Lesson 05-03: TCP

CS 326 E Elements of Networking

Mikyung Han

mhan@cs.utexas.edu

Internet Reference Model

Example Protocols		Responsible for
FTP, HTTP, SMTP	Application	application specific needs
TCP, UDP	Transport	process to process data transfer
IP	Network	host to host data transfer across different network
Ethernet, WiFi	Link	data transfer between physically adjacent nodes
802.3 PHY	Physical	bit-by-bit or symbol-by-symbol delivery

2

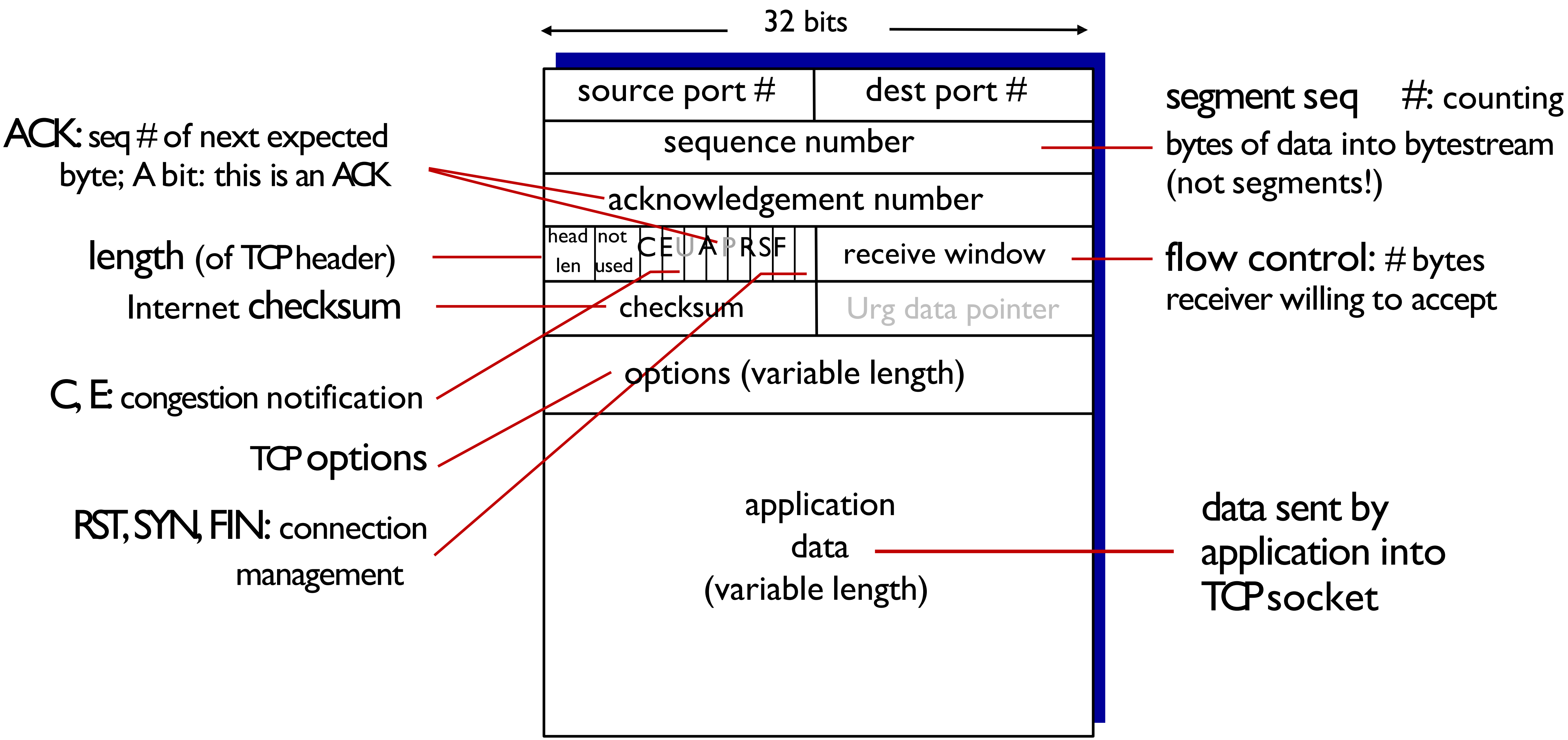
Outline

1. TCP overview

TCP: overview RFCs: 793,1122, 2018, 5681, 7323

- point-to-point:
  - one sender, one receiver
- reliable, in-order byte steam:
  - no “message boundaries"
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- cumulative ACKs
  - ACK N acks all packets till N-1 cumulatively
  - ACK N means send me Seq N next
- timeouts
- pipelining:
  - TCP congestion and flow control set window size
- connection-oriented (handshake)
- flow controlled:
  - sender will not overwhelm receiver

TCP segment structure



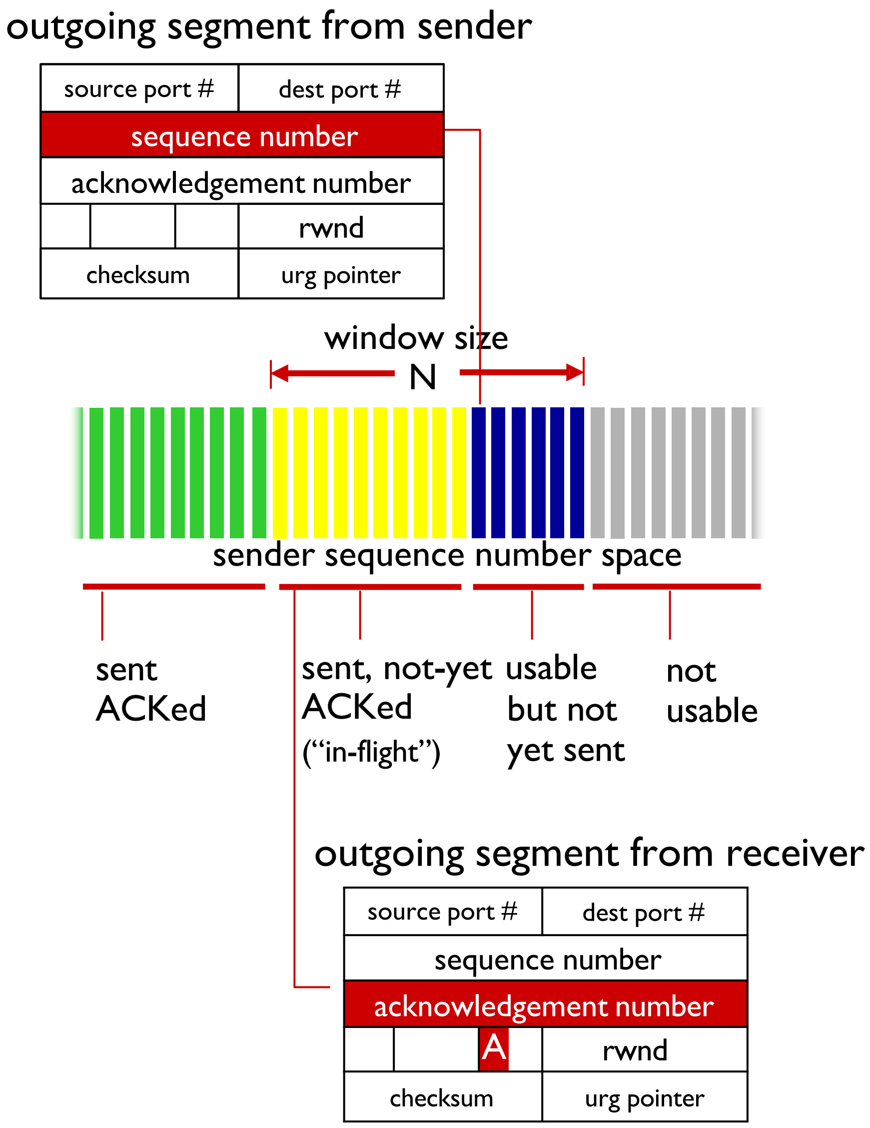
TCP sequence numbers, ACKs

Sequence numbers:

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

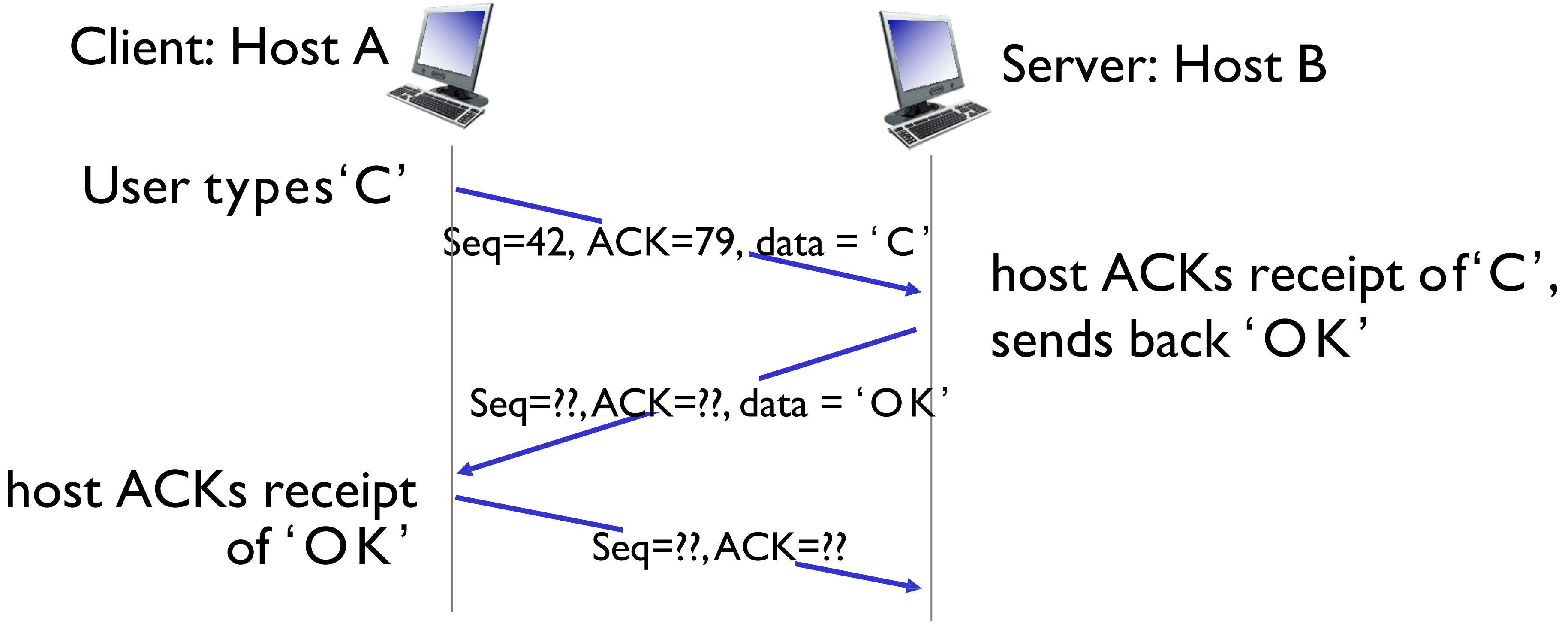
Acknowledgements:

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



TCP ACK *n* means "Got everything till n-1 so send me *n*"

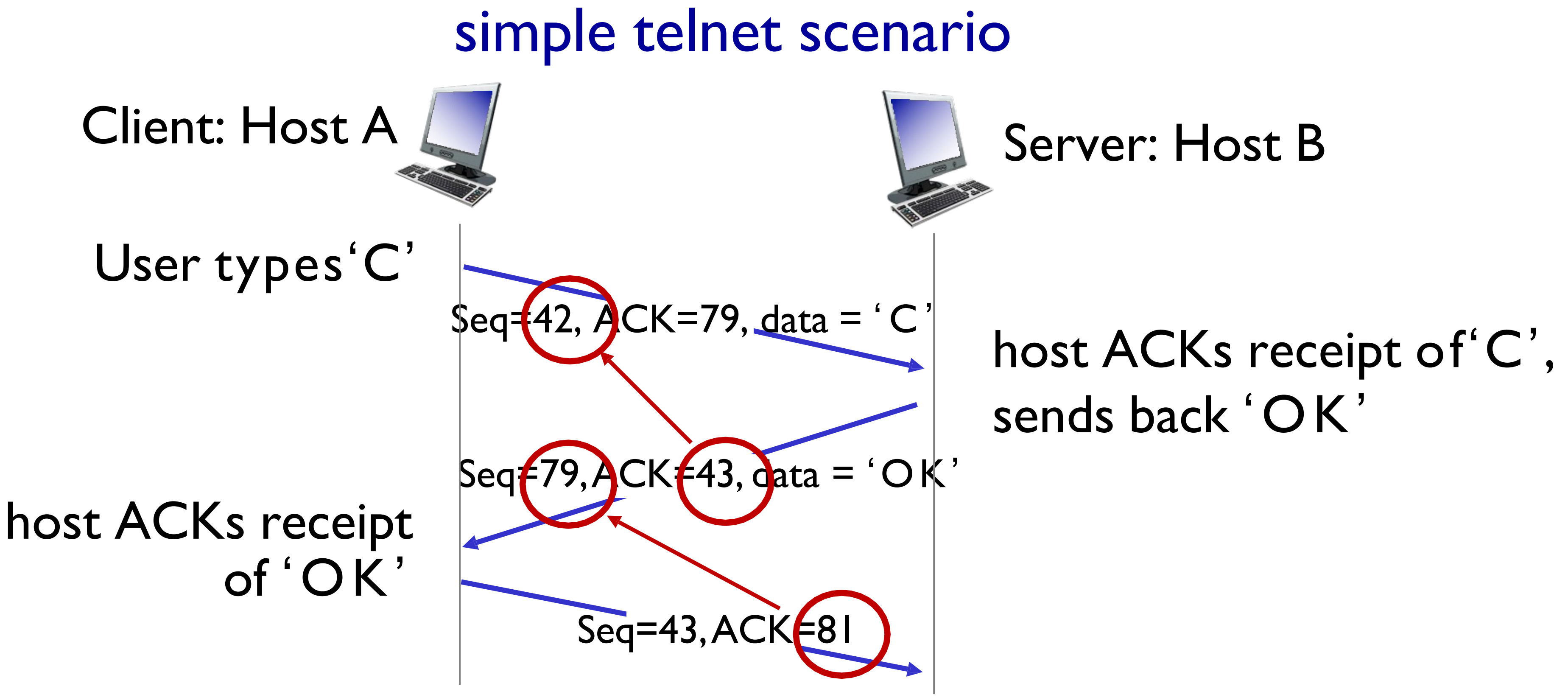
simple telnet scenario



Seq number is calculated based on bytes sent

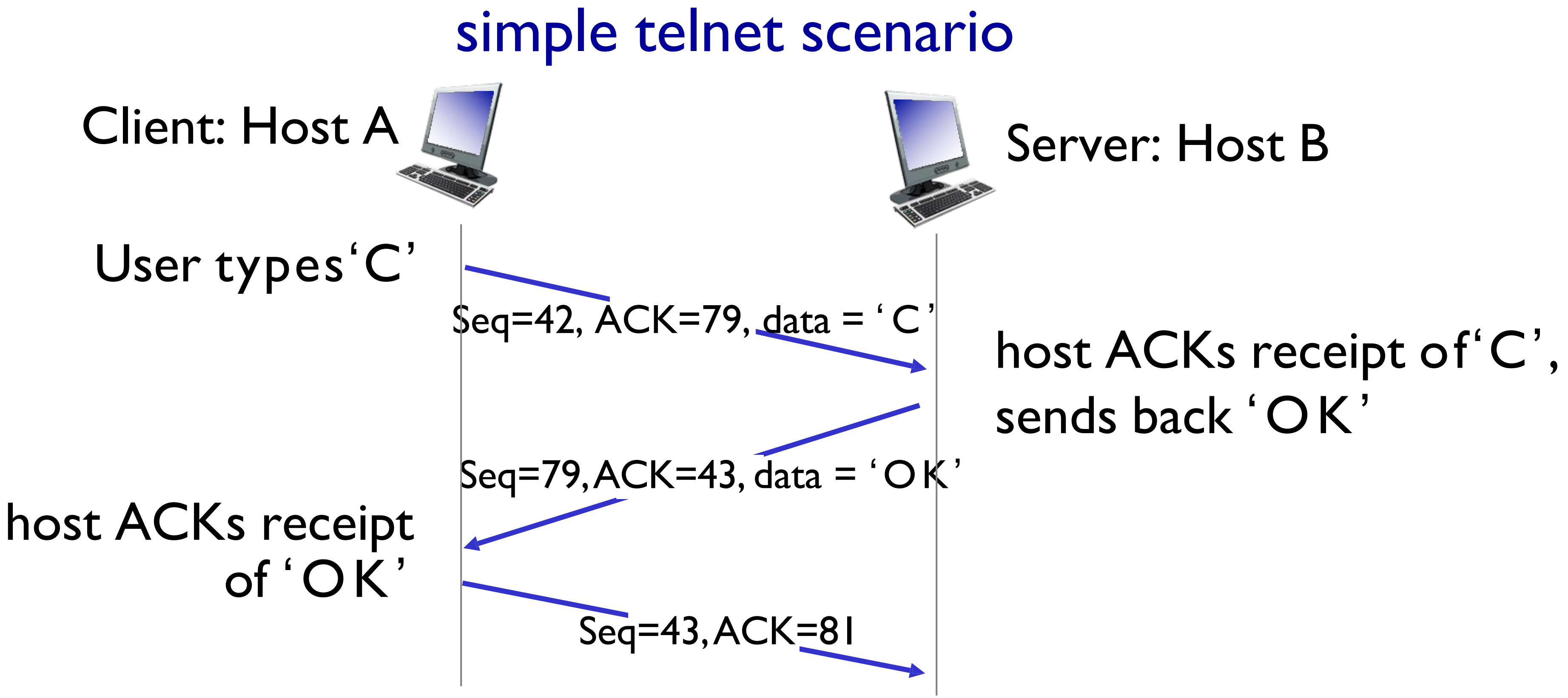
Actual num bytes sent is determined by network condition

TCP ACK *n* means "Got everything till n-1 so send me *n*"



Does the last segment have DATA? Why then seq no?

