Lecture 12: Transport Layer
TCP again

COMP 332, Spring 2018
Victoria Manfredi

WESLEYAN
UNIVERSITY

Acknowledgements: materials adapted from Computer Networking: A Top Down Approach 7th edition: ©1996-2016, J.F Kurose and K.W. Ross, All Rights Reserved as well as from slides by Abraham Matta at Boston University, and some material from Computer Networks by Tannenbaum and Wetherall.

Today

1. Announcements
   - homework 5 extension until Thursday at 11:59p

2. TCP
   - seq #s and ack #s
   - timeouts
   - reliable data transport
   - connection management

3. Midterm
   - covers through whatever we get through today
   - overview of exam format

TCP

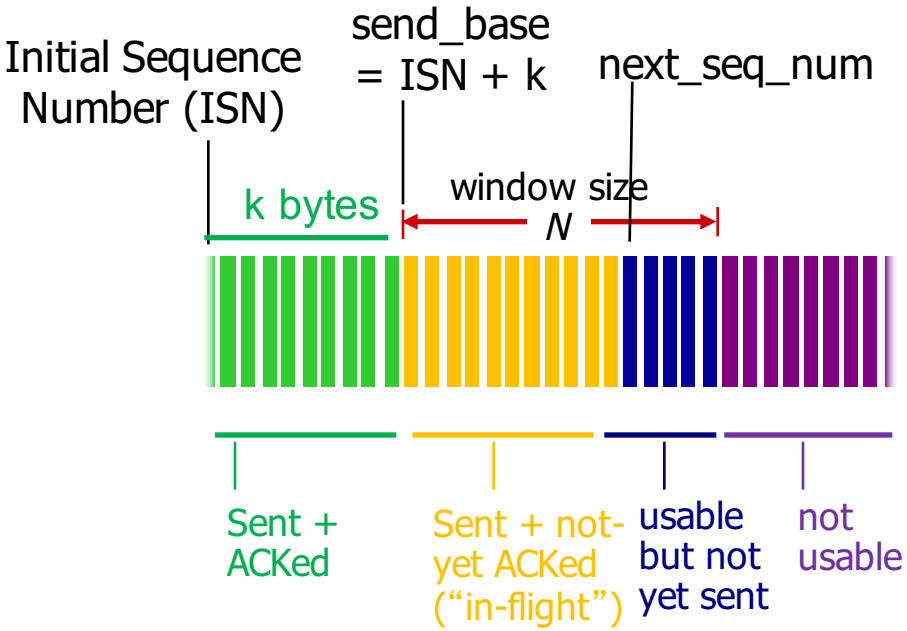
SEQ #S AND ACK #S

TCP ACKs

Cumulative ACKs (but different than in Go-Back-N)

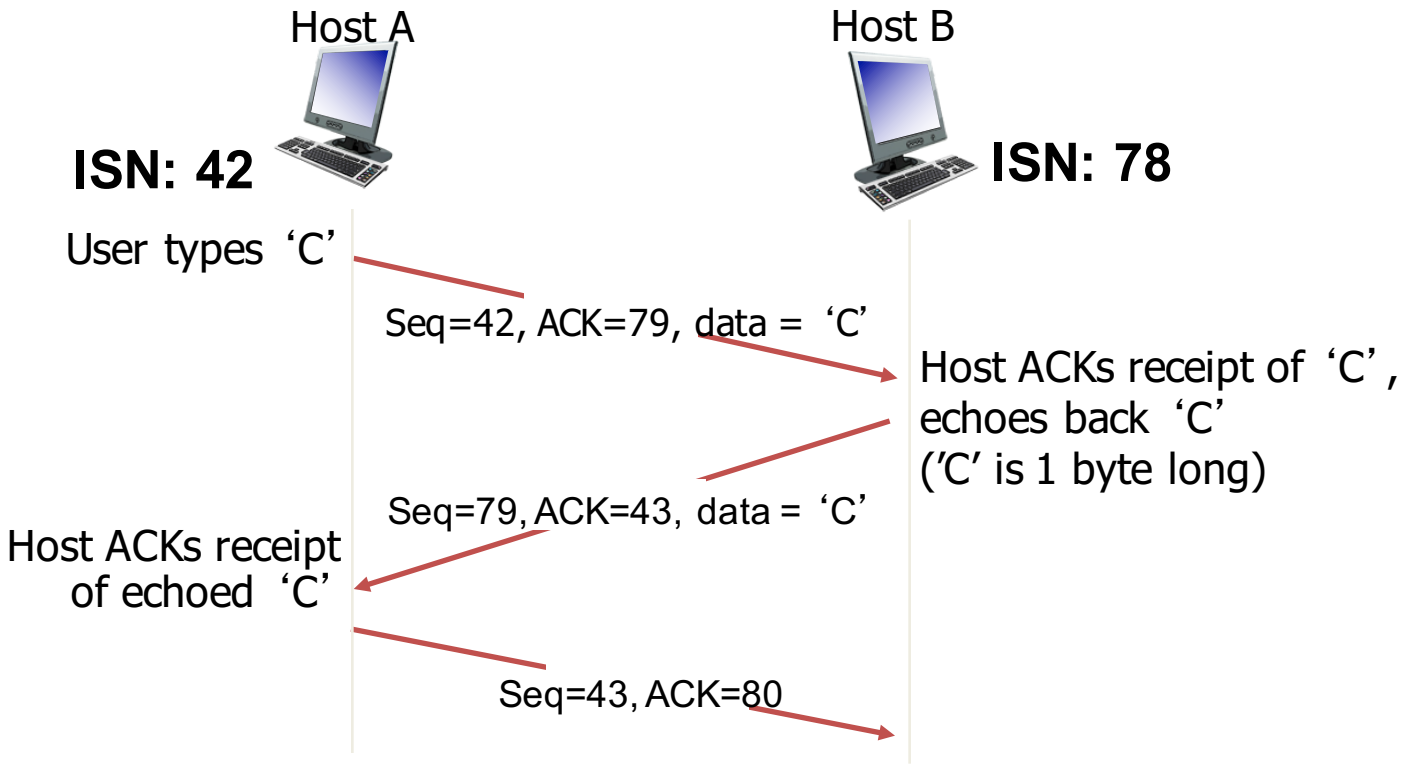
- ACKs everything up to sequence number received
- ACKs what receiver expects next, not last packet received
- Only 1 retransmission timer (for first pkt in window
  - Sender retransmits only first pkt in window if no ack when timer expires

