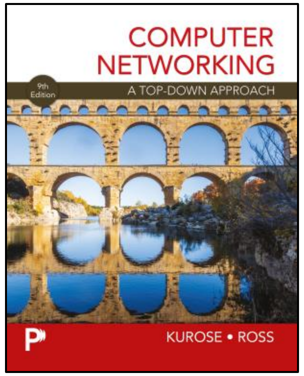


CSC 361
Computer Networks
Transport Layer
(Chapter 3 – TCP Flow Ctrl & Conn Management)

Wenjun Yang

Summer 2026

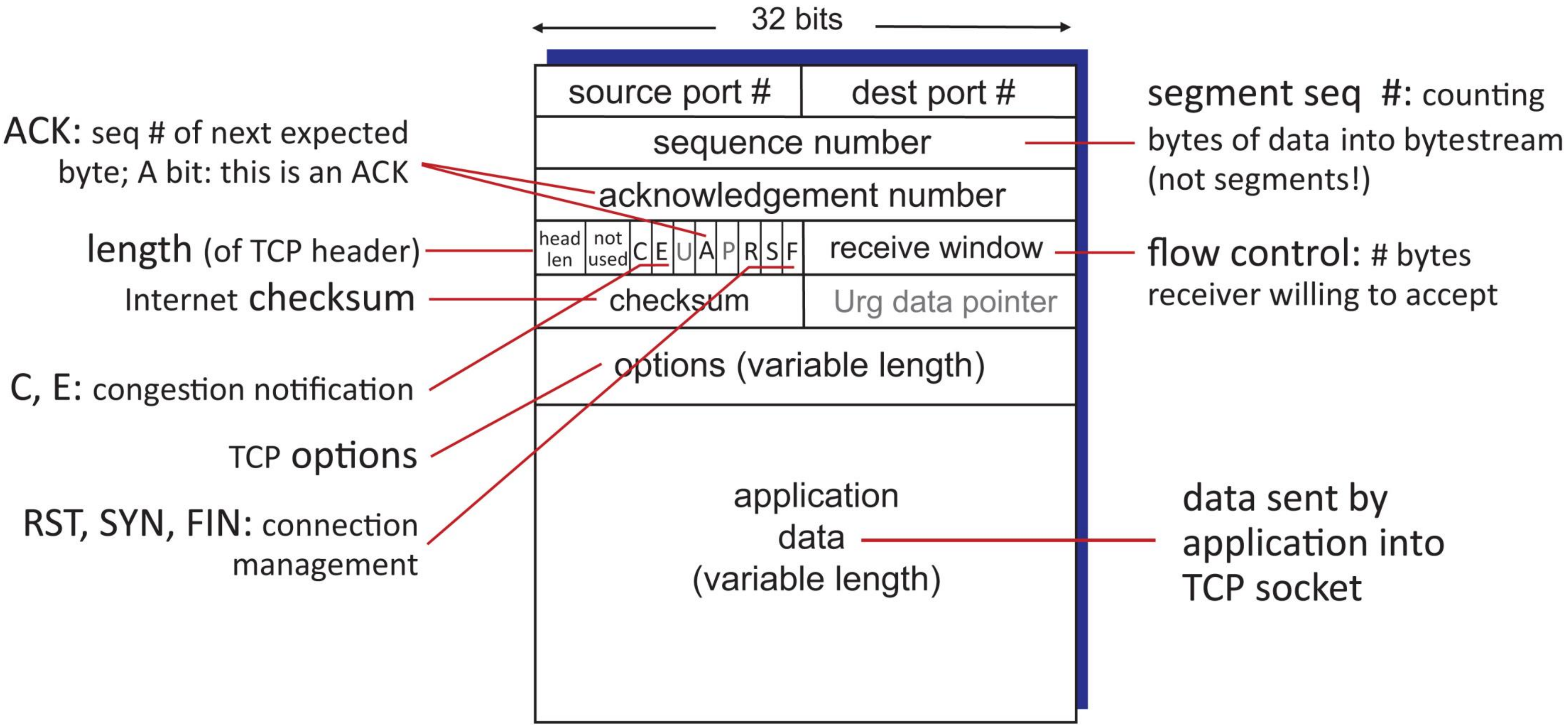
Chapter 3: Roadmap (4 of 7)

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- Principles of congestion control
- TCP congestion control

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
- pipelining:
  - TCP congestion and flow control set window size
- connection-oriented:
  - handshaking (exchange of control messages) initializes sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP Segment Structure



TCP Sequence Numbers, ACKs (1 of 2)

Sequence numbers:

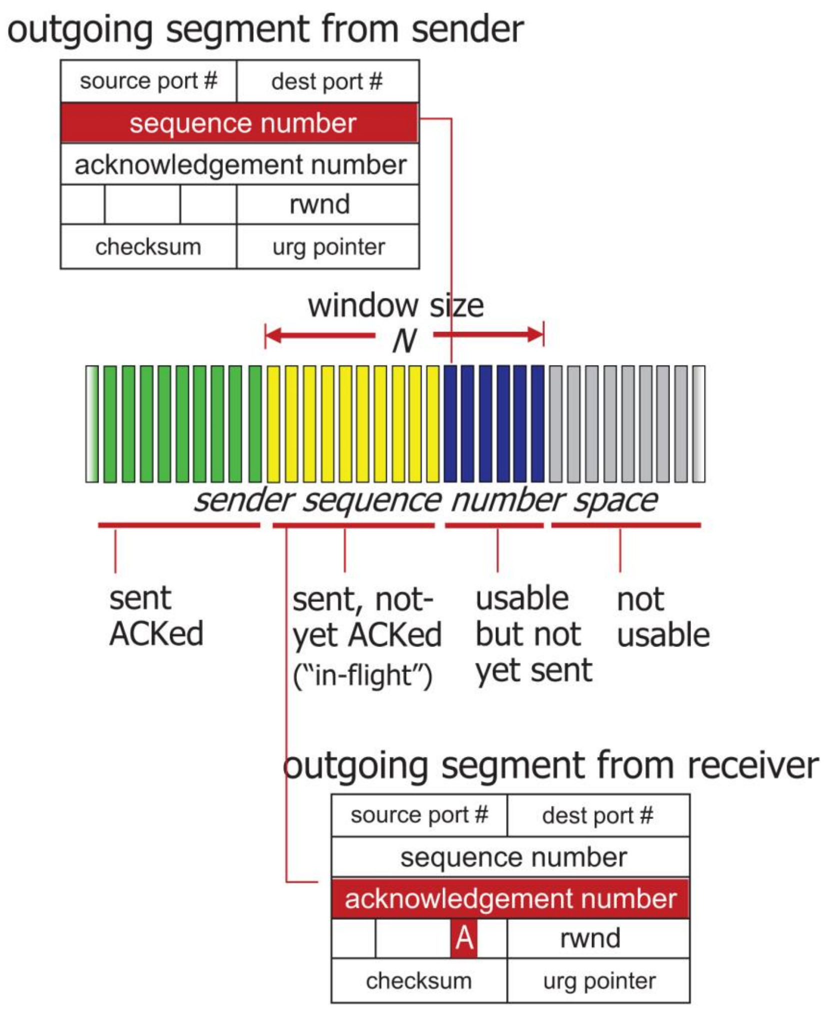
- byte stream "number" of first byte in segment's data

Acknowledgements:

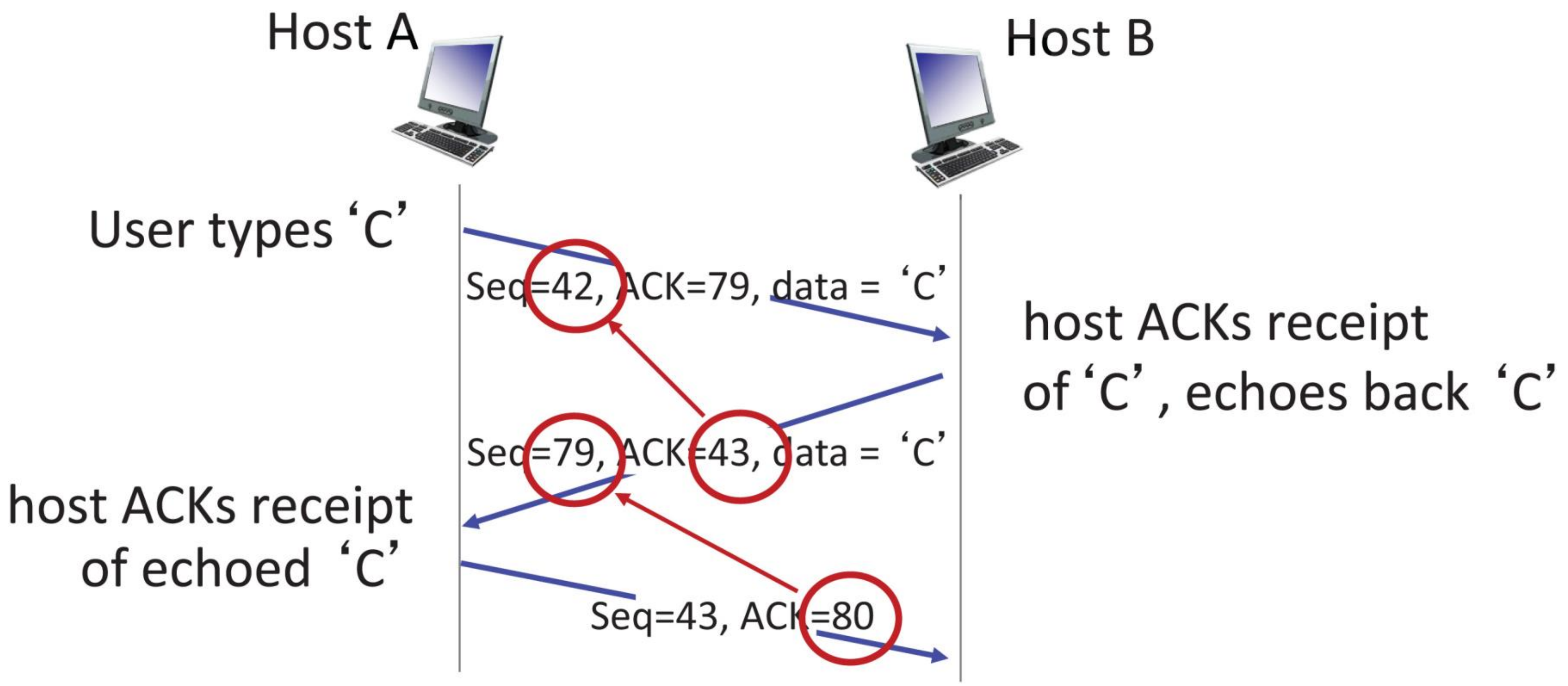
- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

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



TCP Sequence Numbers, ACKs (2 of 2)



simple telnet scenario

TCP Round Trip Time, Timeout (1 of 3)