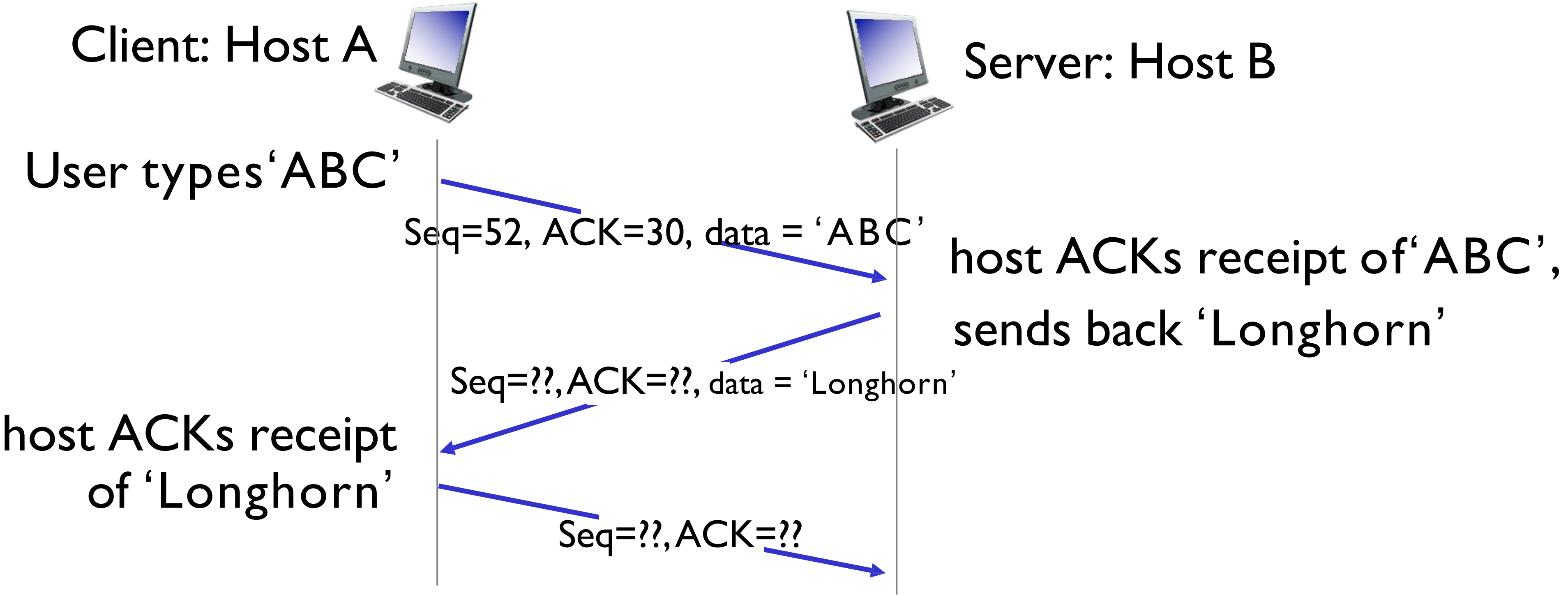
TCP ACKs can piggyback to DATA



Which segments have the ACKs piggybacked to DATA?

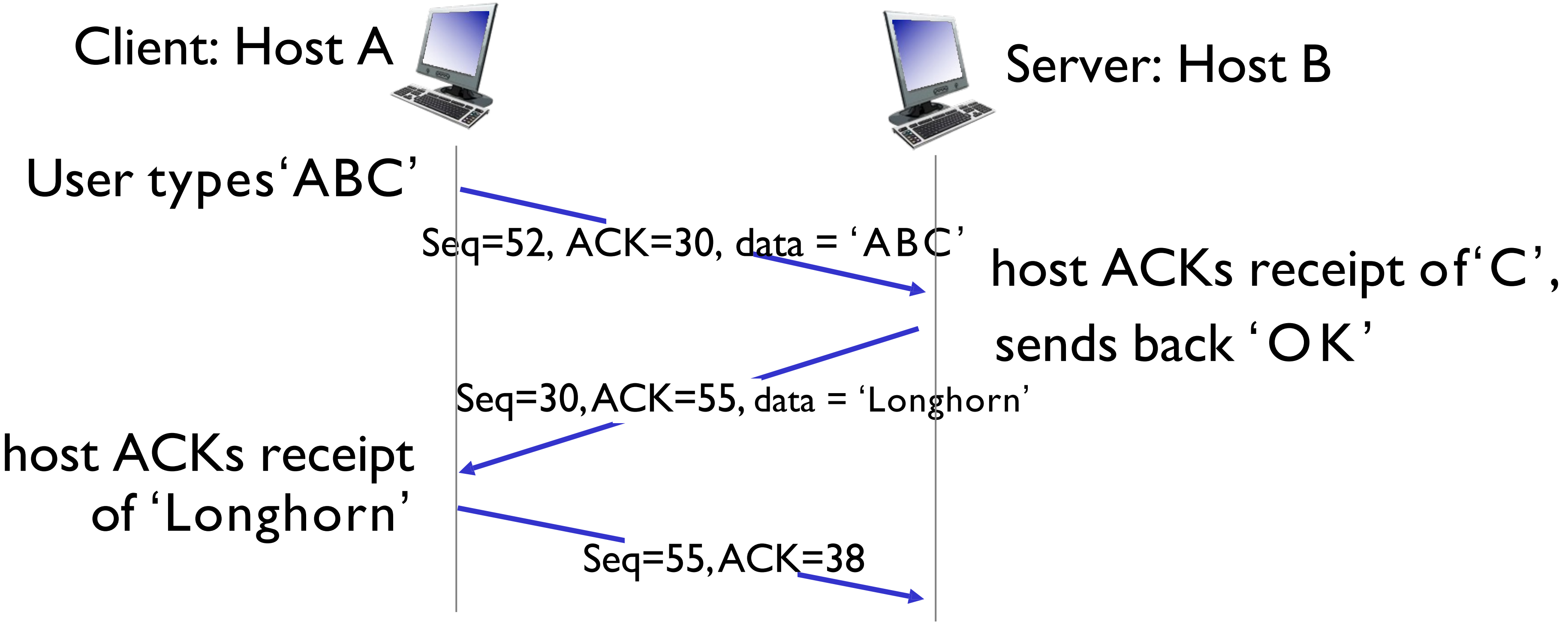
Practice one more time!

simple telnet scenario



Practice one more time!

simple telnet scenario



Outline

1. TCP overview
2. TCP timeout

How to set TCP timeout value?

- What happens if timeout value is too short?
- What happens if timeout value is too long?
- We know it should be at least longer than… what?

How to set TCP timeout value?

