CS 3251- Computer Networks I: TCP (1)

Professor Patrick Traynor
Lecture 09
9/17/13

Announcements

- Homework 2 was posted last week
  - Due 10/1
- Reminder: Project 2 will take time
  - It is posted now. Form your groups and get started soon!
  - Due 10/8

Project 2: GTmyMusic

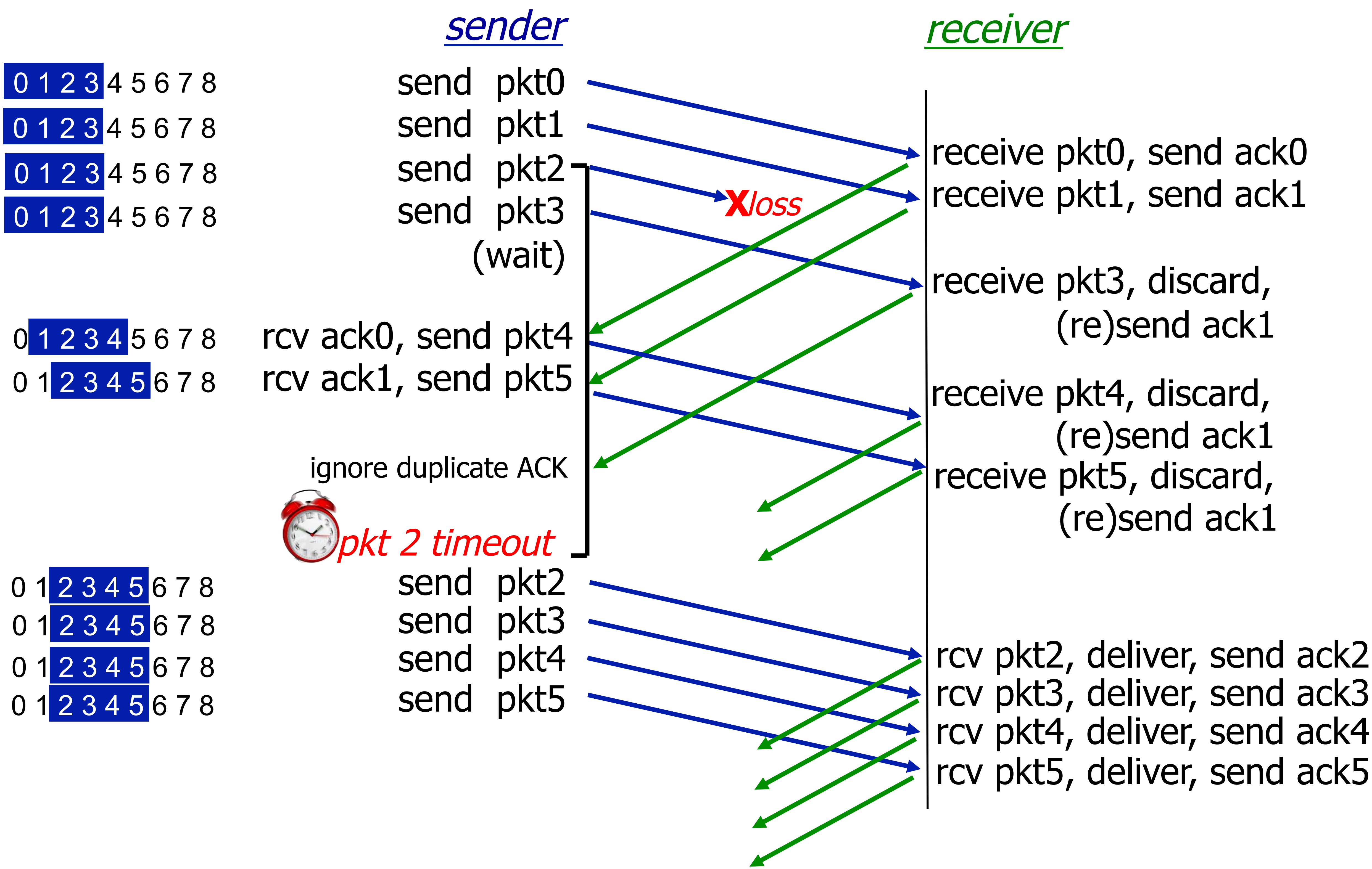
- You will be building an application that allows you to synchronize your music across machines.
  - The details of which are on the course website.
- You must work as a pair.
  - You will both receive the same grade, so work hard.
- Make sure your design is extensible.
  - Future projects will possibly build on this infrastructure.

Last Time

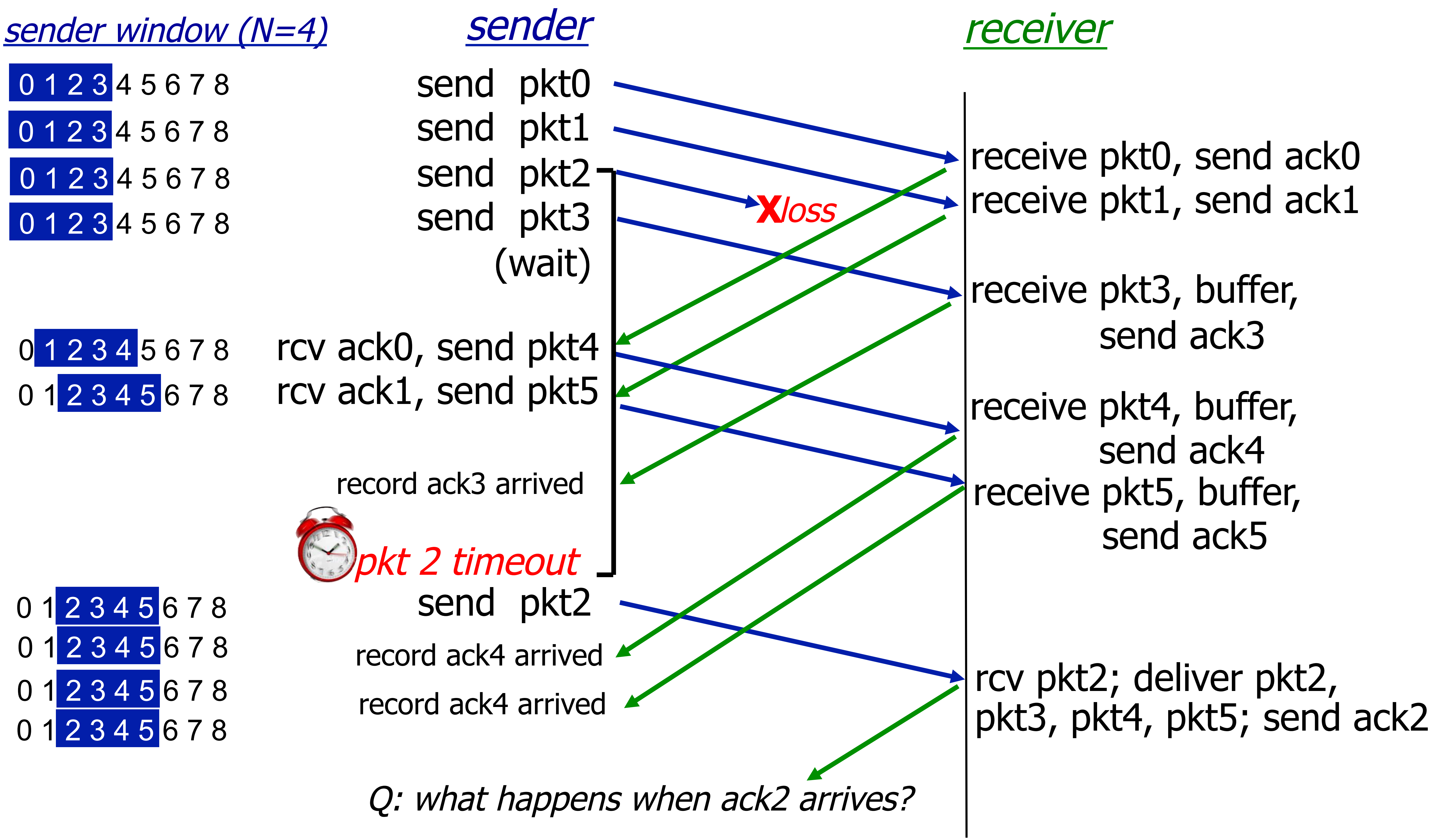
- Discussed a variety of algorithms that can give us guarantees of reliable delivery.
  - What were they?
  - How do they differ?

- Finite State Machines (FSMs) are a powerful means of representing protocols.

