This Lecture will be recorded!!!

# Welcome to

# *CS 3516:*
# *Computer Networks*

Prof. Yanhua Li

Time: 9:00am –9:50am M, T, R, and F
Zoom Lecture
Fall 2020 A-term

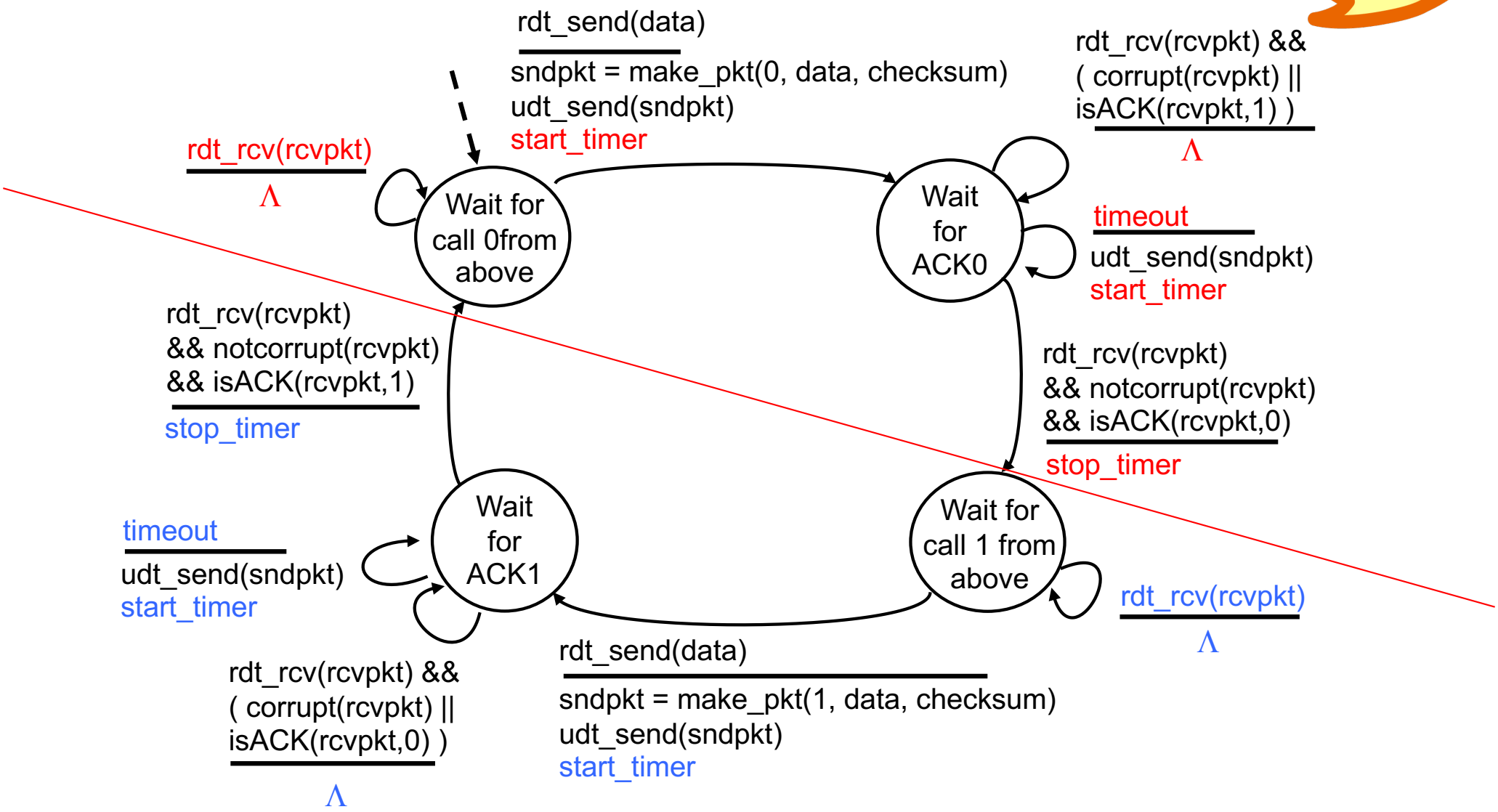


Some slides are originally from the course materials of the textbook “Computer Networking: A Top Down Approach”, 7th edition, by Jim Kurose, Keith Ross, Addison-Wesley March 2016. Copyright 1996-2017 J.F Kurose and K.W. Ross, All Rights Reserved.

# Updates

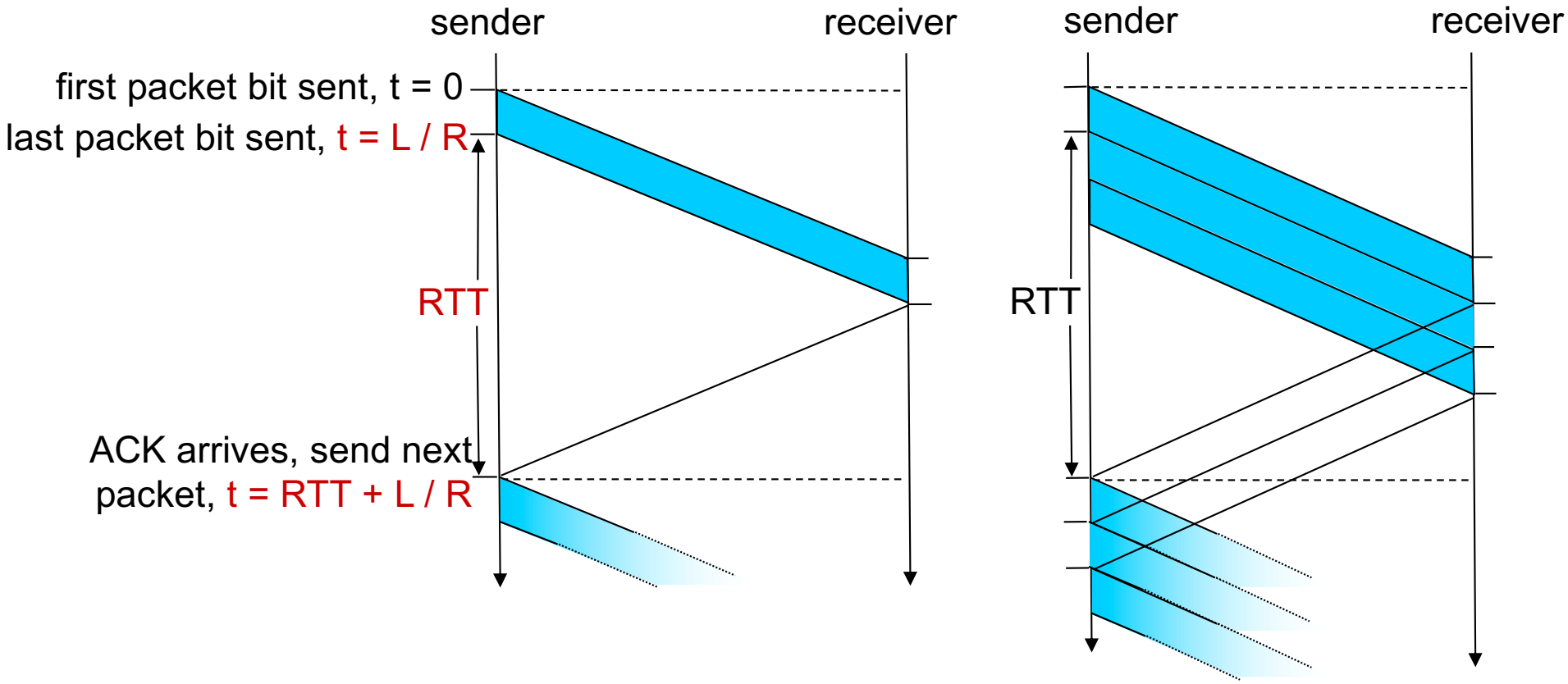
- Quiz 6 Grading Done
- Lab 2 Grading Done
- Project 1
  - Grading (almost done, will be posted today)
- Mid-term
  - Grading by Today
- Project 2
  - Due on 10/5 M
  - Extra office hours (To be announced)
- Quiz 7:
  - This Friday
  - *TCP and Network Layer Intro*

# rdt3.0 sender




rdt3.0 receiver ?

# rdt3.0 vs Pipelining approach




3-packet pipelining increases utilization by a factor of 3!