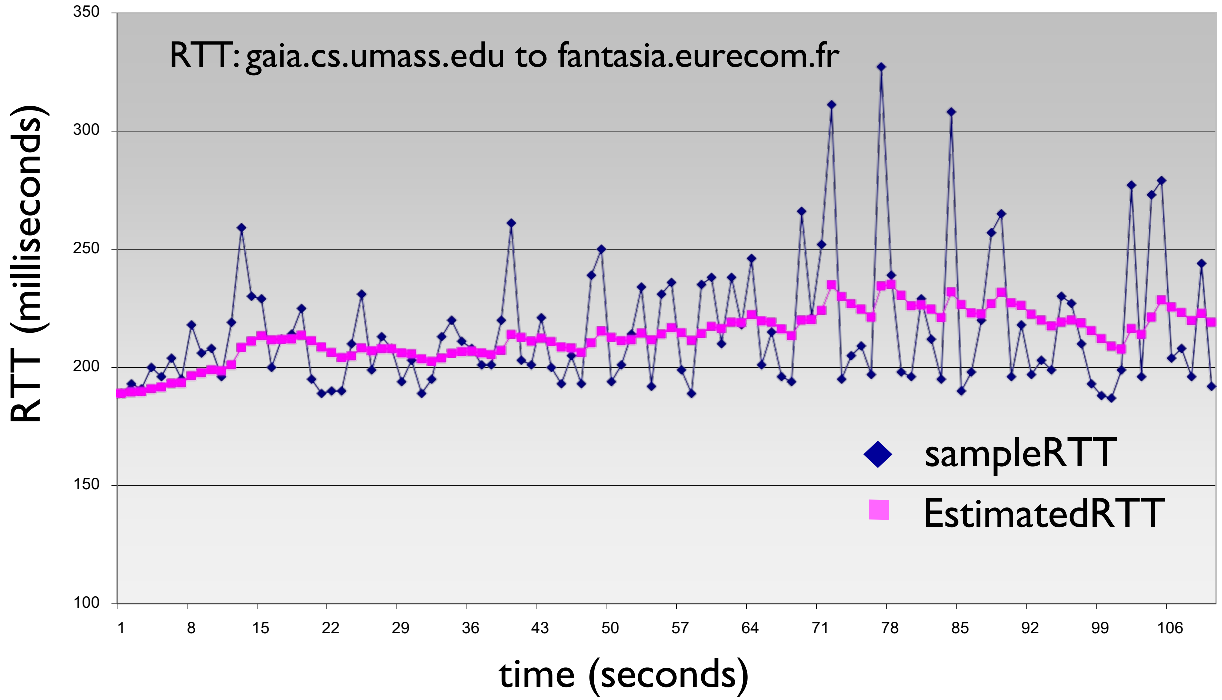
- too short: premature timeout, unnecessary retransmissions
- too long: slow reaction to segment loss
- It should be at least longer than RTT but RTT varies!
- TCP maintains timer for its oldest unACKed segment

TCP uses EWMA of Sample RTT plus safety margin

Estimate RTT uses EWMA to smooth out

$$EstimatedRTT_n = (1- a)*EstimatedRTT_{n-1} + a*SampleRTT_n$$

- exponential weighted moving average (EWMA)
- SampleRTT: measured time from segment transmission until ACK receipt
- influence of past sample decreases exponentially fast
- typical value: a = 0.125



In addition, safety margin is added

- timeout interval: EstimatedRTT plus "safety margin"
  - large variation in EstimatedRTT: want a larger safety margin

TimeoutInterval = EstimatedRTT + 4*DevRTT

estimated RTT

"safety margin"

- DevRTT: EWMA of SampleRTT deviation from EstimatedRTT:

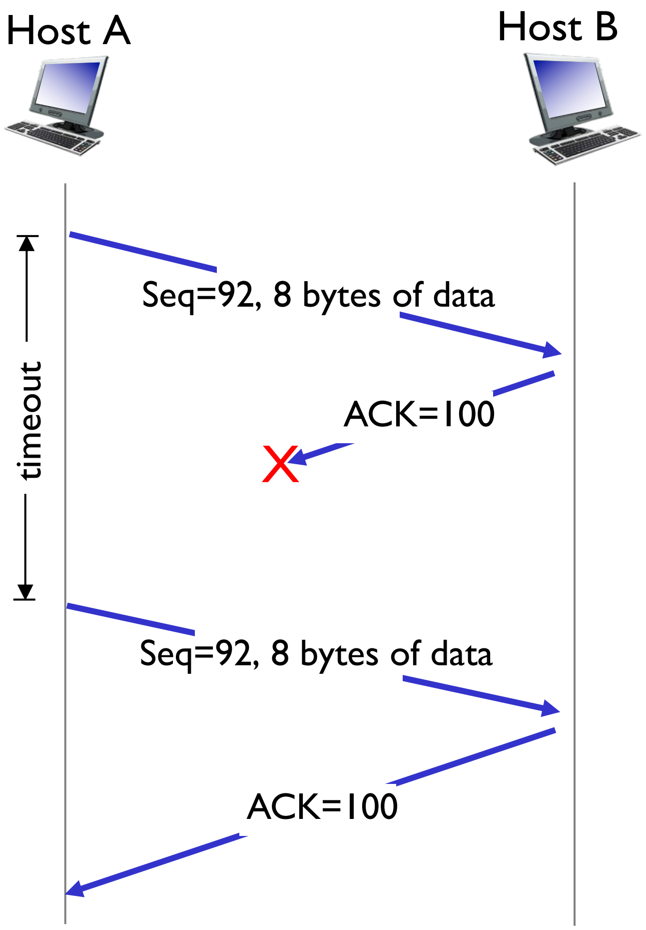
$\text{DevRTT}_n = (1-□)*\text{DevRTT}_{n-1} + □*|\text{SampleRTT}_n-\text{EstimatedRTT}_n|$

(typically, □ = 0.25)