Review: Go-Back-N vs Selective Repeat



Review: Go-Back-N vs Selective Repeat

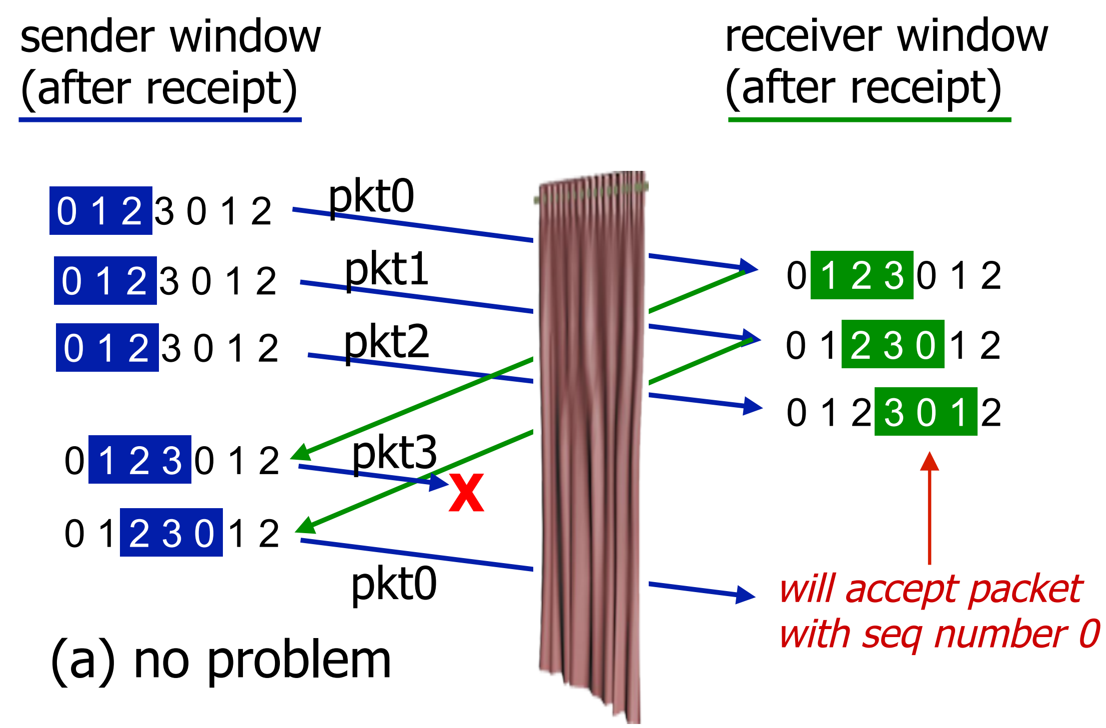


Review: Selective repeat: dilemma

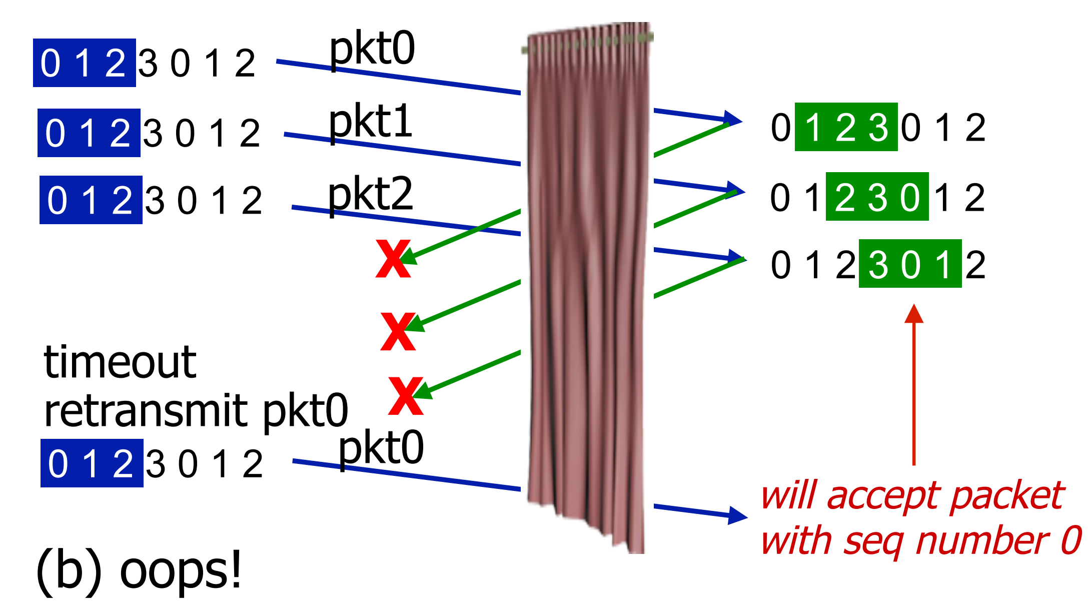
Example:

- seq #'s: 0, 1, 2, 3
- window size=3
- receiver sees no difference in two scenarios!
- incorrectly passes duplicate data as new in (a)

Q: what relationship between seq # size and window size?



receiver can't see sender side.
receiver behavior identical in both cases!
something's (very) wrong!



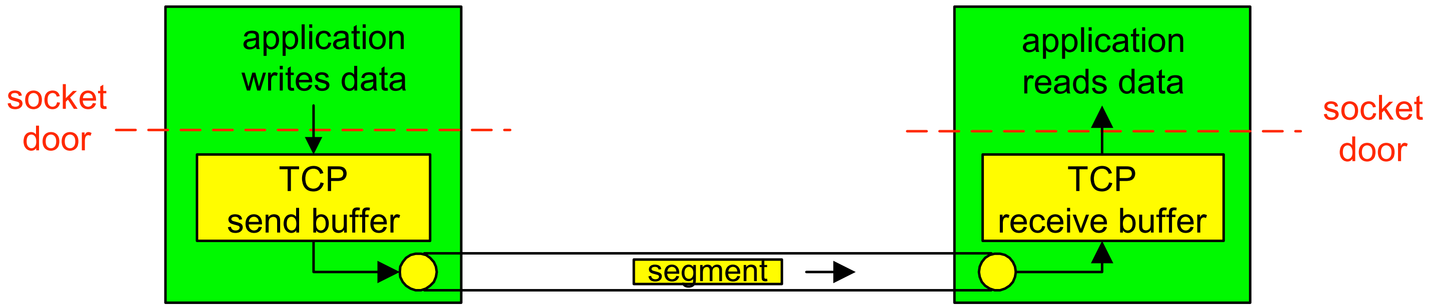
Chapter 3 outline

- 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
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

TCP: Overview

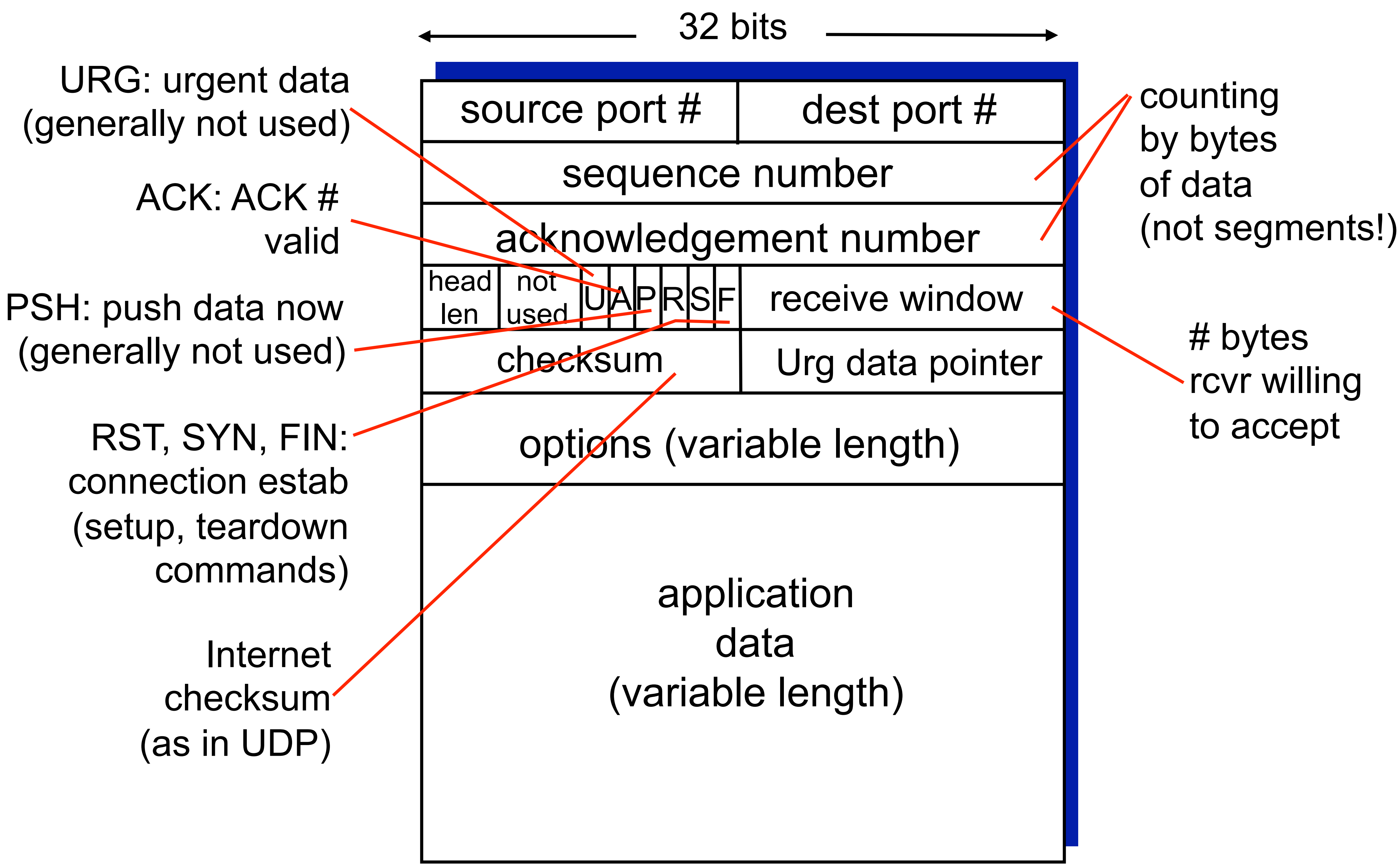
RFCs: 793, 1122, 1323, 2018, 2581

- 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) init’s sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP segment structure