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

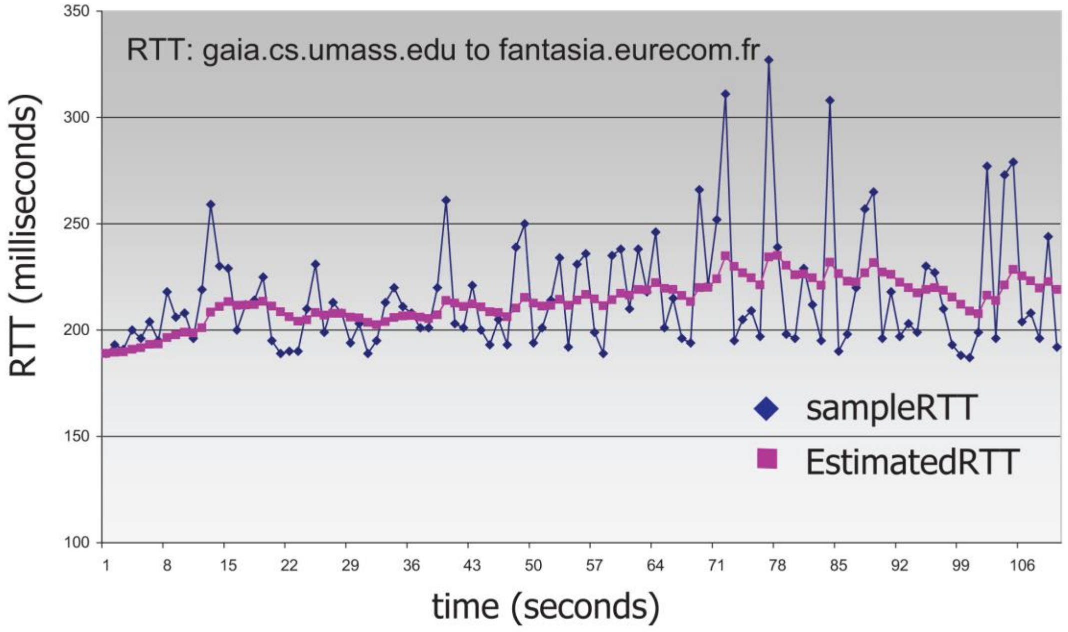
Q: how to estimate RTT?

- `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 (2 of 3)

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

- **e**xponential **w**eighted **m**oving average (EWMA)
- influence of past sample decreases exponentially fast
- typical value: $\alpha = 0.125$



TCP Round Trip Time, Timeout (3 of 3)

- 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**:

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

(typically, β = 0.25)

* Check out the online interactive exercises for more examples:

http://gaia.cs.umass.edu/kurose_ross/interactive/

TCP Sender (Simplified) (1 of 2)

event: data received from application

- 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`

event: timeout

- retransmit segment that caused timeout
- restart timer

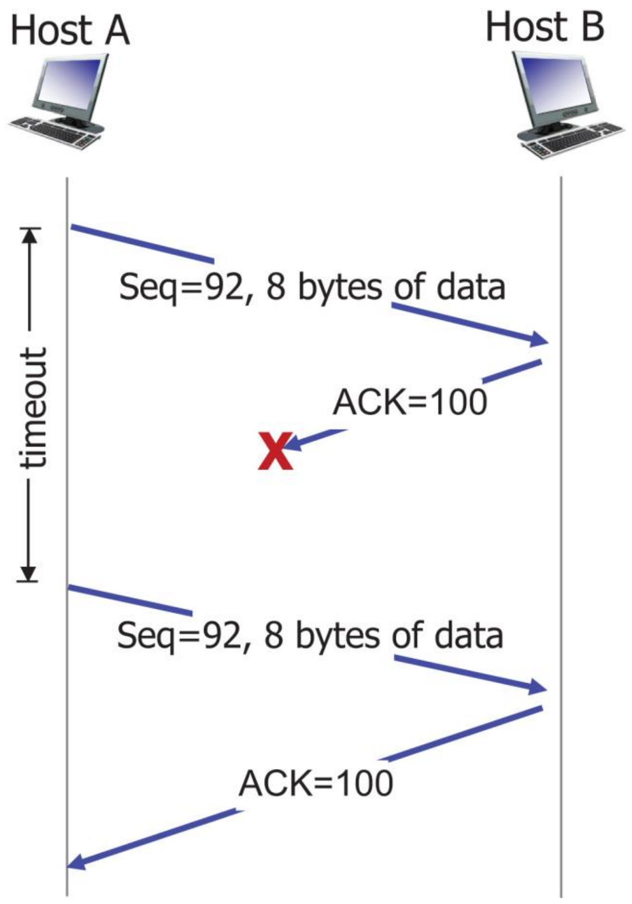
event: ACK received

- if ACK acknowledges previously unACKed segments
  - update what is known to be ACKed
  - start timer if there are still unACKed segments