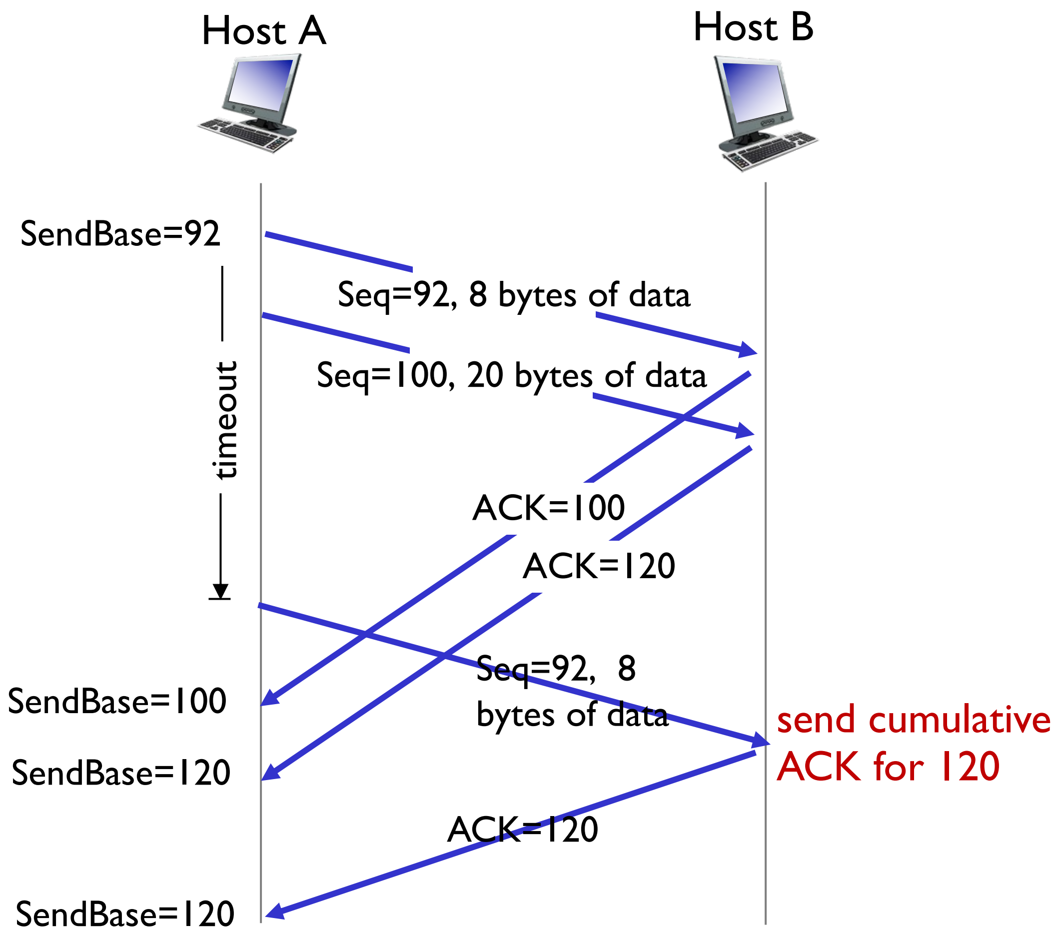
Outline

1. TCP overview
2. TCP timeout
3. TCP retransmissions

TCP: retransmission scenarios

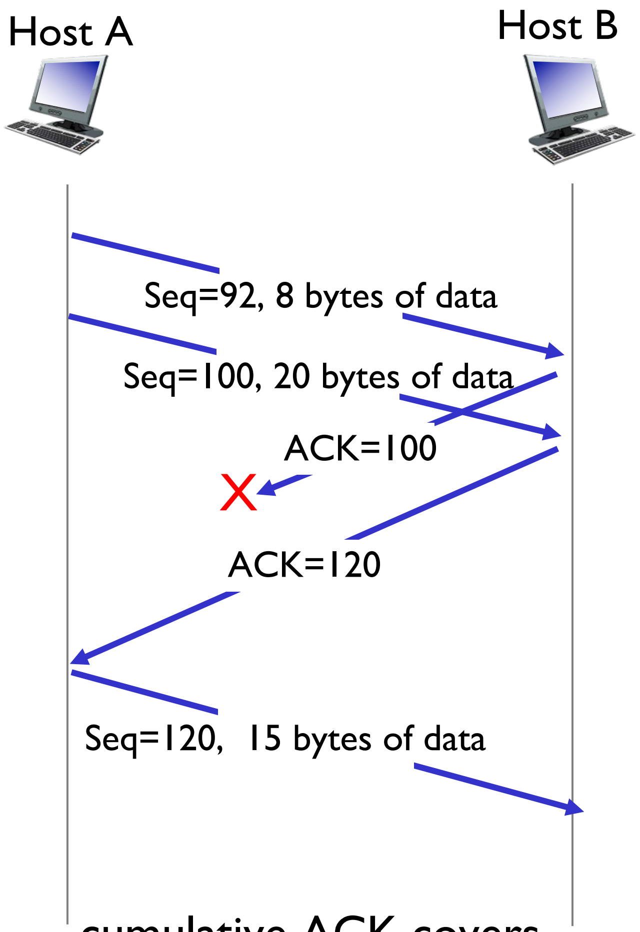


lost ACK scenario



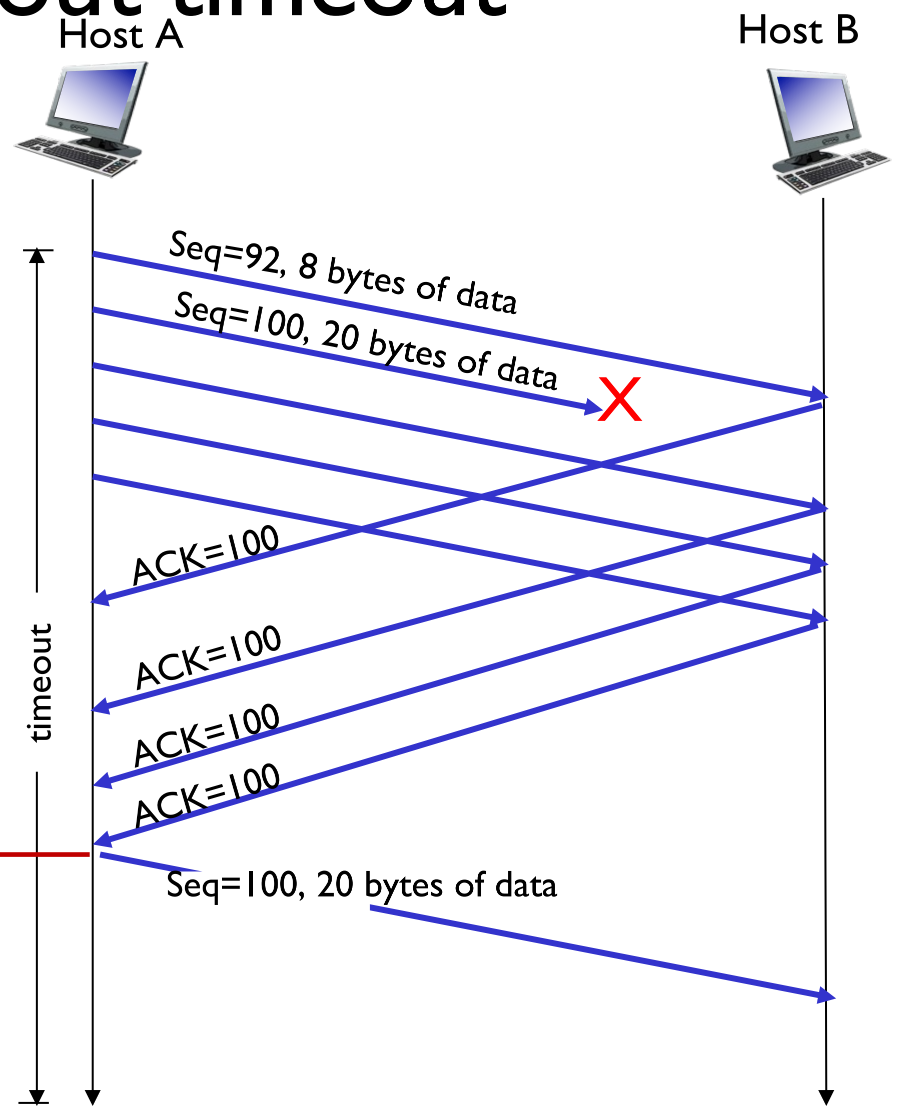
premature timeout

TCP: retransmission scenarios



cumulative ACK covers for earlier lost ACK

TCP fast retransmit: upon receiving triple dup ACKs immediately retransmit without timeout



Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!

Is it a good idea to retransmit as soon as possible?

- TCP assumes packet is lost upon timeout
- TCP assumes the packet is lost due to congestion

Doubles the timeout interval each time TCP retransmits upon timeout!

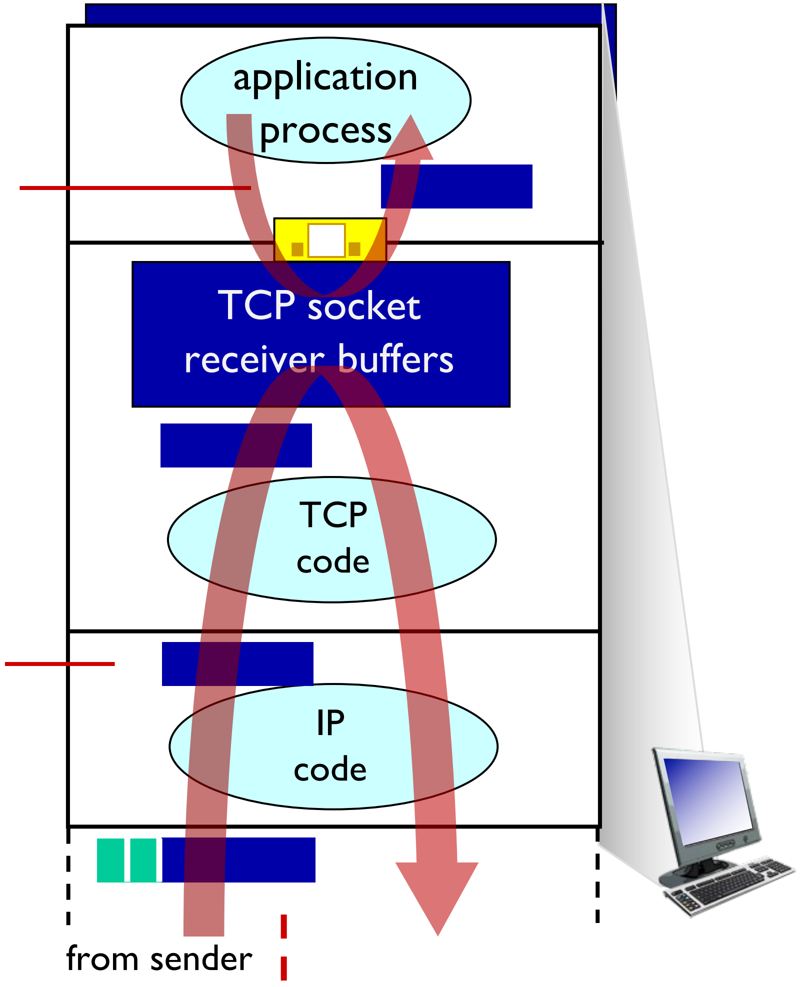
Outline

1. TCP overview
2. TCP timeout
3. TCP interesting scenarios
4. TCP flow control

What happens if network delivers faster than what application layer can process?

