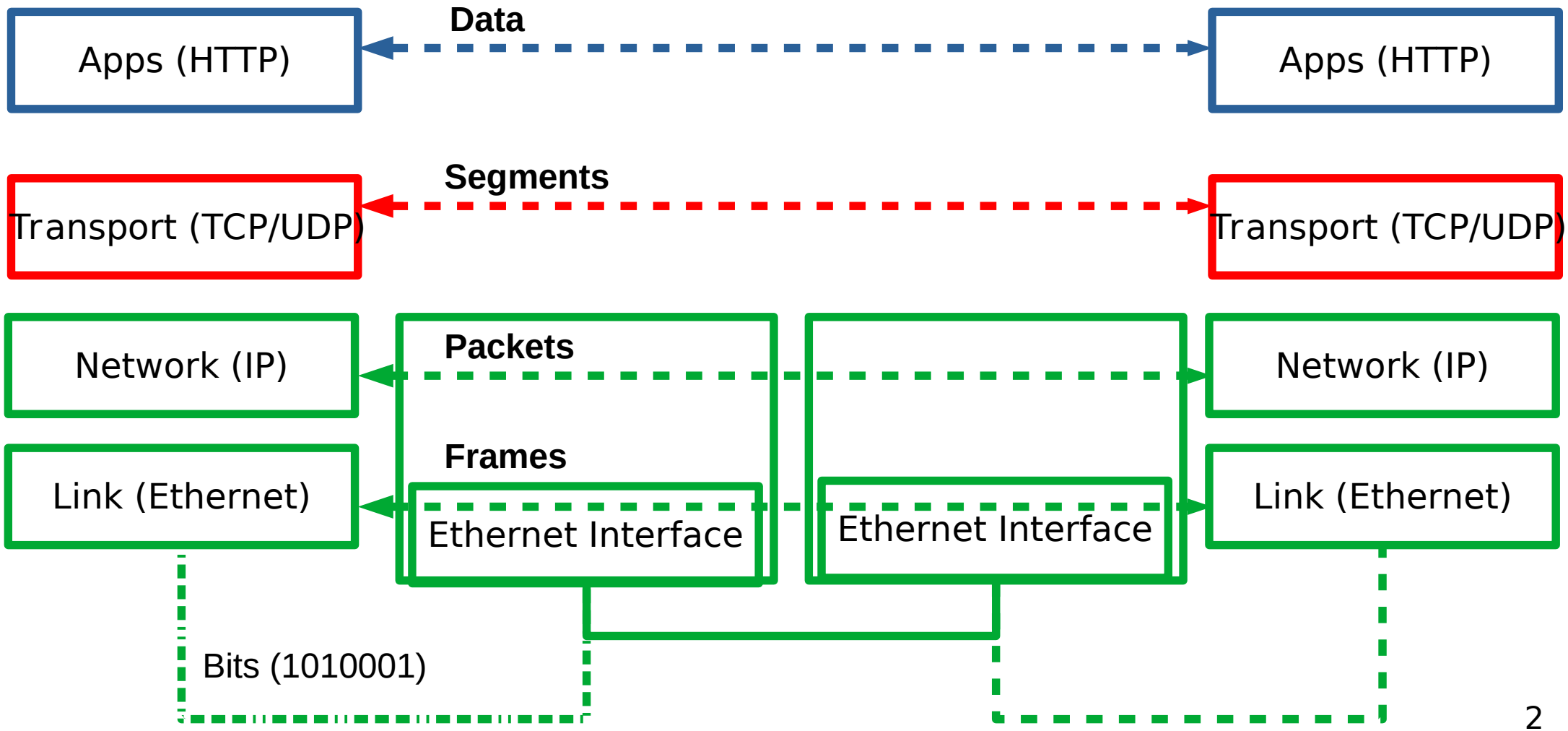
# CSC4200/5200 – COMPUTER NETWORKING

Instructor: Susmit Shannigrahi

## TRANSPORT LAYER PROTOCOLS

sshannigrahi@tntech.edu

Tennessee TECH

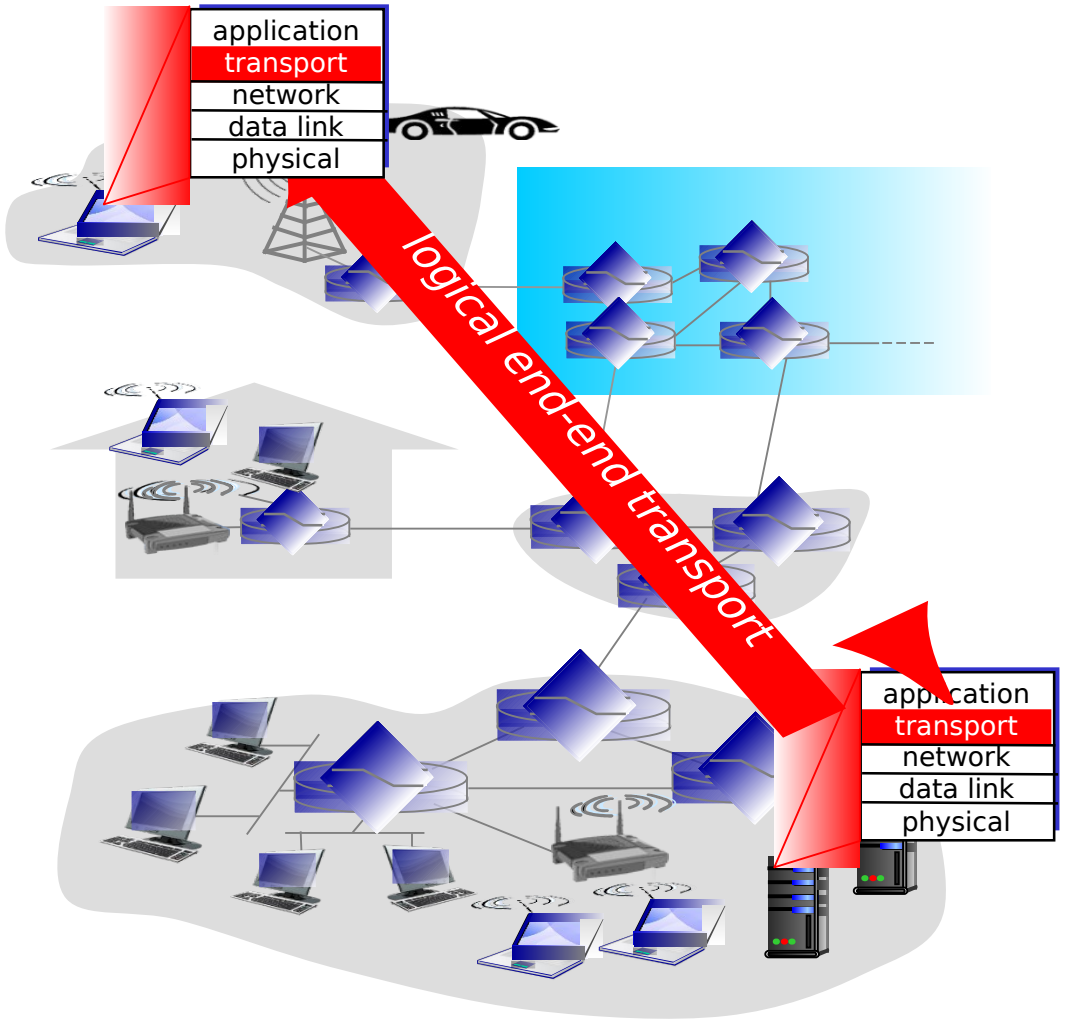

Apps (HTTP)
Data
Apps (HTTP)
Transport (TCP/UDP)
Segments
Transport (TCP/UDP)
Network (IP)
Packets
Network (IP)
Link (Ethernet)
Frames
Ethernet Interface
Ethernet Interface
Link (Ethernet)
Bits (1010001)

# What is transport layer?

- Problem:  How to turn this host-to-host packet delivery service into a process-to-process communication channel?