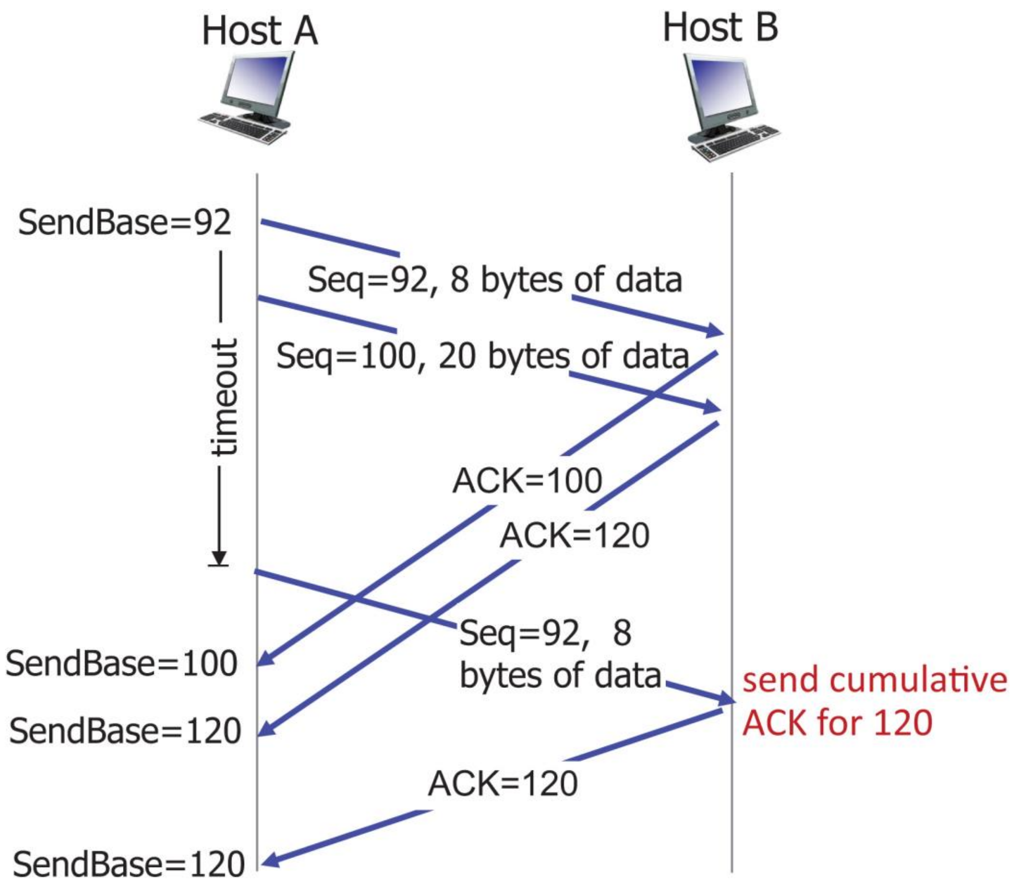
TCP Receiver: ACK Generation [RFC 5681]

Event at receiver	*TCP receiver action*

TCP: Retransmission Scenarios (1 of 2)

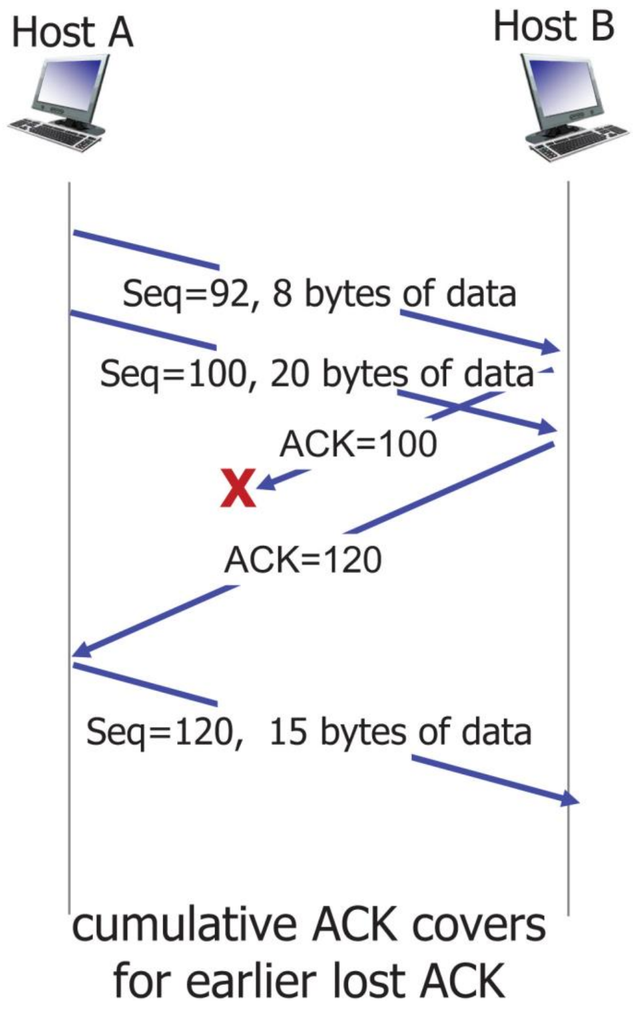


lost ACK scenario



premature timeout

TCP: Retransmission Scenarios (2 of 2)