Sequence #s are not sequential: counting bytes not packets

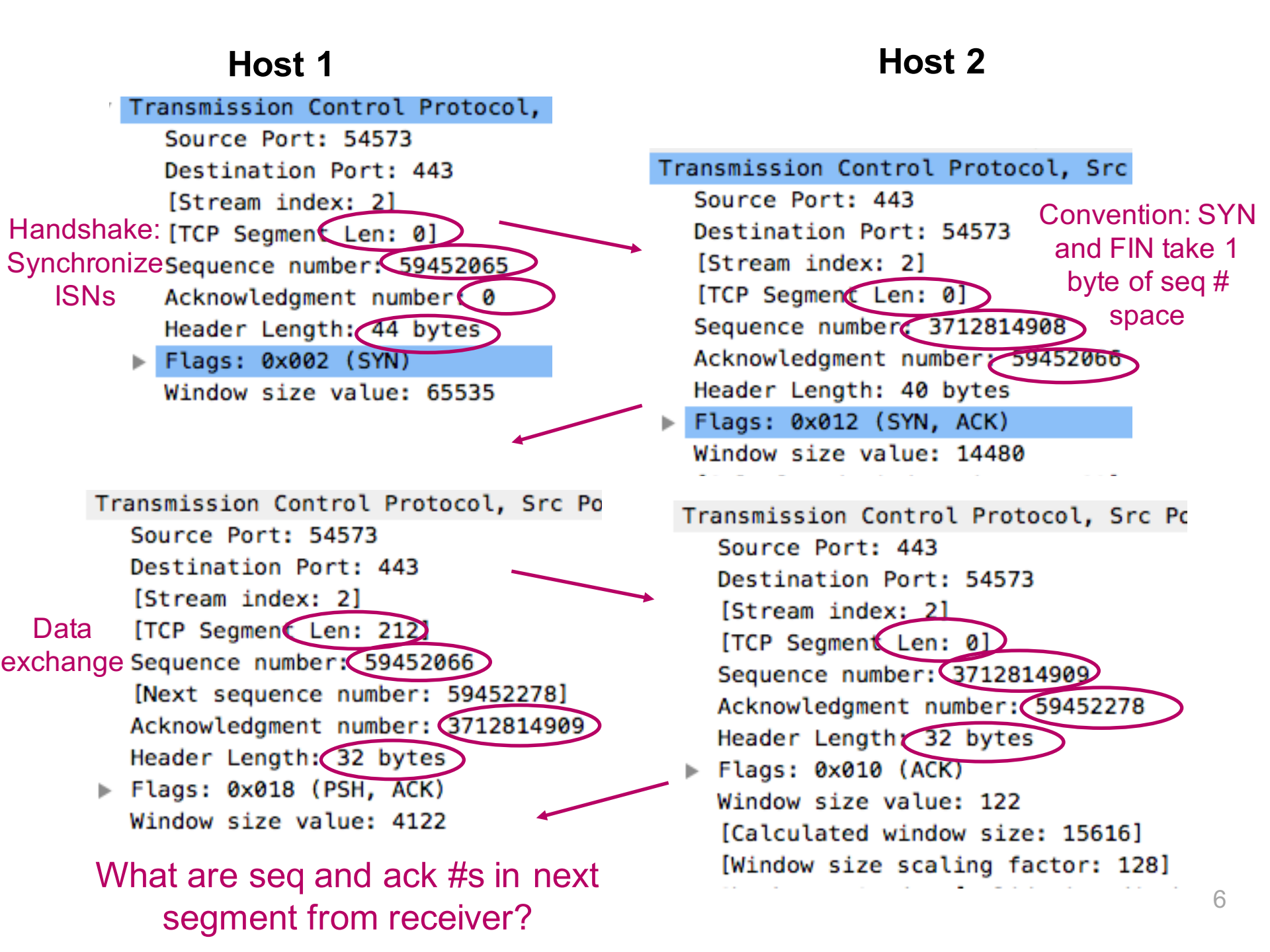


TCP seq. numbers, ACKs

Sequence numbers are synchronized during connection set-up



Simple nc scenario


Host 1
Host 2
Transmission Control Protocol,
Source Port: 54573
Destination Port: 443
[Stream index: 2]
[TCP Segment Len: 0]
Sequence number: 59452065
Acknowledgment number: 0
Header Length: 44 bytes
Flags: 0x002 (SYN)
Window size value: 65535
Handshake: Synchronize ISNs
Transmission Control Protocol, Src
Source Port: 443
Destination Port: 54573
[Stream index: 2]
[TCP Segment Len: 0]
Sequence number: 3712814908
Acknowledgment number: 59452066
Header Length: 40 bytes
Flags: 0x012 (SYN, ACK)
Window size value: 14480
Convention: SYN and FIN take 1 byte of seq # space
Transmission Control Protocol, Src Po
Source Port: 54573
Destination Port: 443
[Stream index: 2]
[TCP Segment Len: 212]
Sequence number: 59452066
[Next sequence number: 59452278]
Acknowledgment number: 3712814909
Header Length: 32 bytes
Flags: 0x018 (PSH, ACK)
Window size value: 4122
Data exchange
Transmission Control Protocol, Src Po
Source Port: 443
Destination Port: 54573
[Stream index: 2]
[TCP Segment Len: 0]
Sequence number: 3712814909
Acknowledgment number: 59452278
Header Length: 32 bytes
Flags: 0x010 (ACK)
Window size value: 122
[Calculated window size: 15616]
[Window size scaling factor: 128]
What are seq and ack #s in next segment from receiver?

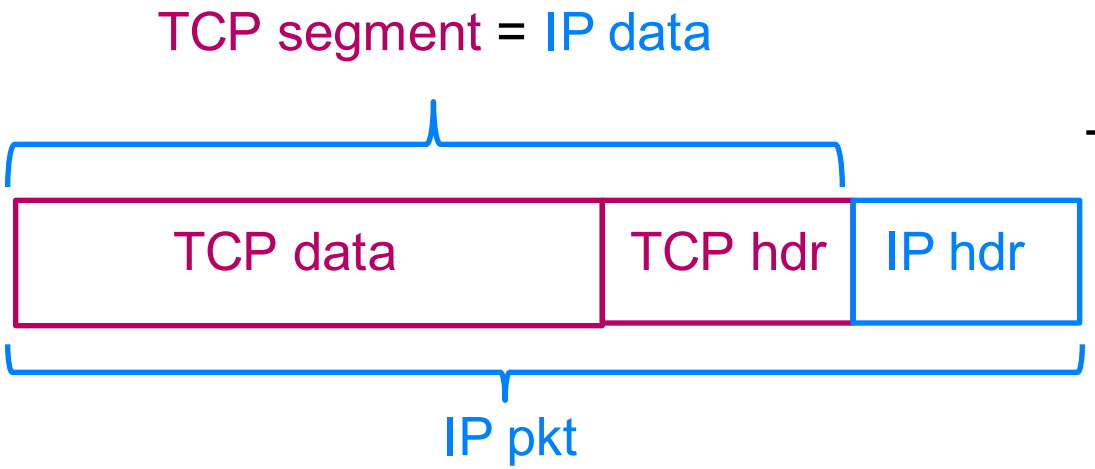
Segment size

Max length of IP packet in bytes

- MTU: Maximum Transmission Unit
- 1500 bytes if Ethernet used as link layer protocol

Max length of TCP data in bytes

- MSS: Maximum Segment Size
- MSS = MTU – IP hdr – TCP hdr
  - TCP header >= 20bytes



TCP segment sent when

- full: meets MSS
- not full: timeout

TCP

TIMEOUTS

TCP timeout

Q: how to set TCP timeout value?

Longer than RTT (ideally proportional)

- but RTT varies ….