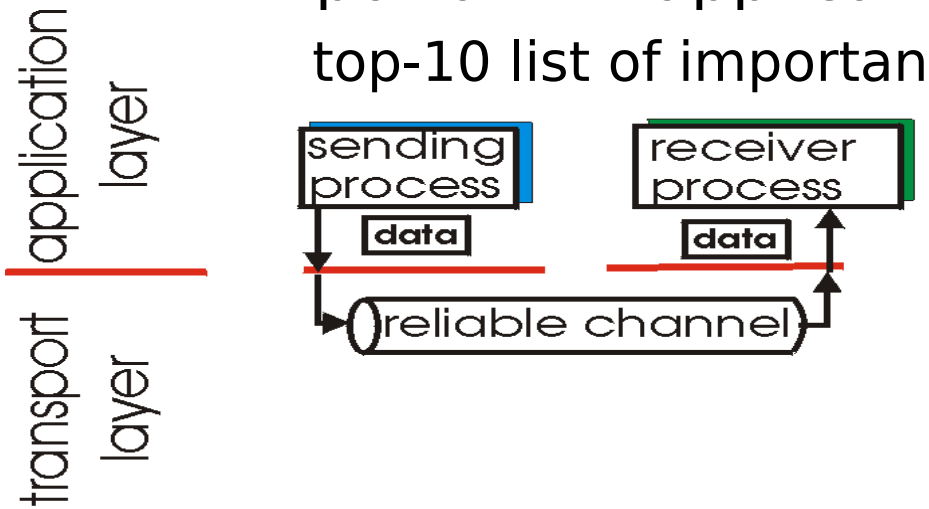
# Transport services and protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into *segments*, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



# Principles of reliable data transfer

- important in application, transport, link layers
  - top-10 list of important networking topics!

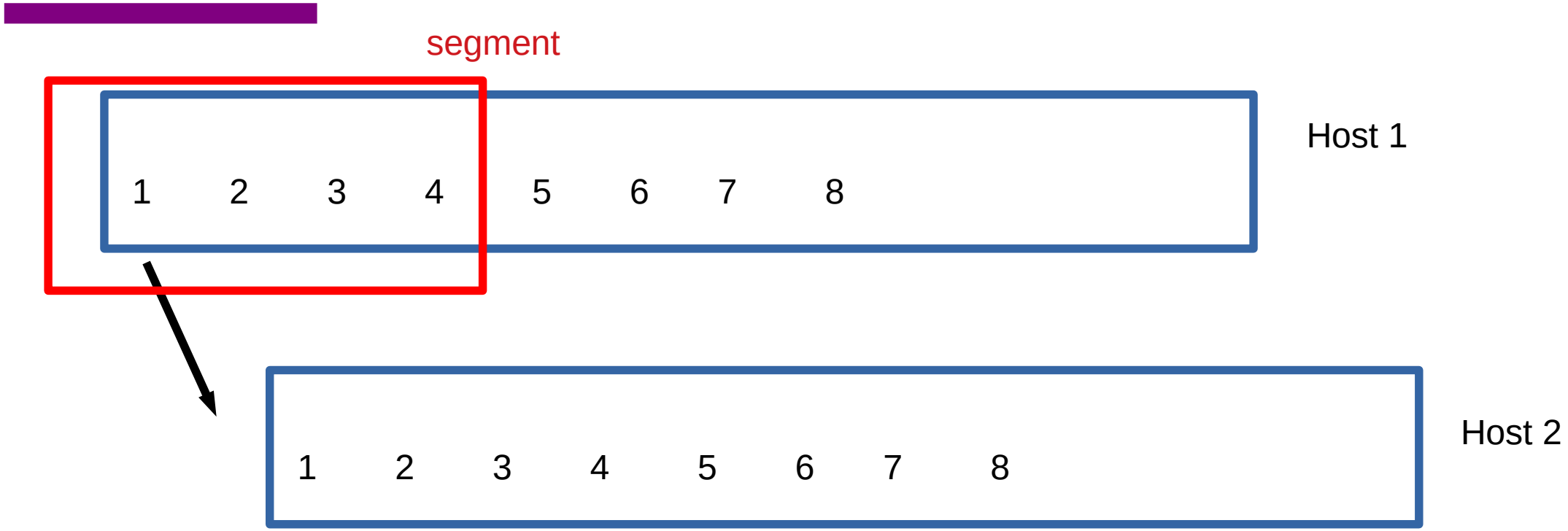



(a) provided service

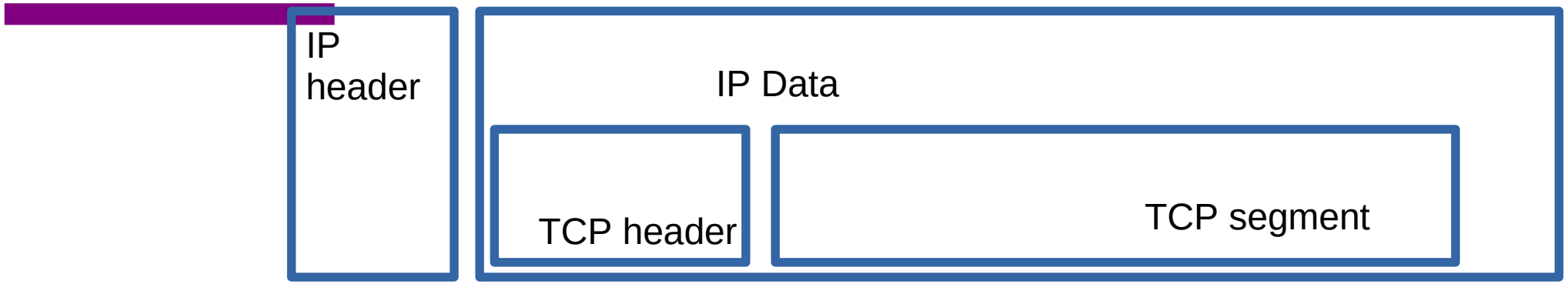
# TCP – Transmission Control Protocol

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - 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 – Transmission Control Protocol



# TCP Segment




IP → No more than MTU (1500 Bytes)

TCP header → 20 bytes

TCP segment → 1460 bytes

Why?