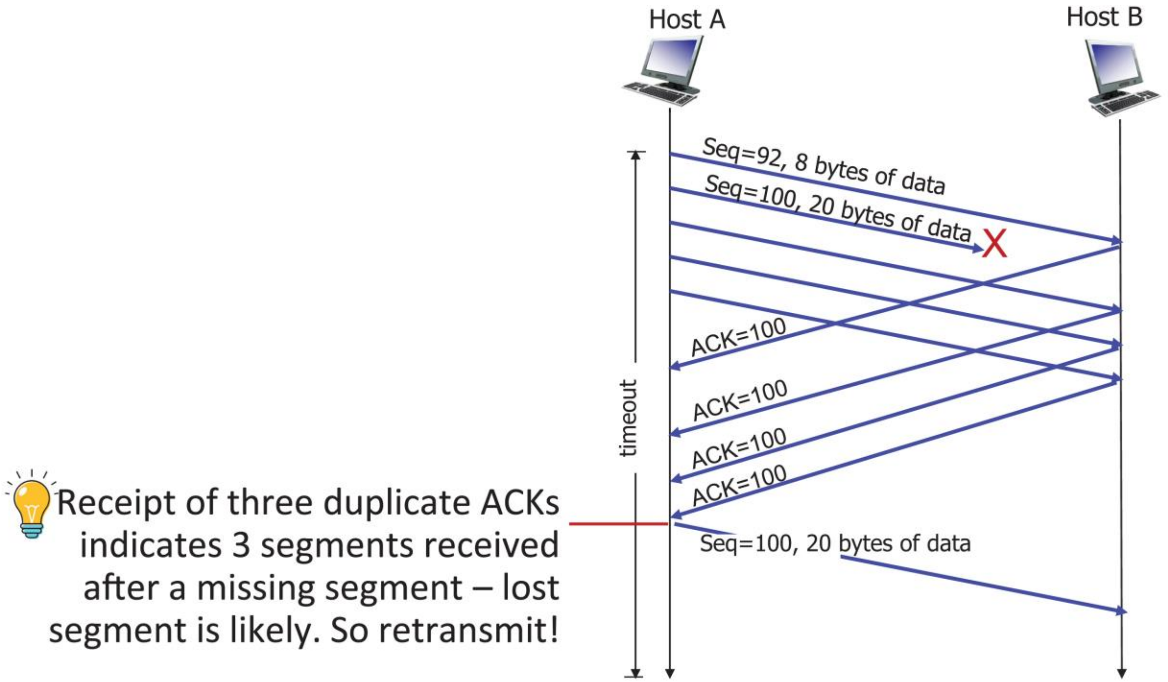


TCP Fast Retransmit

TCP fast retransmit

if sender receives 3 additional ACKs for same data (“triple duplicate ACK s”), resend unACKed segment with smallest seq #

- likely that unACKed segment lost, so don’t wait for timeout

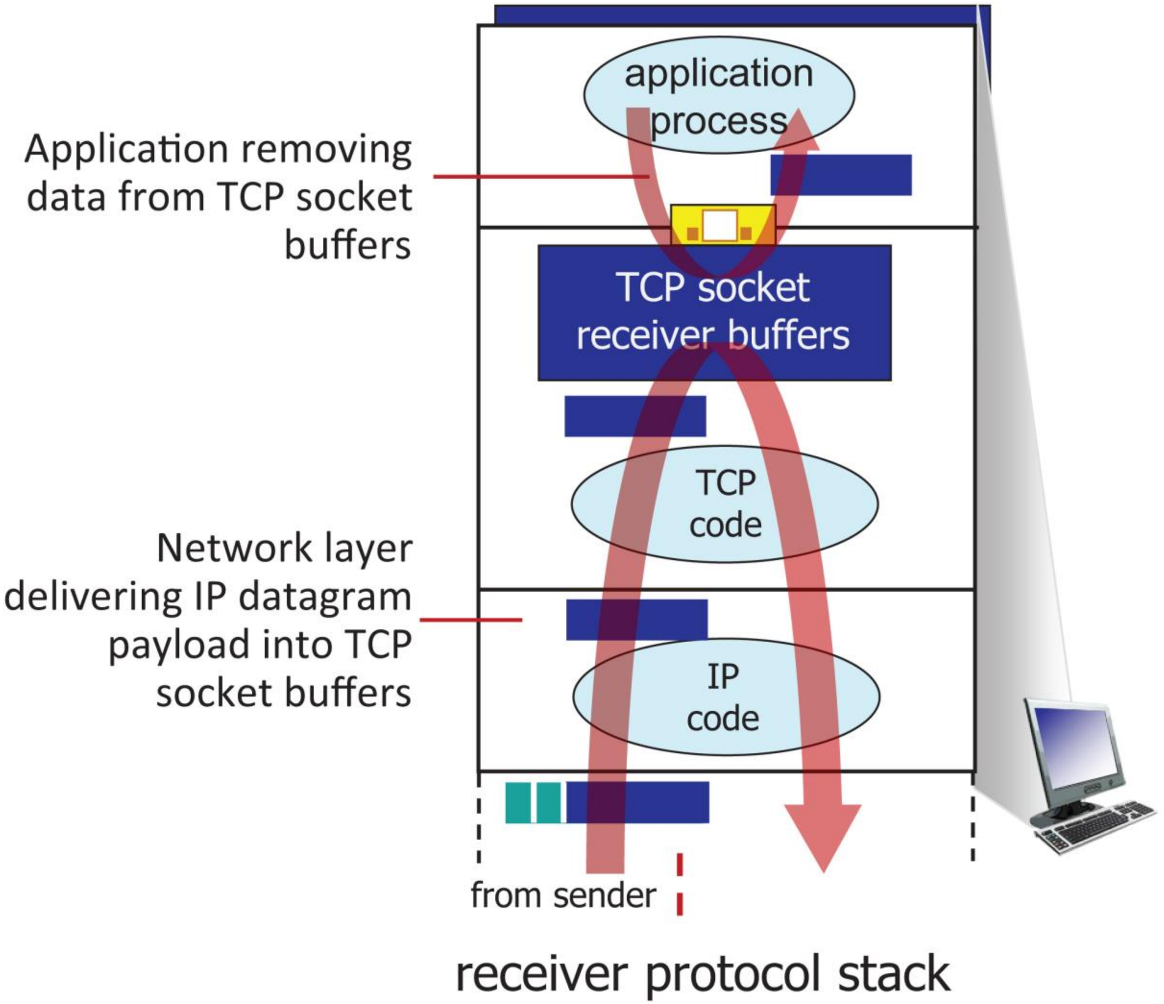


Chapter 3: Roadmap (5 of 7)

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- Principles of congestion control
- TCP congestion control

