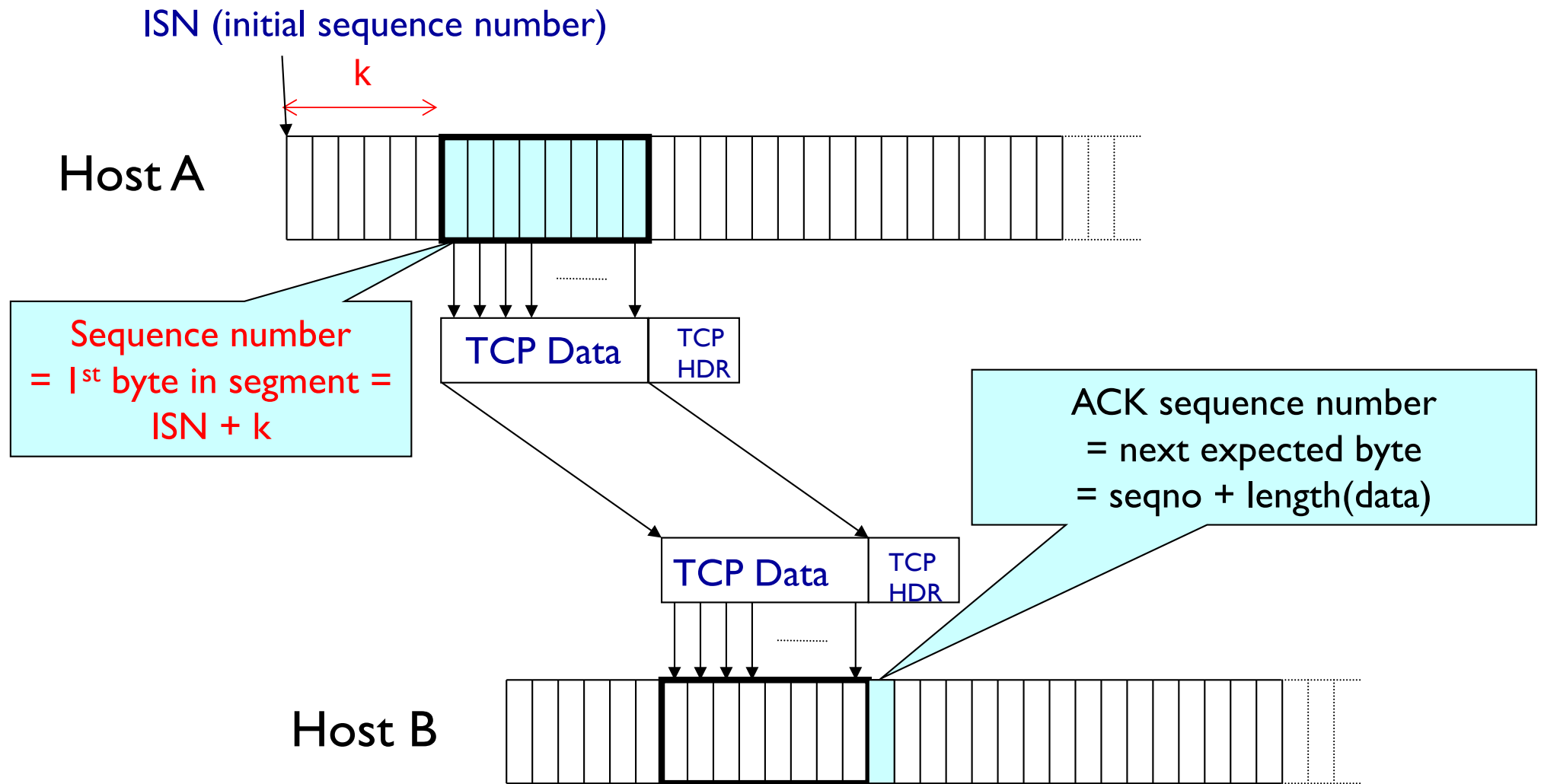
Sequence Numbers



Sequence numbers:

- byte stream “number” of first byte in segment's data

Sequence & Ack Numbers



What does TCP do?

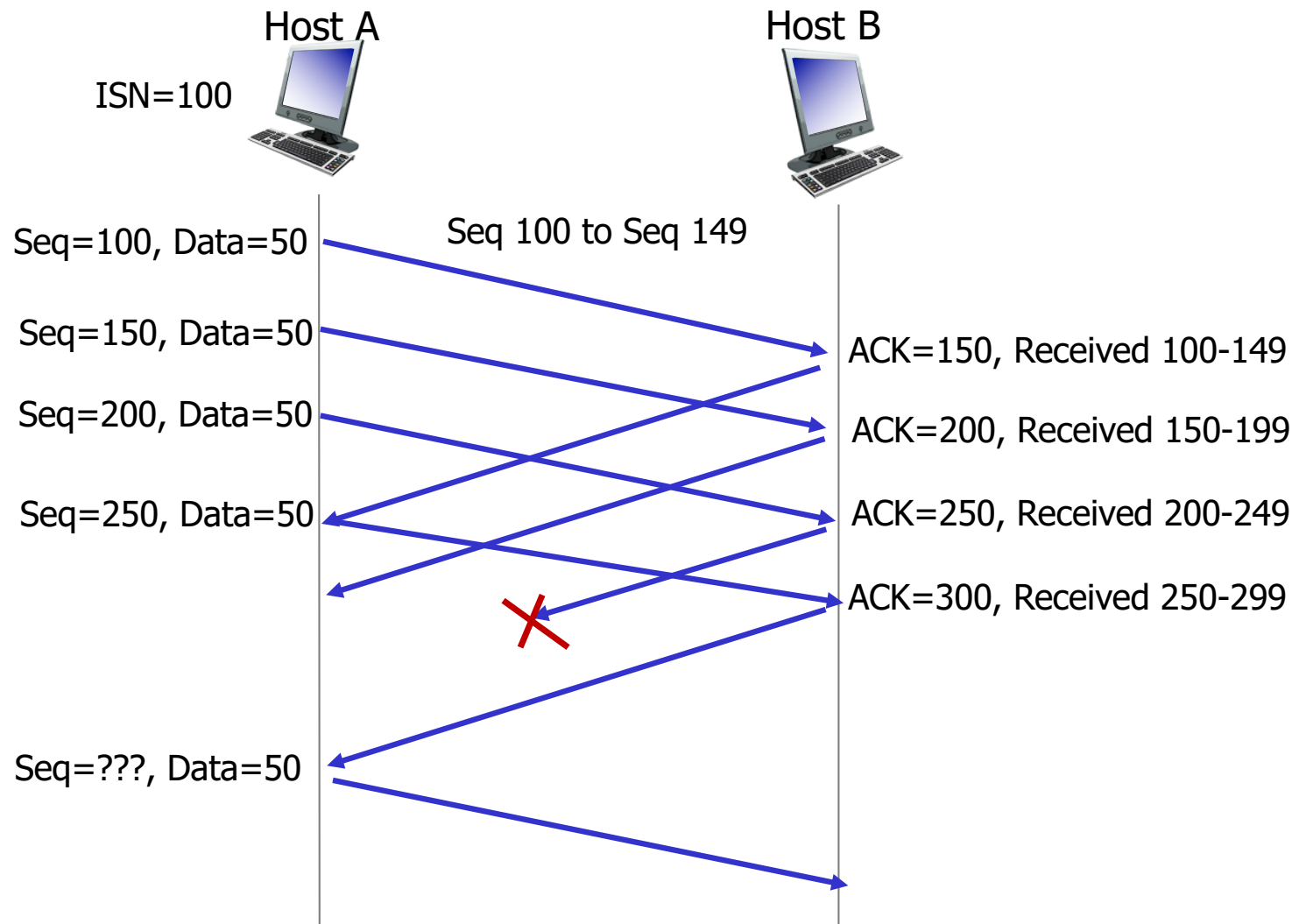
Most of our previous tricks, but a few differences

- ❖ Checksum
- ❖ Sequence numbers are byte offsets
- ❖ Receiver sends cumulative acknowledgements (like GBN)