Application removing data from TCP socket buffers

Network layer delivering IP datagram payload into TCP socket buffers



receiver protocol stack

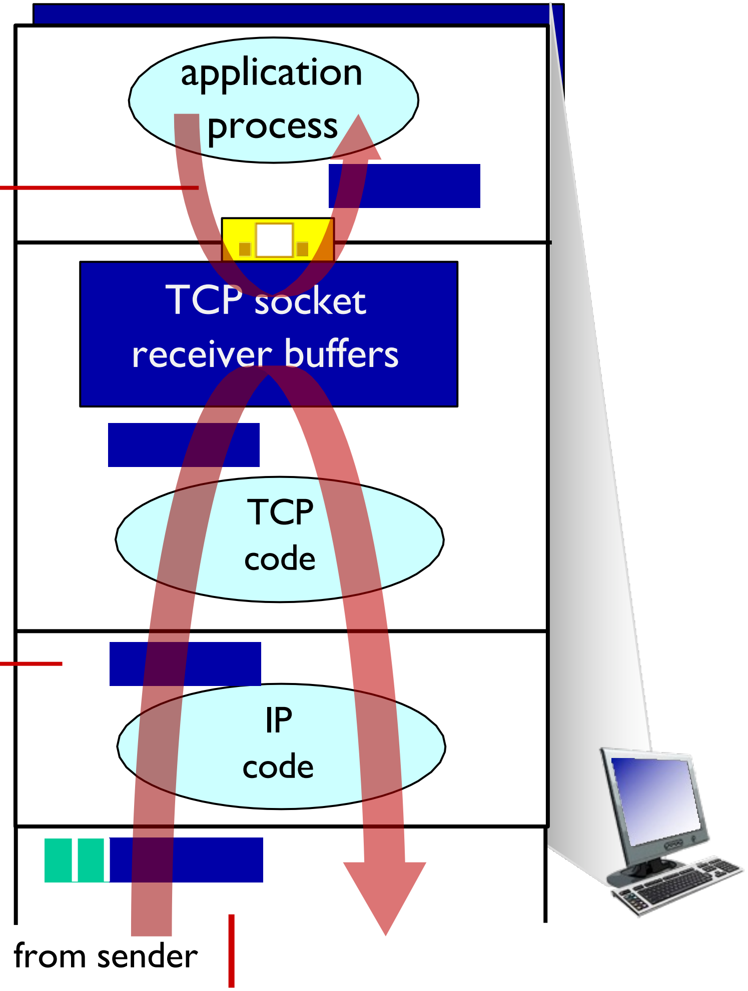
Loss happens when socket's recv buffer is full!

This loss is NOT due to network congestion

You are talking too fast!

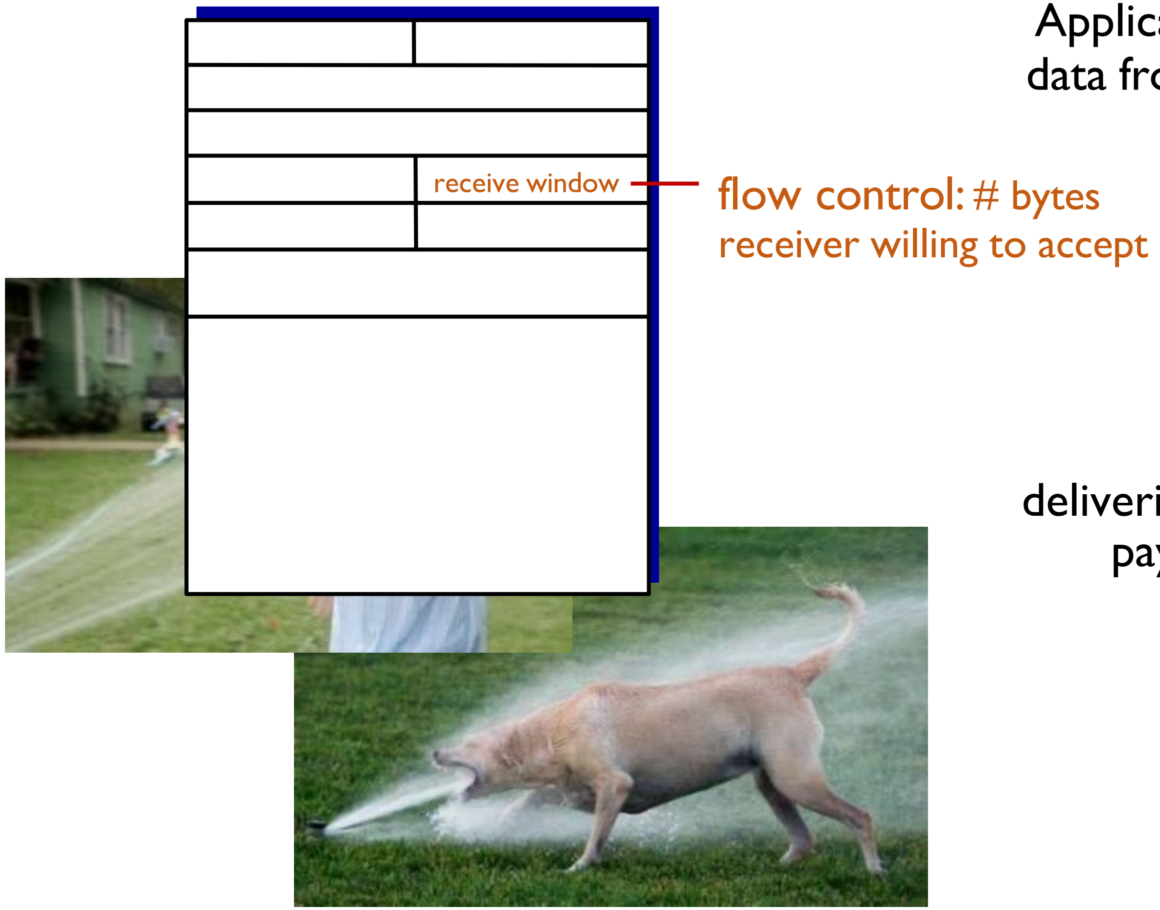
Application removing data from TCP socket buffers

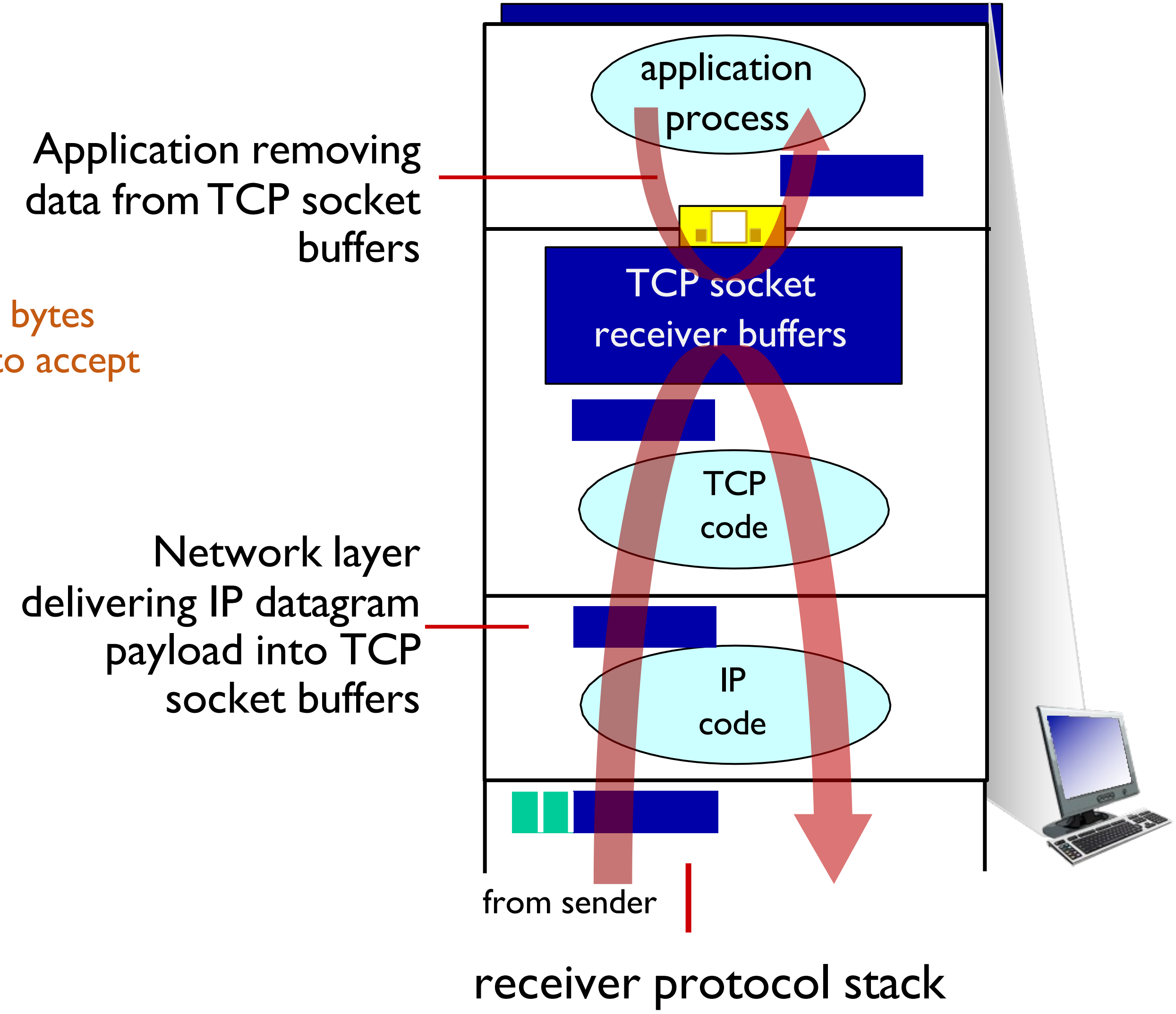
Network layer delivering IP datagram payload into TCP socket buffers