# TCP Header

SYN

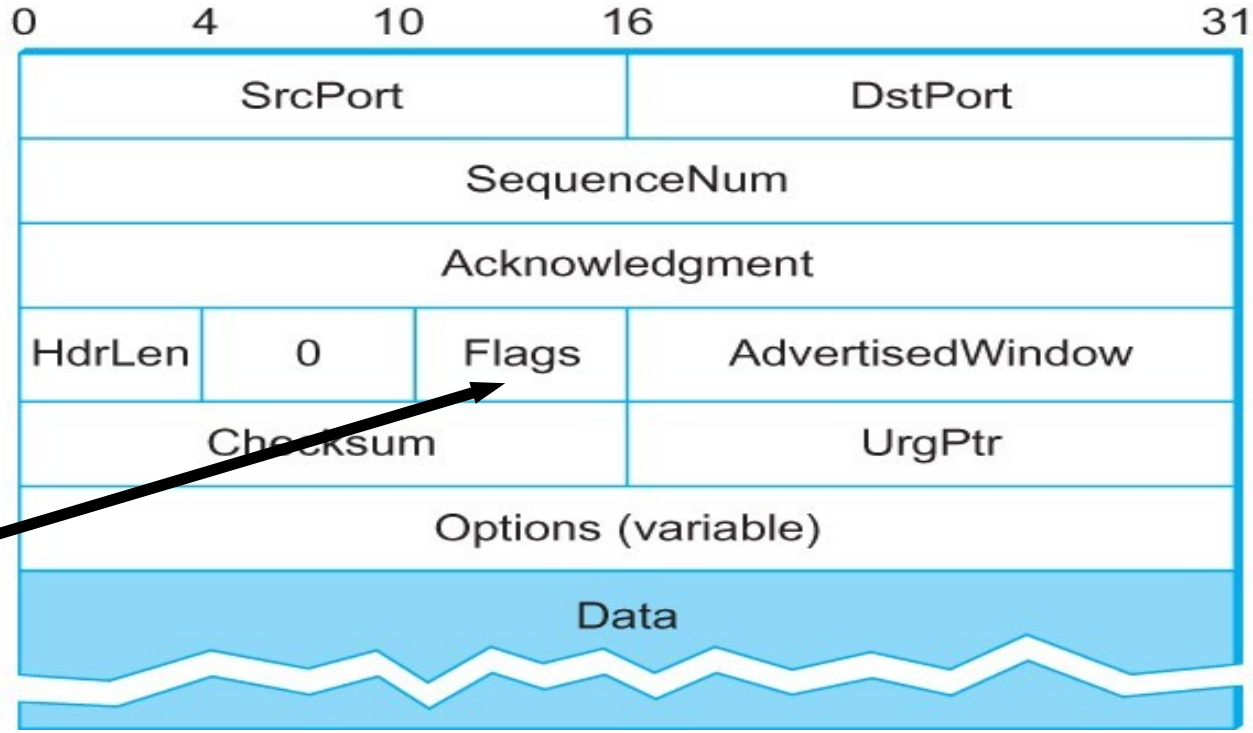
FIN

RST

PSH

URG

ACK

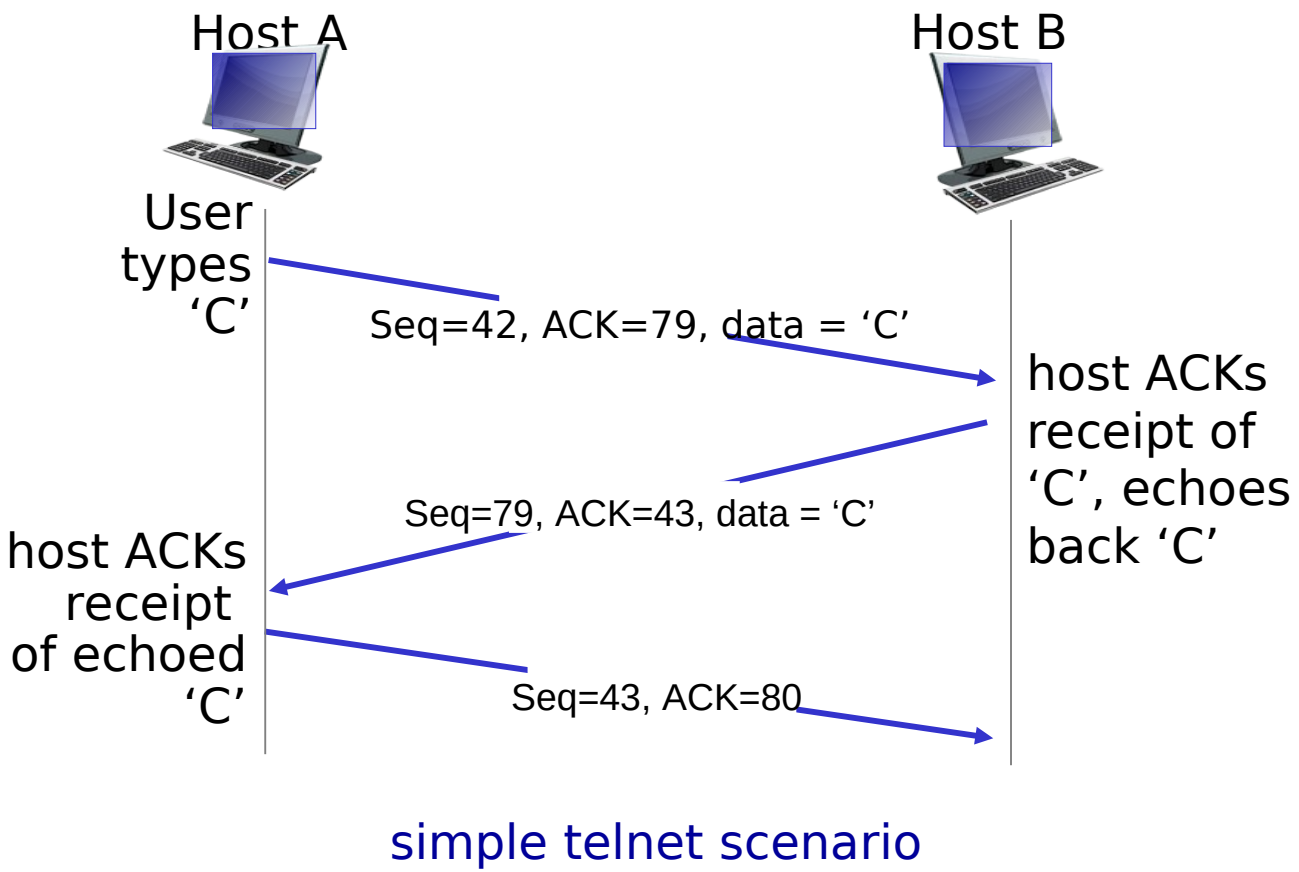



TCP Header Format

# TCP – Transmission Control Protocol

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - 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 seq. numbers, ISNs




simple telnet scenario

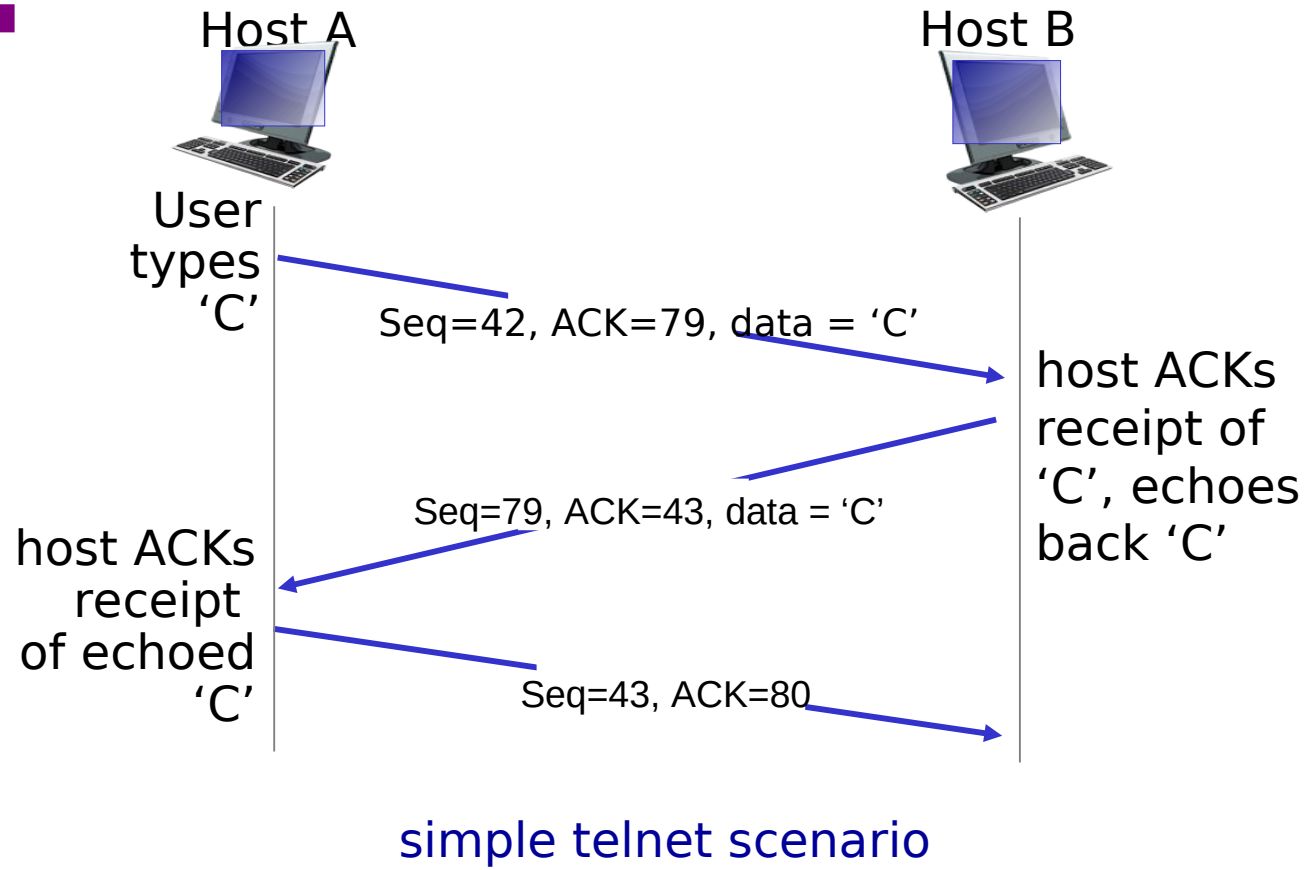
Sequence number for the first byte

Why not use 0 all the time?