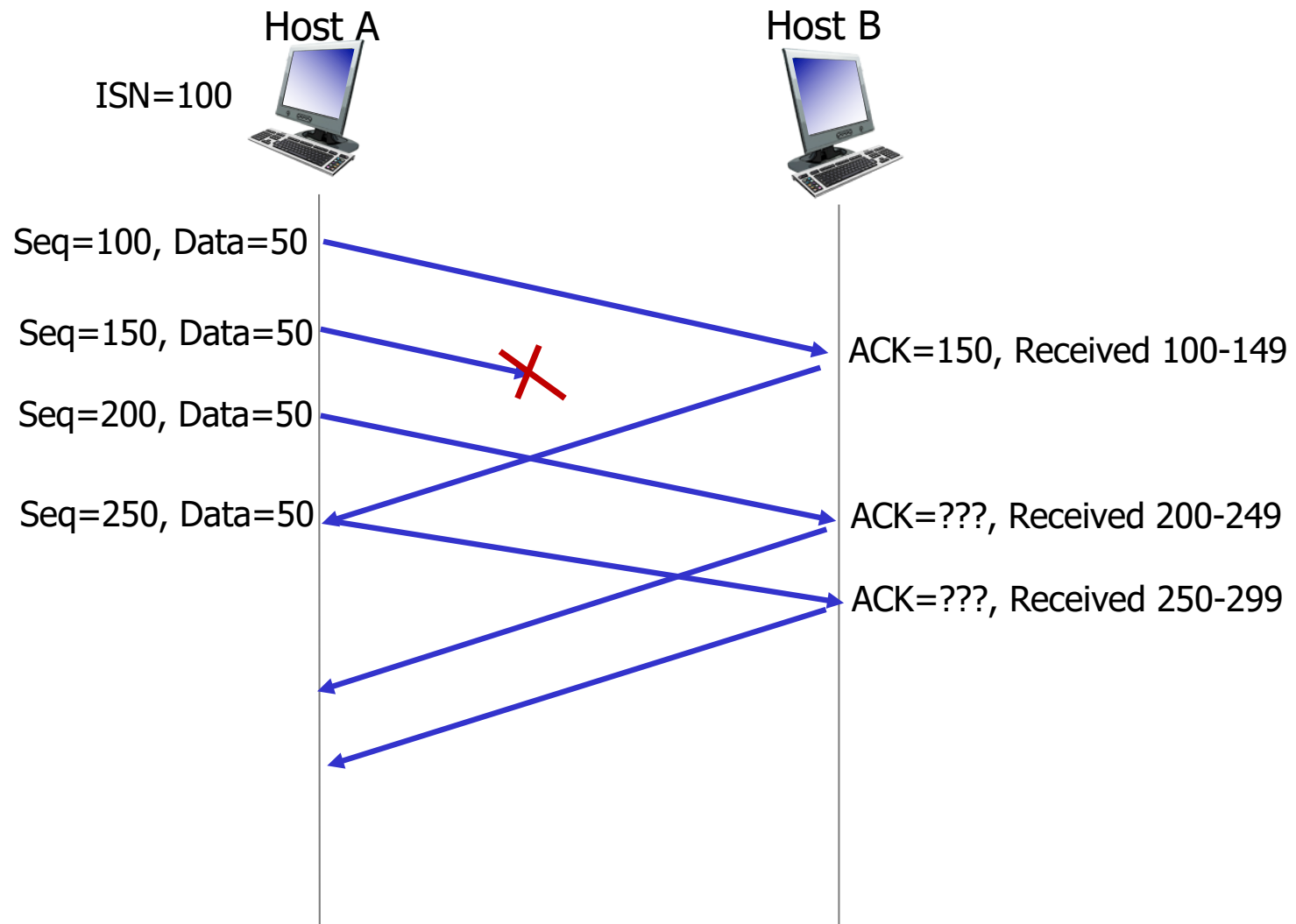
ACKing and Sequence Numbers

- ❖ Sender sends packet
 - Data starts with sequence number X
 - Packet contains B bytes $[X, X+1, X+2, \dots, X+B-1]$
- ❖ Upon receipt of packet, receiver sends an ACK
 - If all data prior to X already received:
 - ACK acknowledges $X+B$ (because that is next expected byte)
 - If highest in-order byte received is Y s.t. $(Y+1) < X$
 - ACK acknowledges $Y+1$
 - Even if this has been ACKed before

TCP seq. numbers, ACKs



TCP seq. numbers, ACKs



Normal Pattern

- ❖ Sender: $\text{seqno} = X$, $\text{length} = B$
- ❖ Receiver: $\text{ACK} = X + B$
- ❖ Sender: $\text{seqno} = X + B$, $\text{length} = B$
- ❖ Receiver: $\text{ACK} = X + 2B$
- ❖ Sender: $\text{seqno} = X + 2B$, $\text{length} = B$

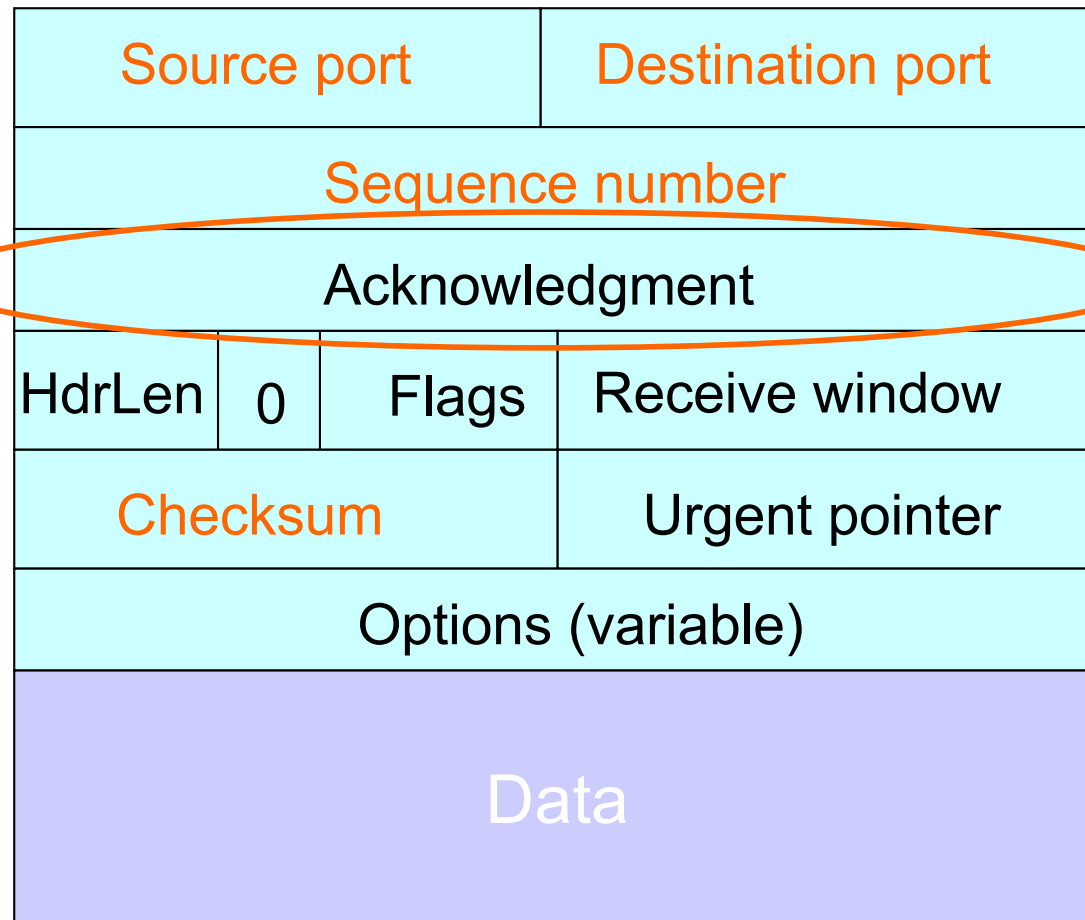
- ❖ Seqno of next packet is same as last ACK field