Too short

- premature timeout
- unnecessary retransmissions

Too long

- slow reaction to segment loss

How to estimate RTT

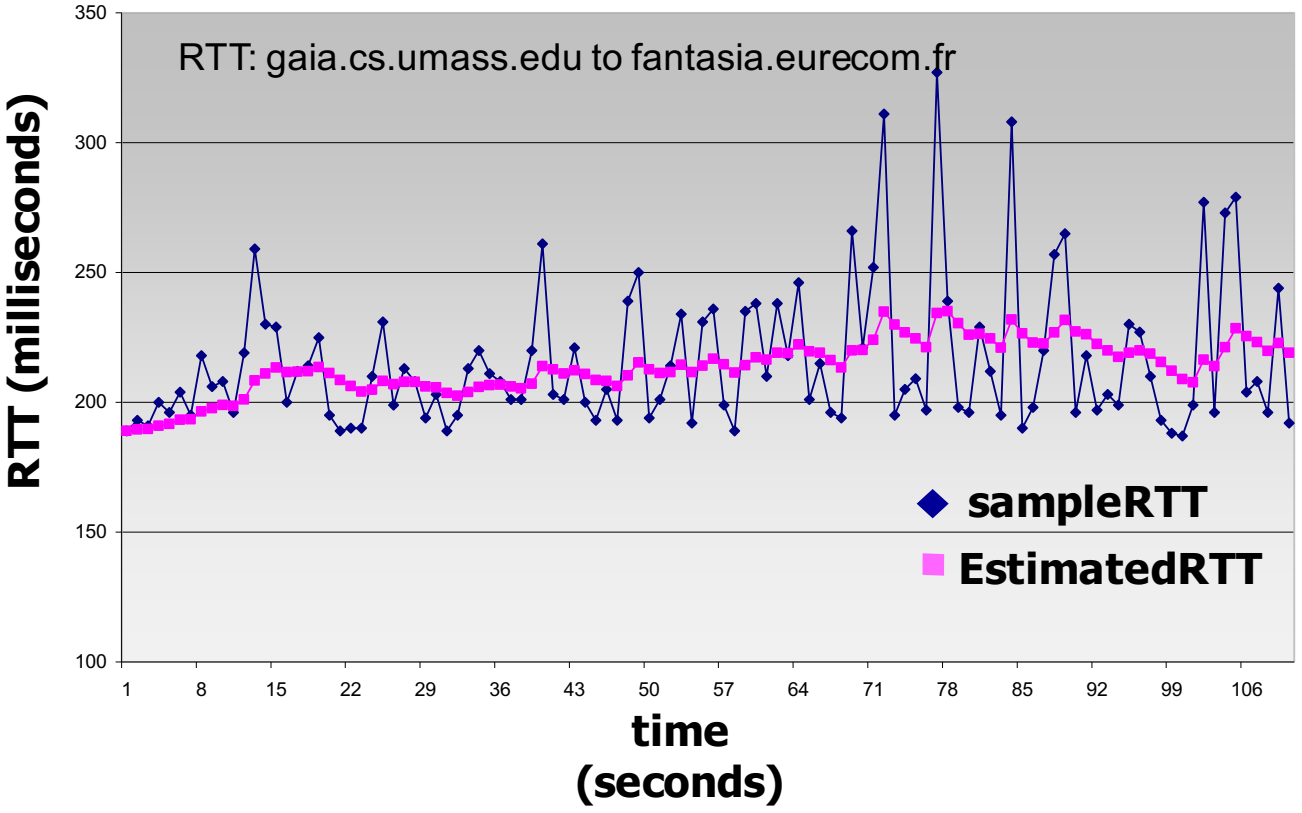
SampleRTT

- time from segment transmission to ACK reception
- ignore retransmissions
  - since problems associating retransmitted ACK with right pkt
  - will vary: use average of several measurements

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

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

Handling variation in RTT



Timeout interval should be ≥ **`EstimatedRTT`**

- because of variation of RTT values
- average several recent measurements, not just current **`SampleRTT`**
- how big should margin of error be?

Handling variation in RTT

Timeout interval should be ≥ **EstimatedRTT**

- because of variation of RTT values
- large variation in **EstimatedRTT** ⇒ larger safety margin

Estimate **SampleRTT** deviation from **EstimatedRTT**

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

(typically, $\beta = 0.25$)

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

"safety margin"

If timeout occurs: timeout interval is doubled to prevent premature timeout for subsequent segments

TCP

RELIABLE DATA TRANSFER

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

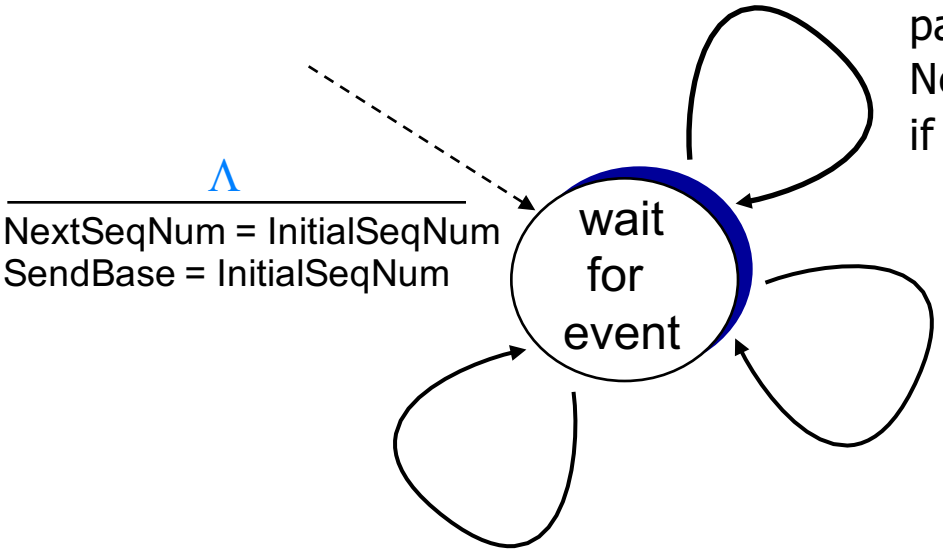
- timeout events
- duplicate ACKs

Let's initially consider simplified TCP sender

- ignore duplicate acks
- ignore flow control, congestion control

TCP sender (simplified)

Seq # is byte-stream # of first data byte in segment. Timer is for oldest unacked segment



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

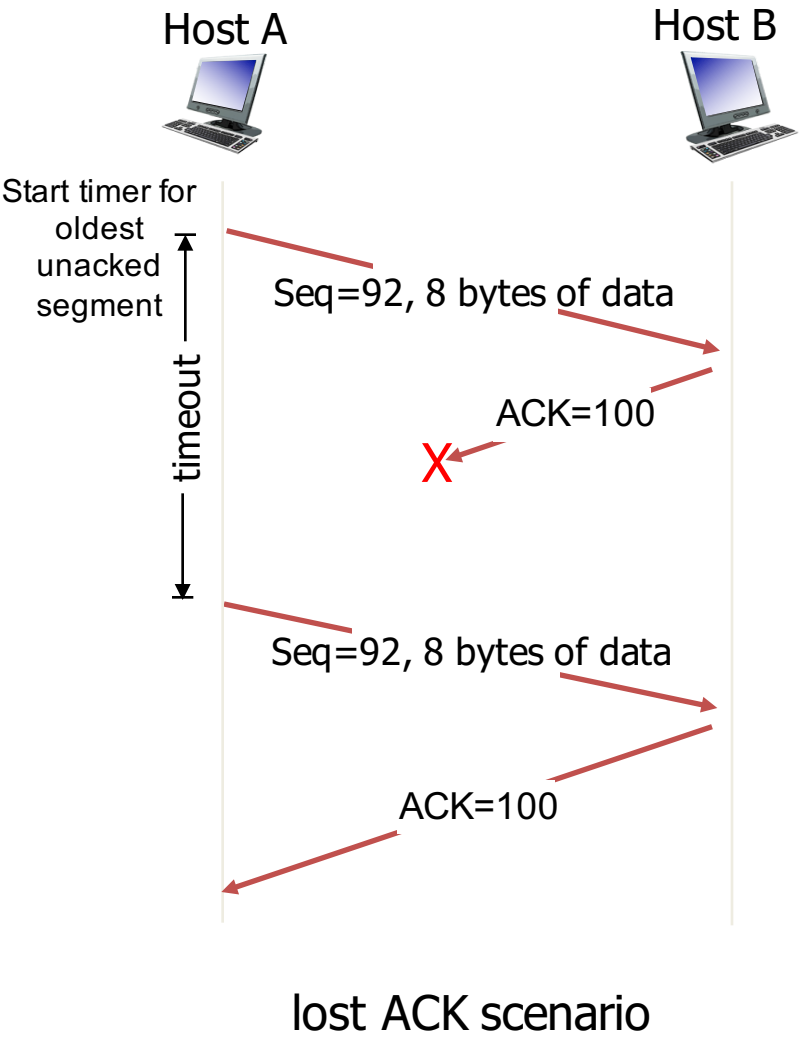
Retransmit first segment in window, restart timer