- Security
- Port are reused, you might end up using someone else's previous connection
- Phone number analogy

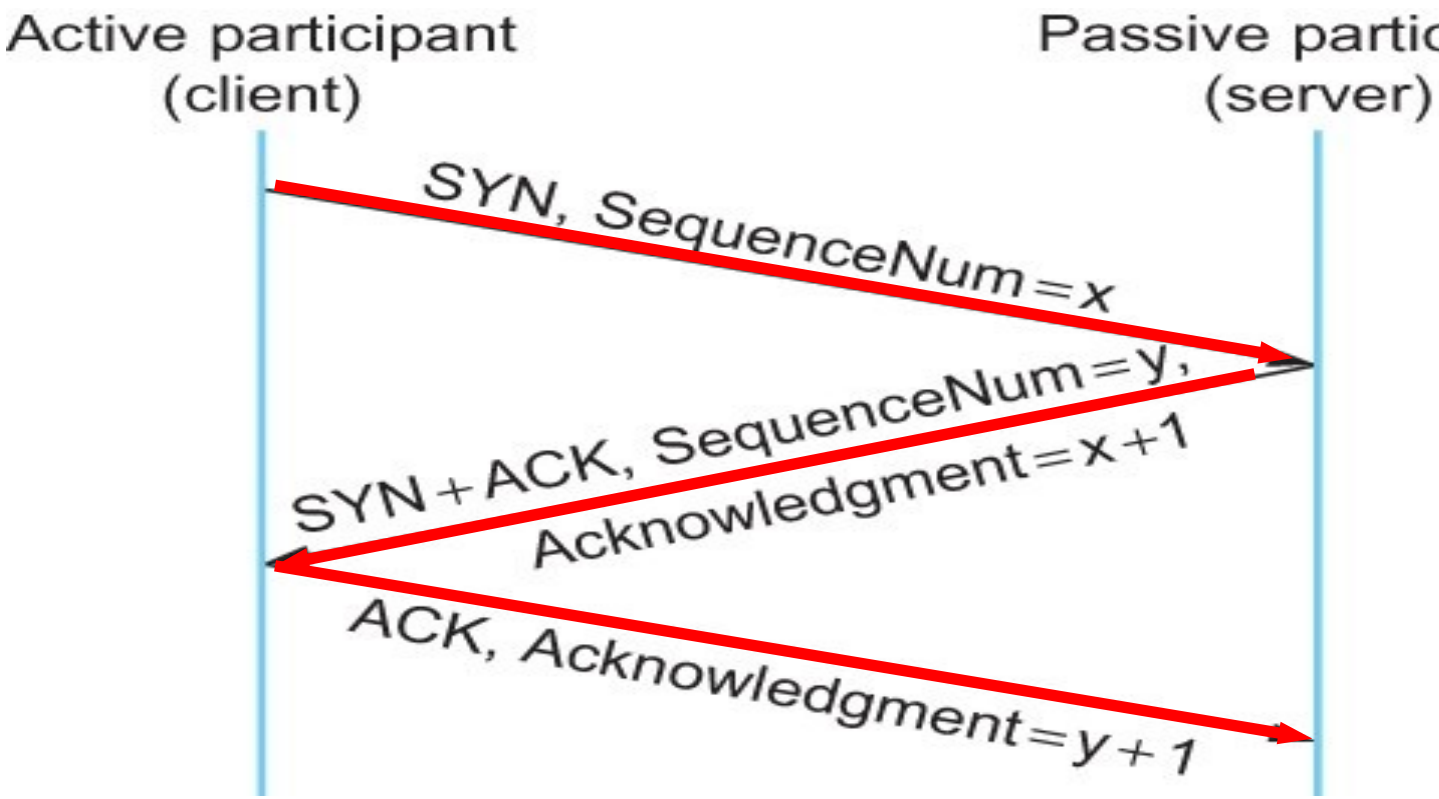
- TCP ISNs are clock based
  - 32 bits, increments in 4 microseconds
  - 4.55 hours wrap around time

# TCP seq. numbers, ACKs




simple telnet scenario

# TCP Three-way Handshake




Timeline for three-way handshake algorithm

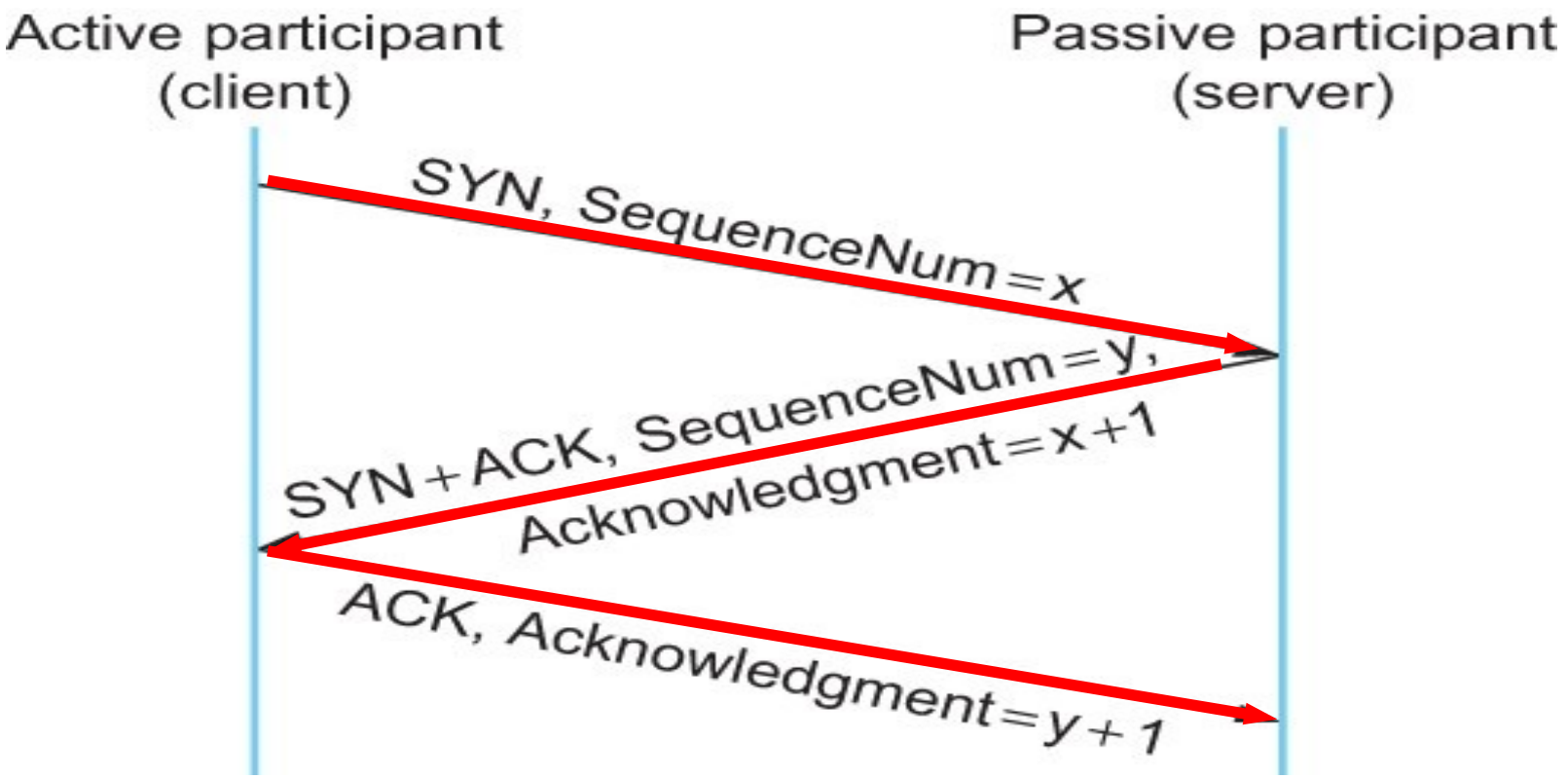
The idea is to tell each other The ISNs

SYN → Client tells server that it wants to open a connection, Client's ISN = x

SYN+ ACK → Server tells Client → Okay → Server's ISN = y, ACK = CLSeq + 1

Why increment by 1?