RDT3.0

Pipelining approach

# Pipelined protocols: overview

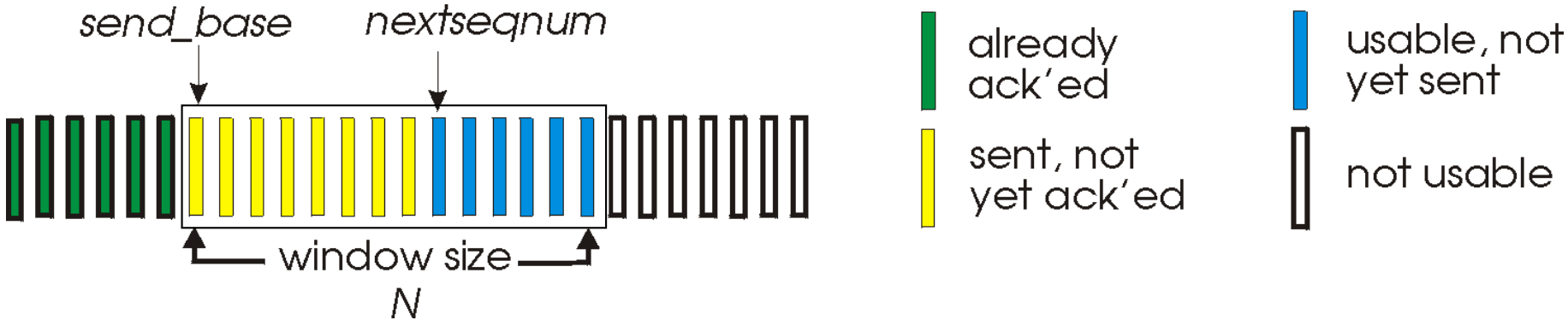
Go-back-N:

- sender can have up to N unacked packets in pipeline
- receiver only sends *cumulative ack*
  - doesn't ack packet if there's a gap
- sender has timer for oldest unacked packet
  - when timer expires, retransmit *all* unacked packets

- Extending from
- one unacknowledged pkt (in RDT3.0) to
- multiple unacknowledged pkts (in pipelining)

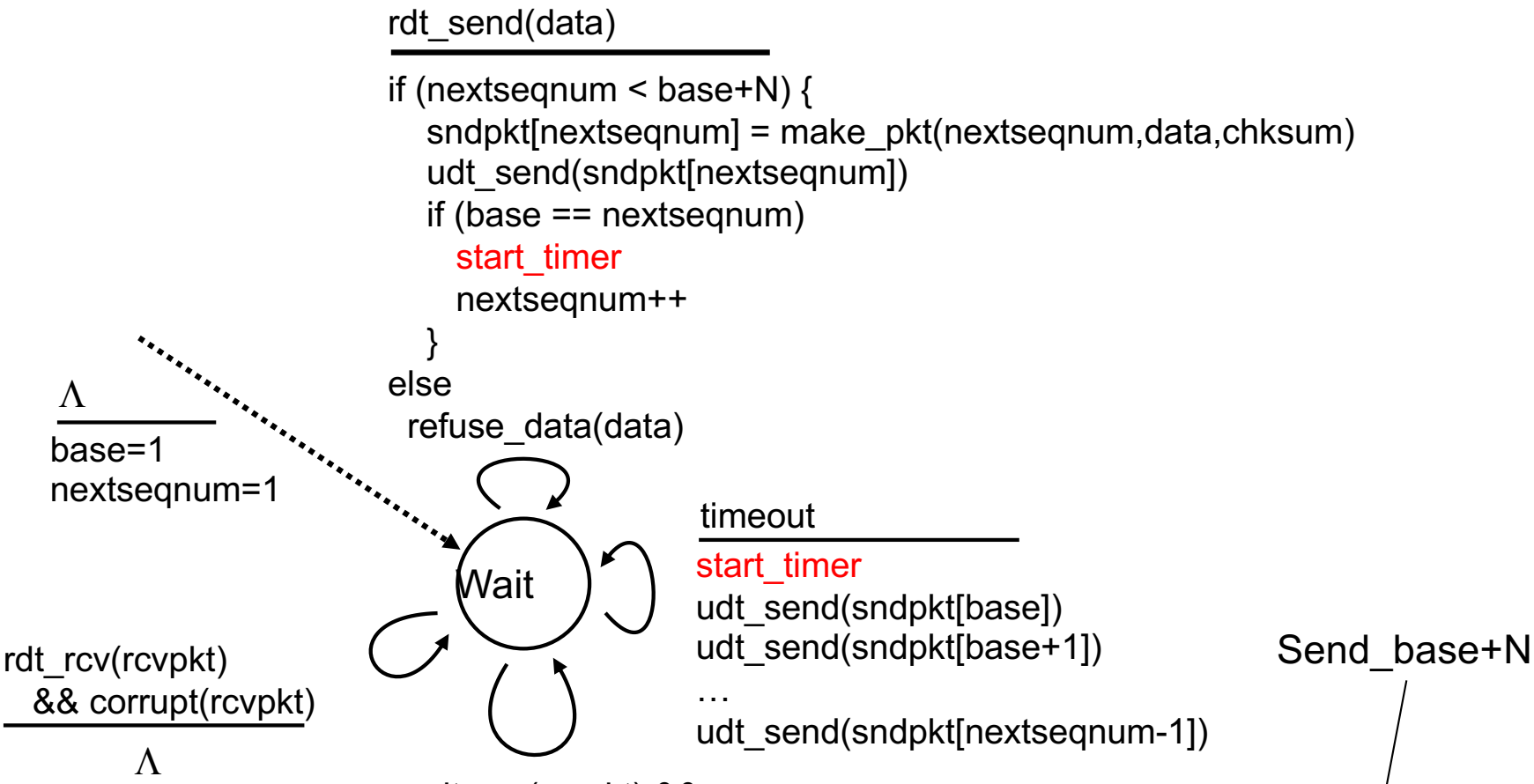
# Go-Back-N: sender

- k-bit seq # in pkt header
- "window" of up to N, consecutive unack'ed pkts allowed

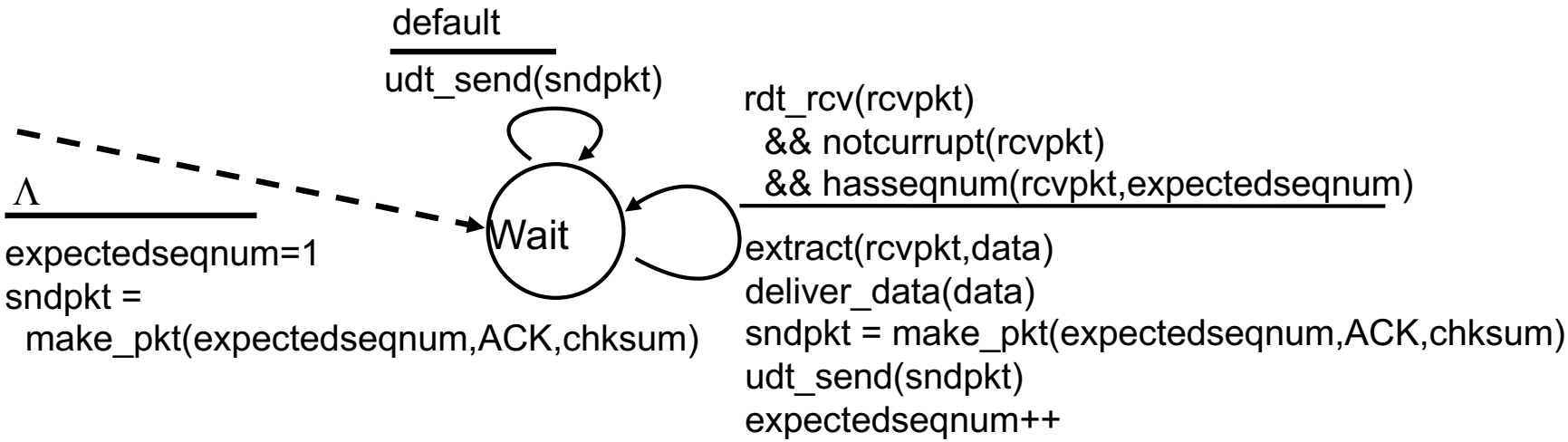



- ACK(n): ACKs all pkts up to, including seq # n - *"cumulative ACK"*
  - may receive duplicate ACKs (see receiver)
- timer for oldest in-flight pkt
- *timeout(n)*: retransmit packet n and all higher seq # pkts in window