Packet Loss

- ❖ Sender: $\text{seqno} = X$, $\text{length} = B$
- ❖ Receiver: $\text{ACK} = X + B$
- ❖ Sender: ~~$\text{seqno} = X + B$, $\text{length} = B$~~ LOST
- ❖ Sender: $\text{seqno} = X + 2B$, $\text{length} = B$
- ❖ Receiver: $\text{ACK} = X + B$

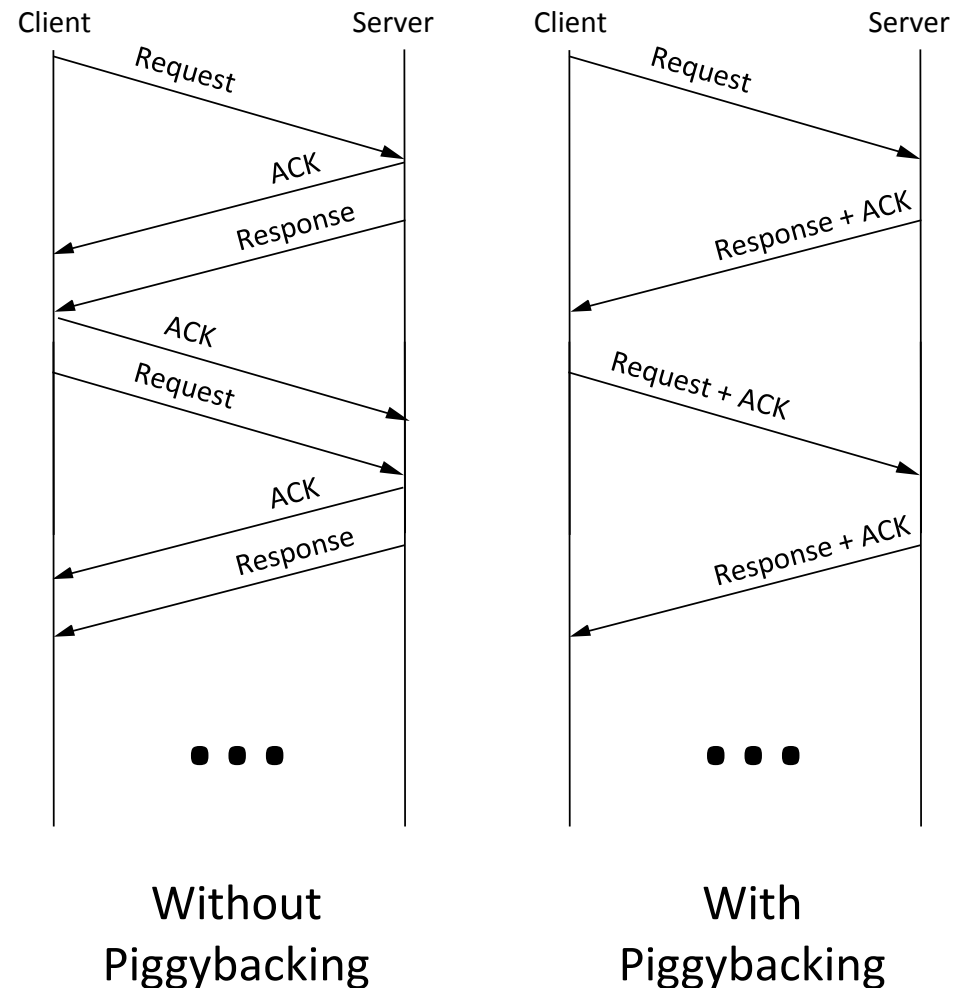
TCP Header

Acknowledgment gives seqno just beyond highest seqno received **in order** (*“What Byte is Next”*)

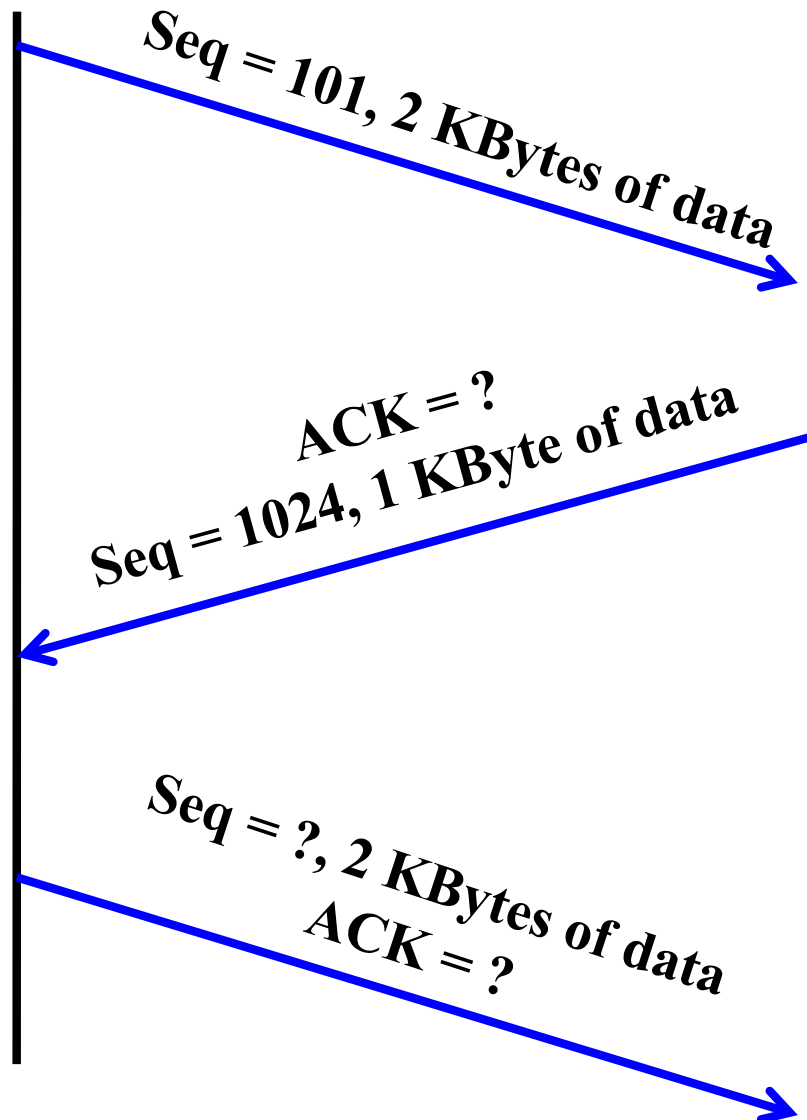


Piggybacking

- ❖ So far, we've assumed distinct “sender” and “receiver” roles
- ❖ In reality, usually both sides of a connection send some data



Quiz



What does TCP do?

Most of our previous tricks, but a few differences

- ❖ Checksum
- ❖ Sequence numbers are byte offsets
- ❖ Receiver sends cumulative acknowledgements (like GBN)
- ❖ Receivers **can** buffer out-of-sequence packets (like SR)