receiver protocol stack

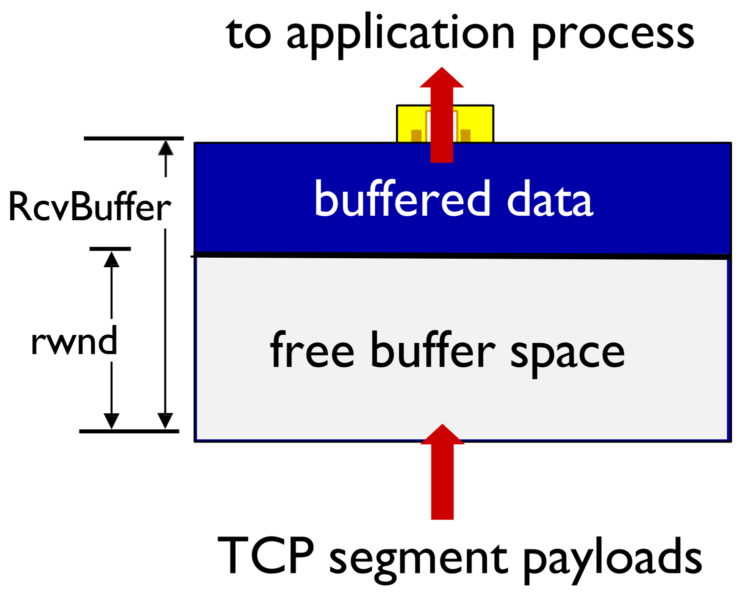
TCP flow control ensures NOT to overflow receiver socket buffer





TCP sender limits in-flight packets smaller than rwnd

- TCP receiver "advertises" free buffer space in rwnd field in TCP header
  - RcvBuffer size set via socket options (default 4096 bytes)



TCP receiver-side buffering

Guarantees receiver buffer will not overflow!

Outline

1. TCP overview
2. TCP timeout
3. TCP interesting scenarios
4. TCP flow control
5. TCP connection management

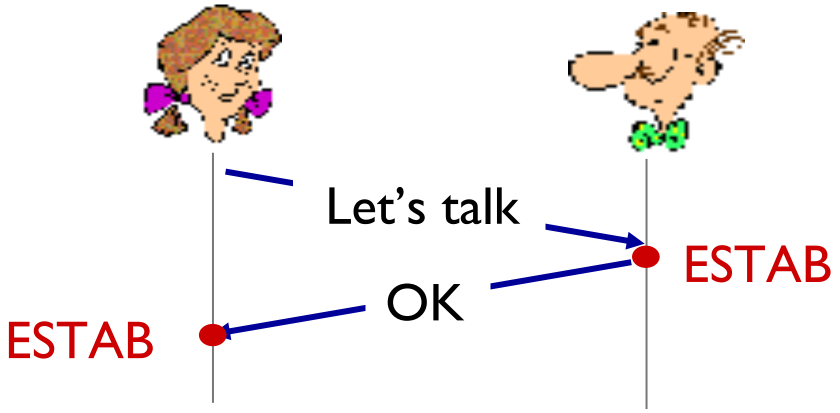
TCP has "handshake" prior to actual data exchange

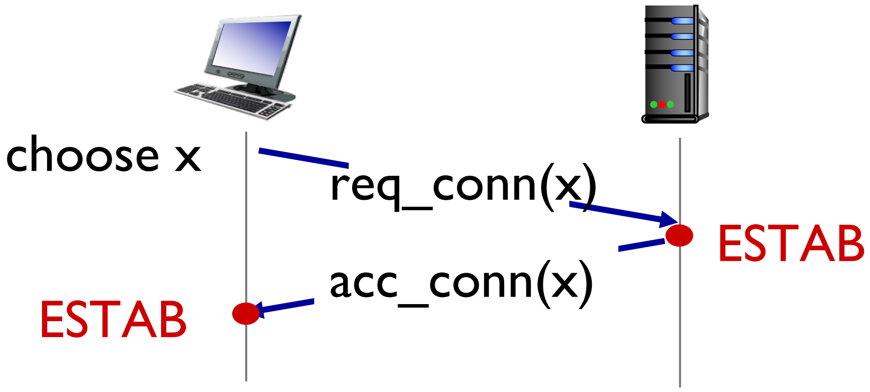
- agree to establish connection
- agree on connection parameters (e.g., starting seq #s, rwnds)

What were the two socket methods to perform this handshake?

Agreeing to establish a connection

2-way handshake:

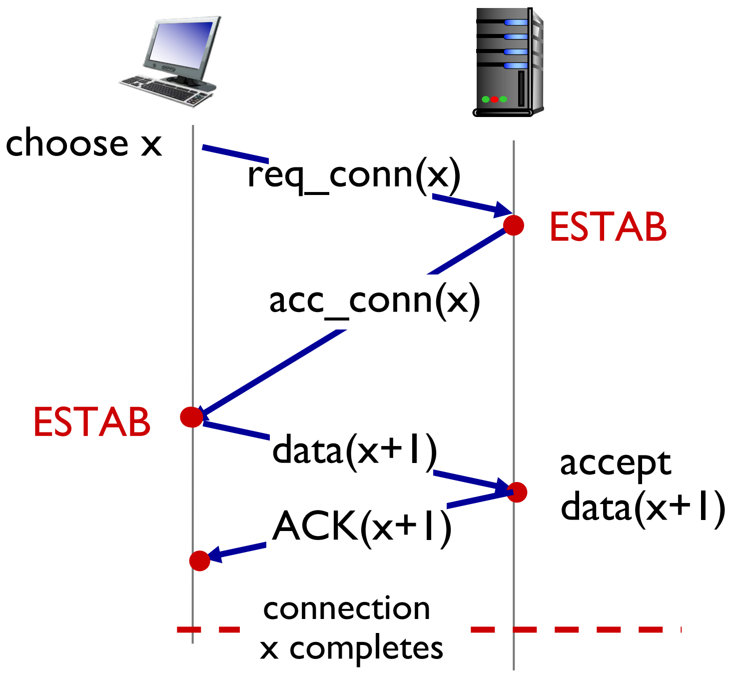




Q: will 2-way handshake always work in network?

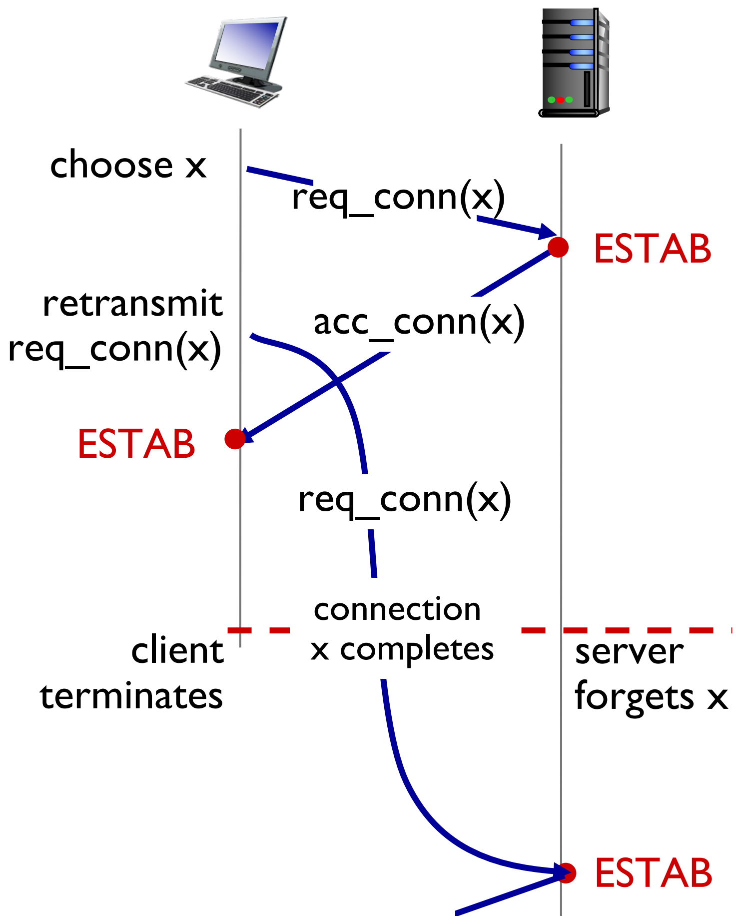
- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