# GBN: sender extended FSM



# GBN: receiver extended FSM

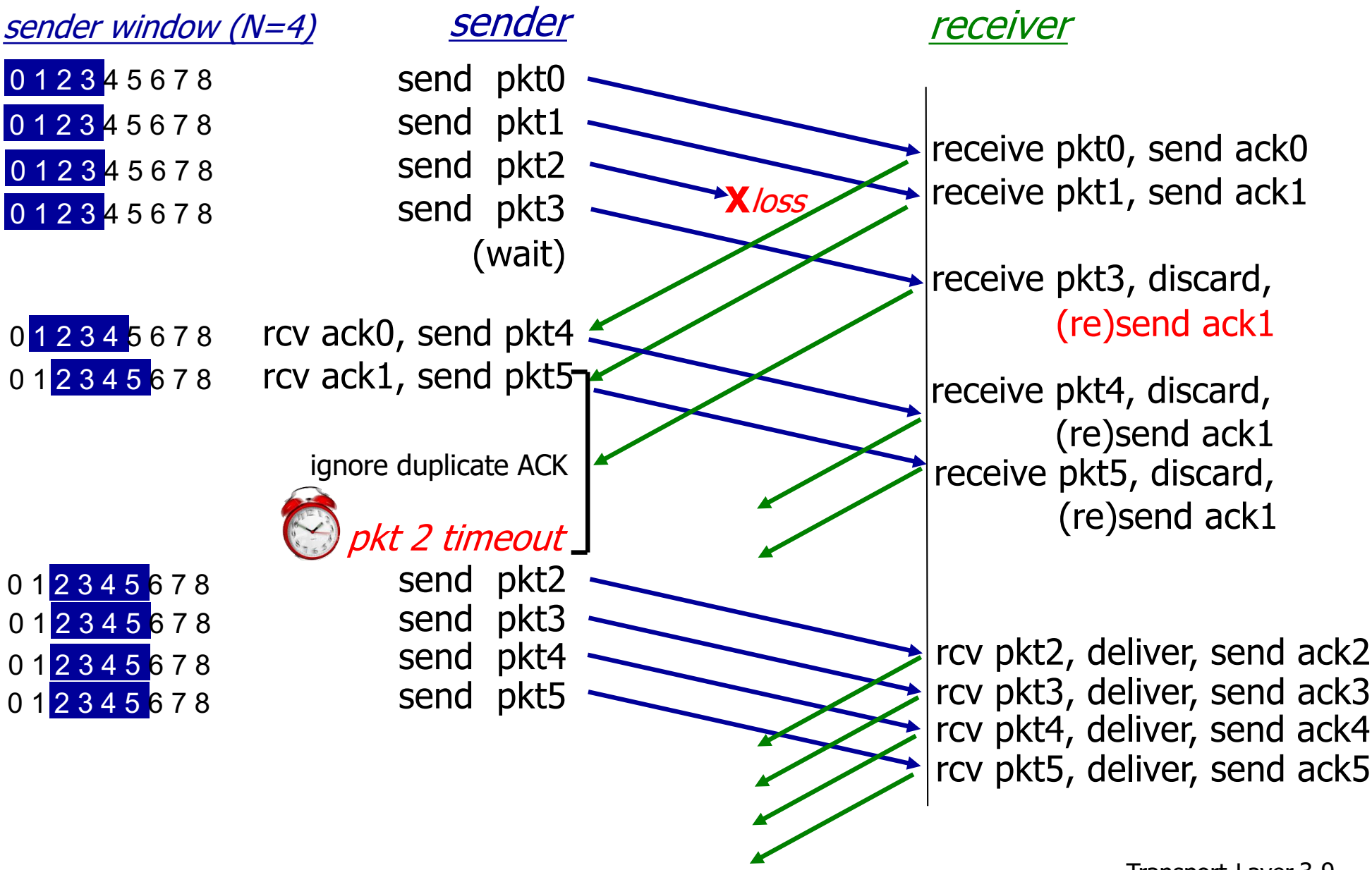



ACK-only: always send ACK for correctly-received pkt with highest *in-order* seq #

- 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



# Chapter 3 outline

3.1 transport-layer services

3.2 multiplexing and demultiplexing

3.3 connectionless transport: UDP

3.4 principles of reliable data transfer

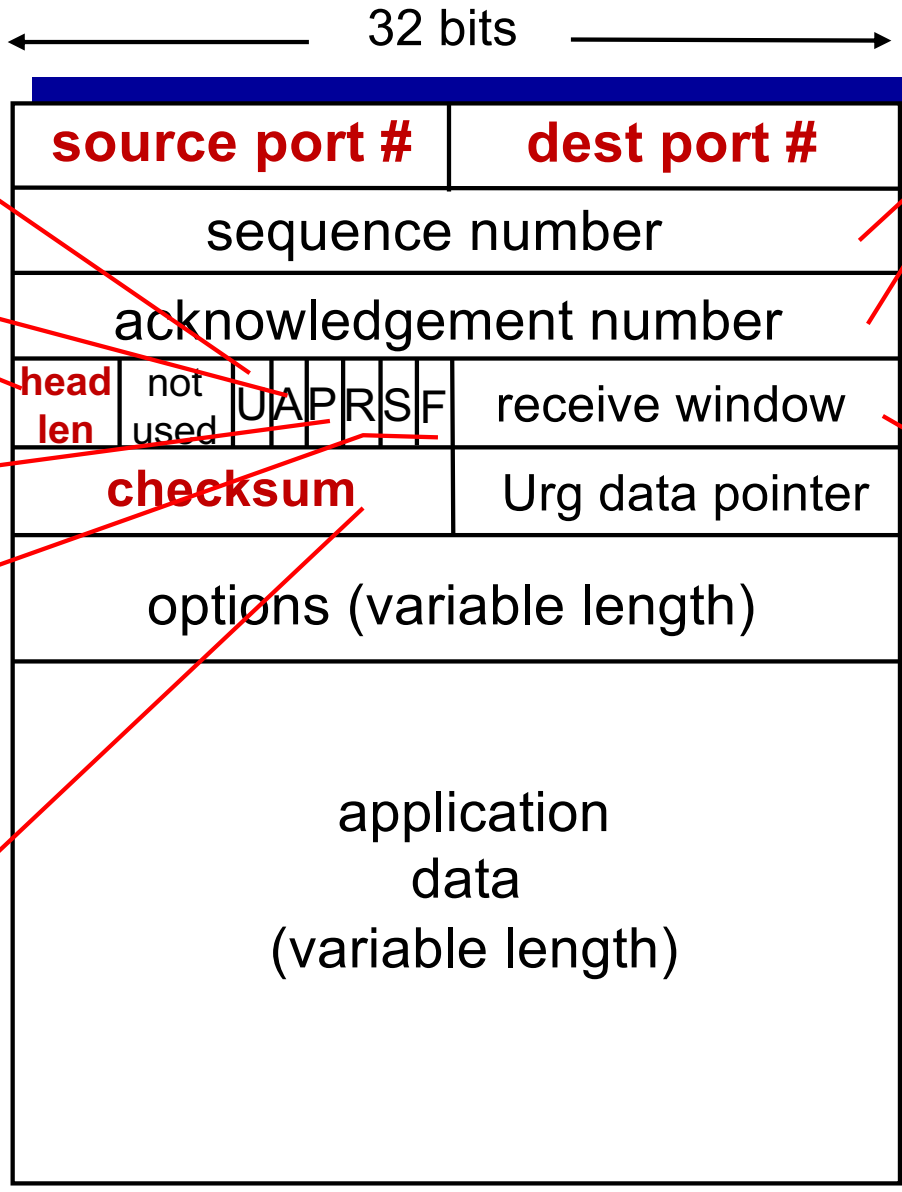
3.5 connection-oriented transport: TCP

- segment structure
  - Segment Format
  - Sequence #
  - Timeout value estimation

# TCP: Overview RFCs: 793,1122,1323, 2018, 2581

- reliable, in-order *byte steam:*
  - no "message boundaries"
- pipelined:
  - TCP window size
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size (e.g., 1460B)
  - MTU: layer 3 maximum transmission unit (e.g., 1500B for Ethernet)
- connection-oriented:
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange

# TCP segment structure (20+ bytes)




URG: urgent data (generally not used)

ACK: ACK # valid

In 4 bytes = 32 bits

PSH: push data now (generally not used)

RST, SYN, FIN: connection estab (setup, teardown commands)

