

50.012 Networks

Lecture 9: TCP Part I

2021 Term 6

Assoc. Prof. CHEN Binbin

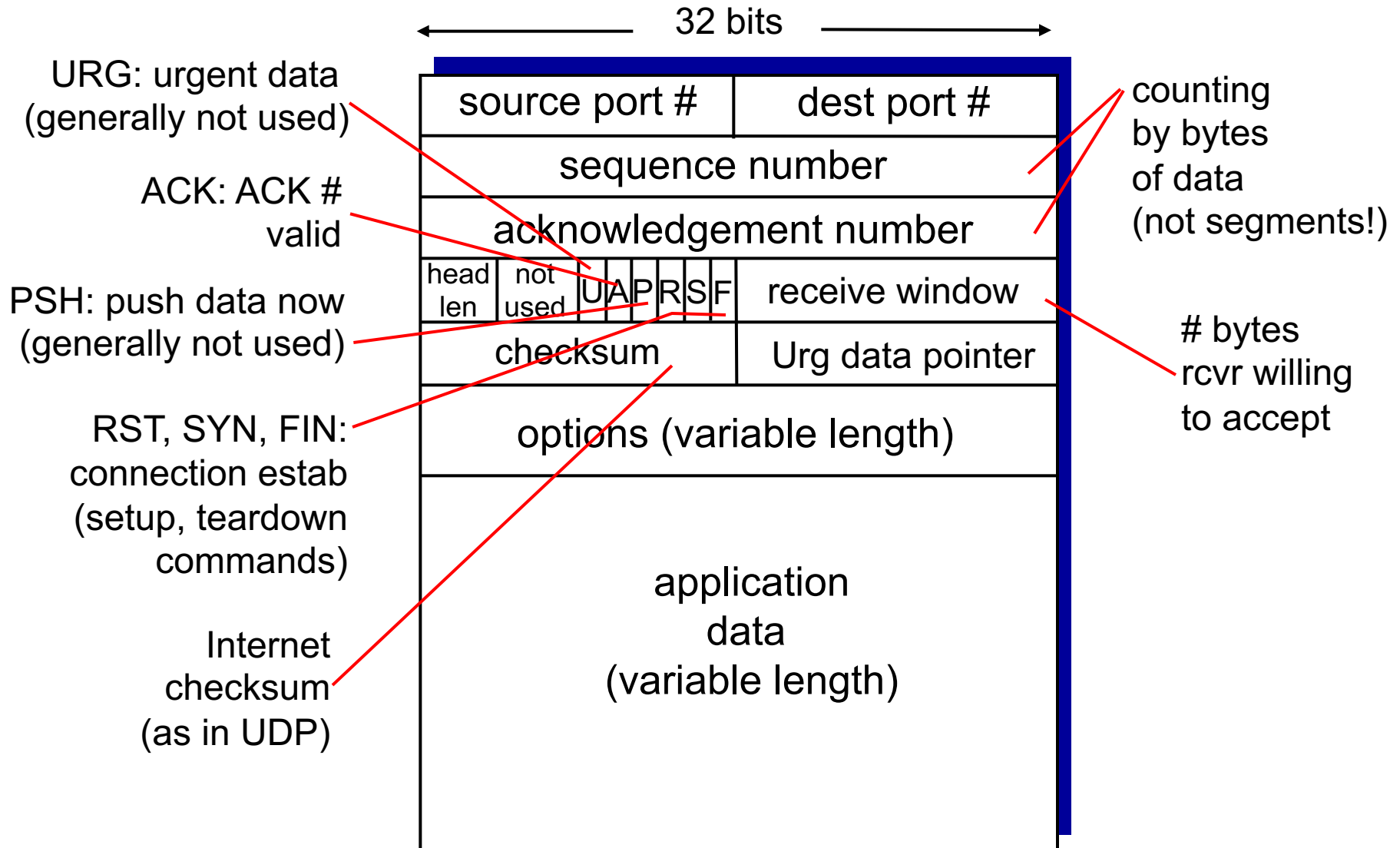


TCP: Overview

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
- **full duplex data:**
 - bi-directional data flow in same connection
 - MSS: maximum segment size
- **connection-oriented:**
 - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- **flow controlled:**
 - sender will not overwhelm receiver

TCP segment structure



TCP seq. numbers, ACKs

sequence numbers:

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

acknowledgements:

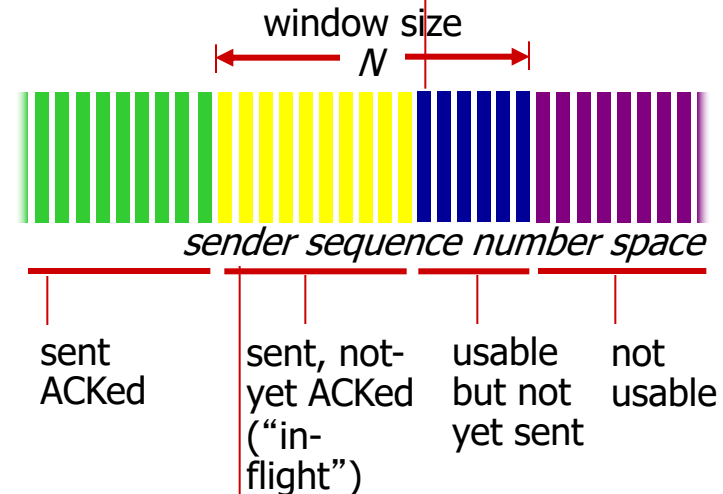
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

- A: TCP spec doesn’t say, - up to implementor

outgoing segment from sender

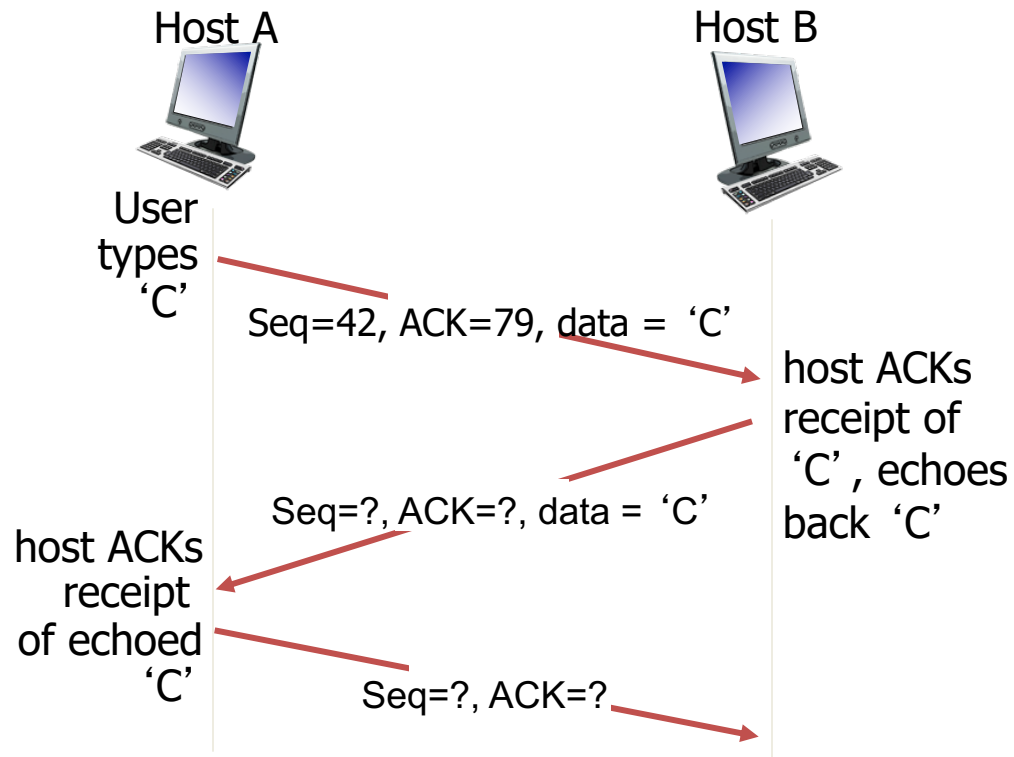
source port #	dest port #
sequence number	
acknowledgement number	
	rwnd
checksum	urg pointer



incoming segment to sender

source port #	dest port #
sequence number	
acknowledgement number	
	A
checksum	urg pointer