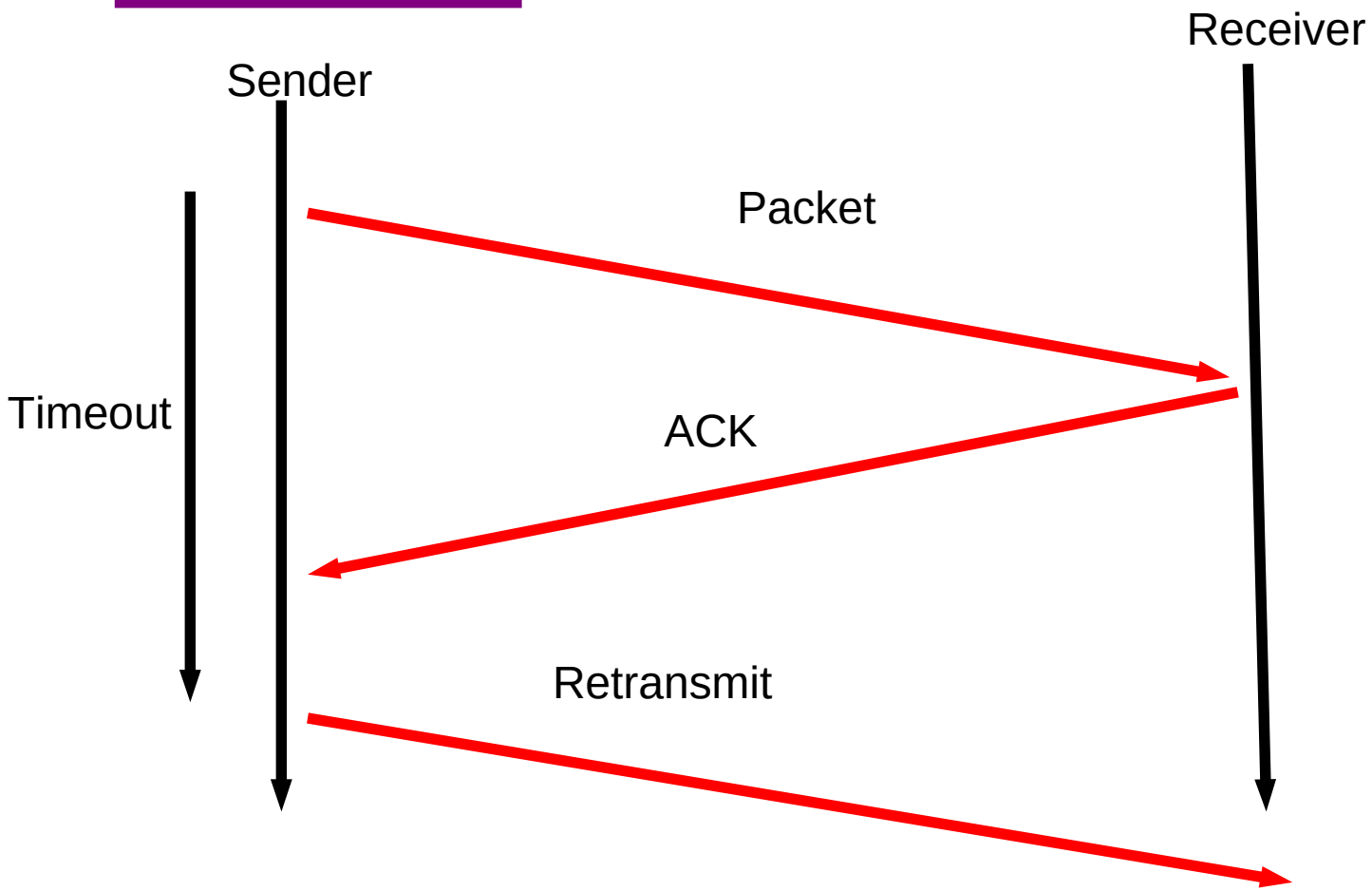
# What if the SYN is lost?