2-way handshake scenarios



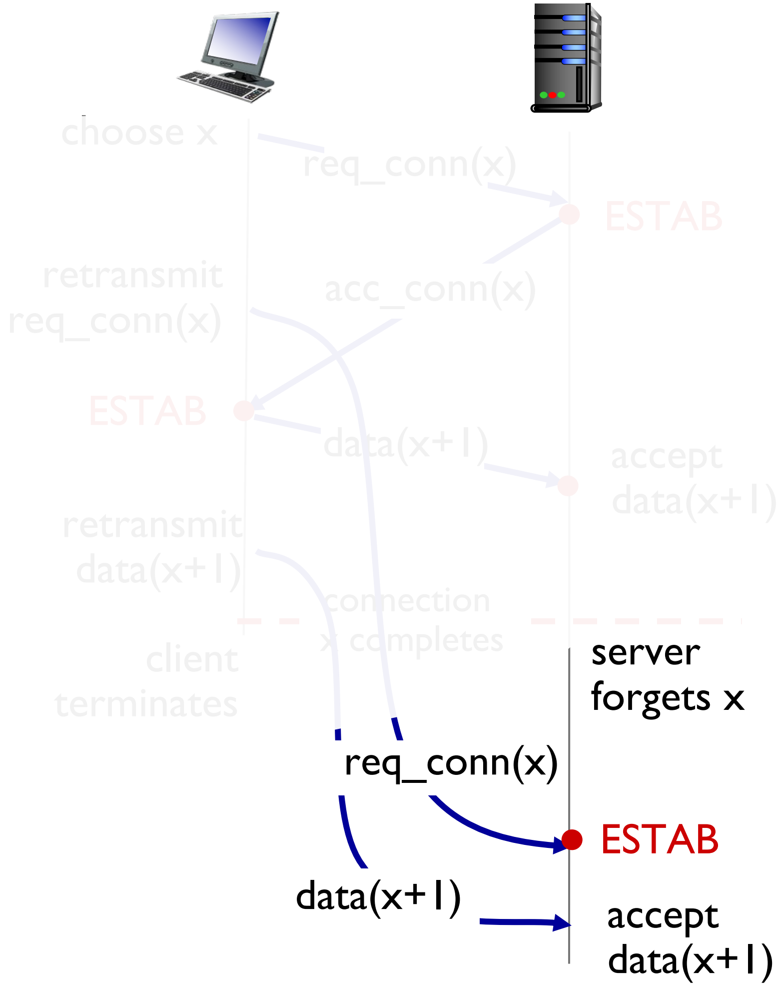
No problem!

2-way handshake scenarios



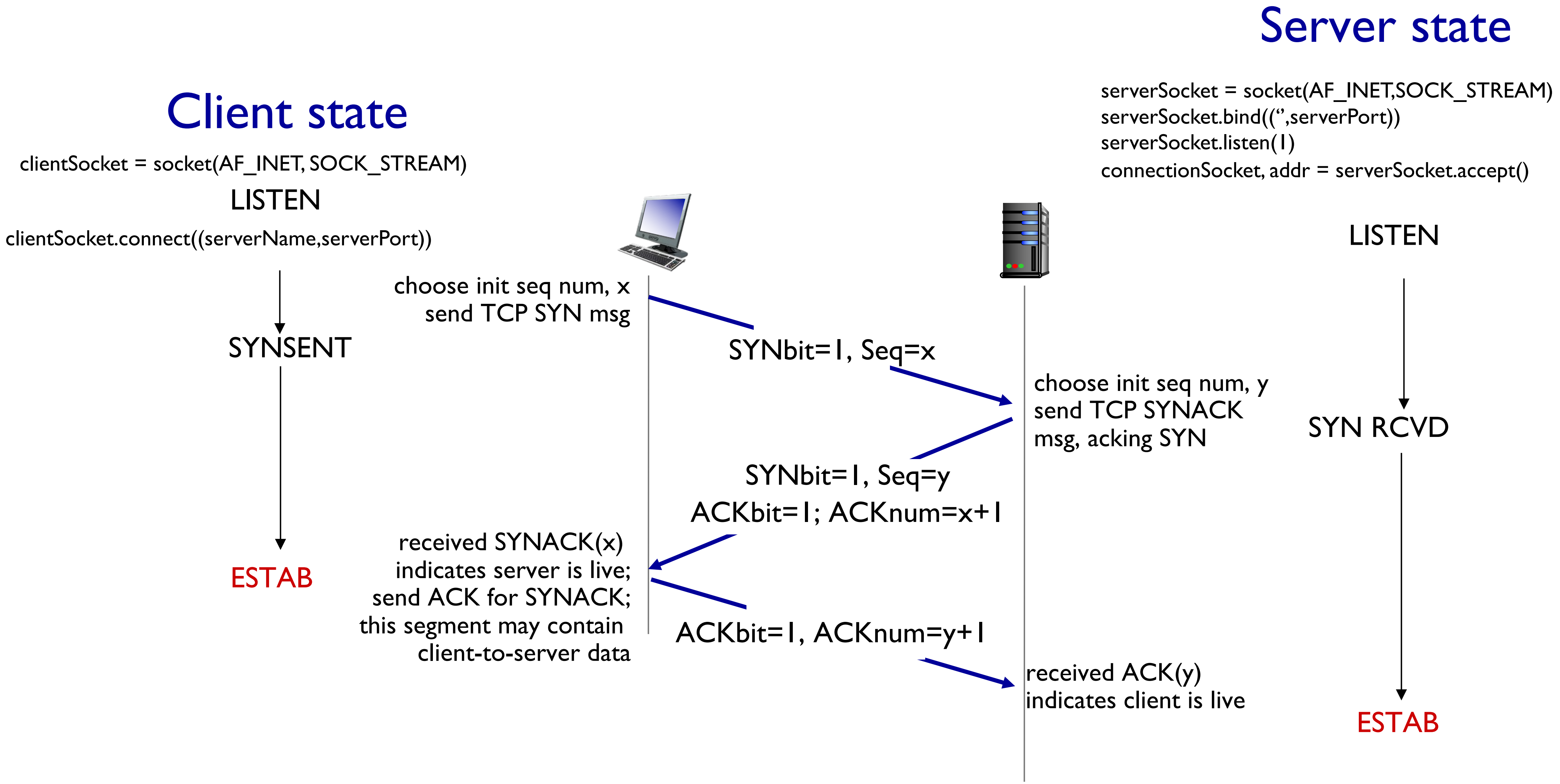
Problem: half open connection! (no client)

2-way handshake scenarios



Problem: dup data accepted!

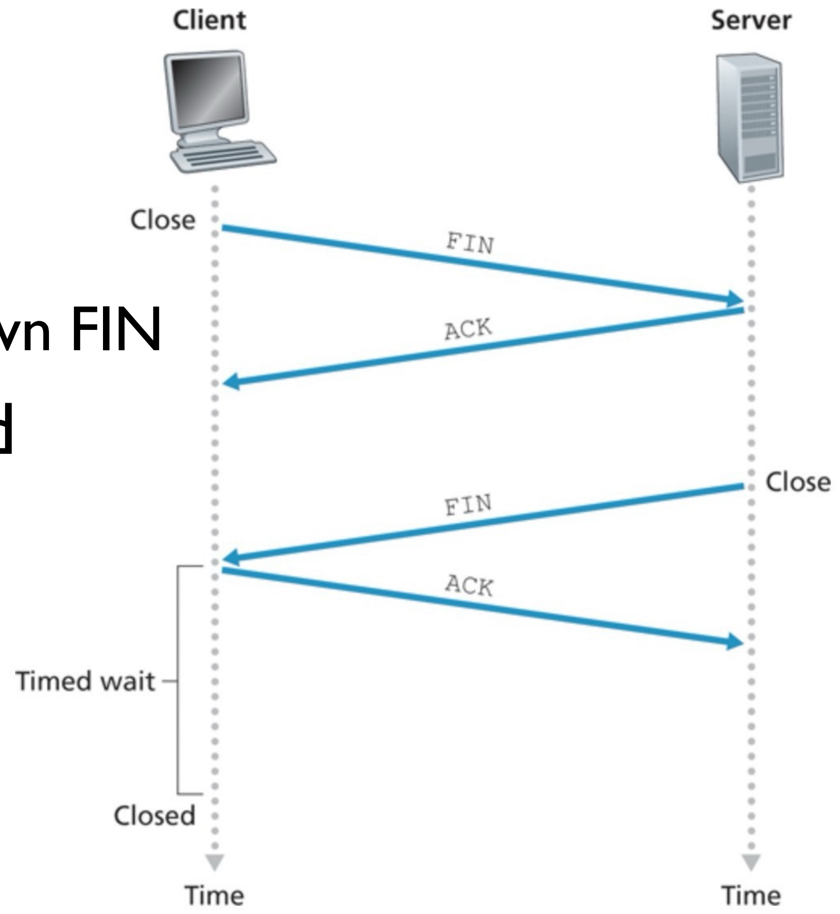
TCP 3-way handshake



A human 3-way handshake protocol

Closing a TCP connection

- Send TCP segment with FIN bit = 1
- respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN
- simultaneous FIN exchanges can be handled



Backup slides

Acknowledgements

Slides are adopted from Kurose’ Computer Networking Slides