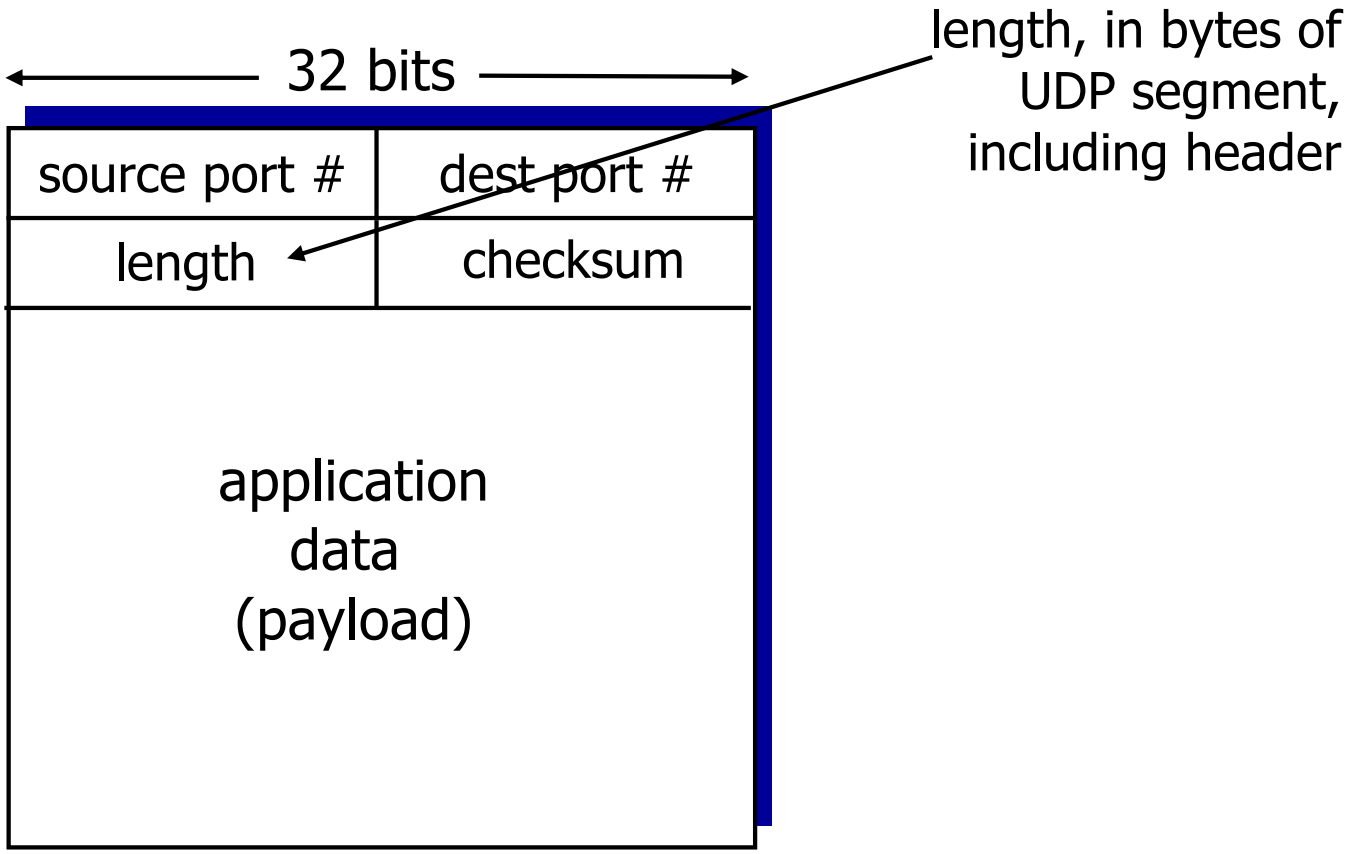
Internet checksum (as in UDP)

counting by bytes of data (not segments!)

# bytes rcvr willing to accept (for flow control)

Length of UDP header?

# UDP: segment header (8 bytes)




UDP segment format

# Chapter 3 outline

3.5 connection-oriented transport: TCP

- segment structure
  - Segment Format
  - Sequence #
  - Timeout value estimation

# TCP seq. numbers, ACKs

**sequence numbers:**

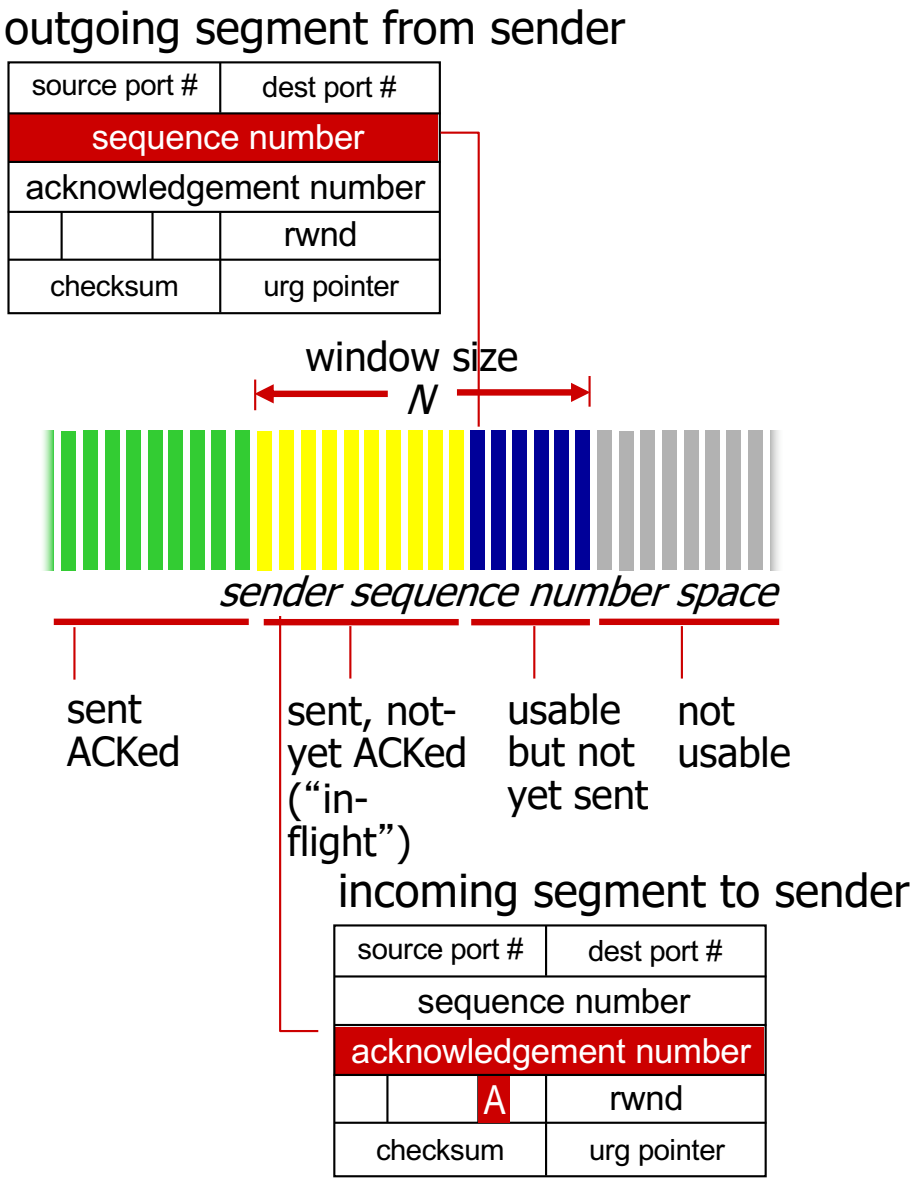
- byte stream "number" of first byte in segment's data