Loss with cumulative ACKs

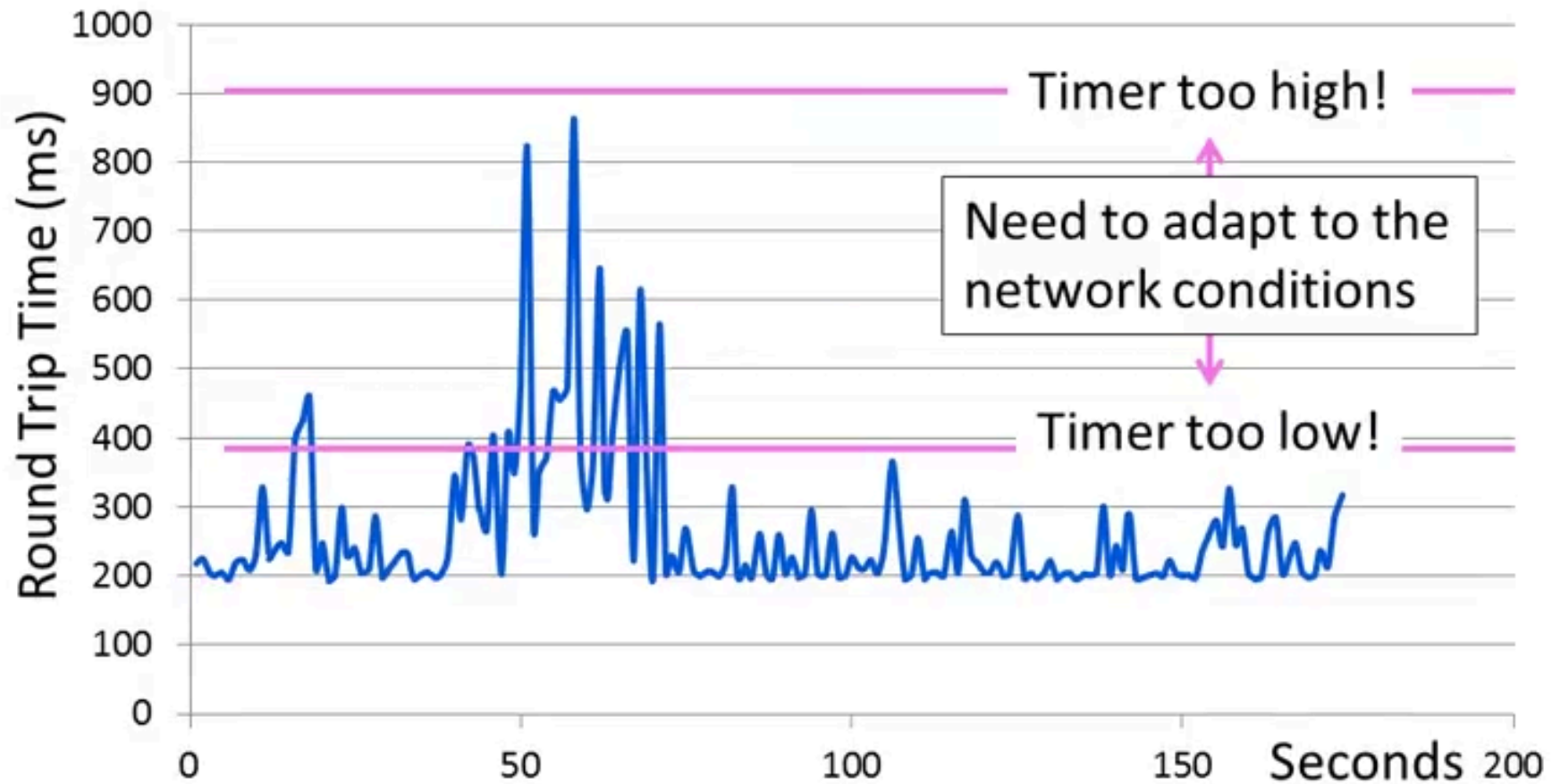
- ❖ Sender sends packets with 100Bytes and sequence numbers:
 - 100, 200, 300, 400, 500, 600, 700, 800, 900, ...
- ❖ Assume the fifth packet (seq. no. 500) is lost, but no others
- ❖ Stream of ACKs will be:
 - 200, 300, 400, 500, 500, 500, 500,...

What does TCP do?

Most of our previous tricks, but a few differences

- ❖ Checksum
- ❖ Sequence numbers are byte offsets
- ❖ Receiver sends cumulative acknowledgements (like GBN)
- ❖ Receivers do not drop out-of-sequence packets (like SR)
- ❖ Sender maintains a single retransmission timer (like GBN) and retransmits on timeout

TCP round trip time, timeout



TCP round trip time, timeout

Q: how to set TCP timeout value?

- ❖ longer than RTT
 - but RTT varies
- ❖ *too short*: premature timeout, unnecessary retransmissions
- ❖ *too long*: slow reaction to segment loss and connection has lower throughput

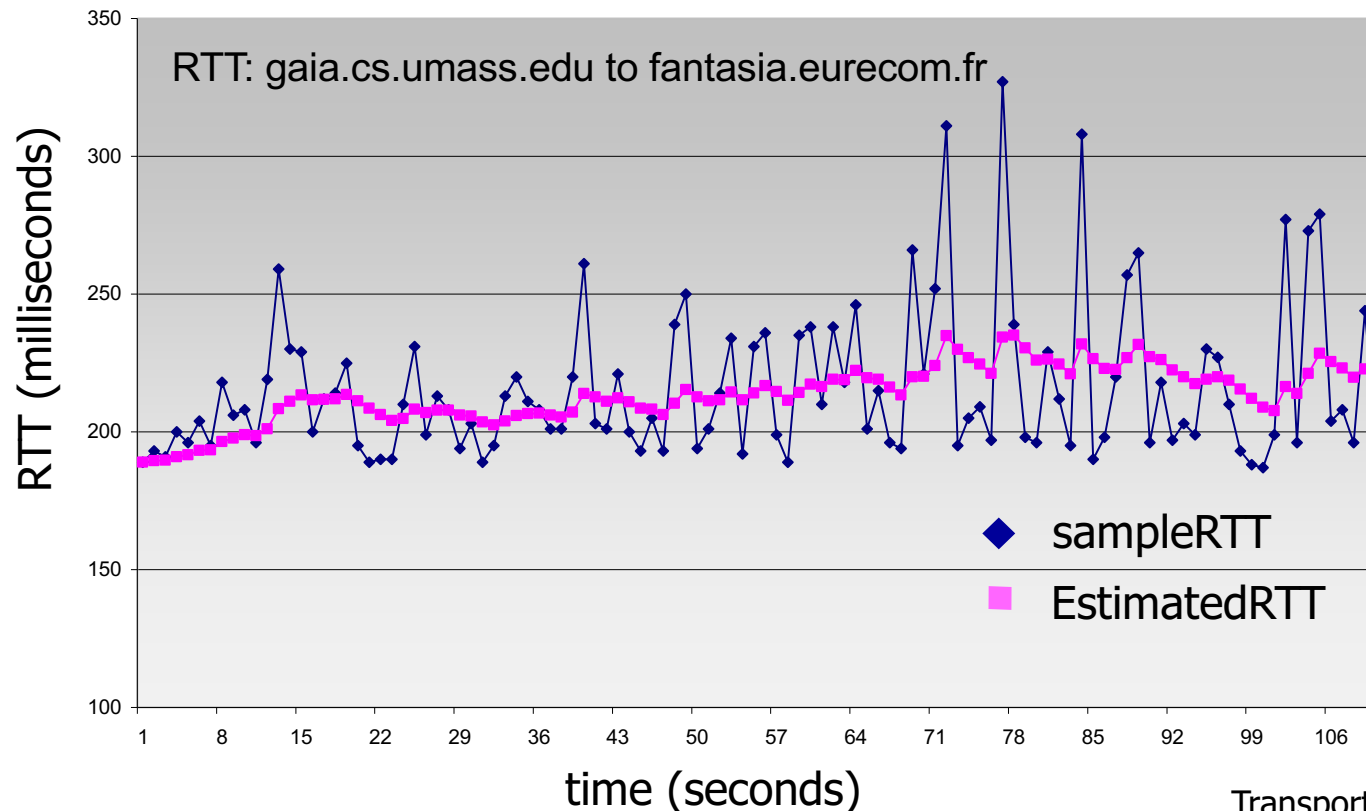
Q: how to estimate RTT?