TCP seq. #'s and ACKs

Seq. #'s:

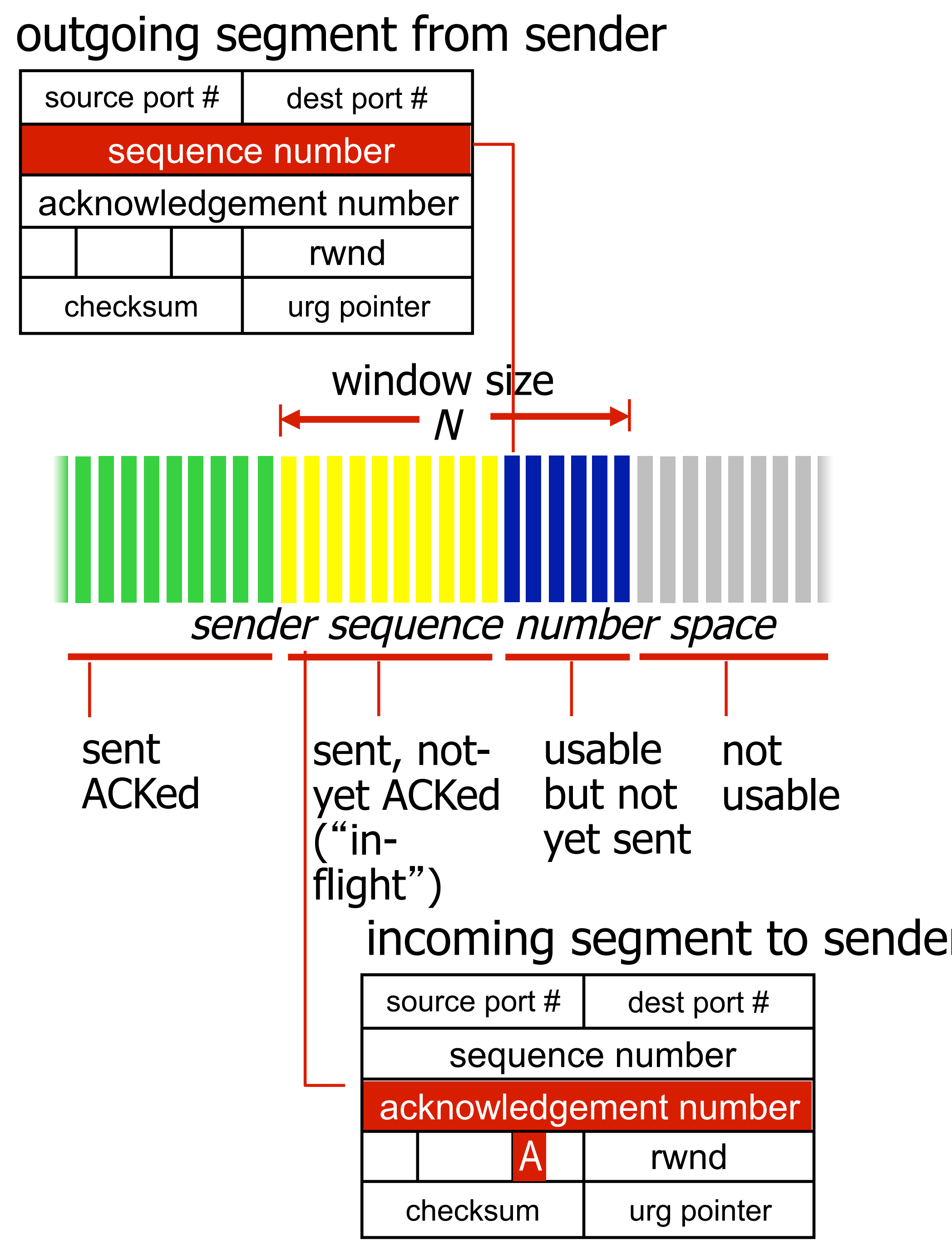
- byte stream "number" of first byte in segment's data

ACKs:

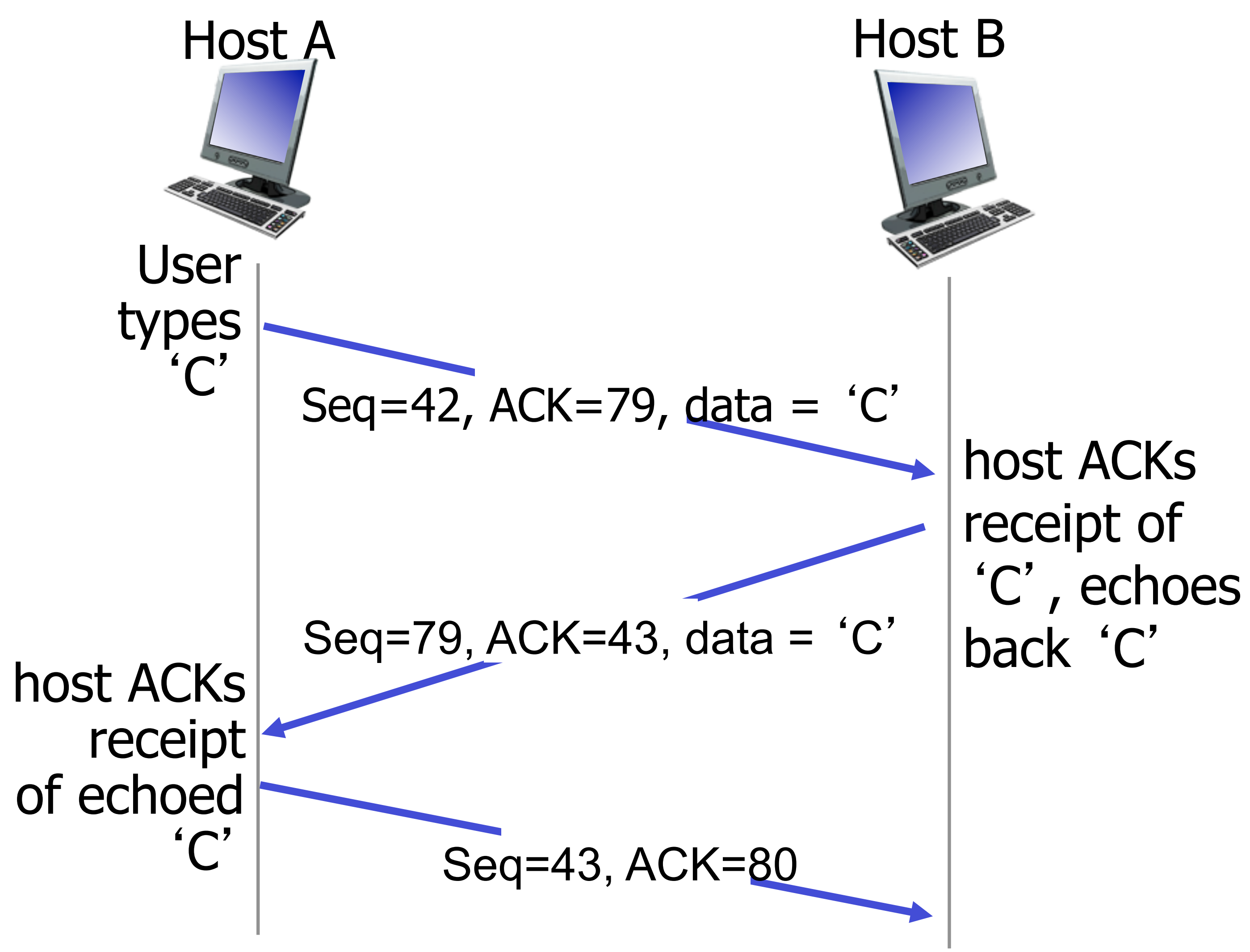
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

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



TCP Sequence Numbers, Acks



simple telnet scenario

TCP Round Trip Time and 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?