**acknowledgements:**

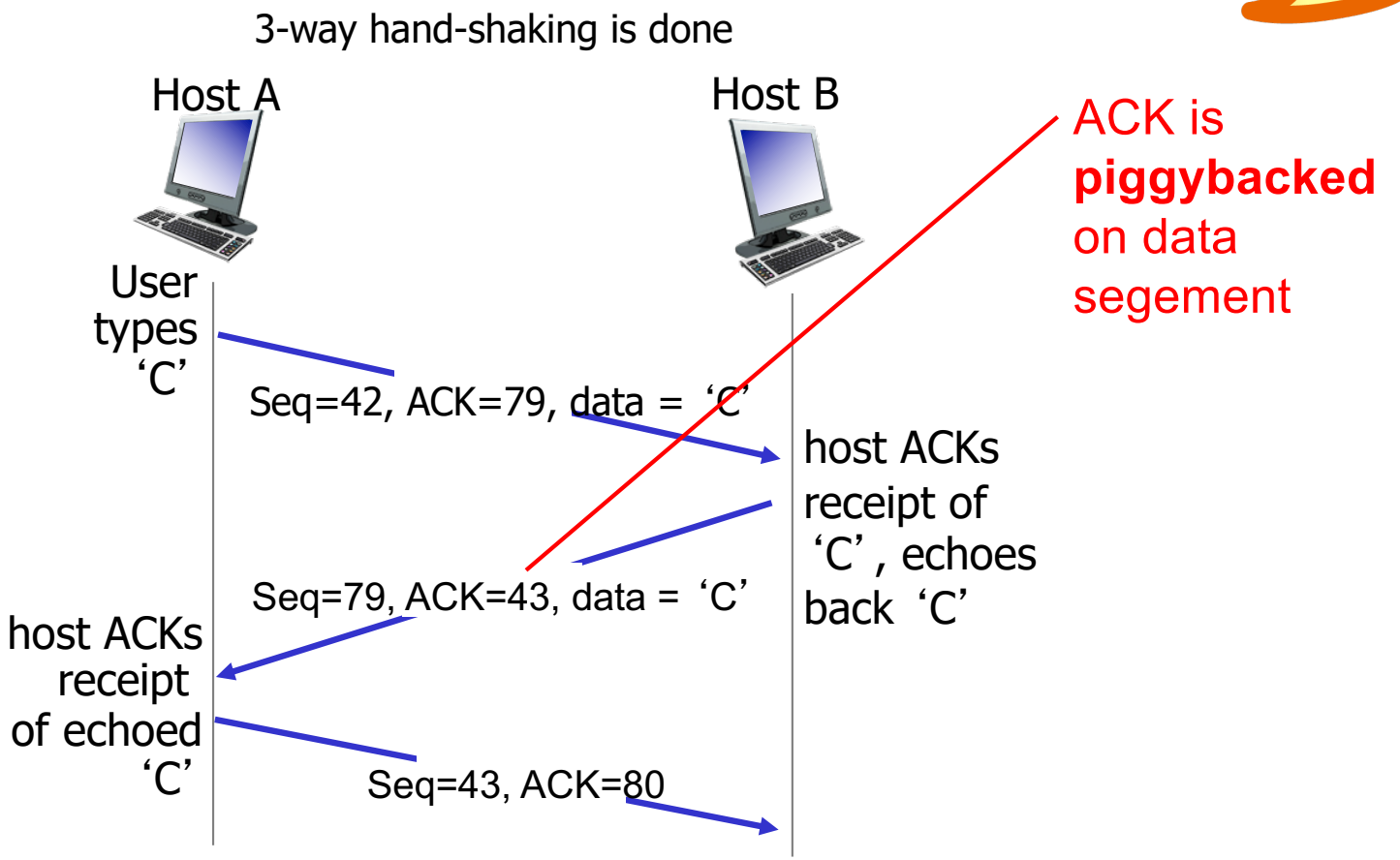
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

- A: TCP spec doesn't say, - up to implementor
- rdt 3.0 & GBN & more



# TCP seq. numbers, ACKs




simple telnet scenario

How to determine a packet loss?

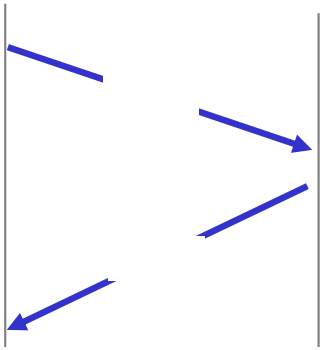
# TCP round trip time, timeout

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?

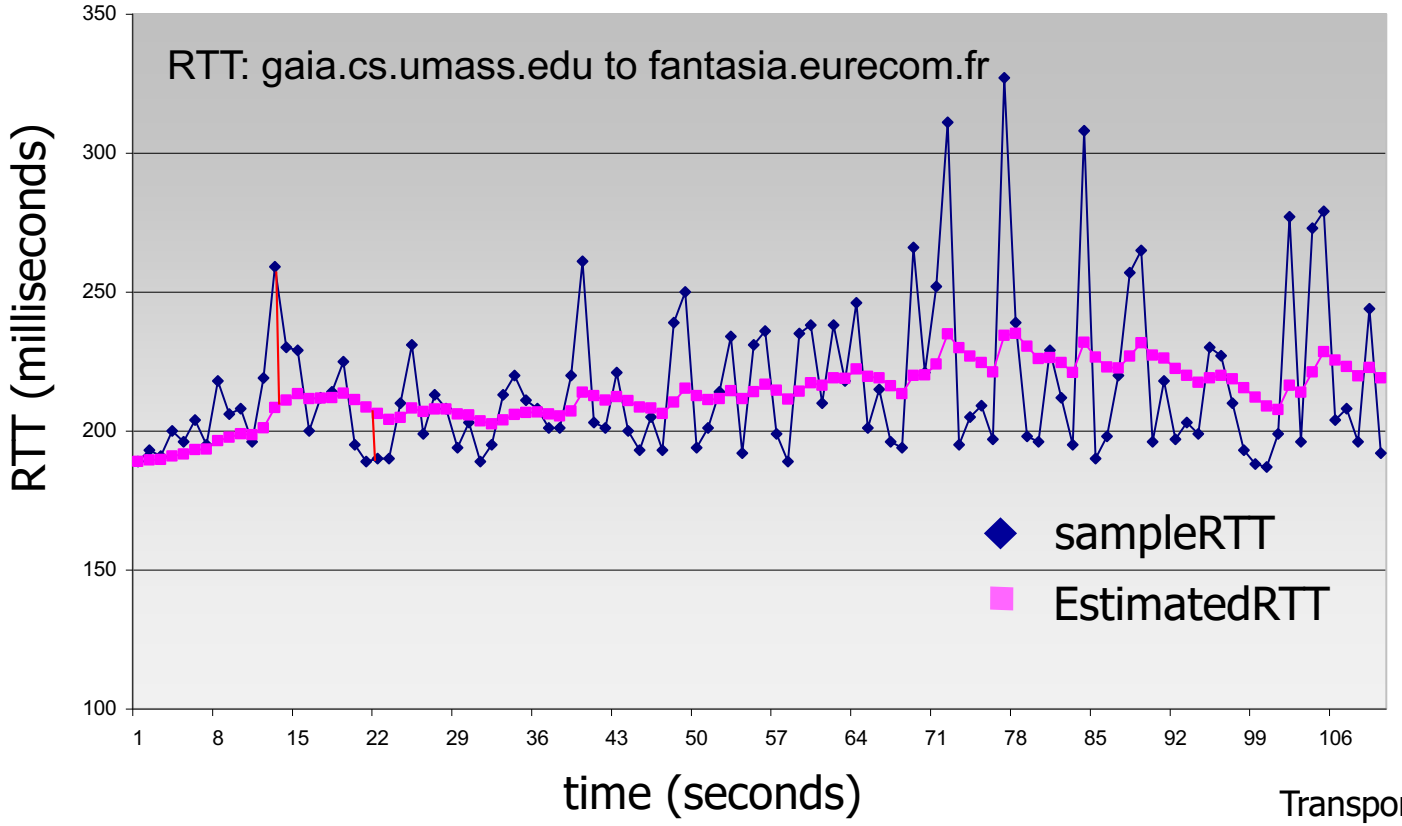
- **`SampleRTT`**: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **`SampleRTT`** will vary, want estimated RTT "smoother"
  - average several *recent* measurements, not just current **`SampleRTT`**



# TCP round trip time, timeout

**EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT**

- exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: α = 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:

**DevRTT = (1-β)*DevRTT +**
**β*|SampleRTT-EstimatedRTT|**

**(typically, β = 0.25)**

**TimeoutInterval = EstimatedRTT + 4*DevRTT**

estimated RTT

"safety margin"

# Chapter 3 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
  - Segment Format
  - Sequence #
  - Timeout value estimation
- reliable data transfer
  - FSM
  - TCP Retransmissions