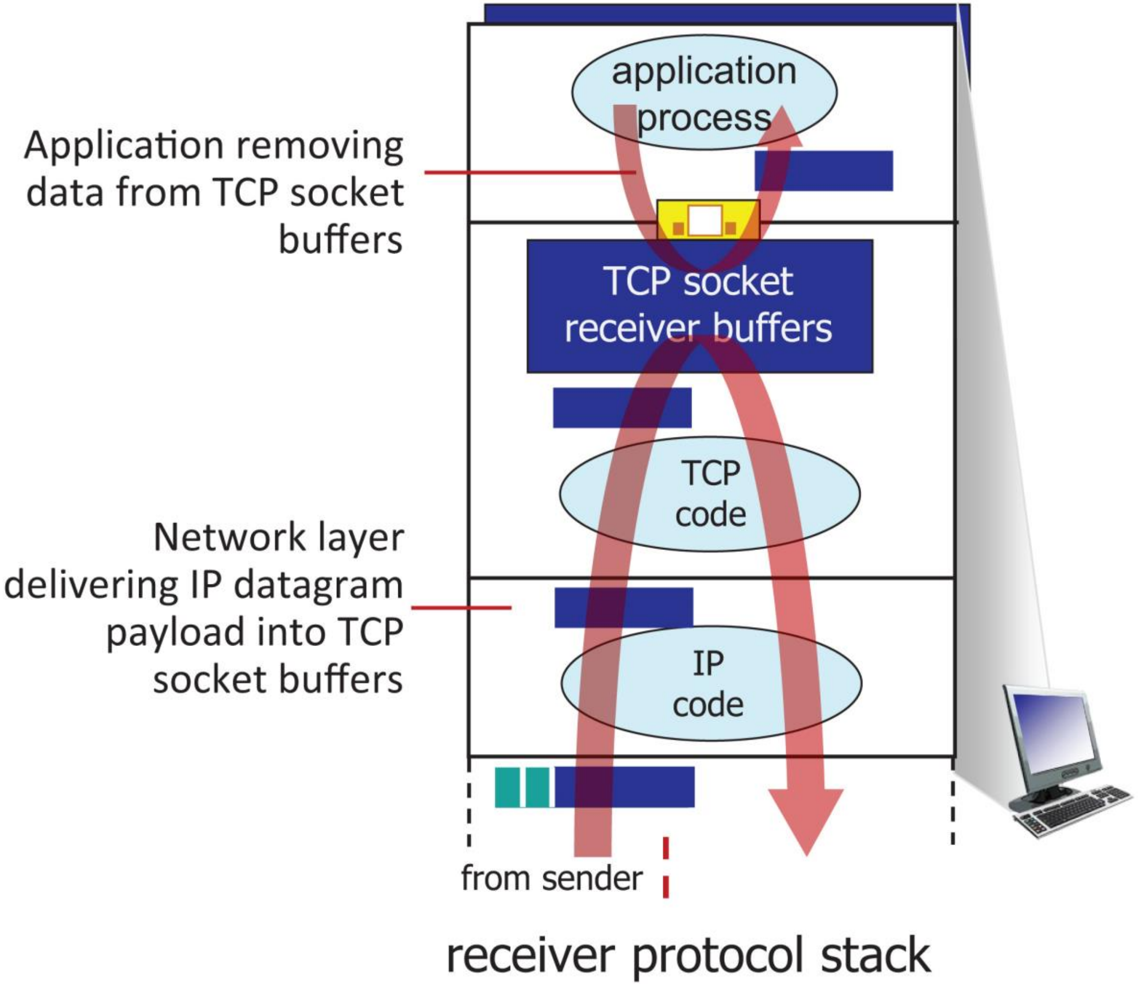
TCP Flow Control (1 of 6)

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?



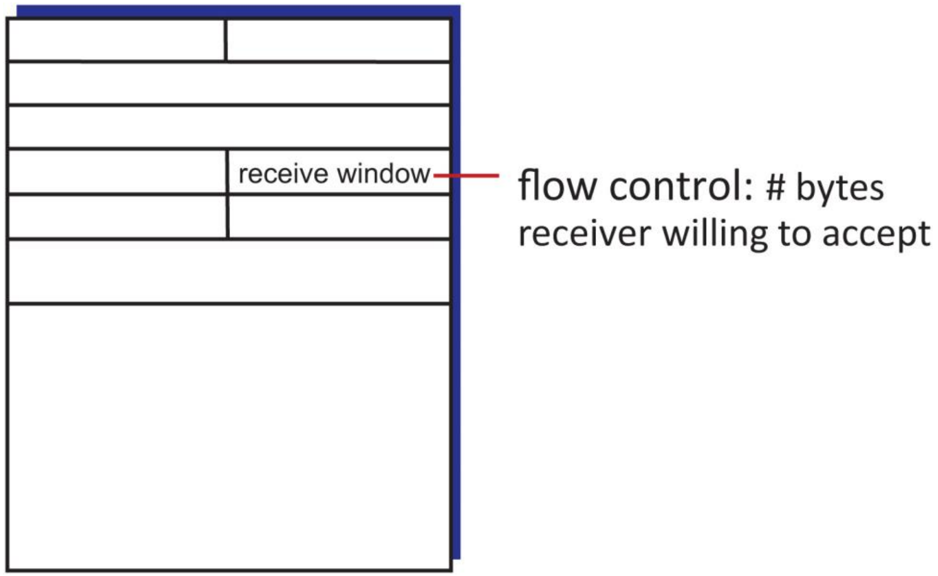
TCP Flow Control (2 of 6)