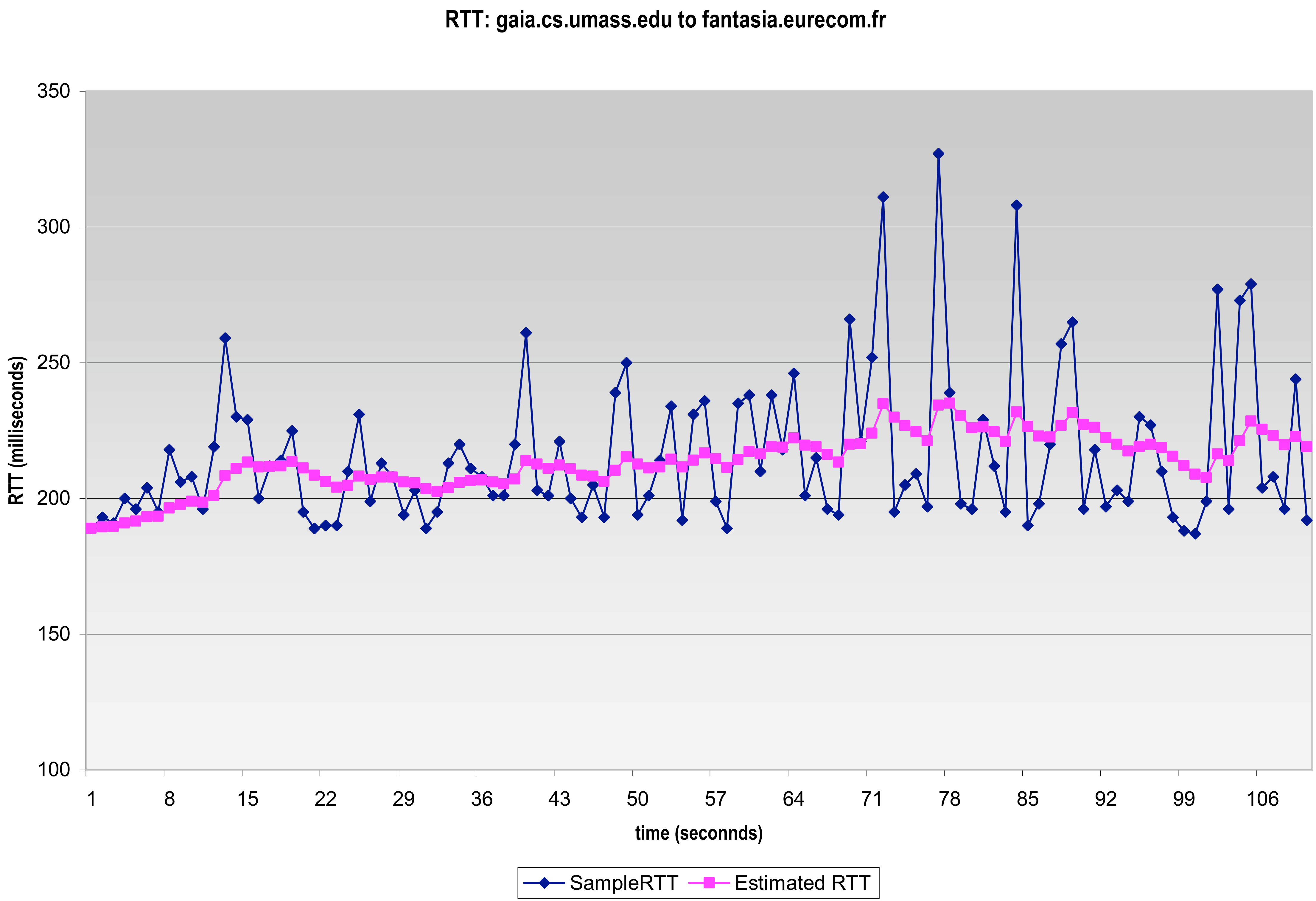
- `SampleRTT`: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **`SampleRTT`** will vary, want estimated RTT "smoother"
  - average several recent measurements, not just current **`SampleRTT`**

TCP Round Trip Time and Timeout

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

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

Example RTT estimation:



TCP Round Trip Time and Timeout

Setting the timeout

- **`EstimtedRTT`** plus "safety margin"
  - large variation in **`EstimatedRTT`** **->** larger safety margin
- first estimate of how much SampleRTT deviates from EstimatedRTT:

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

(typically, β = 0.25)
```

Then set timeout interval:

```
TimeoutInterval = EstimatedRTT + 4*DevRTT
```

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
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

TCP reliable data transfer

- TCP creates rdt service on top of IP's unreliable service
  - Pipelined segments
  - Cumulative acks
  - Single retransmission timer
- TCP uses single retransmission timer

- Retransmissions are triggered by:
  - timeout events
  - duplicate acks
- Initially consider simplified TCP sender:
  - ignore duplicate acks
  - ignore flow control, congestion control

TCP sender events:

<u>data rcvd from app:</u>

- 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 oldest unacked segment
  - expiration interval: `TimeOutInterval`

<u>timeout:</u>

- retransmit segment that caused timeout
- restart timer

<u>Ack rcvd:</u>

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

TCP sender (simplified)

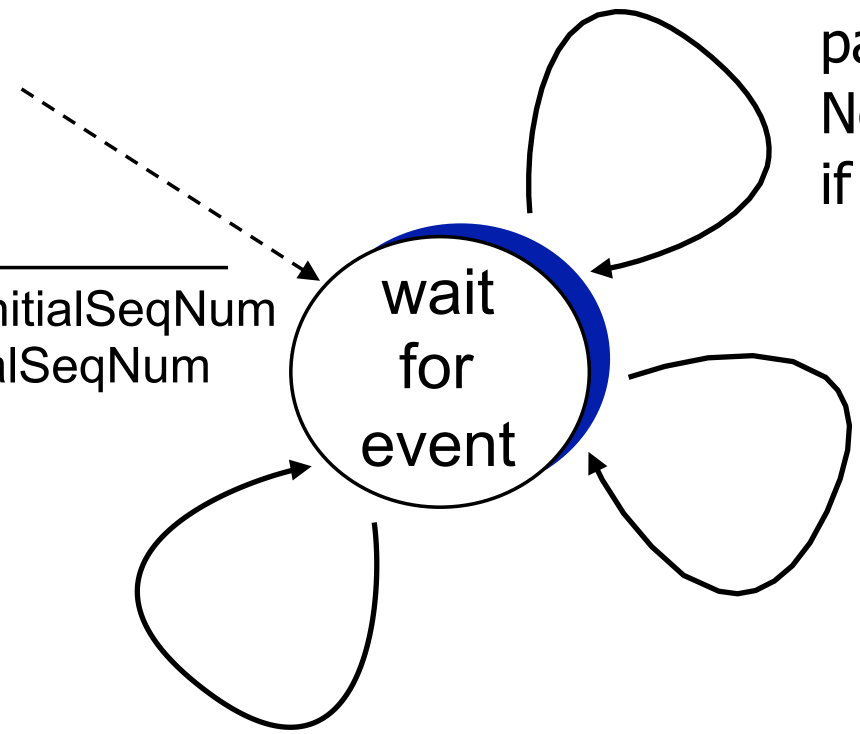
data received from application above

---

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

Λ

NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum



timeout

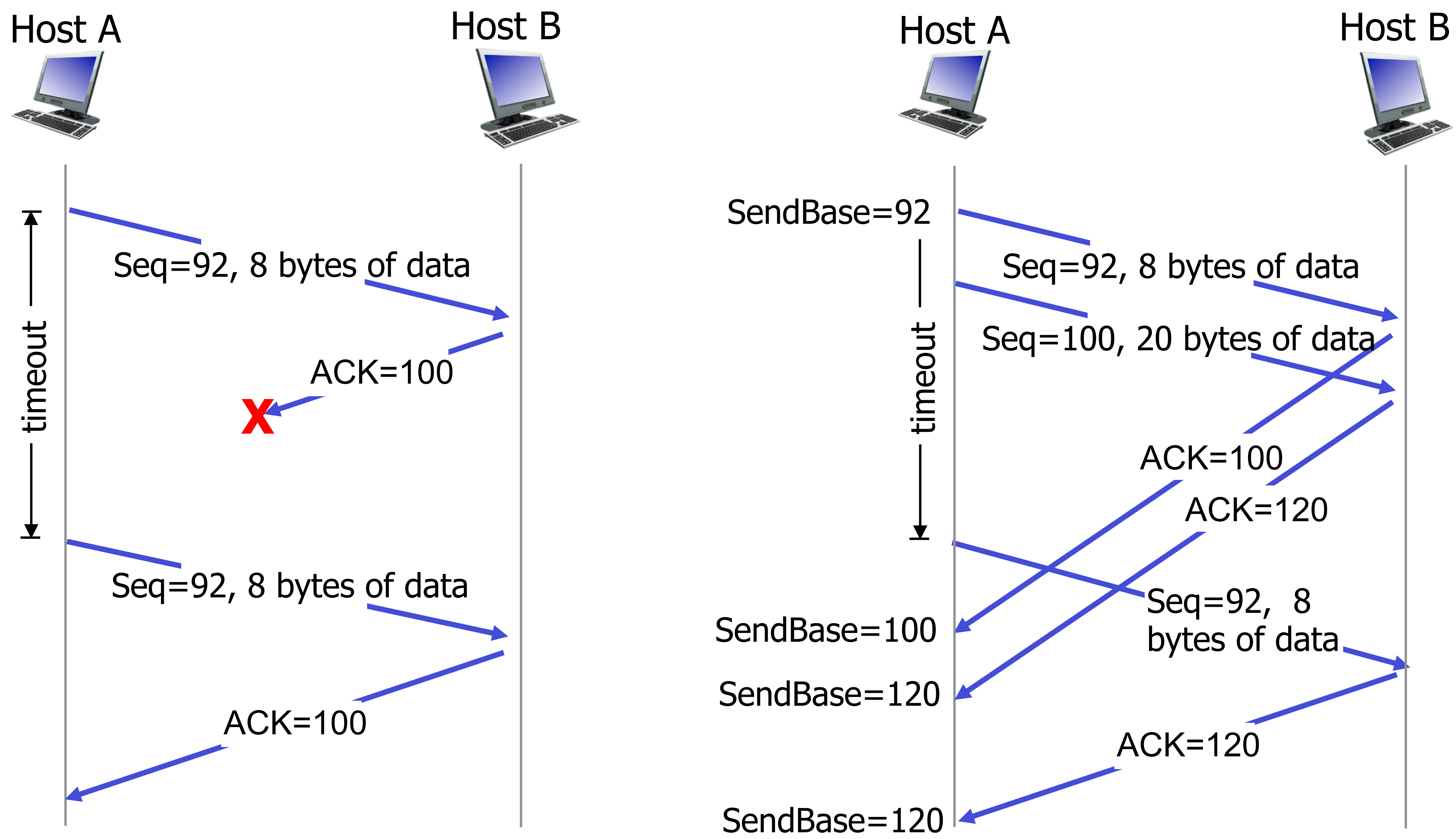
---

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
   }
```