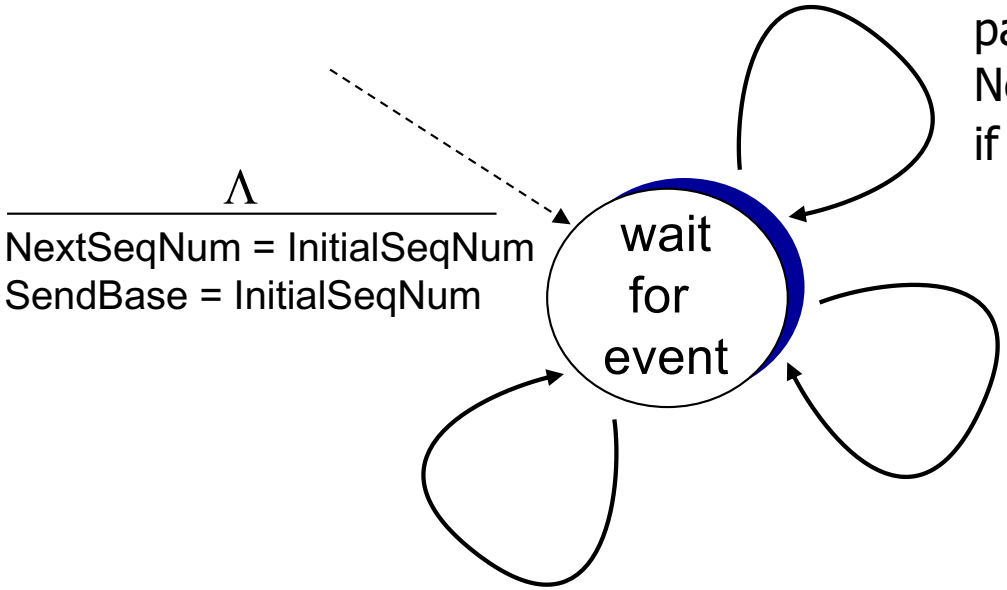
# TCP reliable data transfer (Sim. GBN)

- 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

# TCP sender (simplified)




data received from application above

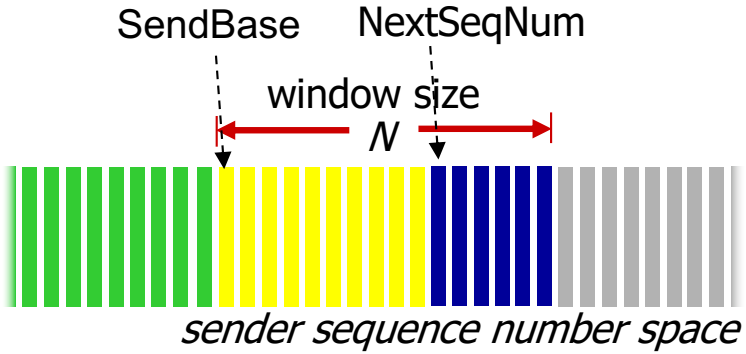
create segment, seq. #: NextSeqNum
pass segment to IP (i.e., "send")
NextSeqNum = NextSeqNum + length(data)
if (timer currently not running)
    start timer

timeout

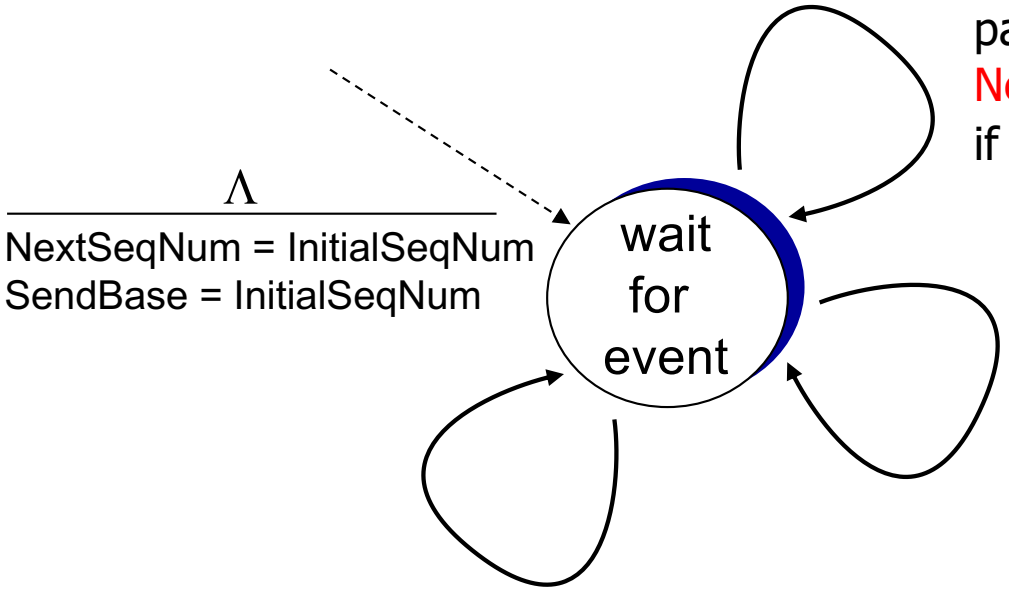
retransmit not-yet-acked segment
        with smallest seq. #
start timer

ACK received, with ACK field value y

```
if (y > SendBase) {
    SendBase = y
    /* SendBase–1: last cumulatively ACKed byte */
    if (there are currently not-yet-acked segments)
        start timer
      else stop timer
     }
```



# TCP sender (simplified)




data received from application above

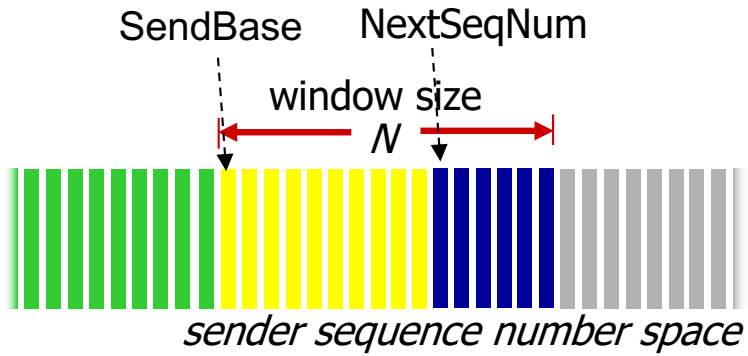
create segment, seq. #: NextSeqNum
pass segment to IP (i.e., "send")
NextSeqNum = NextSeqNum + length(data)
if (timer currently not running)
    start timer

timeout

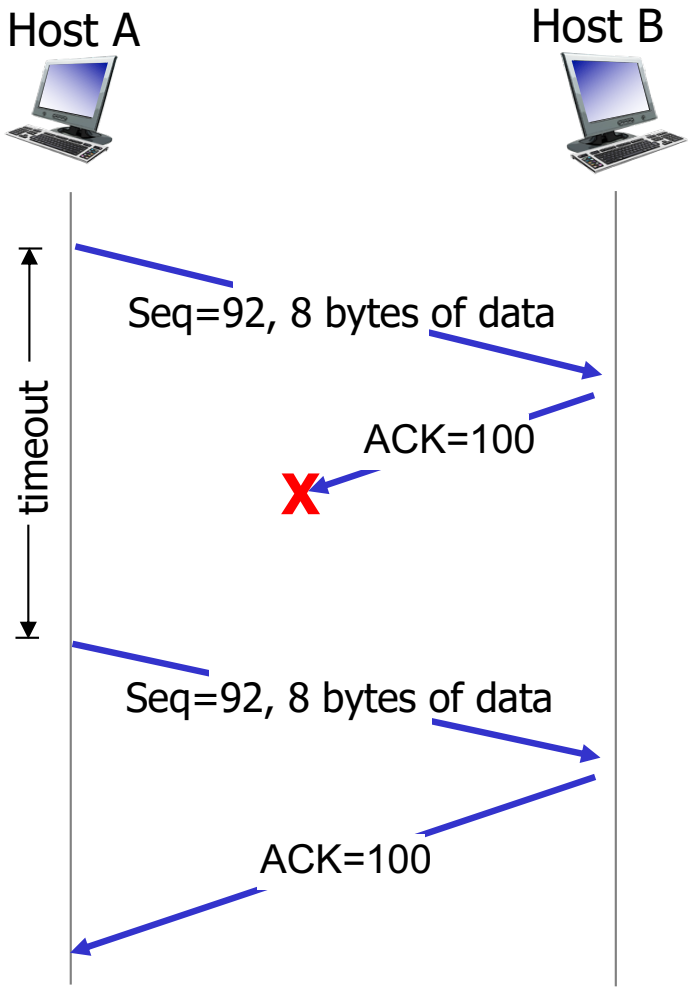
retransmit not-yet-acked segment
        with smallest seq. #
start timer