TCP seq. numbers, ACKs



simple telnet scenario

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

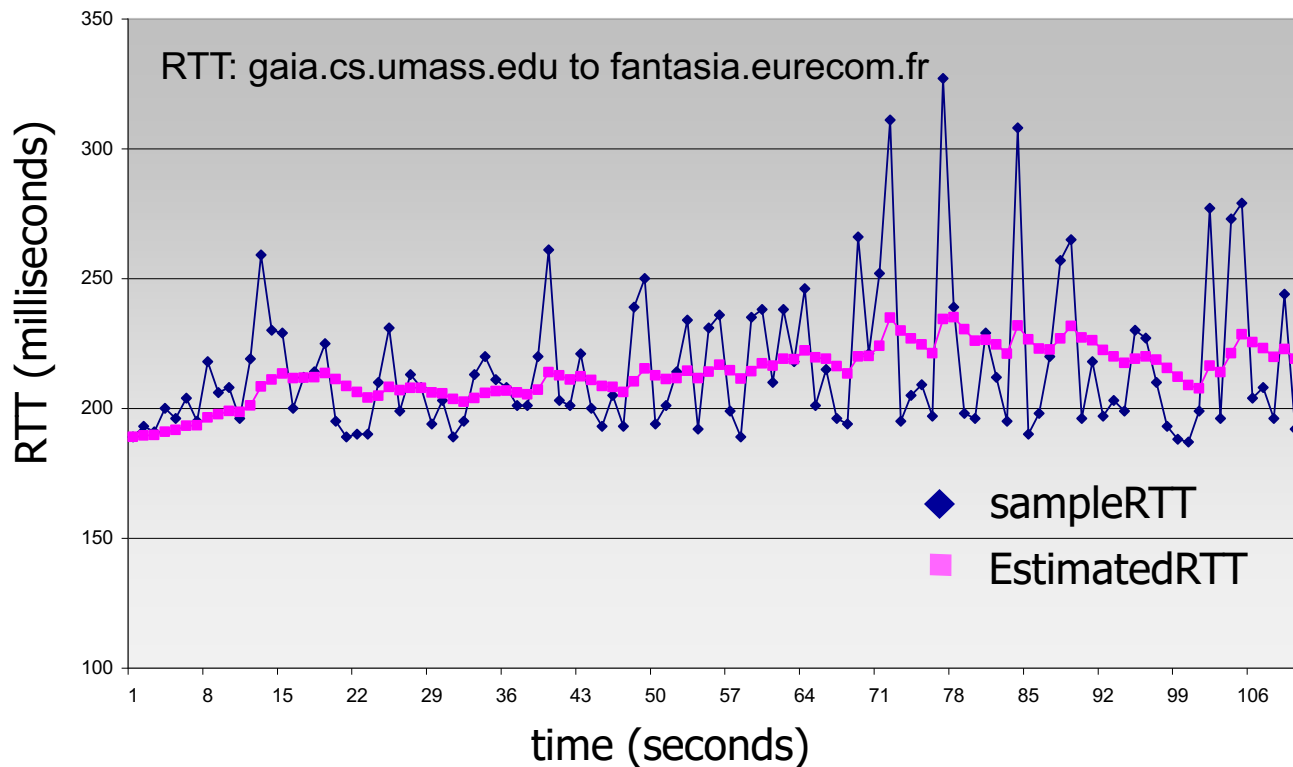
Q: how to estimate RTT?

- **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”

- Check out the online interactive exercises for more examples:
http://gaia.cs.umass.edu/kurose_ross/interactive/TCP_RTT.php

TCP reliable data transfer

- TCP creates rdt service on top of IP' s unreliable service
 - pipelined segments
 - cumulative acks
 - single retransmission timer
- retransmissions triggered by:
 - timeout events
 - duplicate acks

let' s initially consider simplified TCP sender:

- ignore duplicate acks
- ignore flow control, congestion control

TCP sender events:

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 the oldest unacked segment
 - expiration interval: **TimeoutInterval**