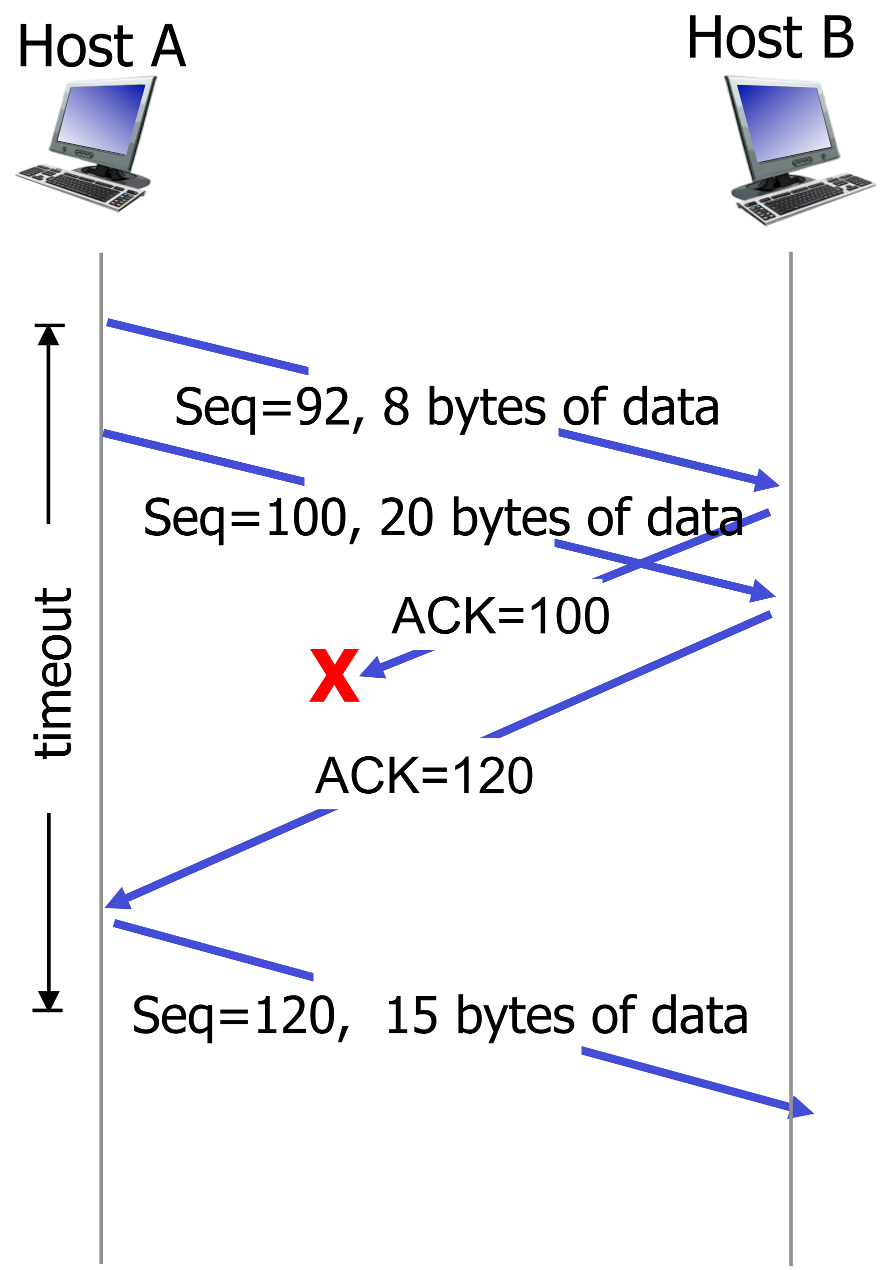
TCP: retransmission scenarios



lost ACK scenario

premature timeout

TCP retransmission scenarios (more)



cumulative ACK

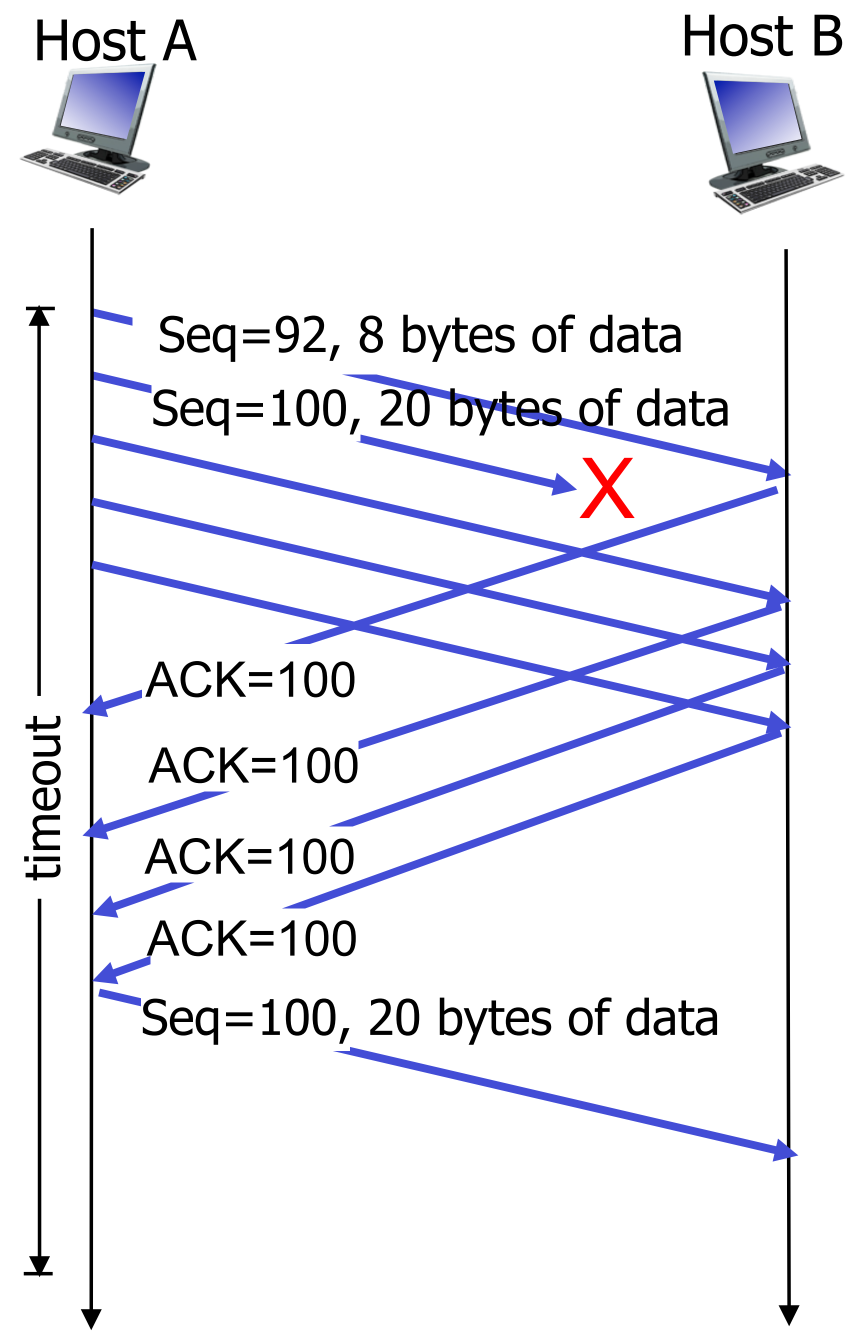
TCP ACK generation [RFC 1122, RFC 2581]

Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK. Wait up to 500ms 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 *duplicate ACK*, 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

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.
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - fast retransmit: resend segment before timer expires