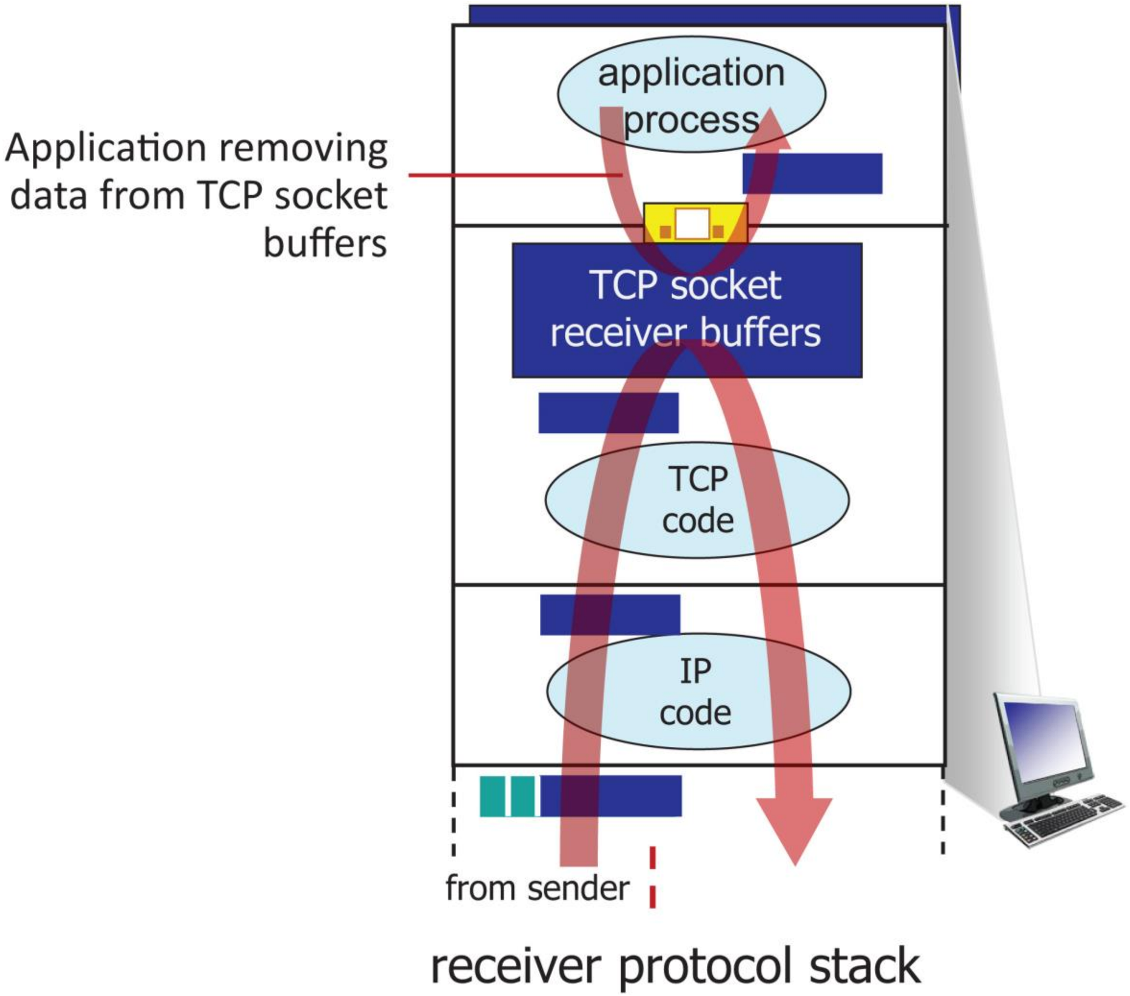
Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?



TCP Flow Control (3 of 6)

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?



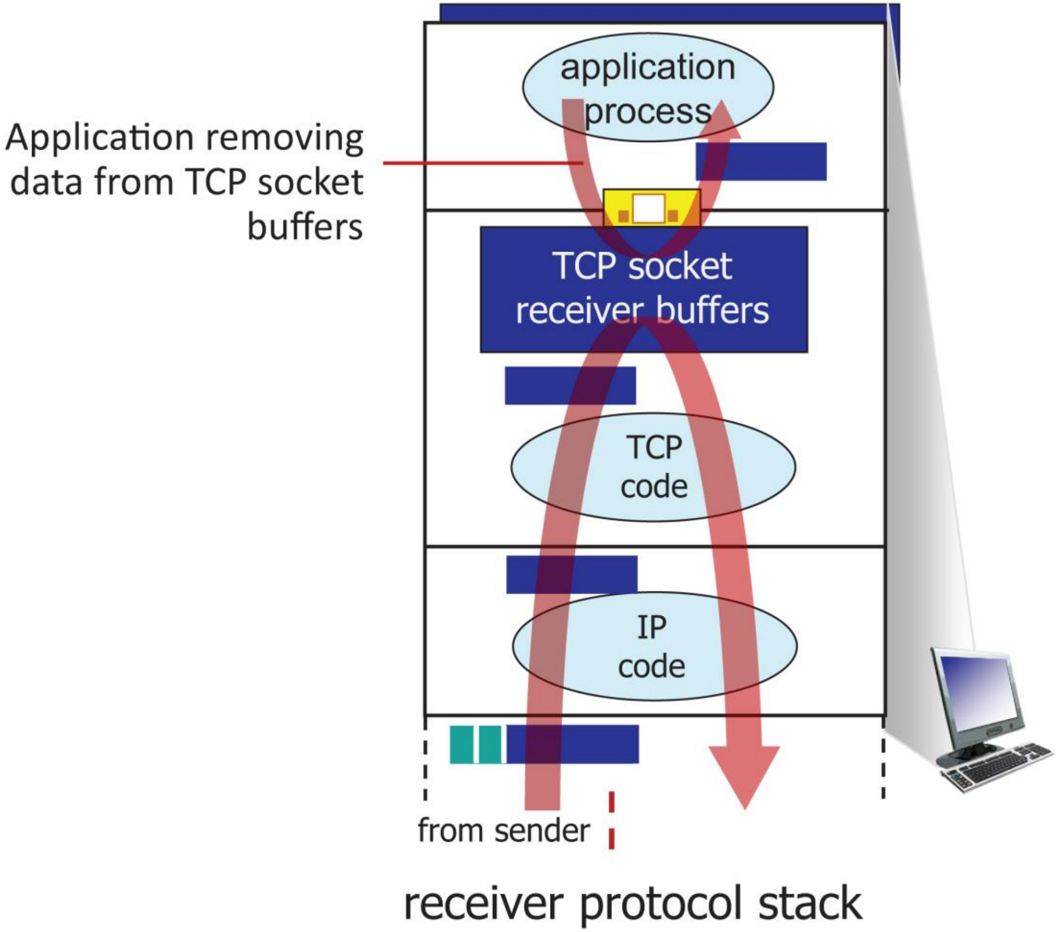


TCP Flow Control (4 of 6)

Q: What happens if network layer delivers data faster than application layer removes data from socket buffers?