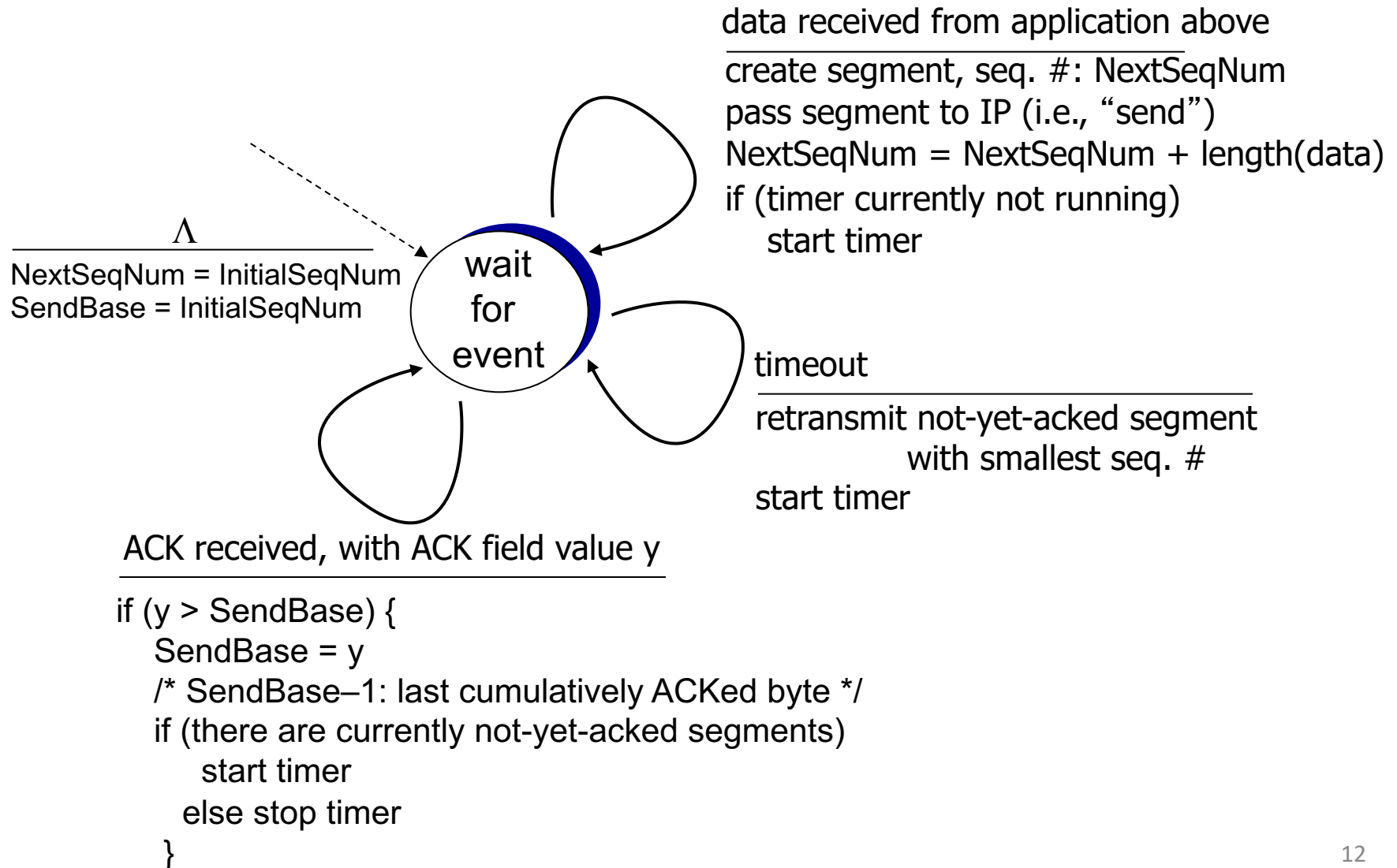
timeout:

- retransmit segment that caused timeout
- restart timer

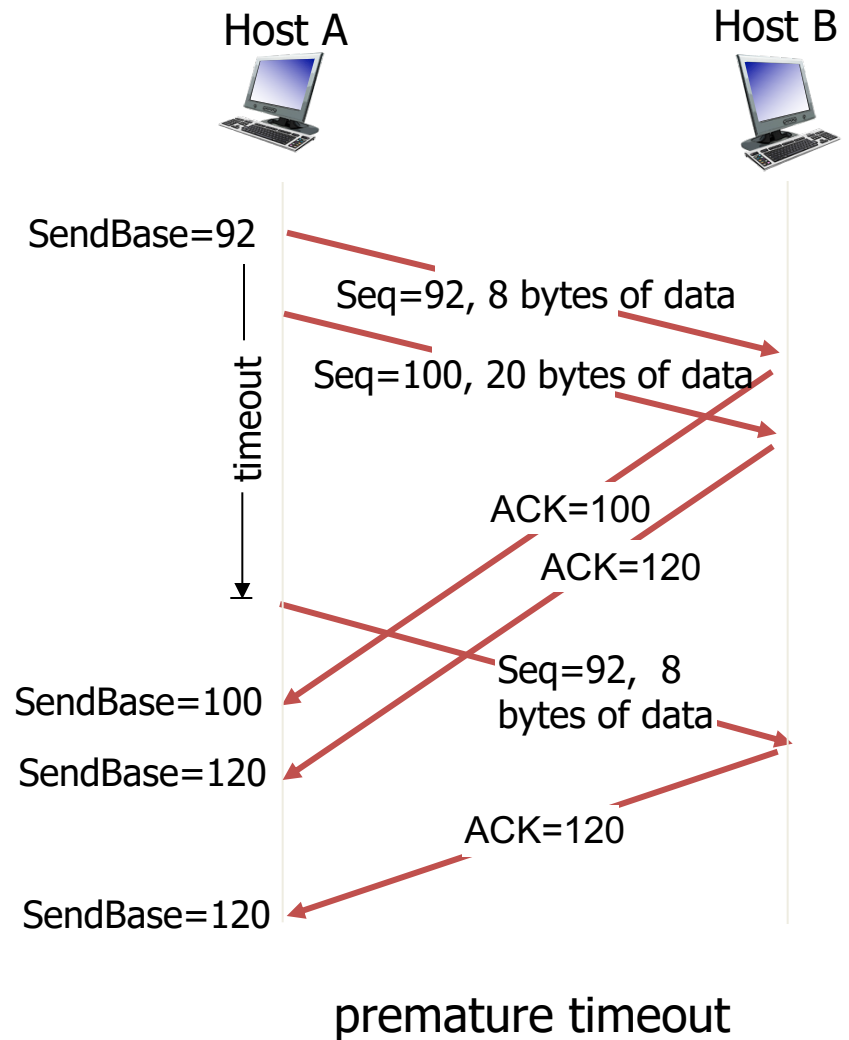
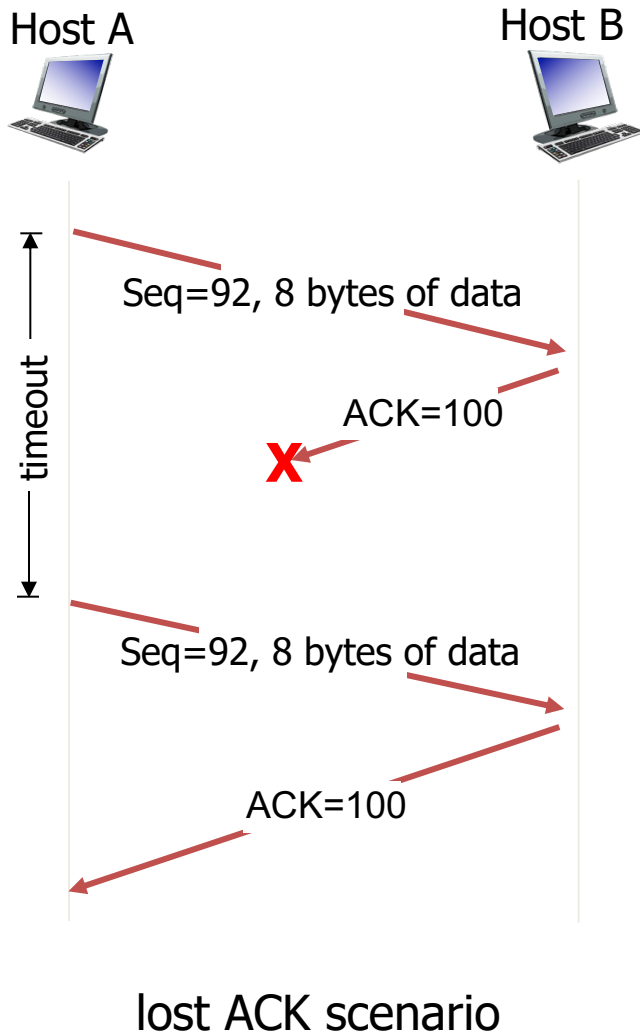
ack rcvd:

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

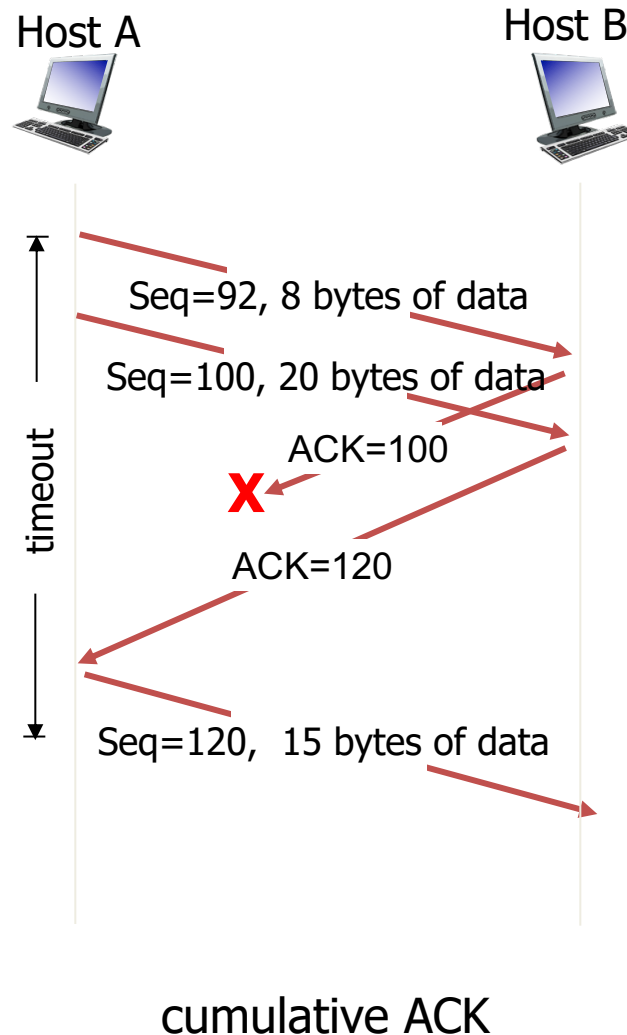
TCP sender (simplified)



TCP: retransmission scenarios



TCP: retransmission scenarios



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. <i>Wait up to 500ms</i> 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

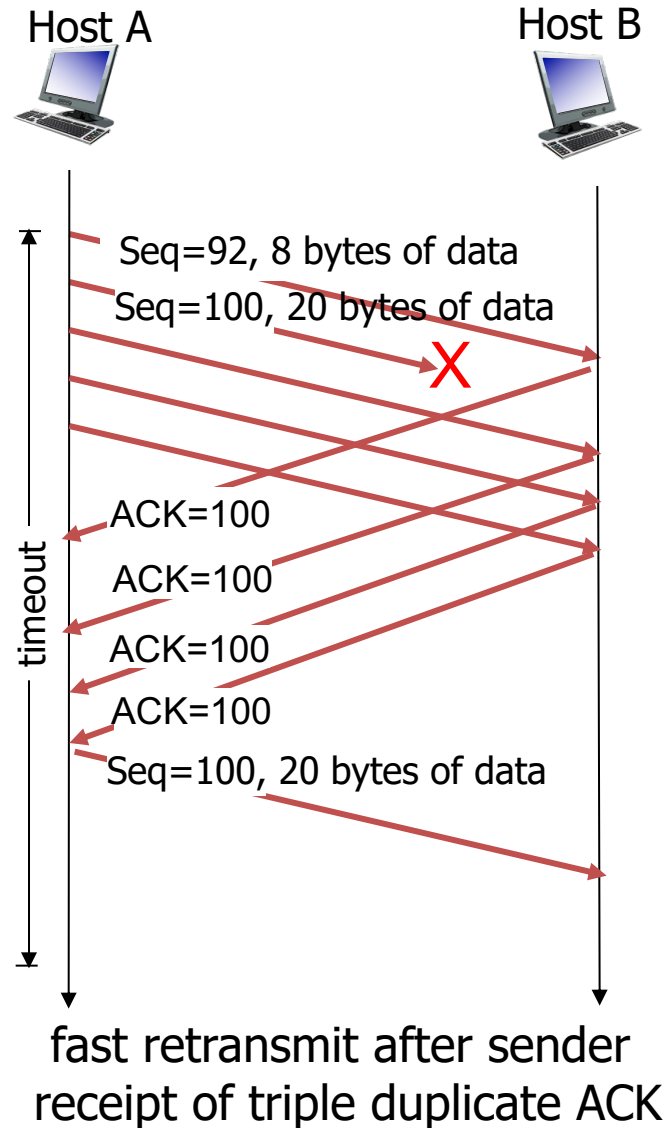
TCP fast retransmit

- time-out period often relatively long:
 - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
 - sender often sends many segments back-to-back
 - if segment is lost, there will likely be many duplicate ACKs.

TCP fast retransmit

- if sender receives **4** ACKs for same data (“triple duplicate ACKs”), resend unacked segment with smallest seq #
- likely that unacked segment lost, so don't wait for timeout

TCP fast retransmit



Revisit: TCP Congestion Control