Fast Retransmit

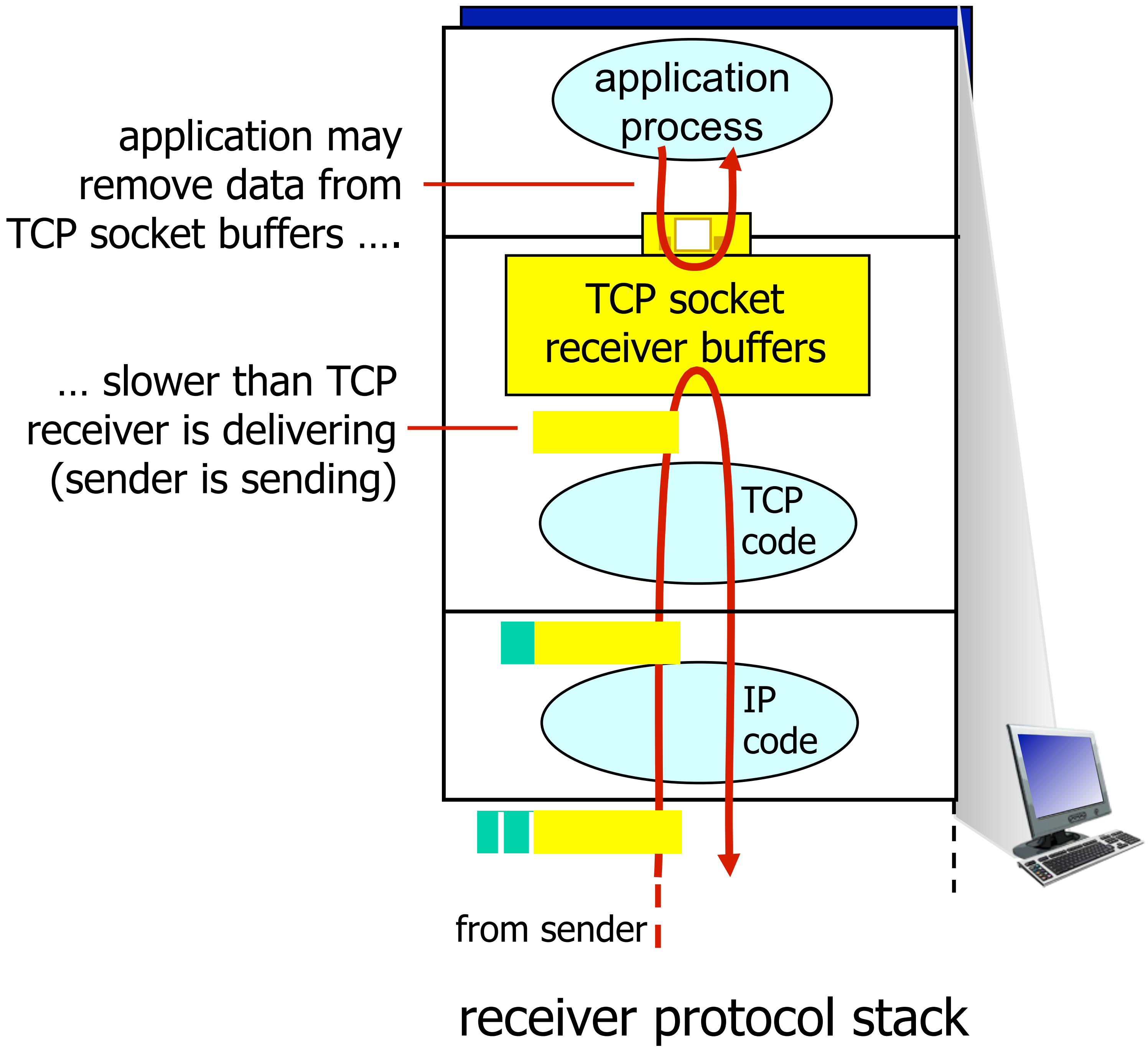


fast retransmit after sender receipt of triple duplicate ACK

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
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

TCP Flow Control

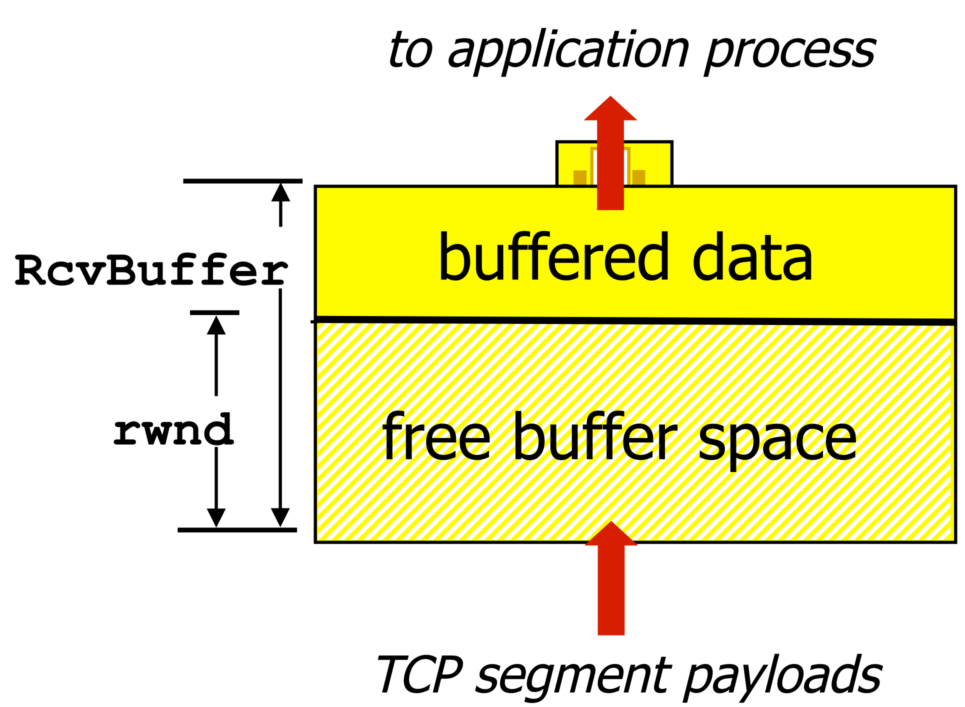


flow control

receiver controls sender, so sender won't overflow receiver's buffer by transmitting too much, too fast

TCP Flow control: how it works

- Receiver "advertises" free buffer space by including **`rwnd`** value in TCP header of receiver-to-sender segments
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems autoadjust **`RcvBuffer`**
- sender limits amount of unacked ("in-flight") data to receiver's **`rwnd`** value
- guarantees receive buffer will not overflow



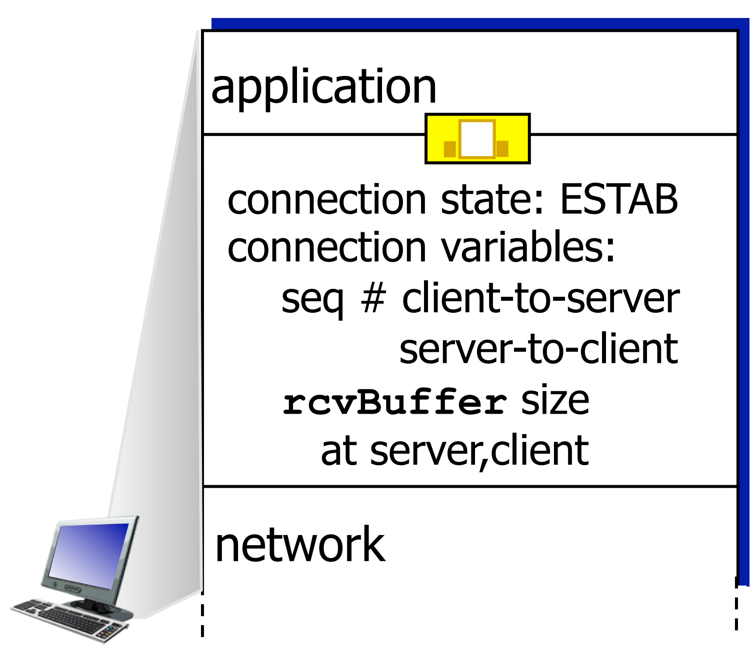
receiver-side buffering

Chapter 3 outline

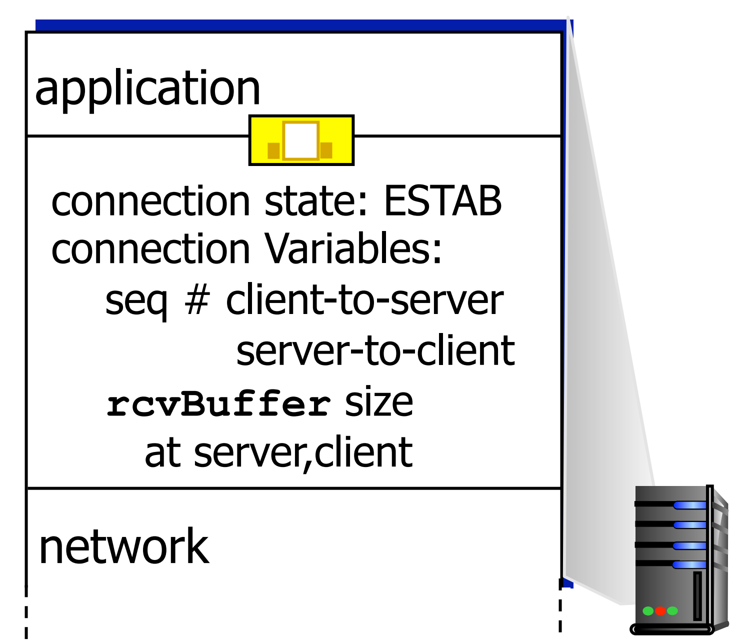
- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

TCP Connection Management

- Before exchanging data, sender/receiver "handshake":
  - agree to establish connection (each knowing the other willing to establish connection)
  - agree on connection parameters