ACK received, with ACK field value y

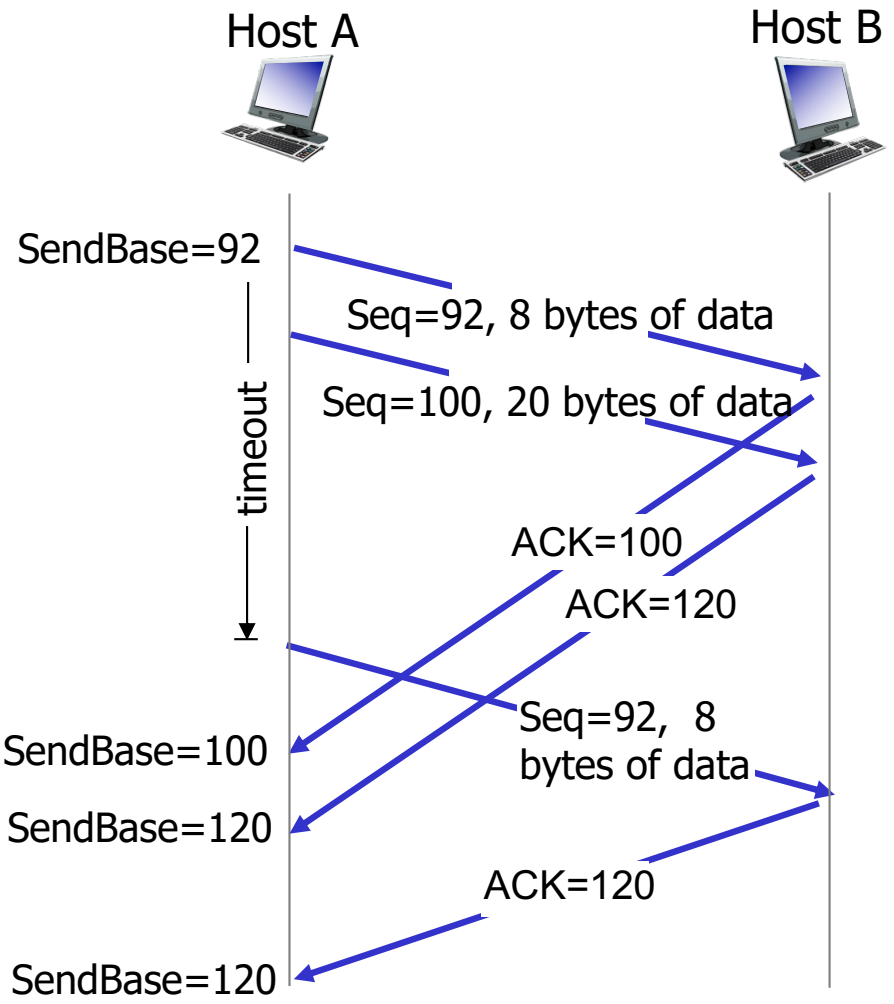
```
if (y > SendBase) {
    SendBase = y
    /* SendBase–1: last cumulatively ACKed byte */
    if (there are currently not-yet-acked segments)
        start timer
      else stop timer
     }
```