Start Timer and resend

Timeline for three-way handshake algorithm

# TCP Retransmission - ARQ

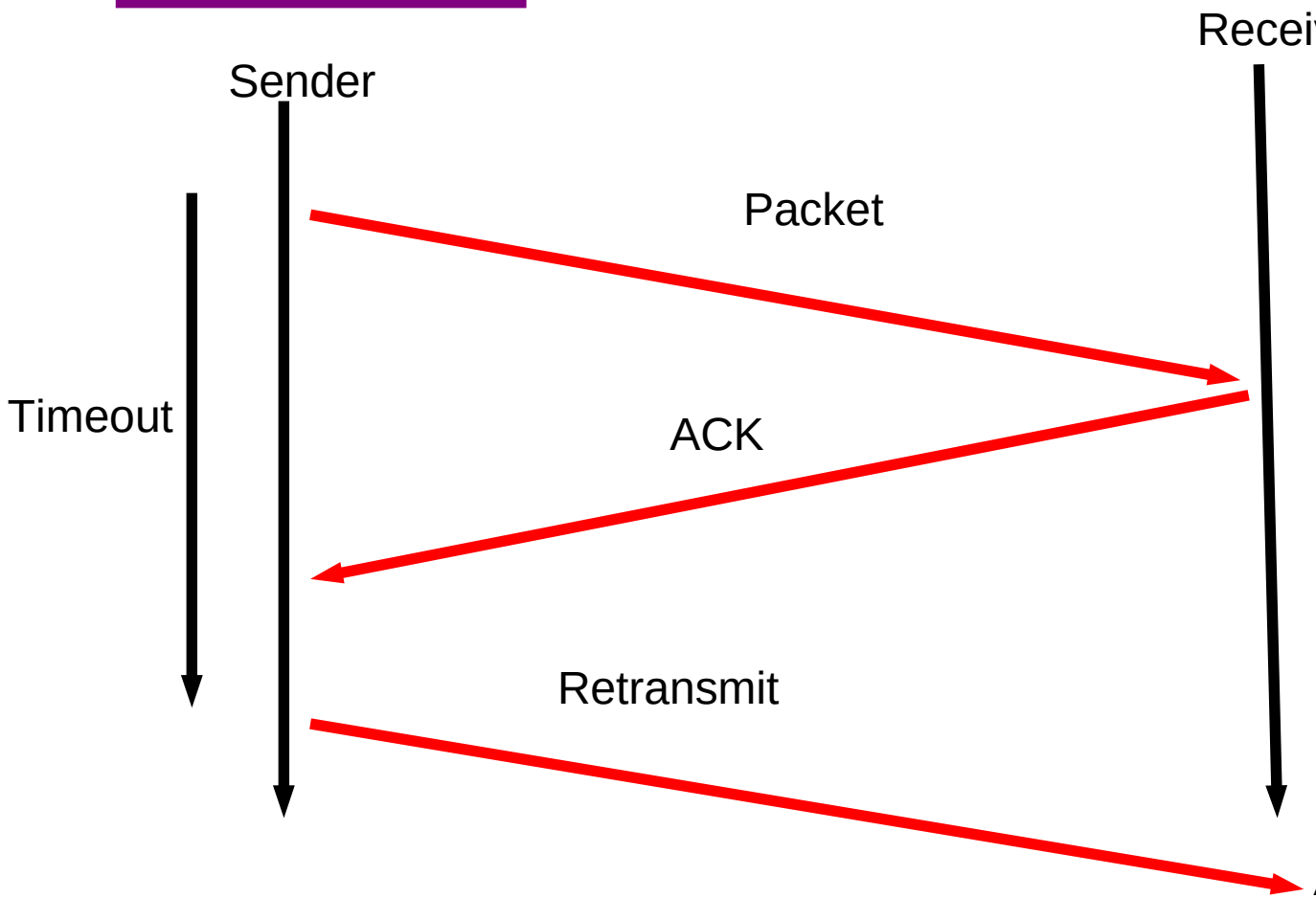



Each packet is “ACK”ed by the receiver

If ACK isn’t received by timeout, resend

Example, Stop-n-wait

# How long should the sender wait?




For TCP, timeout is a function of RTT

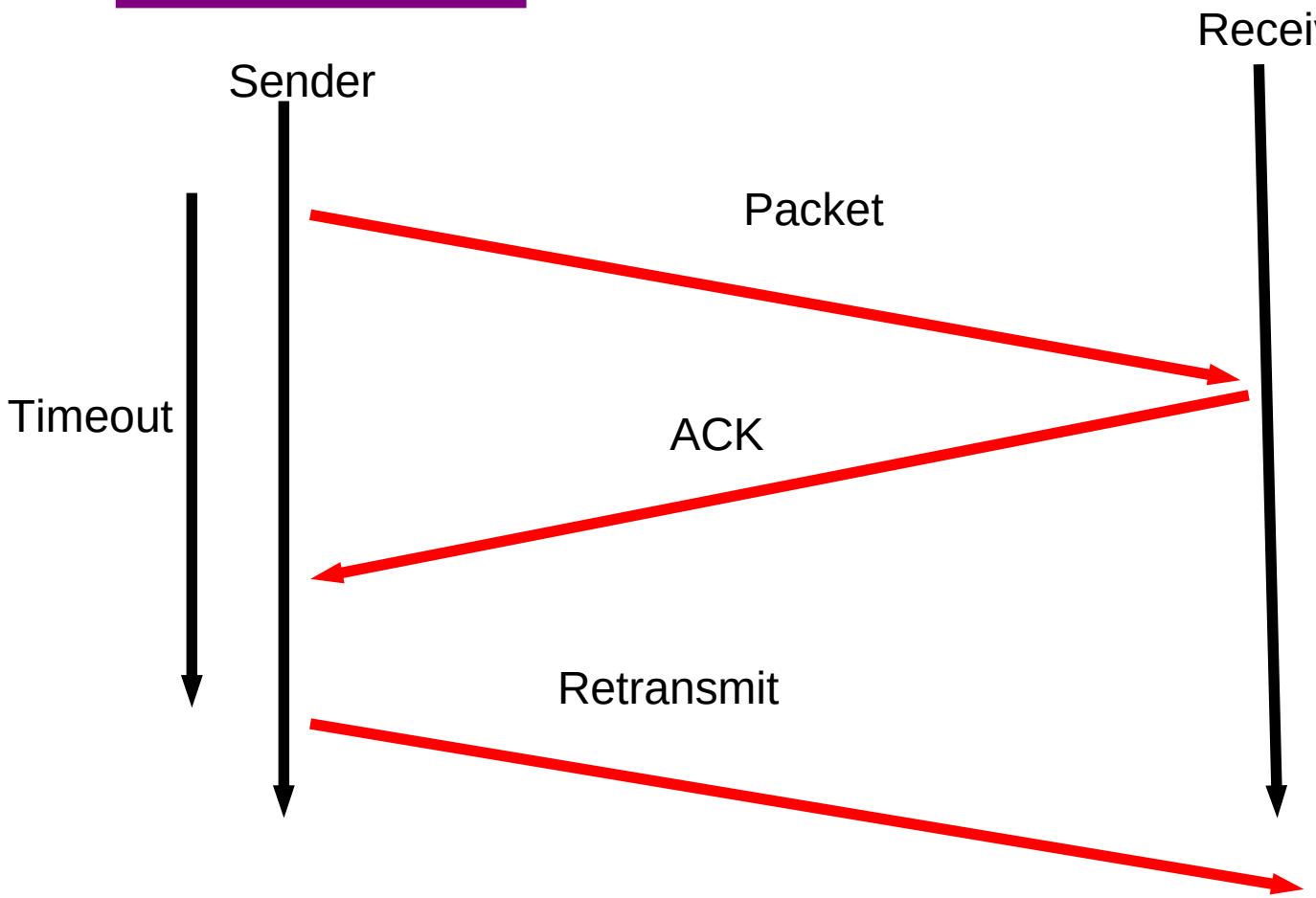
Keep a running estimate of RTTs by watching the ACKs

EstimatedRTT = (1 – α) • EstimatedRTT
+ α • SampleRTT

Alpha is generally 0.125

Timeout = 2* EstimatedRTT

# But stop and wait is inefficient




Only one segment in-flight

Especially bad if delay is high!