- ❖ **SampleRTT**: measured time from segment transmission until ACK receipt
 - ignore retransmissions
- ❖ **SampleRTT** will vary, want estimated RTT “smoother”
 - average several *recent* measurements, not just current **SampleRTT**

TCP round trip time, timeout

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

- ❖ exponential weighted moving average
- ❖ influence of past sample decreases exponentially fast
- ❖ typical value: $\alpha = 0.125$



TCP round trip time, timeout

- ❖ **timeout interval:** `EstimatedRTT` plus “safety margin”
 - large variation in `EstimatedRTT` -> larger safety margin
- ❖ estimate `SampleRTT` deviation from `EstimatedRTT`:

$$\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$

(typically, $\beta = 0.25$)

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$



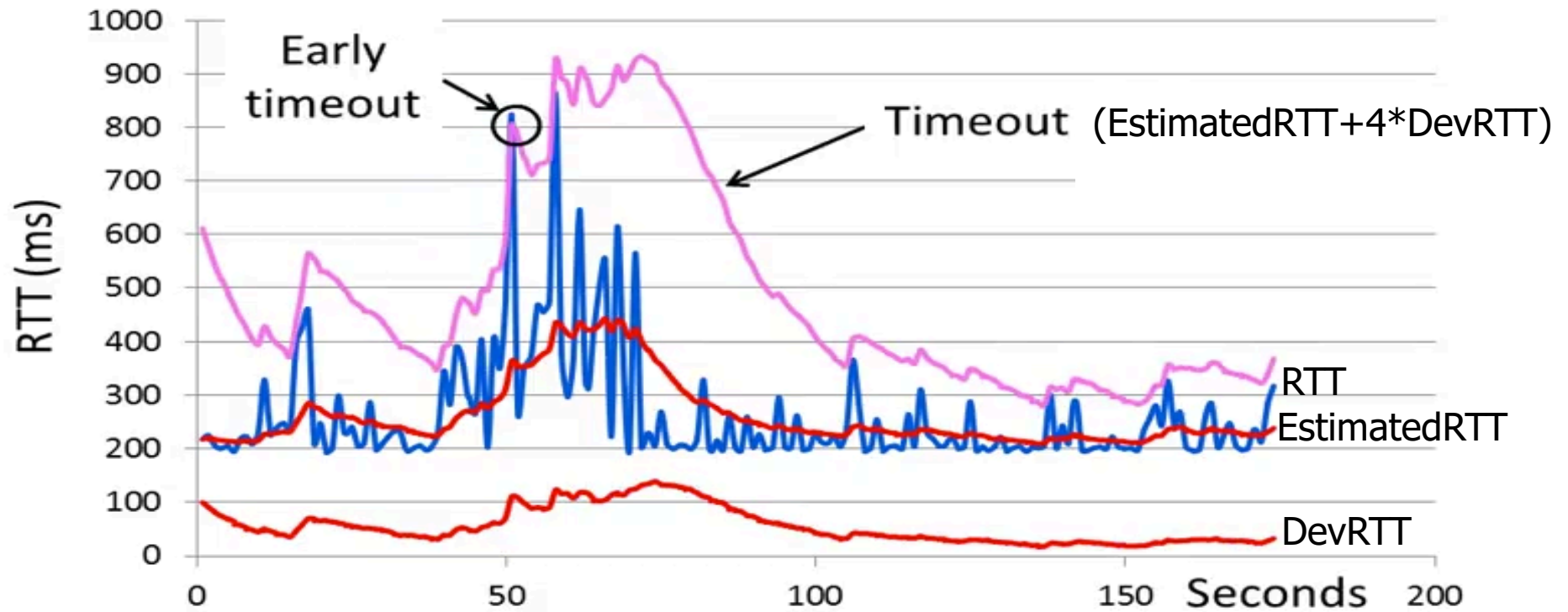
↑
estimated RTT

↑
“safety margin”

Practice Problem:

http://wps.pearsoned.com/ecs_kurose_compnetw_6/216/55463/14198700.cw/index.html

TCP round trip time, timeout



$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$

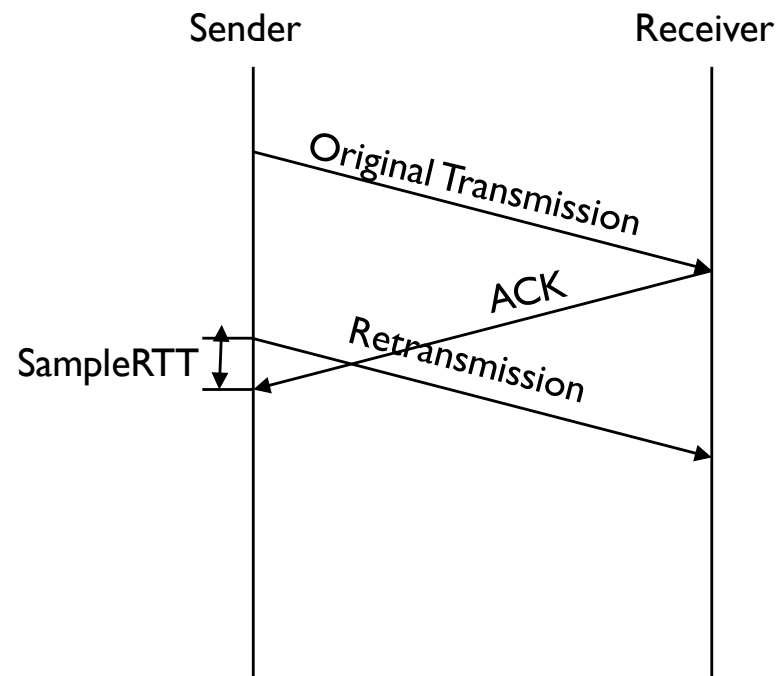
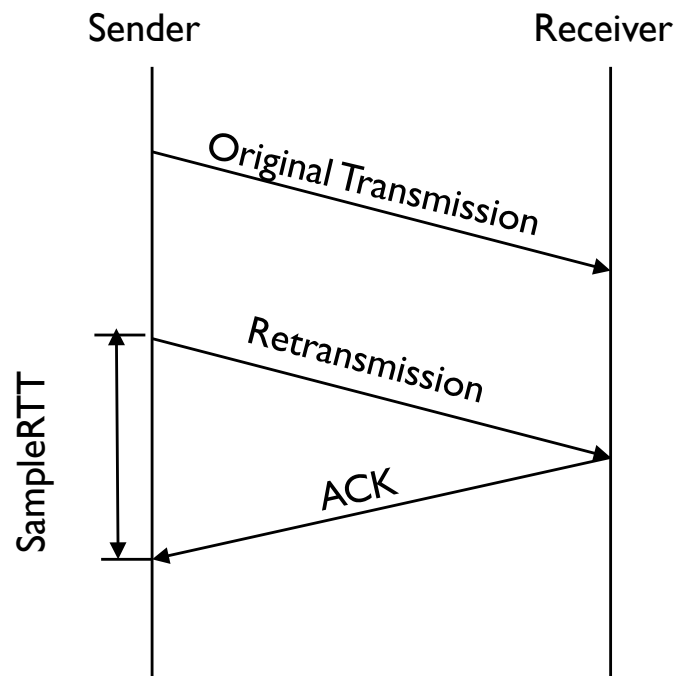


↑
estimated RTT

↑
“safety margin”

Why exclude retransmissions in RTT computation?

- ❖ How do we differentiate between the real ACK, and ACK of the retransmitted packet?