ACK received, with ACK field value y

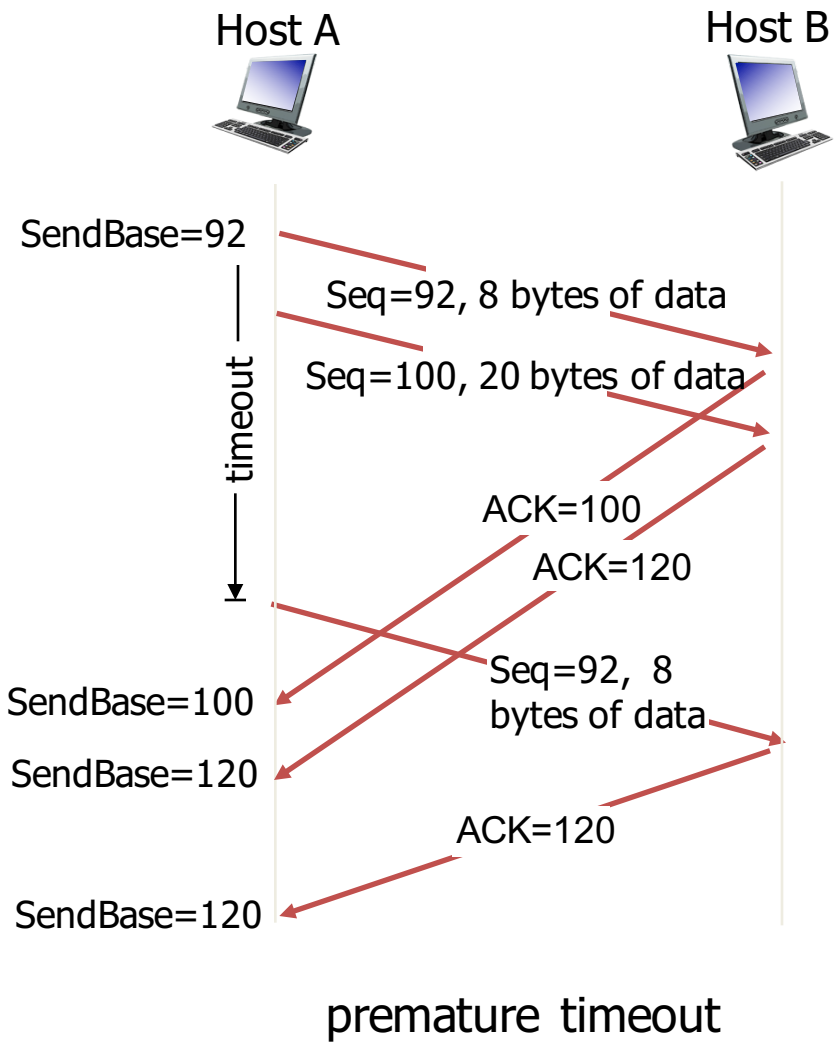
```
if (y > SendBase) {
    SendBase = y
    /* SendBase–1: last cumulatively ACKed byte */
    if (there are currently not-yet-acked segments)
        start timer
    else stop timer
    }
```

If acks previously unacked segments, update what is known to be ACKed, start timer if still unacked segments