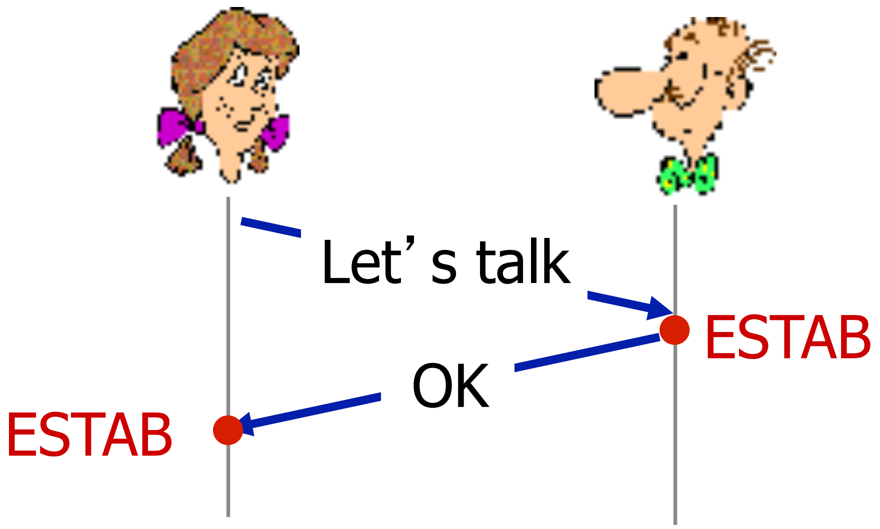

```
Socket clientSocket =
  newSocket("hostname","port
  number");
```

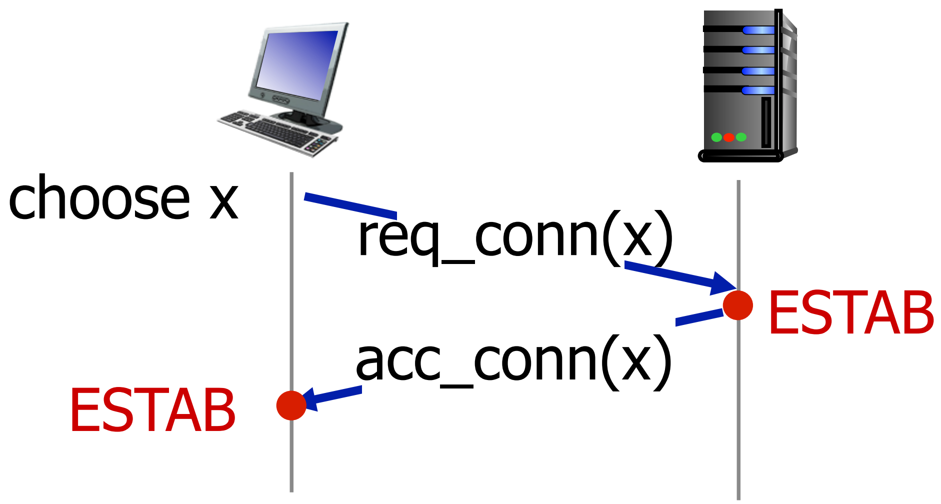



```
Socket connectionSocket =
  welcomeSocket.accept();
```

Agreeing to Establish a Connection

2-way handshake:

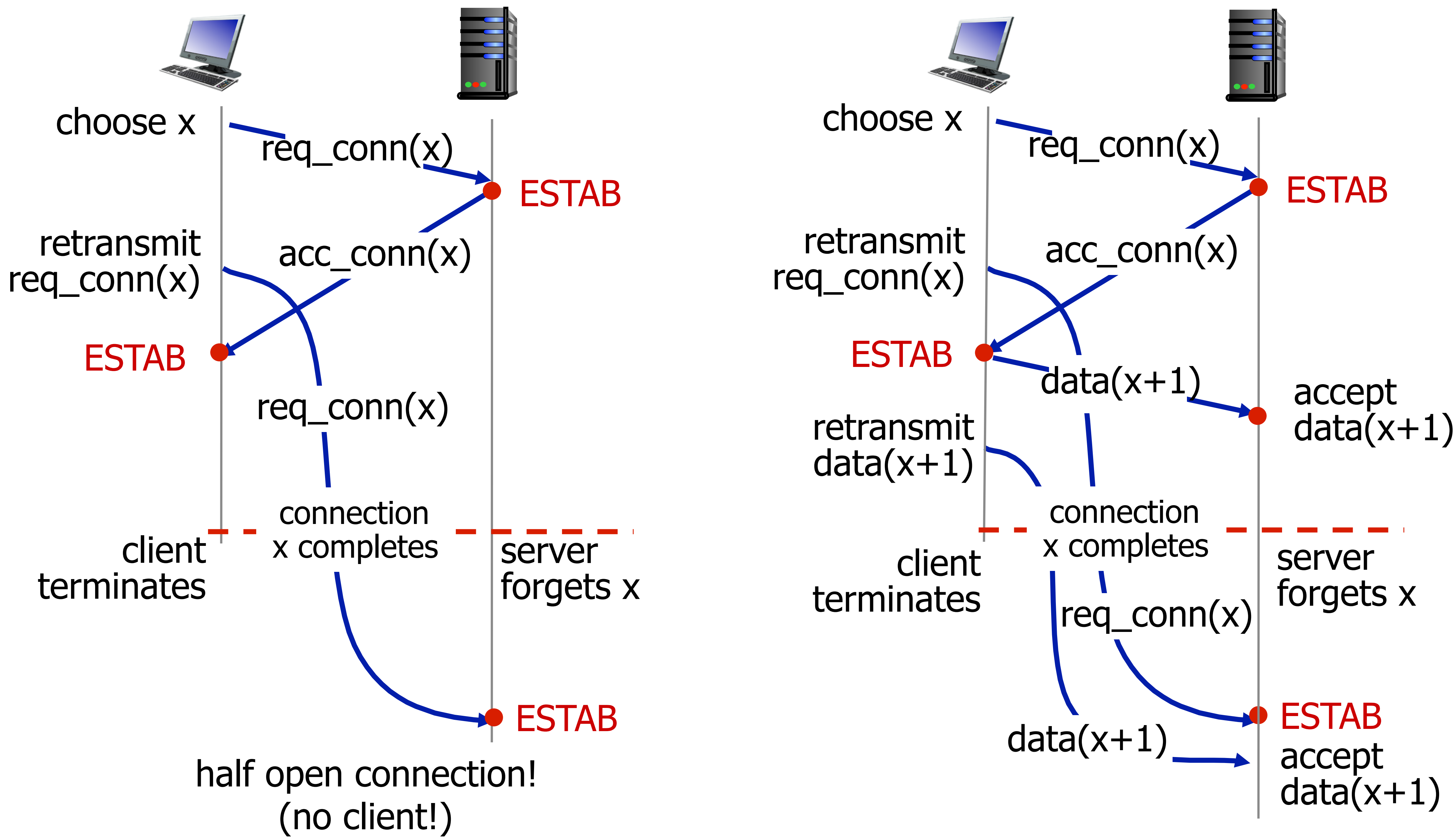




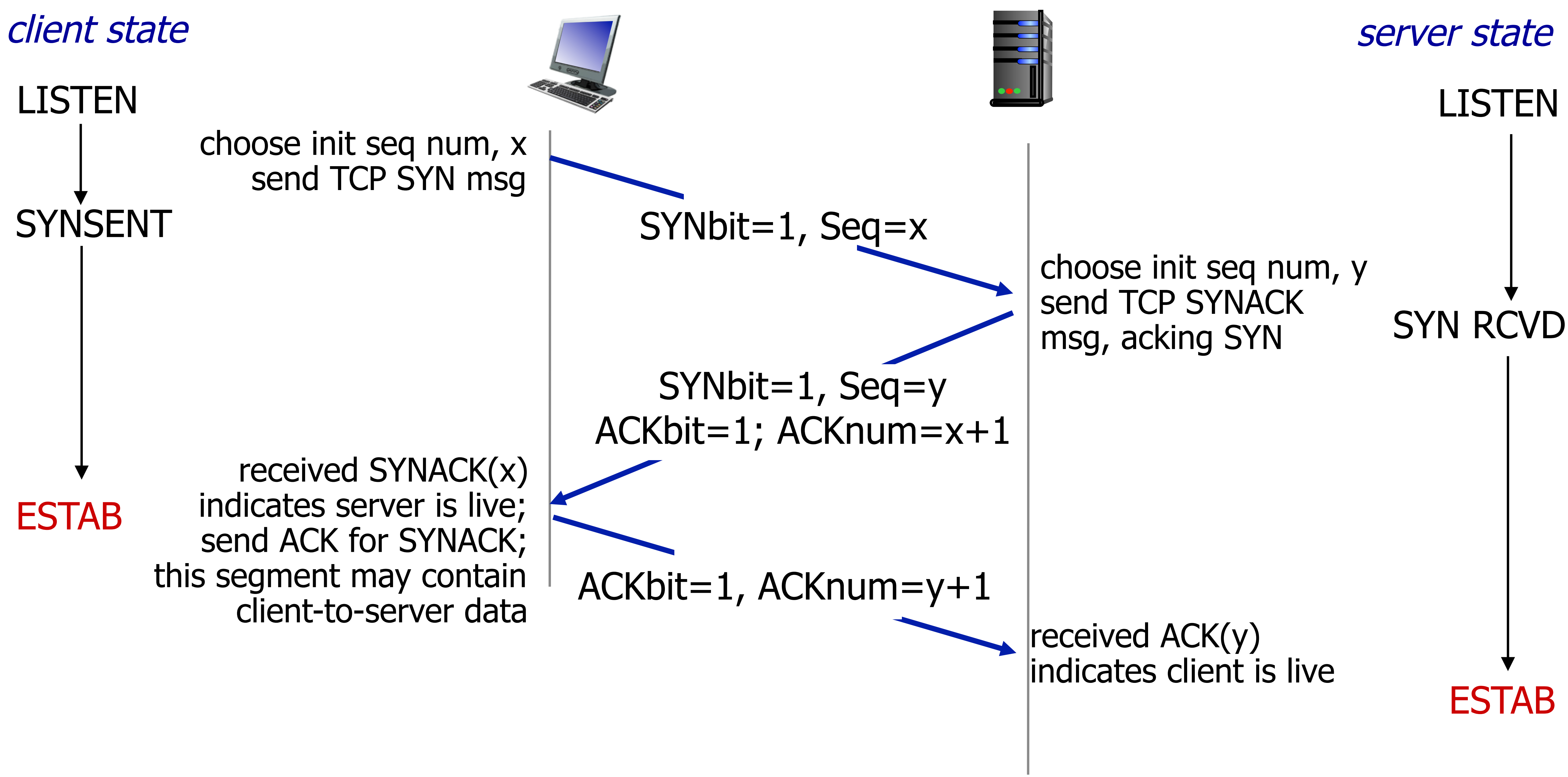
- Q: Will 2-way handshake always work in network?
  - variable delays
  - retransmitted messages (e.g. req_conn(x)) due to message loss
  - message reordering
  - can’t “see” other side