TCP sender events:

PUTTING IT
TOGETHER

data rcvd from app:

- ❖ create segment with seq #
- ❖ seq # is byte-stream number of first data byte in segment
- ❖ start timer if not already running
 - think of timer as for oldest unacked segment
 - expiration interval: `TimeoutInterval`

timeout:

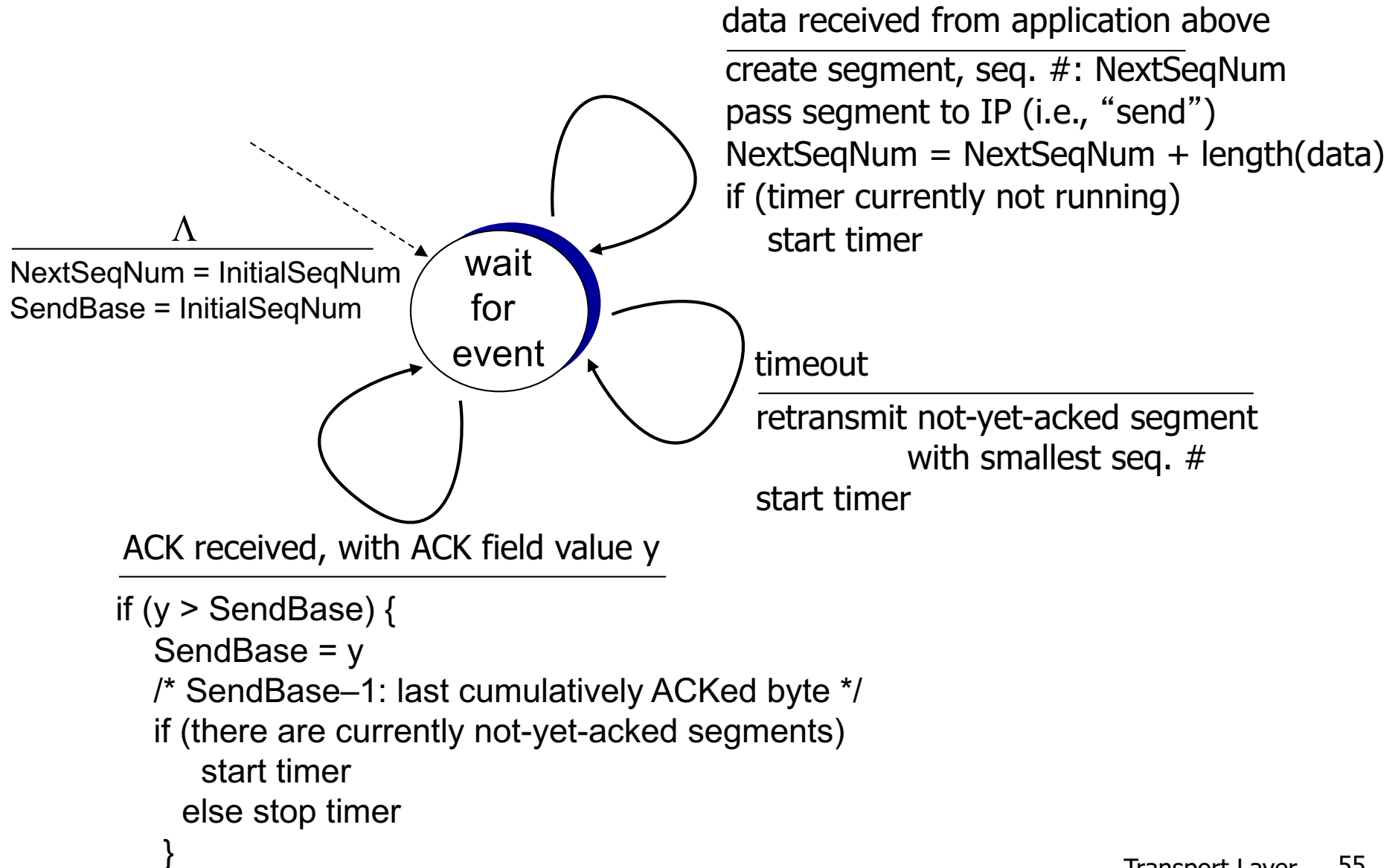
- ❖ retransmit segment that caused timeout
- ❖ restart timer

ack rcvd:

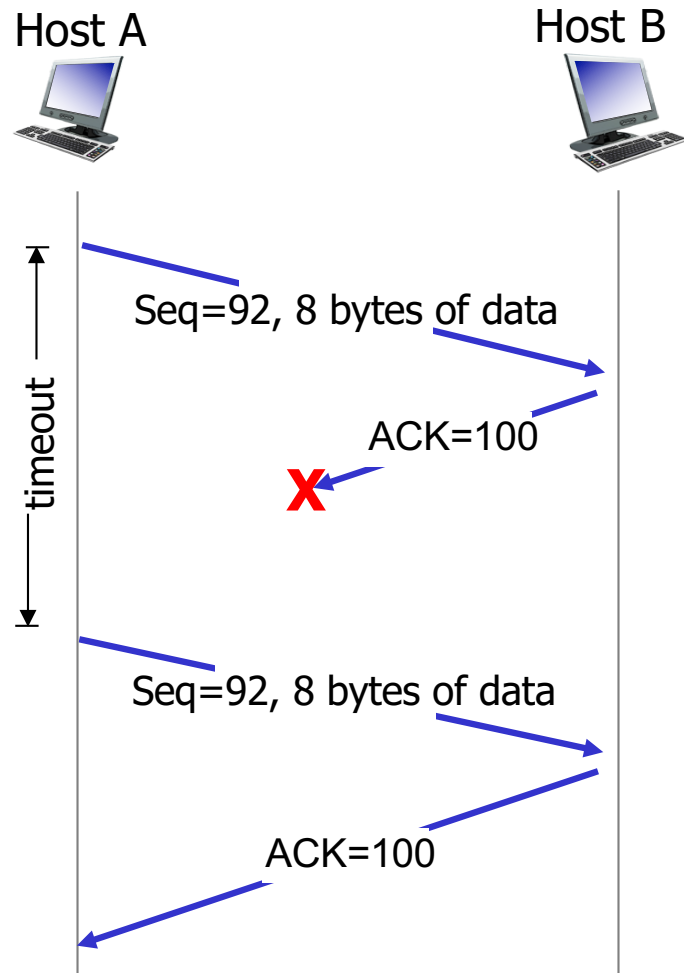
- ❖ if ack acknowledges previously unacked segments
 - update what is known to be ACKed
 - start timer if there are still unacked segments

TCP sender (simplified)