# TCP: retransmission scenarios

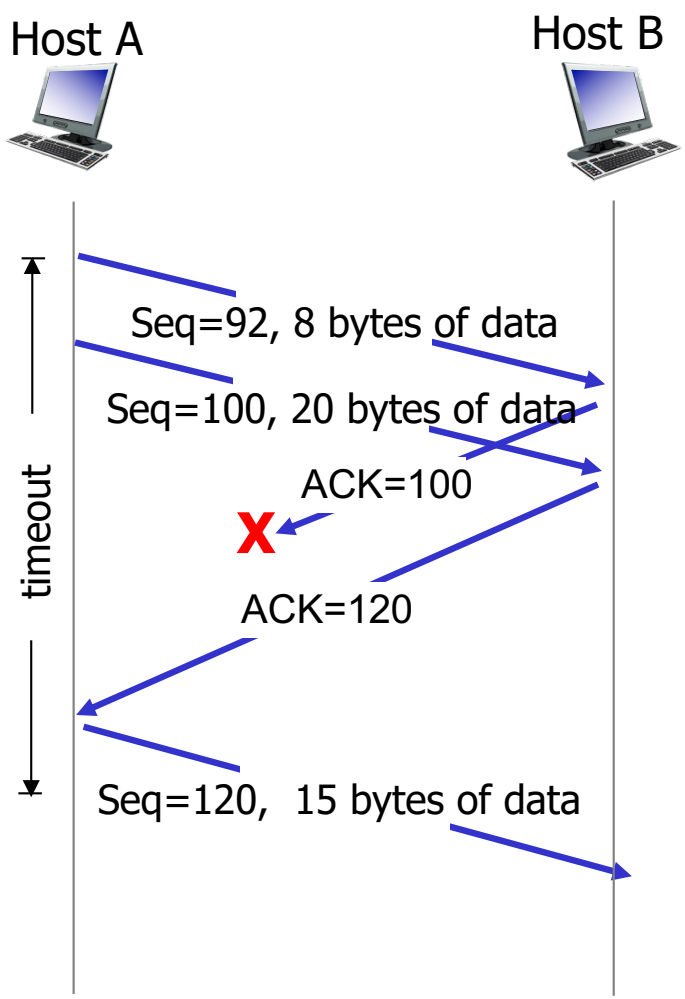



lost ACK scenario

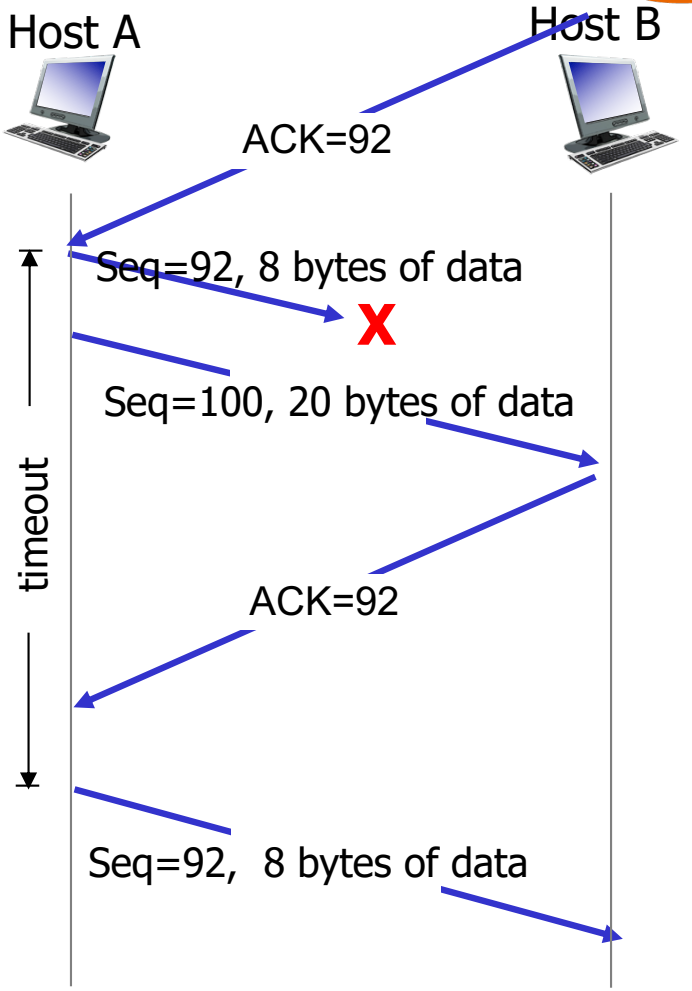



premature timeout

# TCP: retransmission scenarios

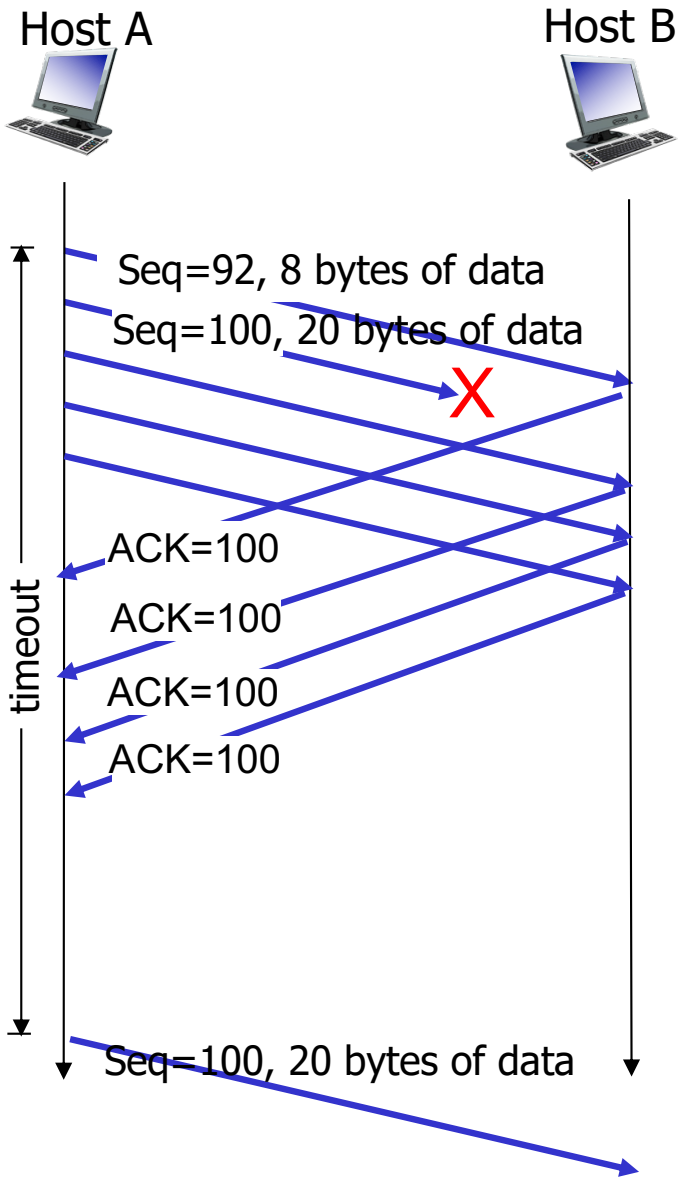



cumulative ACK




duplicate ACK

# TCP without fast retransmit



# TCP fast retransmit

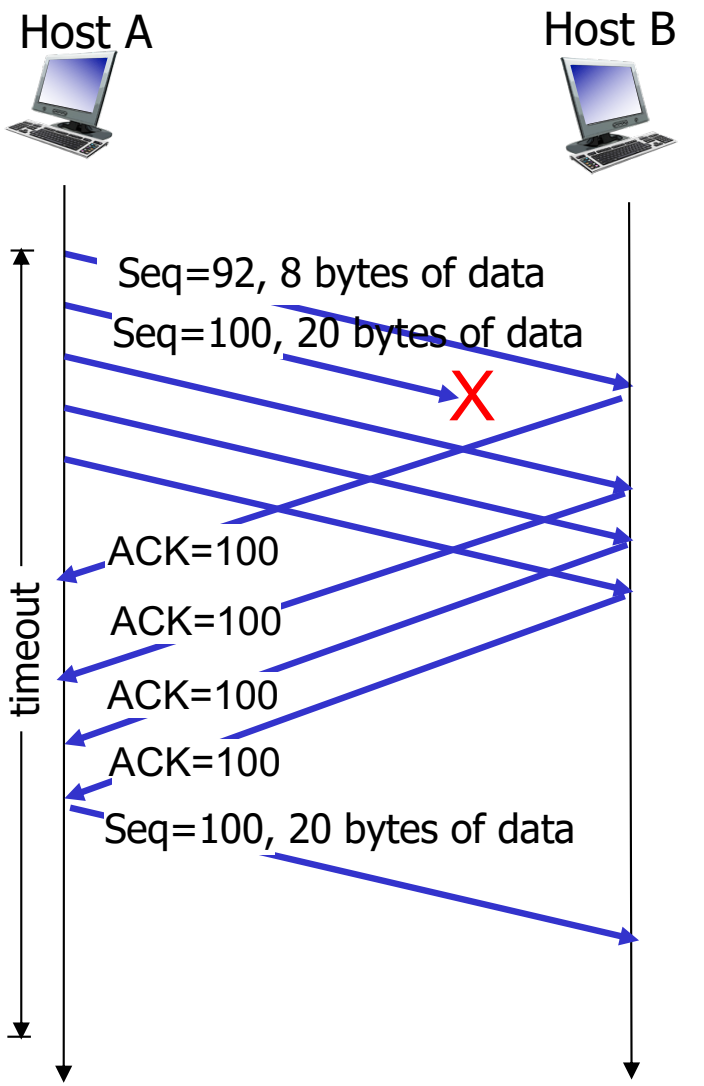
- time-out period often relatively long:
  - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs.

*TCP fast retransmit*

if sender receives 3 dup 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




fast retransmit after sender receipt of triple duplicate ACK

# Chapter 3: summary

- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
- instantiation, implementation in the Internet
  - UDP
  - TCP

next:

- leaving the network "edge" (application, transport layers)
- into the network "core"

# Questions?

# Optional Slides:

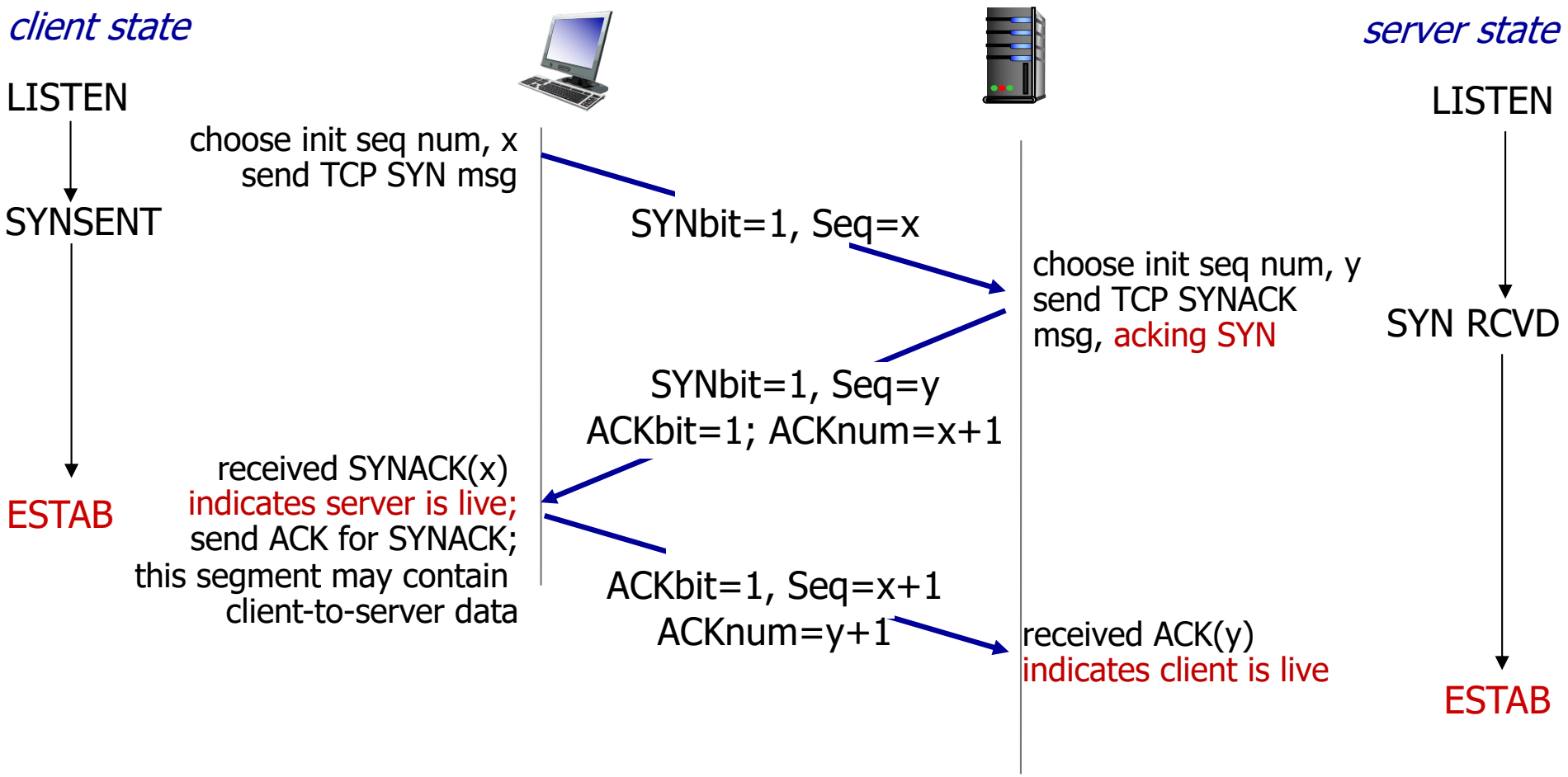
(additional materials for those who are interested)

TCP 3 way handshake

This will not be included in any Quiz nor final exam

# TCP 3-way handshake

| head len | not used | U | A | P | R | S | F | receive window |
|---|---|---|---|---|---|---|---|---|




Random initial seq # are generated on both sides

# TCP 3-way handshake: FSM