Solution – sliding window

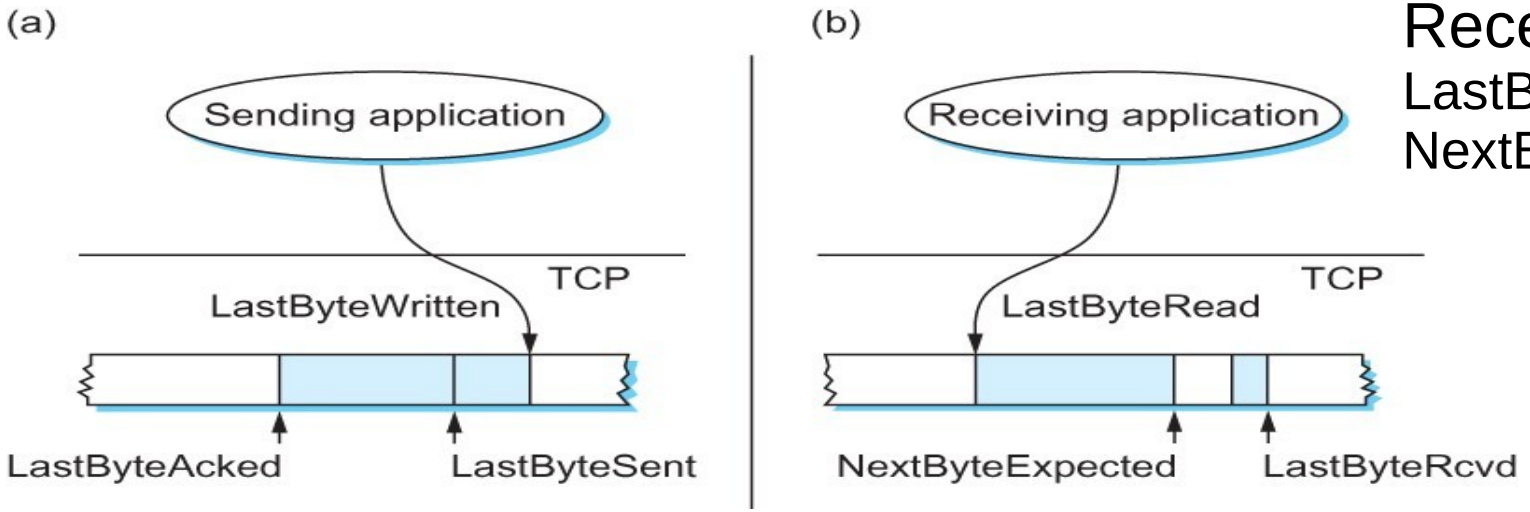
# Sliding Window Revisited

## Sending Side

LastByteAcked ≤ LastByteSent
LastByteSent ≤ LastByteWritten

## Receiving Side

LastByteRead < NextByteExpected
NextByteExpected ≤ LastByteRcvd + 1

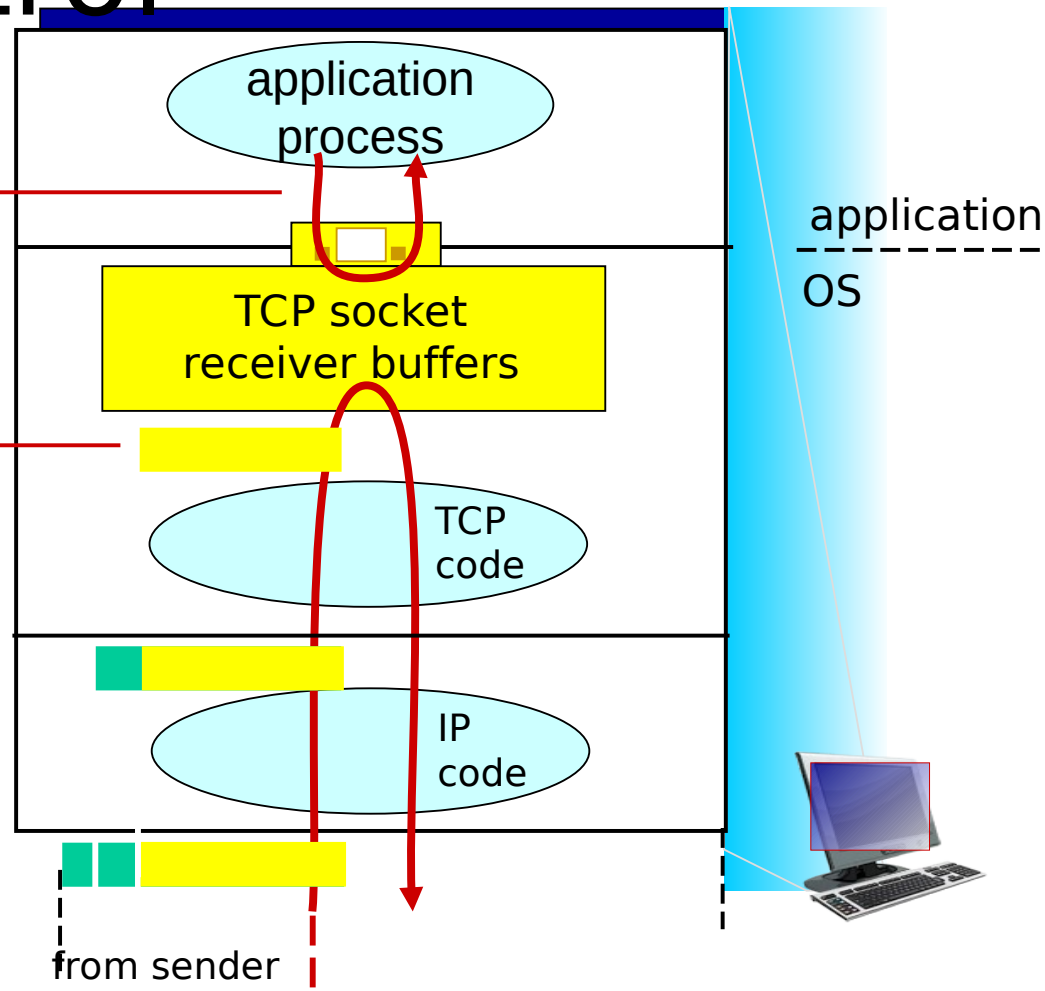



Relationship between TCP send buffer (a) and receive buffer (b).

# Used for TCP flow control

application may remove data from TCP socket buffers ....

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



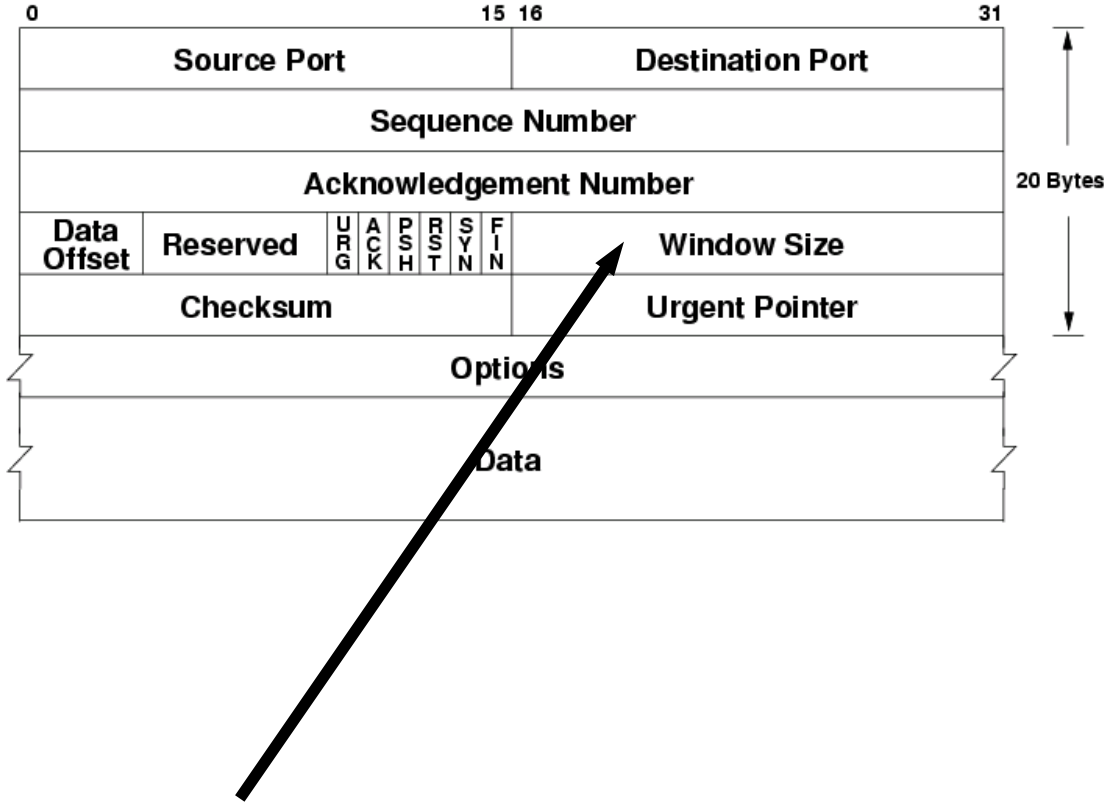

receiver protocol stack

*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 in the header
- sender limits amount of unacked (“in-flight”) data to receiver’s **`rwnd`** value
- guarantees receive buffer will not overflow