TCP: retransmission scenarios

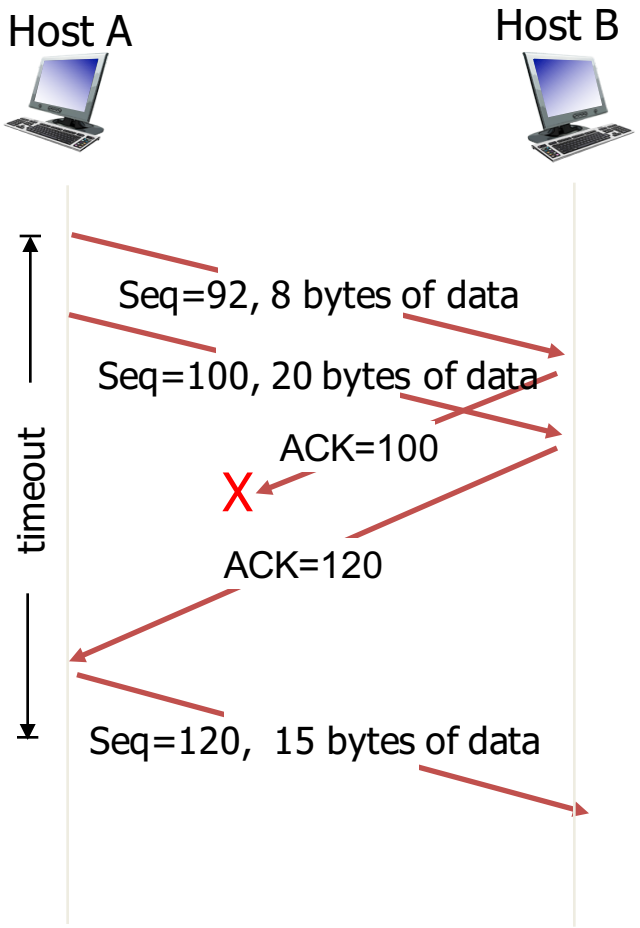


lost ACK scenario



premature timeout

TCP: retransmission scenarios



cumulative ACK

Duplicate ACKs

Time-out period often relatively long

- long delay before resending lost packet

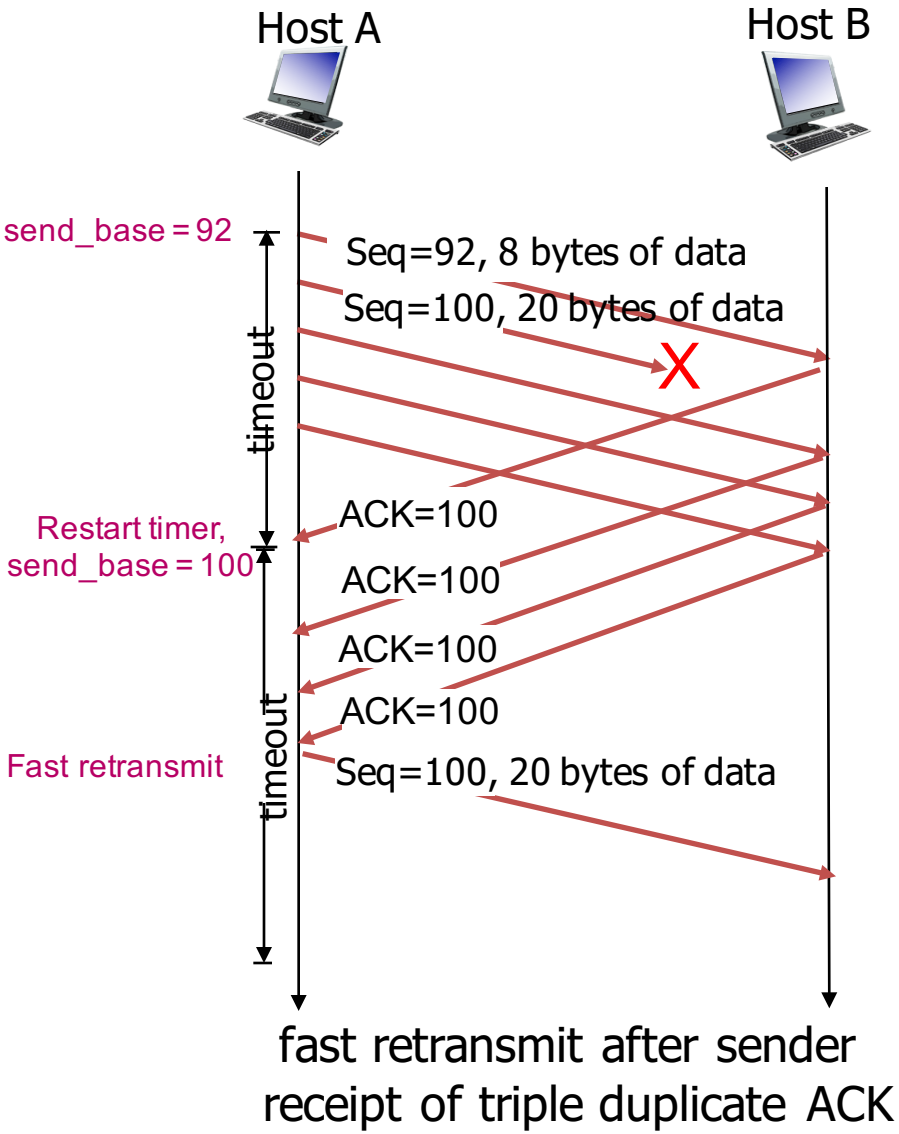
Duplicate ACKs

- indicate isolated loss (rather than congestion causing many losses)
  - sender often sends many segments back-to-back
  - if segment is lost, likely many duplicate ACKs
  - ACKs being received indicates some packets received at destination since ACK sent for every packet: so not congestion

TCP fast retransmit

- if sender receives 3 ACKs for same data (triple duplicate ACKs)
  - resend unacked segment with smallest seq #
- Why 3?
  - pkts may just have been reordered otherwise
  - likely that unacked segment lost, so don't wait for timeout

TCP fast retransmit



fast retransmit after sender receipt of triple duplicate ACK

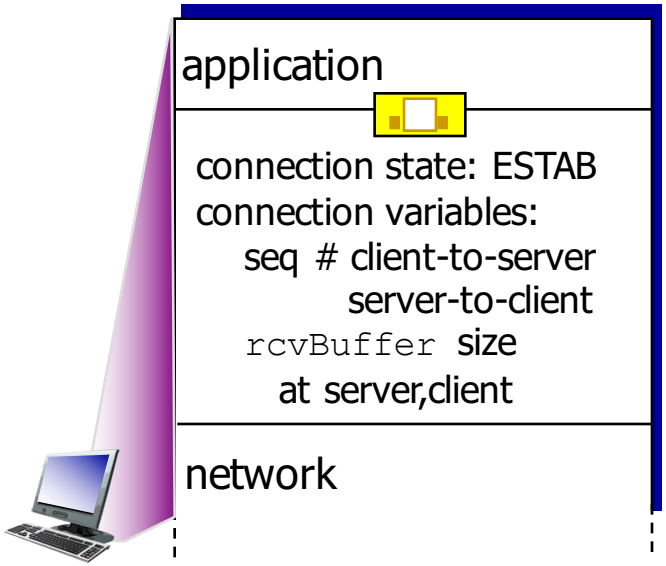
TCP

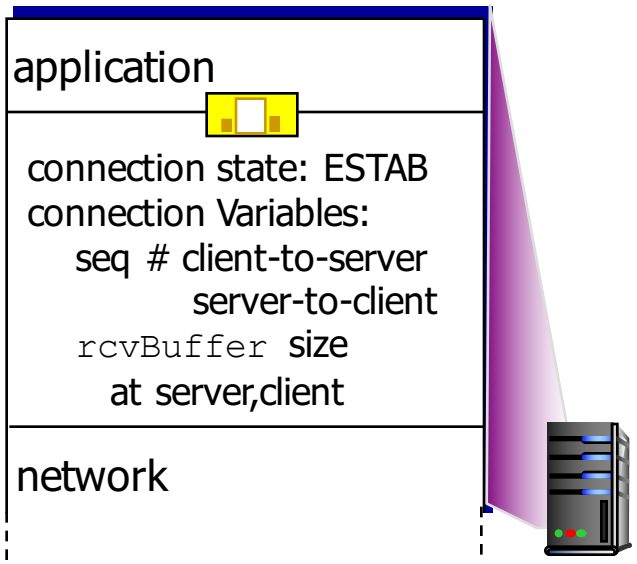
CONNECTION MANAGEMENT

Connection Management

Before exchanging data, sender/receiver handshake

- establish connection and connection parameters
  - each knowing the other willing to establish connection
- tear down connection when done



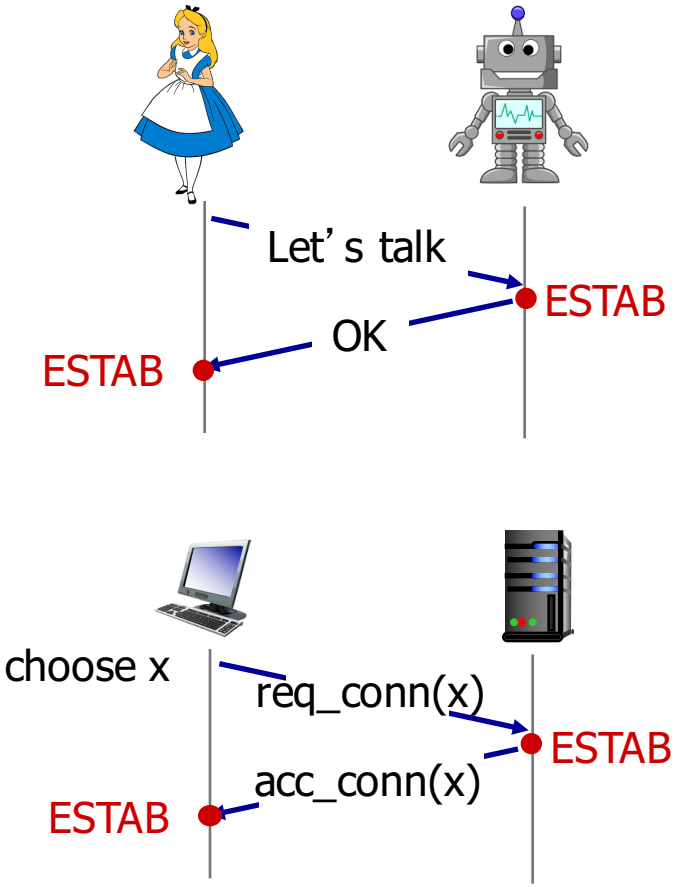


sock = sock.connect((host, port))

conn, addr = server_sock.accept()

Agreeing to establish a connection

2-way handshake:

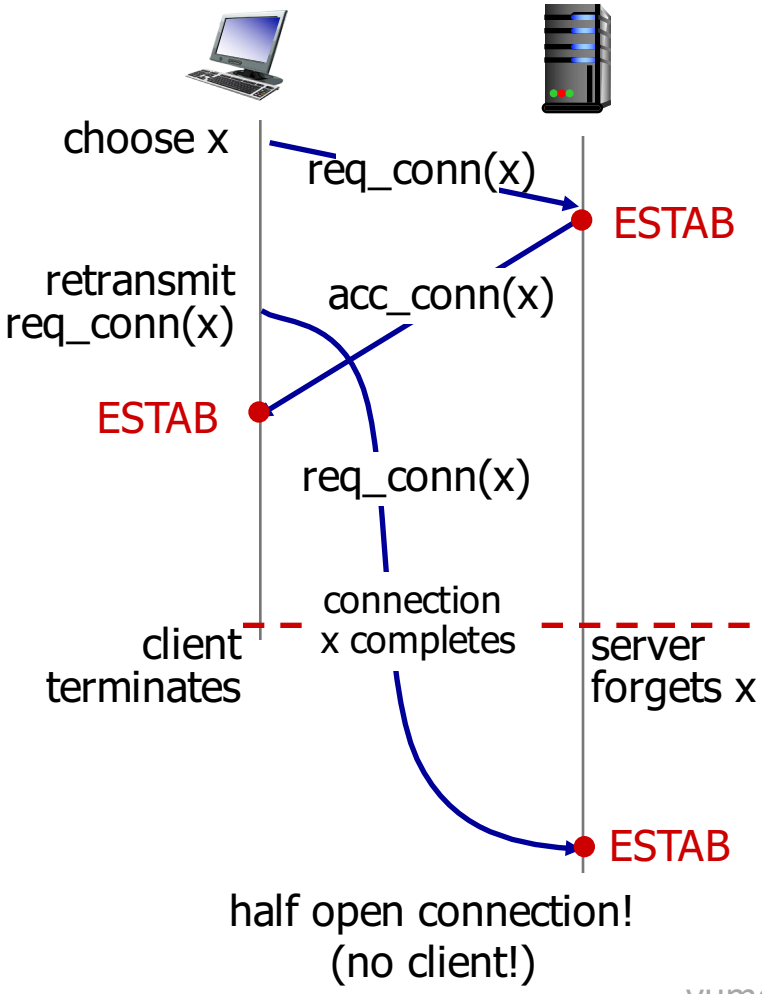


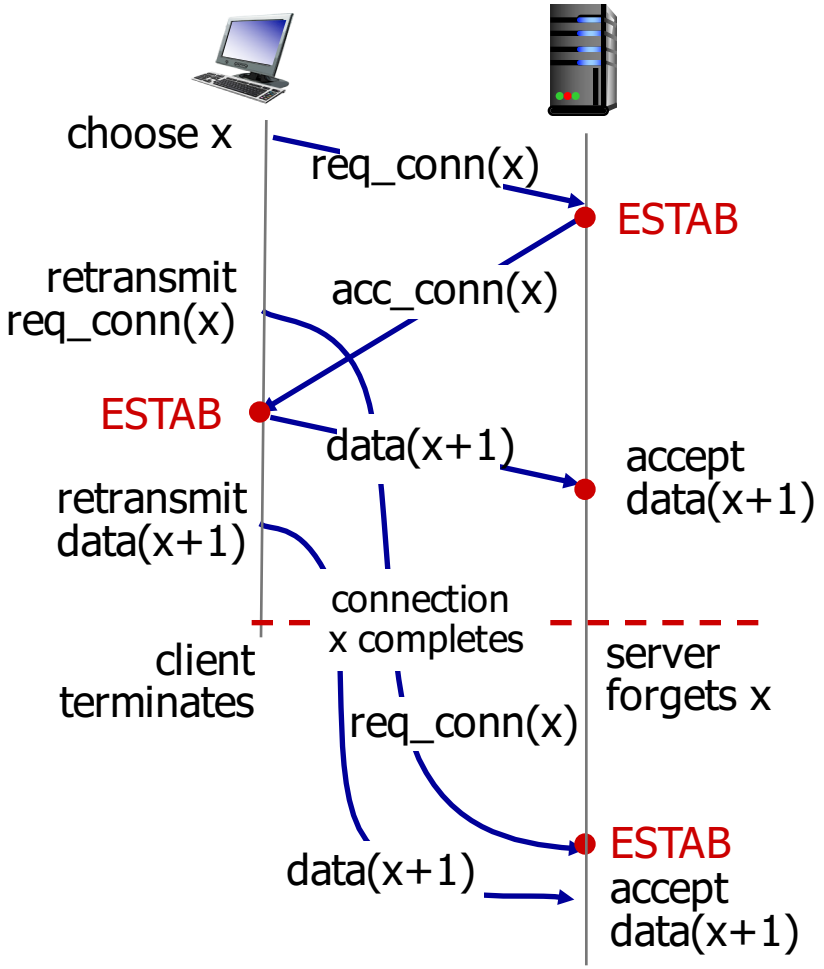
Q: will 2-way handshake always work in network?

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

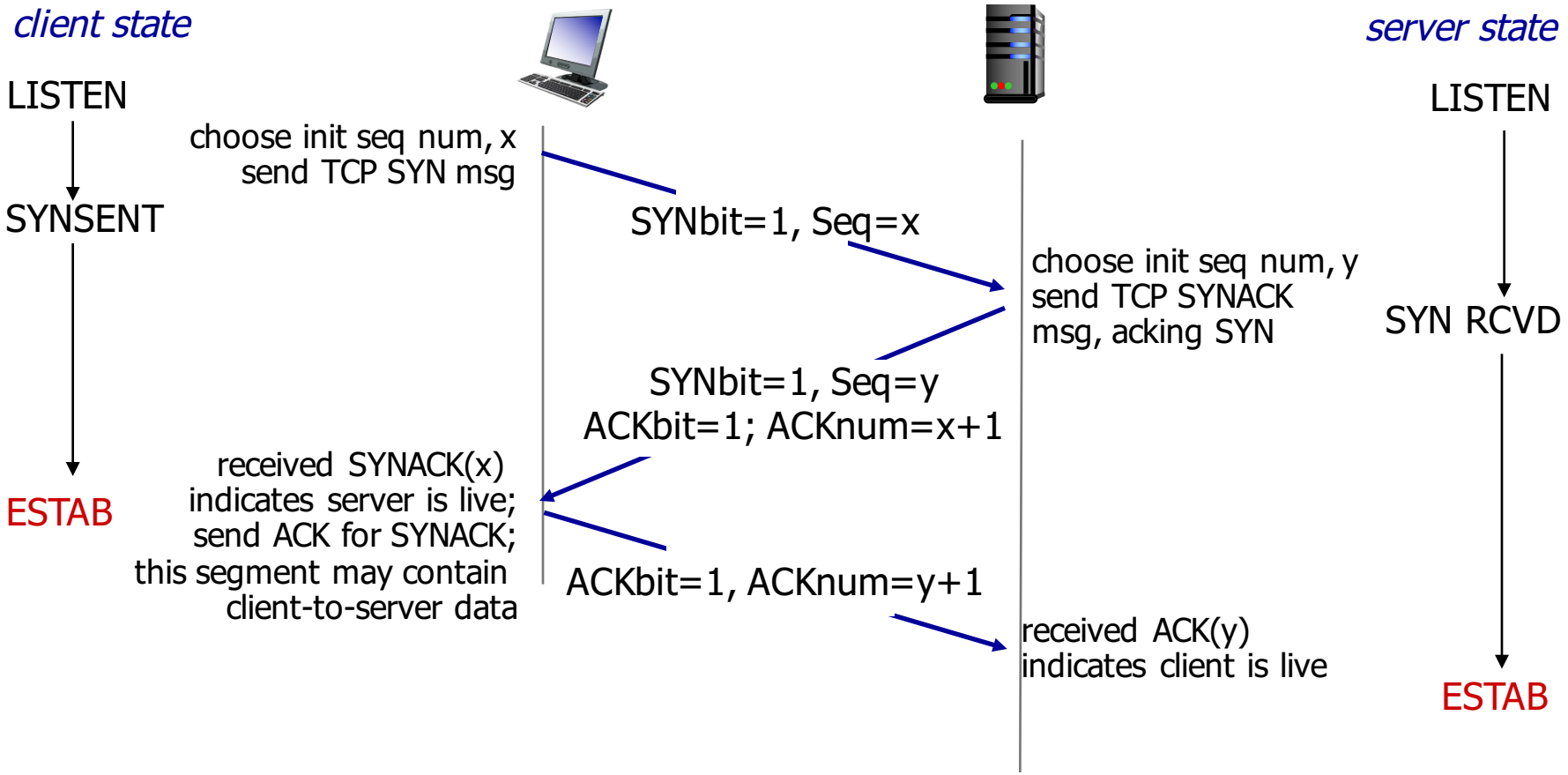
Agreeing to establish a connection

2-way handshake failure scenarios:



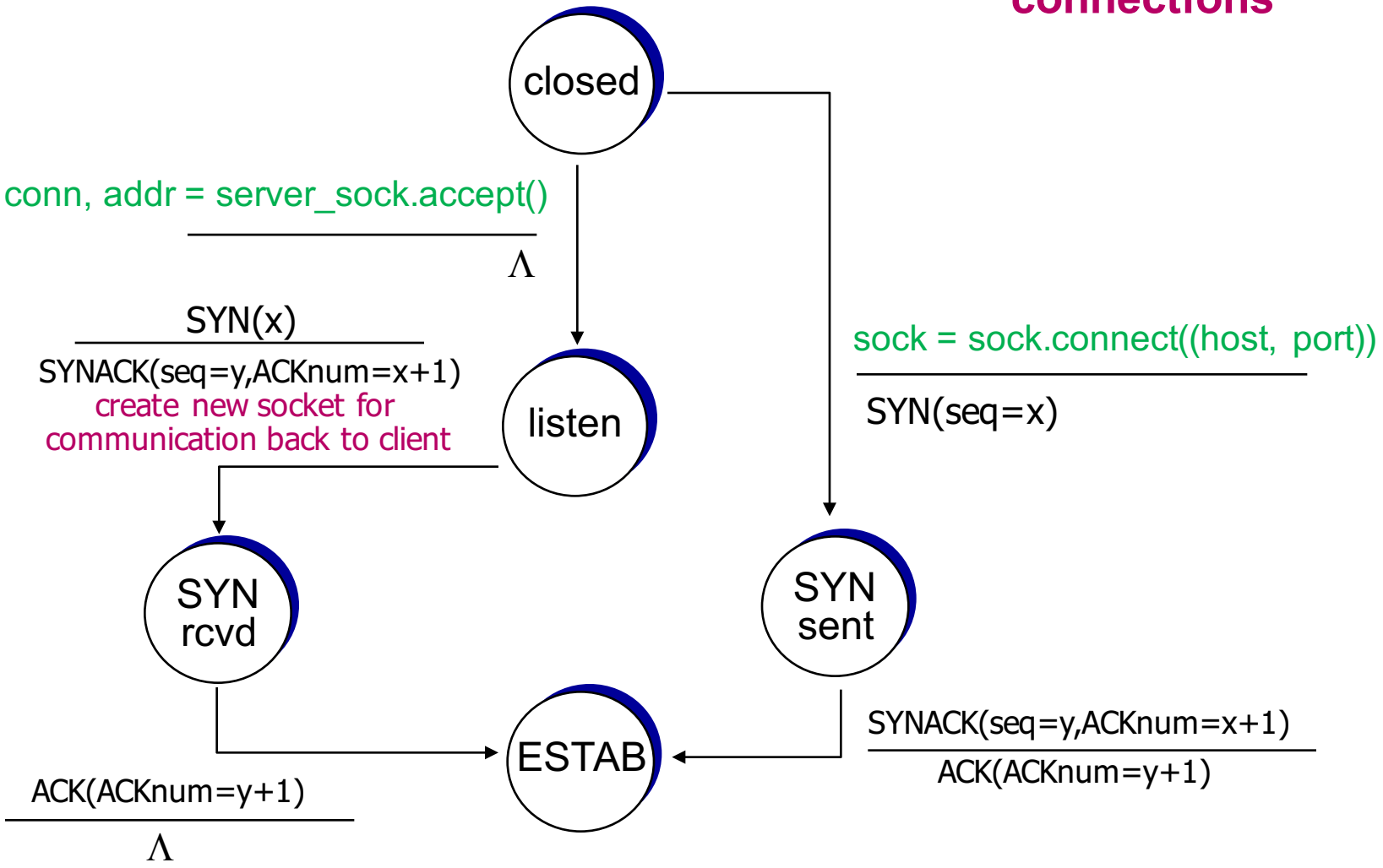


TCP 3-way handshake



TCP 3-way handshake: FSM

netstat -ta to see state of TCP connections



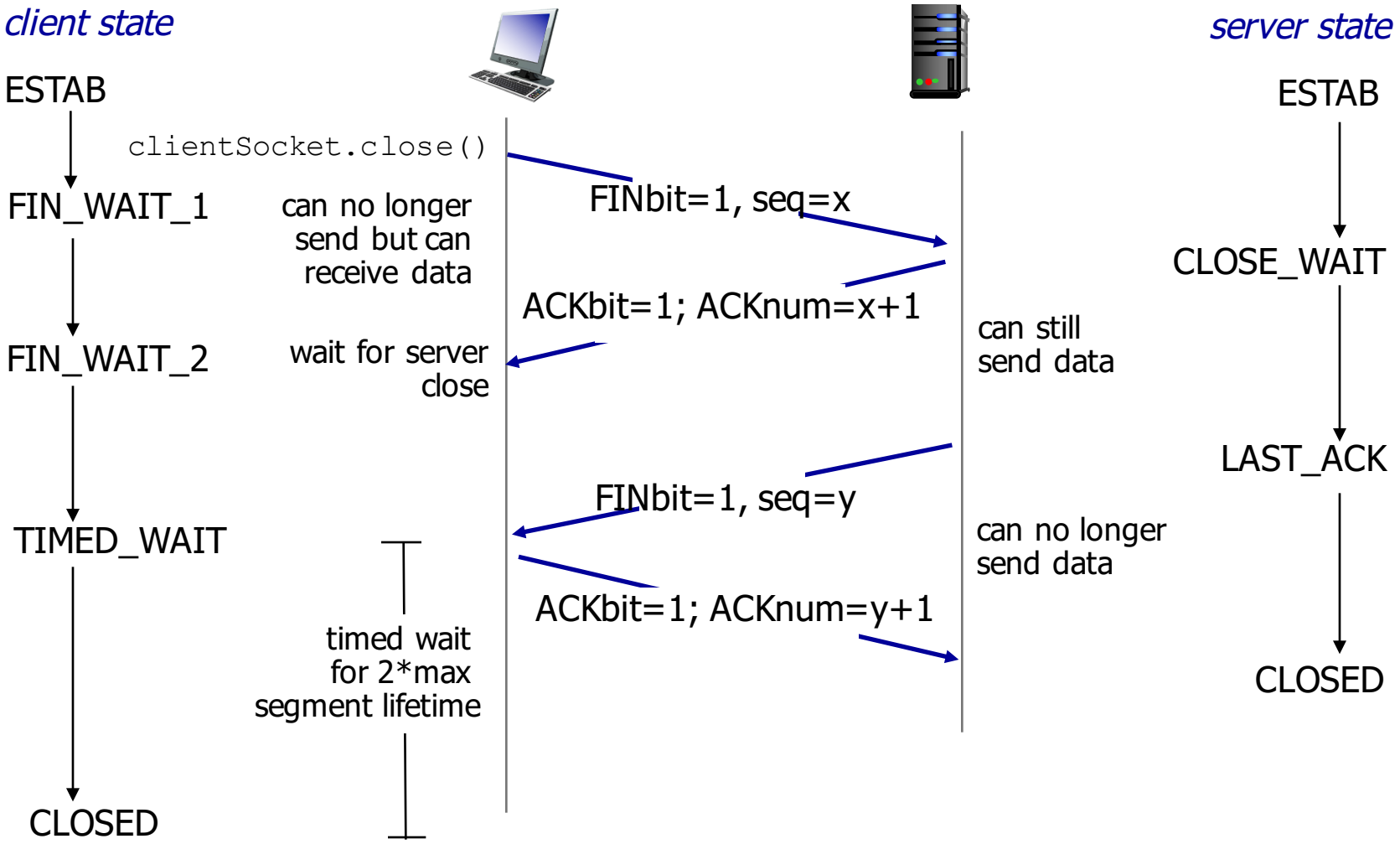
Look at the state of tcp connections

```
> netstat -ta
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0  vmanfredismbp2.w.55777 lga25s60-in-f5.1.https ESTABLISHED
tcp4      31      0  vmanfredismbp2.w.55736 162.125.34.6.https     CLOSE_WAIT
tcp4       0      0  vmanfredismbp2.w.55717 a104-110-151-148.https ESTABLISHED
tcp4       0      0  vmanfredismbp2.w.55716 a104-110-151-148.https ESTABLISHED
tcp4       0      0  vmanfredismbp2.w.55715 a104-110-151-148.https ESTABLISHED
tcp4       0      0  vmanfredismbp2.w.55714 a104-110-151-148.https ESTABLISHED
tcp4       0      0  vmanfredismbp2.w.55713 a104-110-151-148.https ESTABLISHED
tcp4       0      0  vmanfredismbp2.w.55668 wesfiles.wesleya.http  CLOSE_WAIT
tcp4       0      0  vmanfredismbp2.w.55486 162.125.18.133.https   ESTABLISHED
tcp4       0      0  vmanfredismbp2.w.55322 162.125.18.133.https   ESTABLISHED
tcp4      31      0  vmanfredismbp2.w.55250 162.125.4.3.https      CLOSE_WAIT
tcp4       0      0  vmanfredismbp2.w.55170 ec2-52-20-75-192.https CLOSE_WAIT