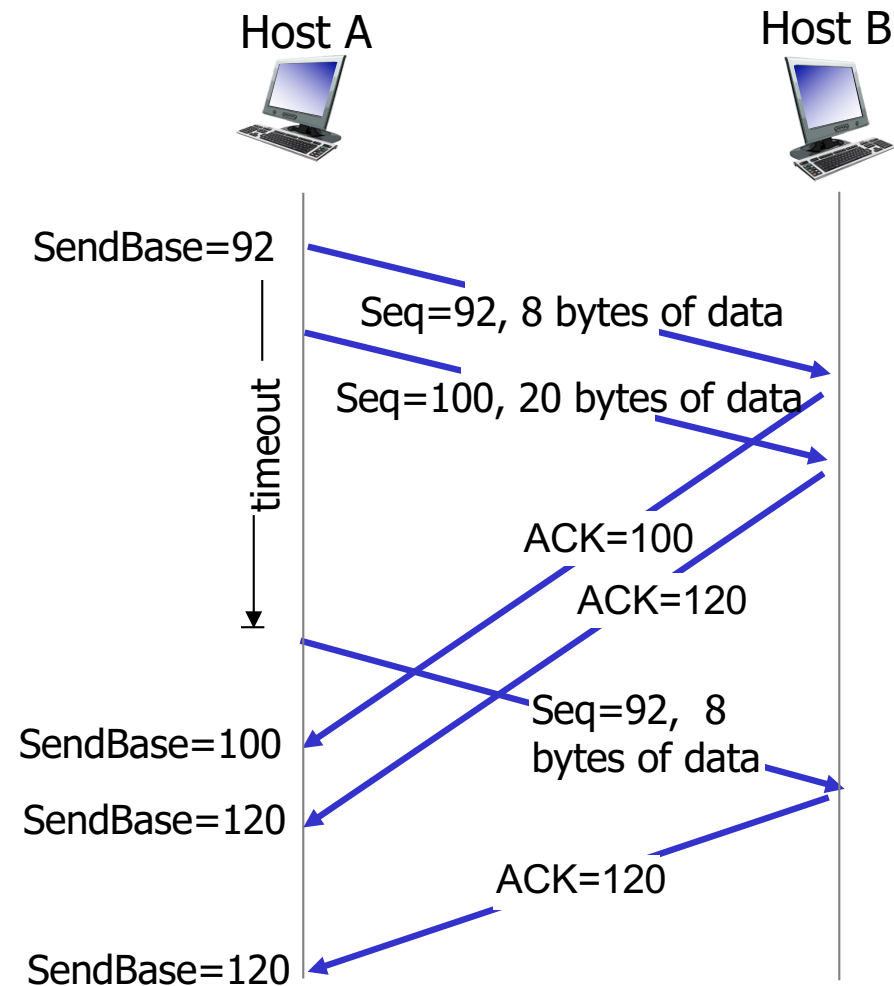
PUTTING IT TOGETHER



TCP: retransmission scenarios

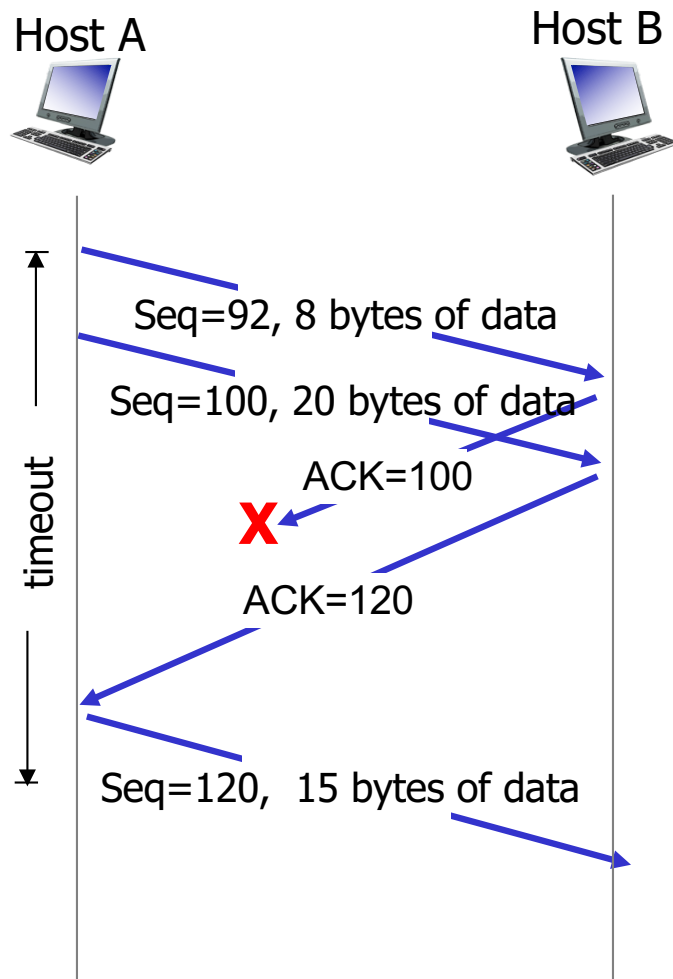


lost ACK scenario

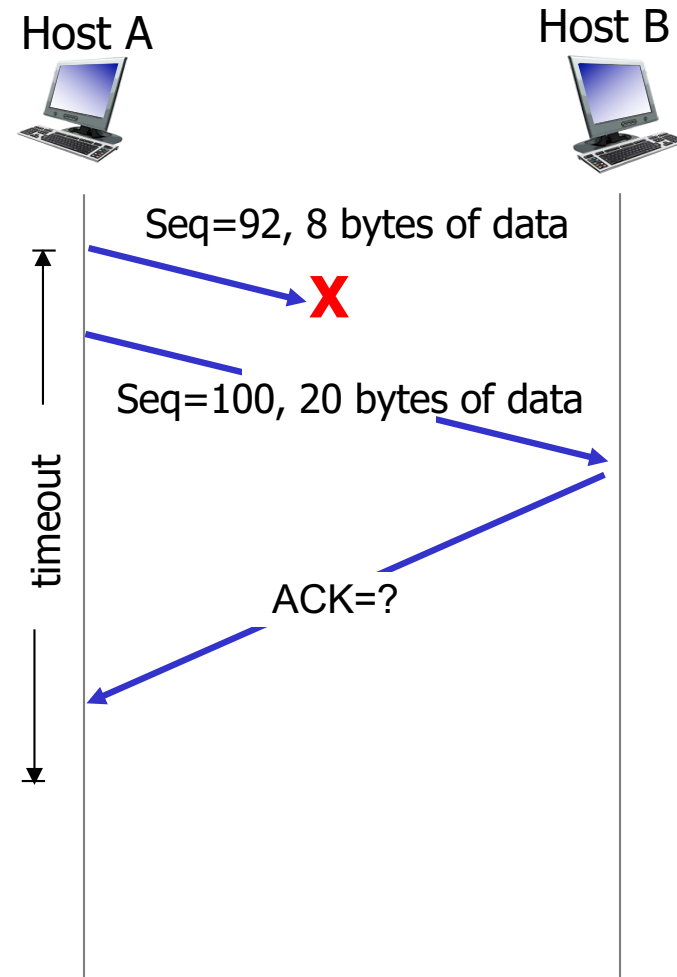


premature timeout

TCP: retransmission scenarios



cumulative ACK



cumulative ACK

Quiz



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.

- ❖ Q1: How much data is in the first segment ?
- ❖ Q2: Suppose that the first segment is lost but the second segment arrives at B. In the acknowledgement that Host B sends to Host A, what will be the acknowledgement number?

TCP ACK generation [RFC 1122, RFC 2581]

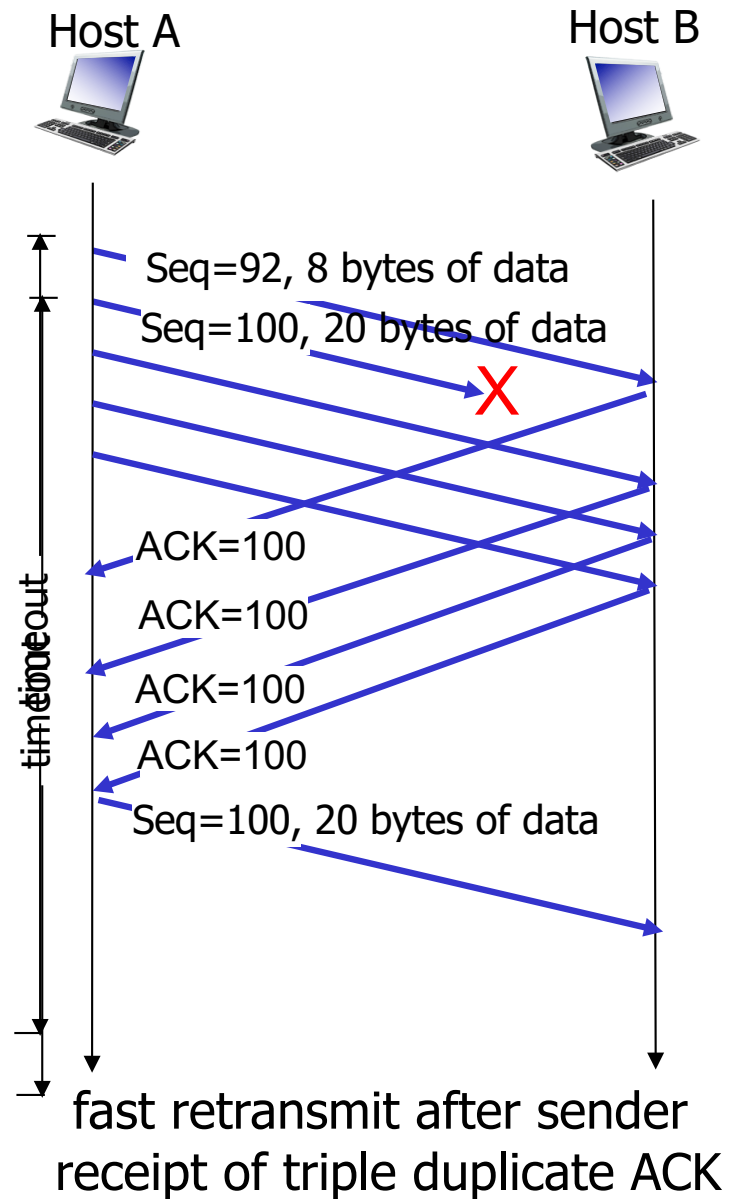
<i>event at receiver</i>	<i>TCP receiver action</i>
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send single cumulative ACK, ACKing both in-order segments
arrival of out-of-order segment higher-than-expect seq. # . Gap detected	immediately send <i>duplicate ACK</i> , indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	immediate send ACK, provided that segment starts at lower end of gap

What does TCP do?