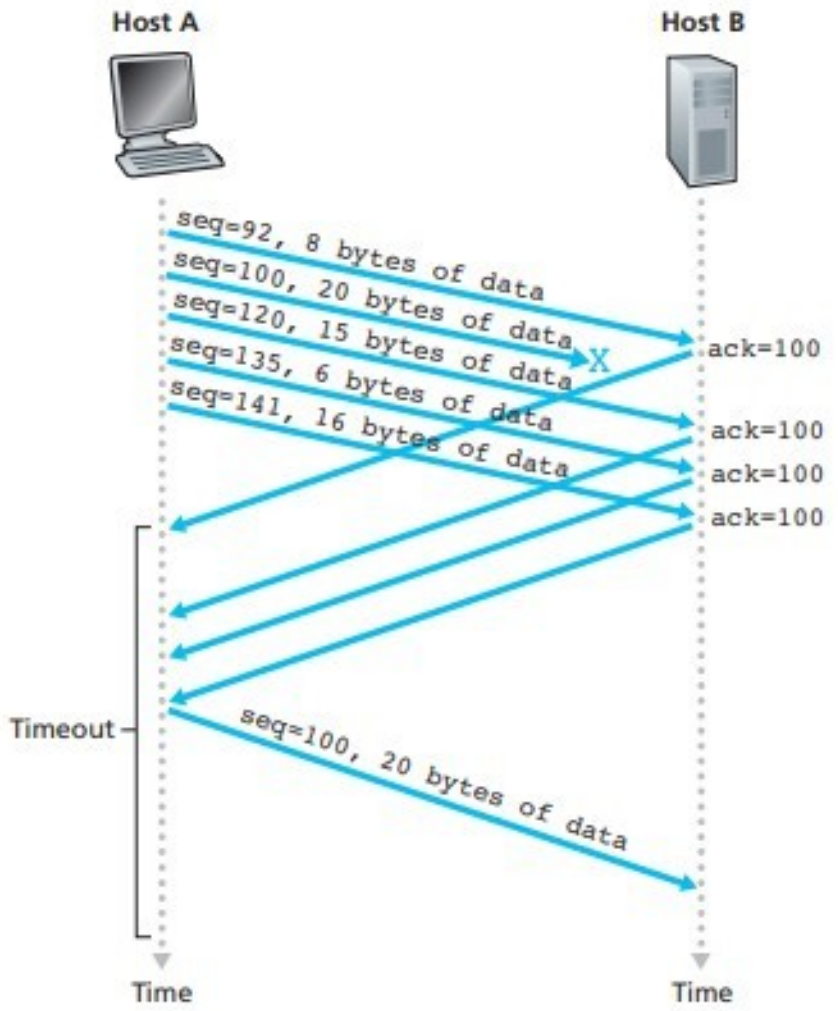


# TCP Fast Retransmission

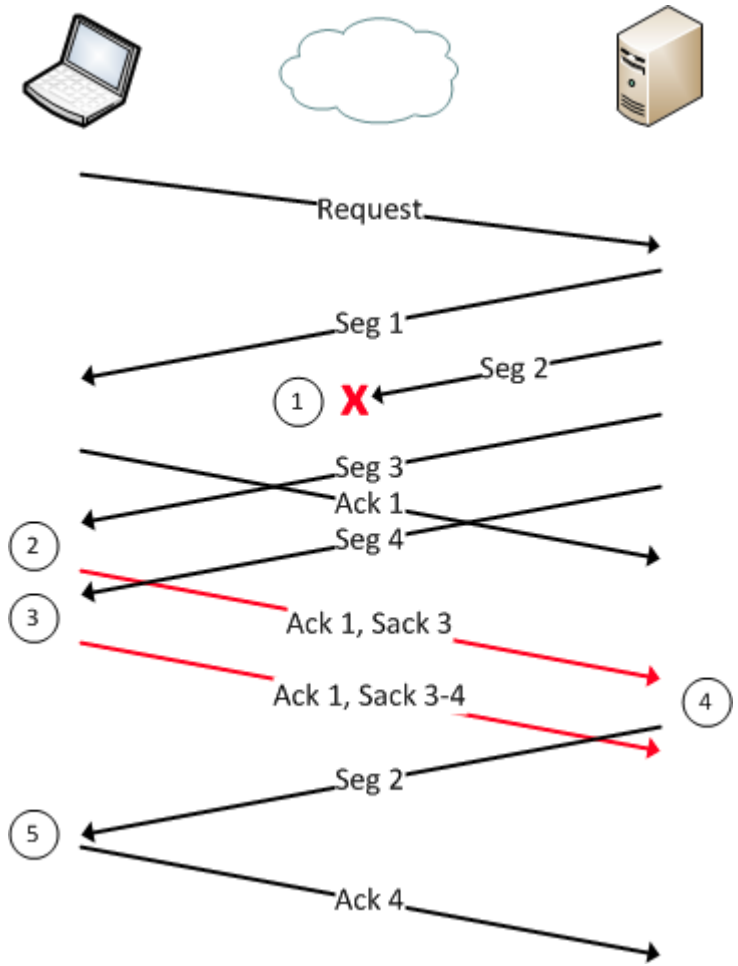
Timeouts are wasteful

Triple duplicate ACKs

Retransmits before timeout



# TCP Fast Retransmission - SACK



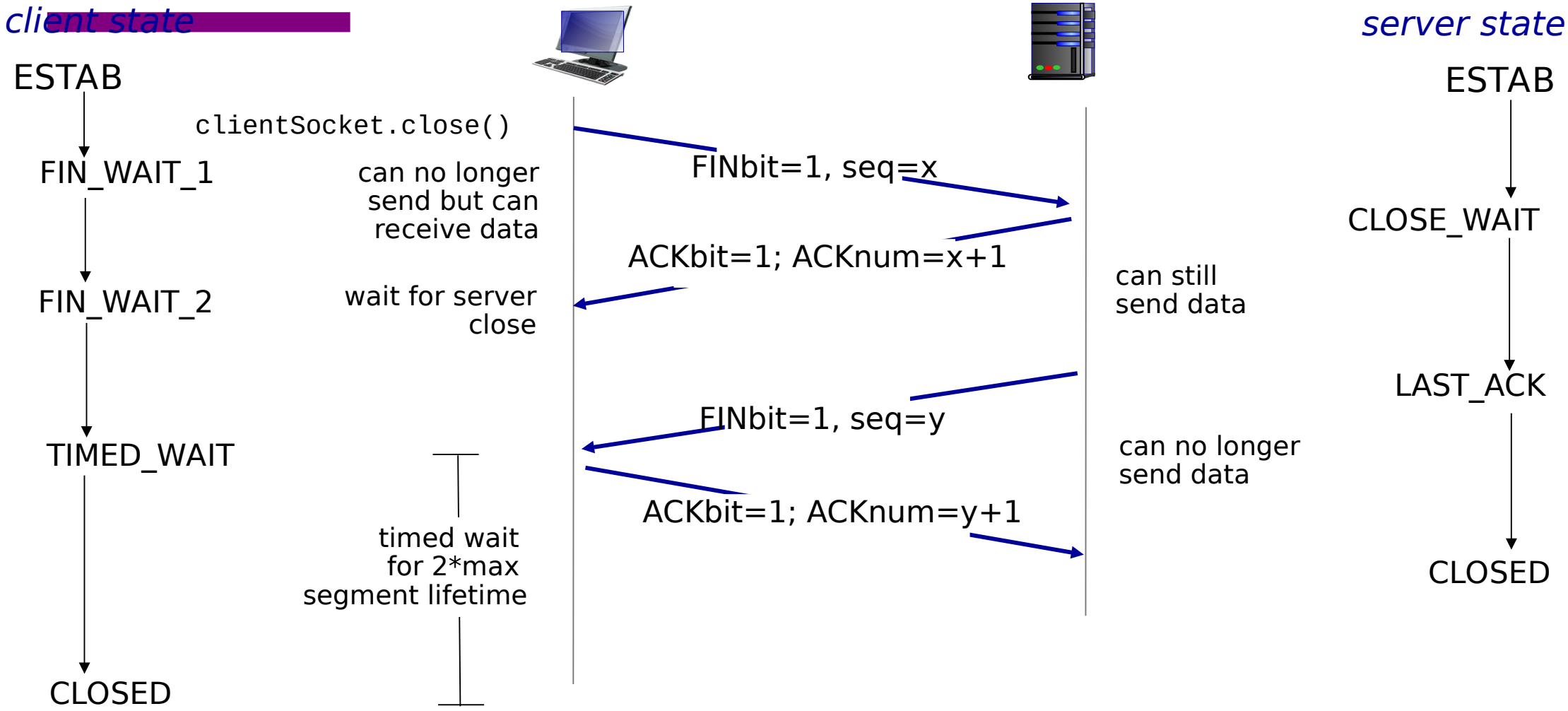

What if multiple segments are lost?

Very good explanation:
https://packetlife.net/blog/2010/jun/17/tcp-selective-acknowledgments-sack/

# TCP: closing a connection

- client, server each close their side of 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

# TCP: closing a connection



# Reading

https://book.systemsapproach.org/e2e/tcp.html#segment-format
https://book.systemsapproach.org/e2e/tcp.html#connection-establishment-and-termination

https://book.systemsapproach.org/e2e/tcp.html#sliding-window-revisited