Agreeing to Establish a Connection

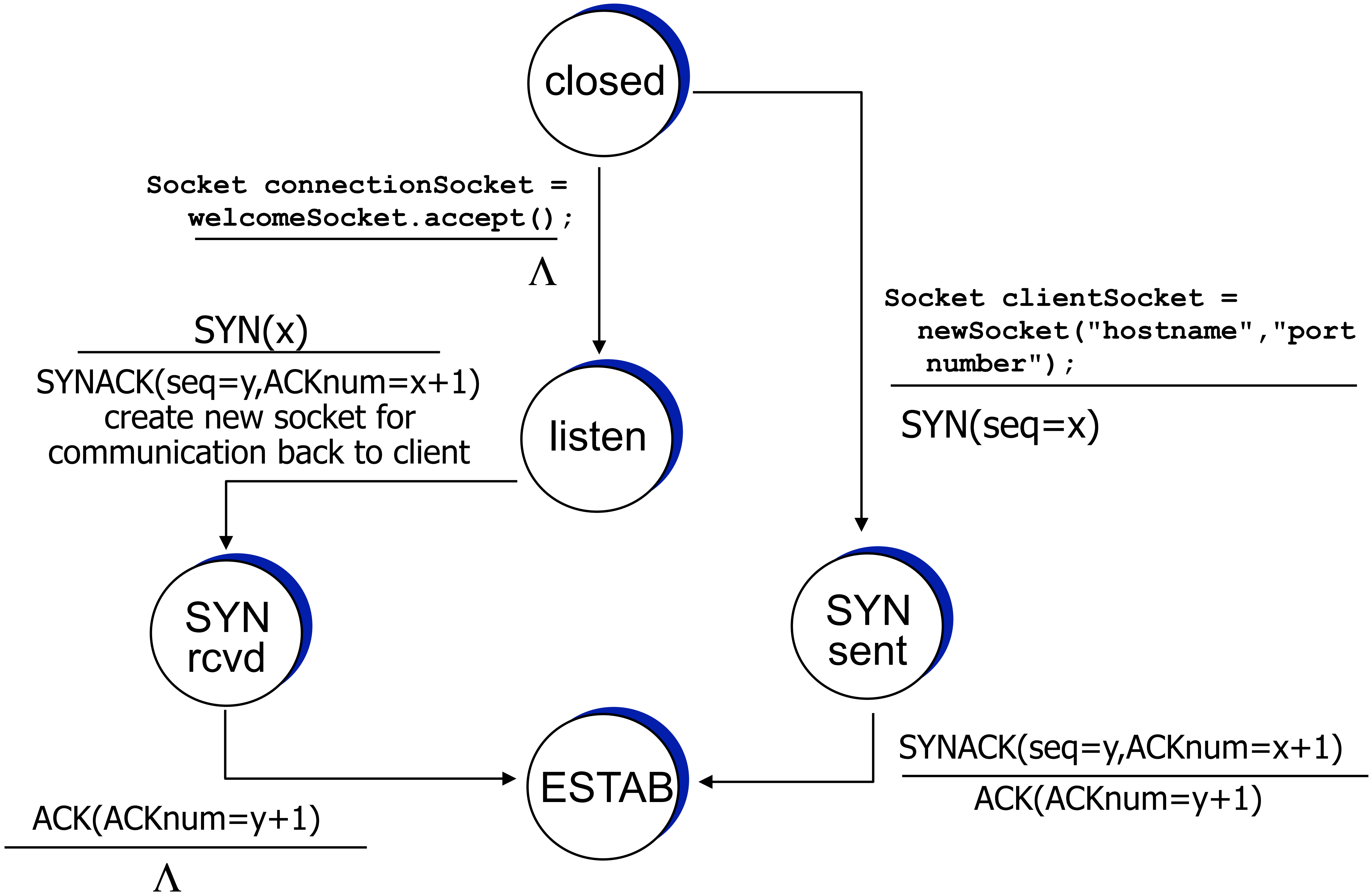
2-way handshake failure scenarios:



TCP 3-Way Handshake



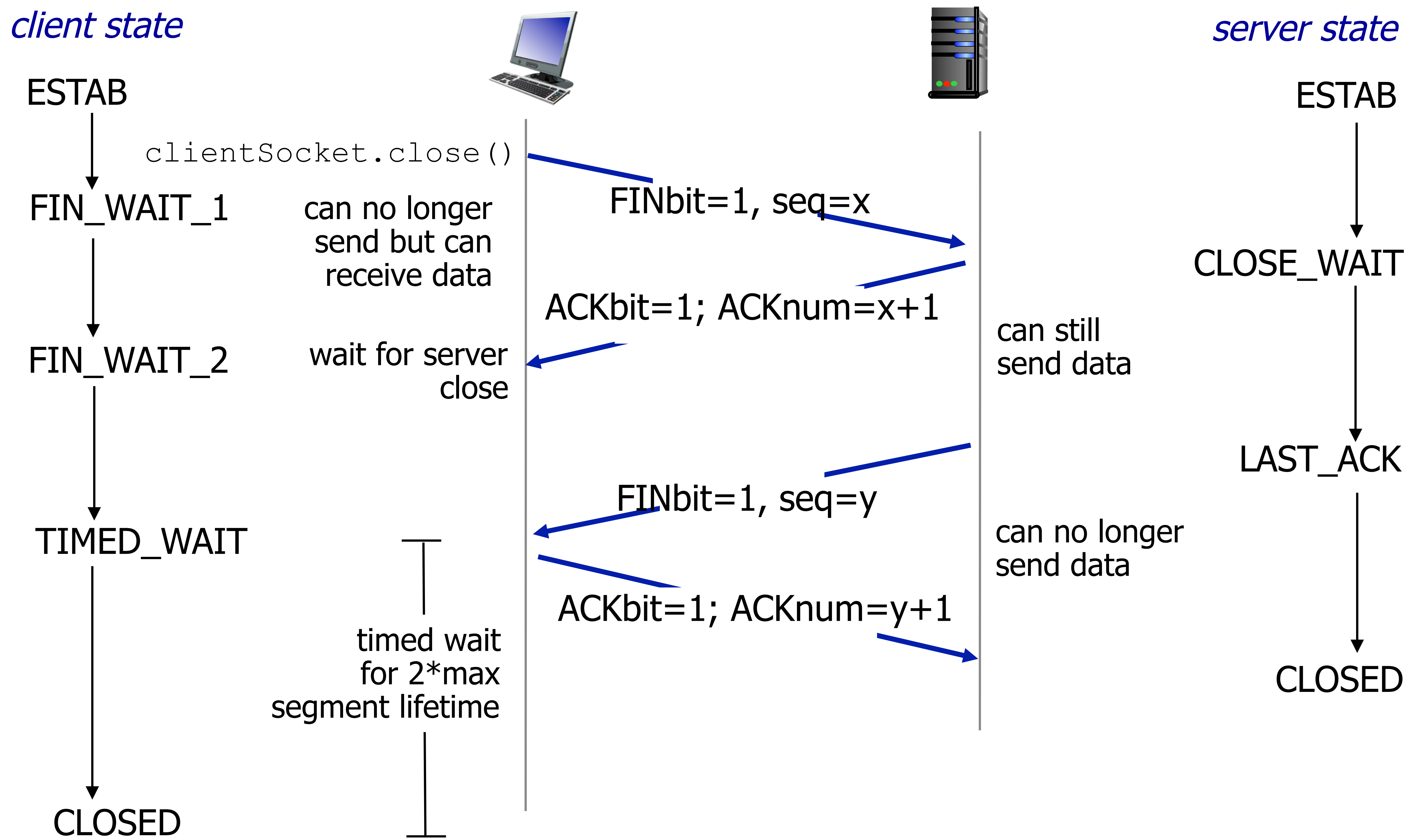
TCP 3-Way Handshake: FSM



TCP: Closing a Connection

- Client, Server each close their side of the connection
  - Send TCP segment with FIN bit = 1
- Respond to received FIN with FIN, ACK

TCP Connection Management (cont.)



SYN Flooding

- Classic Internet attack sends a huge number of SYN packets to a host, but never responds with the third handshake message.
- In so doing, an adversary forces a receiver to dedicate a huge amount of resources to bogus requests.
  - And therefore makes those resources unavailable to legitimate users.
- There are ways to prevent this (SYN Cookies), but a surprising number of systems are still vulnerable.

Next Time

- Read Section 3.6 and 3.7
  - Congestion control in TCP