Most of our previous tricks, but a few differences

- ❖ Checksum
- ❖ Sequence numbers are byte offsets
- ❖ Receiver sends cumulative acknowledgements (like GBN)
- ❖ Receivers may not drop out-of-sequence packets (like SR)
- ❖ Sender maintains a single retransmission timer (like GBN) and retransmits on timeout
- ❖ Introduces **fast retransmit**: optimisation that uses duplicate ACKs to trigger early retransmission

TCP fast retransmit



TCP fast retransmit

- ❖ time-out period often relatively long:
 - long delay before resending lost packet
- ❖ “Duplicate ACKs” are a sign of an isolated loss
 - The lack of ACK progress means that packet hasn't been delivered
 - Stream of ACKs means some packets are being delivered
 - Could trigger resend on receiving “k” duplicate ACKs (TCP uses $k = 3$)

TCP fast retransmit

if sender receives 3 duplicate ACKs for same data

(“triple duplicate ACKs”),
resend unacked
segment with smallest
seq #

- likely that unacked segment is lost, so don't wait for timeout

What does TCP do?

Most of our previous ideas, but some key differences

- ❖ Checksum
- ❖ Sequence numbers are byte offsets
- ❖ Receiver sends cumulative acknowledgements (like GBN)
- ❖ Receivers do not drop out-of-sequence packets (like SR)
- ❖ Sender maintains a single retransmission timer (like GBN) and retransmits on timeout
- ❖ Introduces fast retransmit: optimization that uses duplicate ACKs to trigger early retransmission

Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

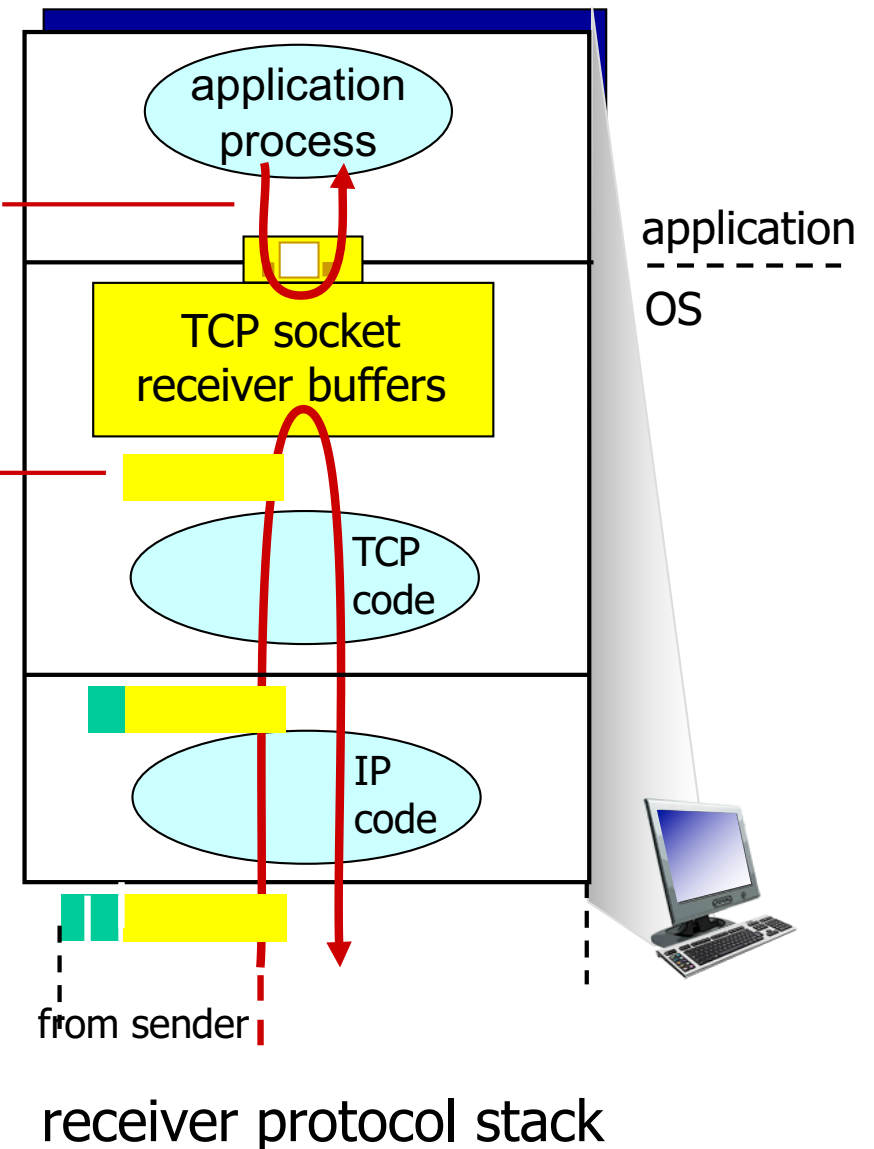
3.7 TCP congestion control

TCP flow control

application may
remove data from
TCP socket buffers

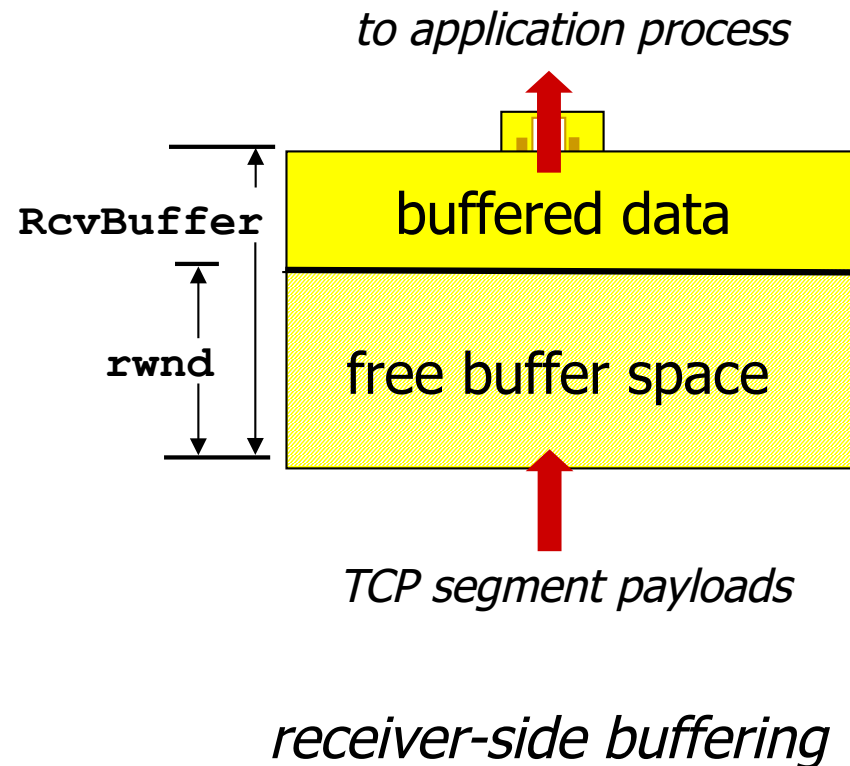
... slower than TCP
receiver is delivering
(sender is sending)

flow control
receiver controls sender, so
sender won't overflow
receiver's buffer by transmitting
too much, too fast



TCP flow control

- ❖ receiver “advertises” free buffer space by including **rwnd** value in TCP header of receiver-to-sender segments
 - **RcvBuffer** size set via socket options (typical default is 4096 bytes)
 - many operating systems autoadjust **RcvBuffer**
- ❖ sender limits amount of unacked (“in-flight”) data to receiver’s **rwnd** value
- ❖ guarantees receive buffer will not overflow



TCP Header

Source port		Destination port	
Sequence number			
Acknowledgment			
HdrLen	0	Flags	Receive window
Checksum		Urgent pointer	
Options (variable)			
Data			

Transport Layer Outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control