tcp4       0      0  vmanfredismbp2.w.55072 85.97.201.35.bc..https ESTABLISHED
tcp4       0      0  localhost.ipp          *.*                    LISTEN
tcp6       0      0  localhost.ipp          *.*                    LISTEN
tcp4       0      0  vmanfredismbp2.w.53453 6.97.a86c.ip4.st.https ESTABLISHED
```

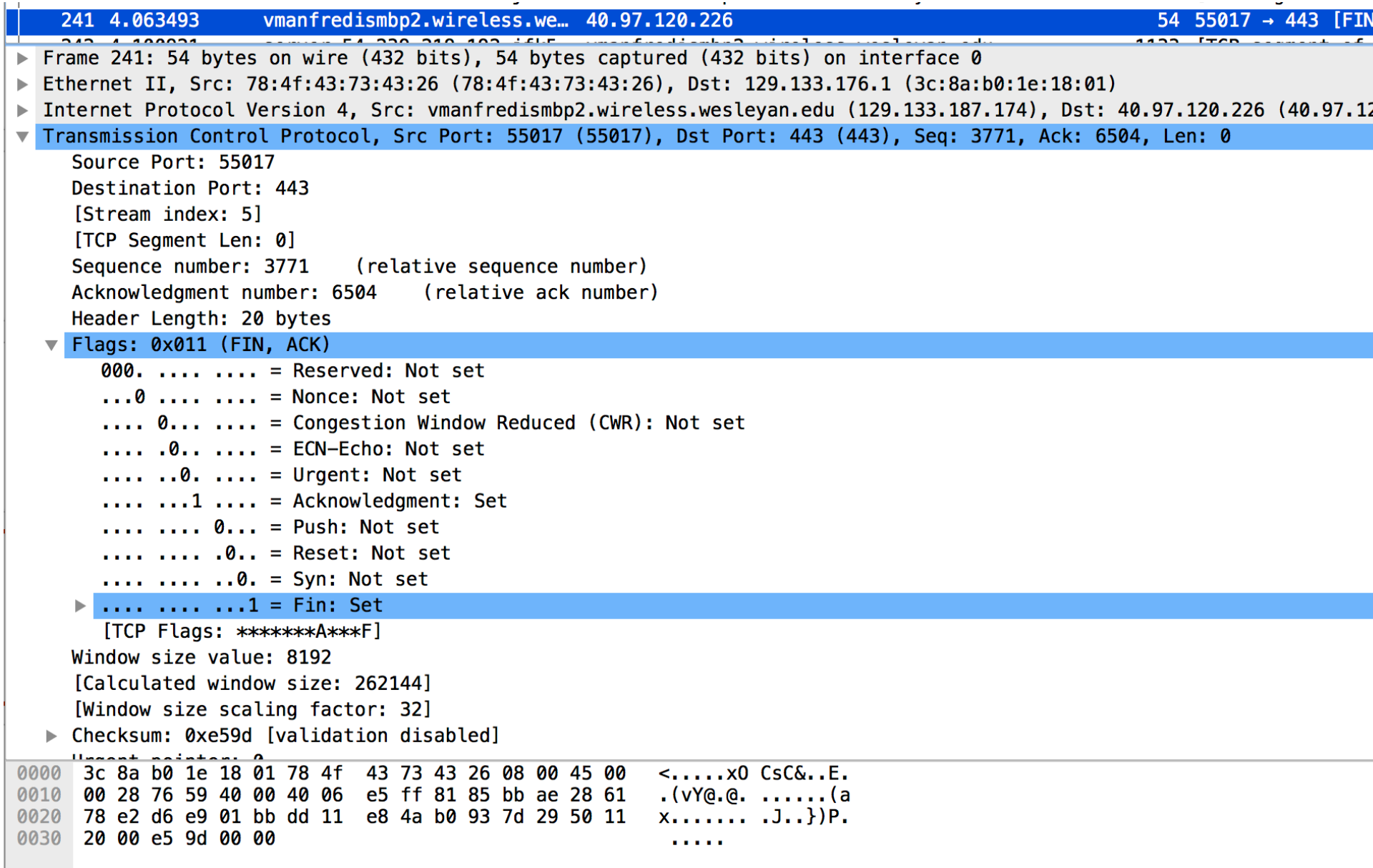
TCP: politely closing a connection

Client, server close connection: each sends TCP segment with FIN bit = 1

- respond to received FIN with ACK (ACK can be combined with own FIN)



FIN segment in Wireshark



Midterm

OVERVIEW

Midterm overview

In class on Wednesday Mar. 28

- closed book, closed notes
- covers material in lectures 1 to 12

Tentative exam format

Still under development but …

No probability questions

Questions for which you need only provide short answers

- E.g.,
  - what is the difference between a recursive vs. iterated query in the DNS?
  - how are ports numbers used by UDP to demultiplex incoming segments?

Question on reliable Data Transfer

- Hint
  - Given channel characteristics design a protocol
  - be able to design a reliable data transfer protocol like the Stop-and-wait protocol, know your timeline diagrams

+ 2 other longer questions