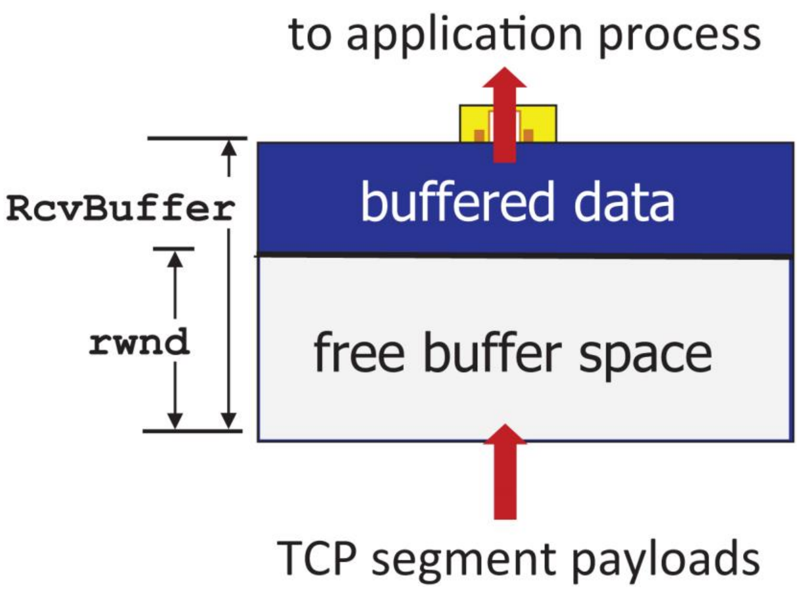
flow control

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



TCP Flow Control (5 of 6)

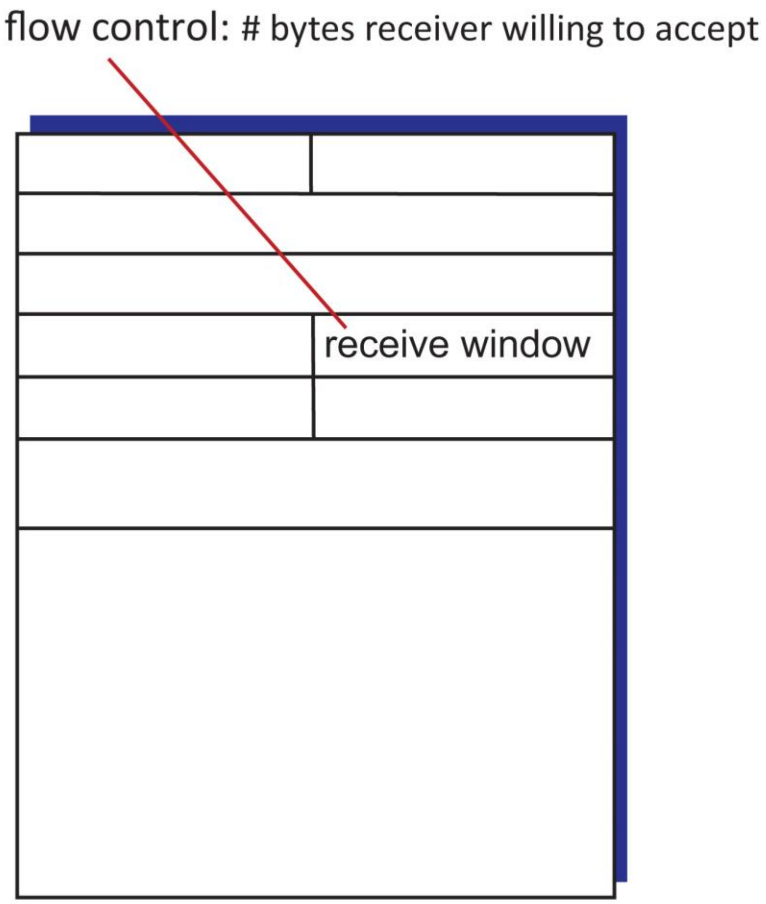
- TCP receiver "advertises" free buffer space in **`rwnd`** field in TCP header
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems auto-adjust **`RcvBuffer`**
- sender limits amount of unACKed ("in-flight") data to received **`rwnd`**
- guarantees receive buffer will not overflow



TCP receiver-side buffering

TCP Flow Control (6 of 6)

- TCP receiver "advertises" free buffer space in **`rwnd`** field in TCP header
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems auto-adjust **`RcvBuffer`**
- sender limits amount of unACKed ("in-flight") data to received **`rwnd`**
- guarantees receive buffer will not overflow

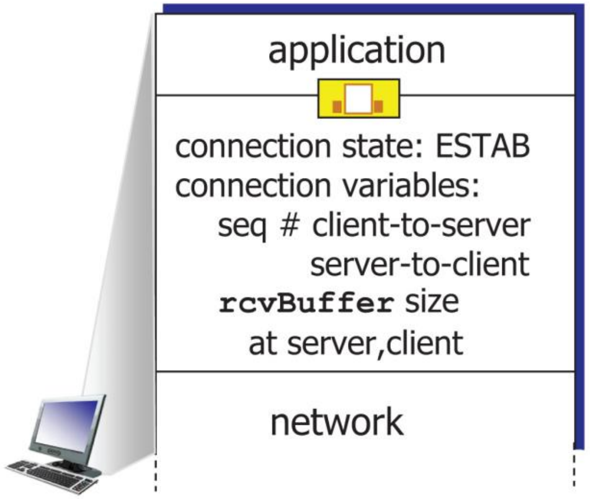


TCP segment format

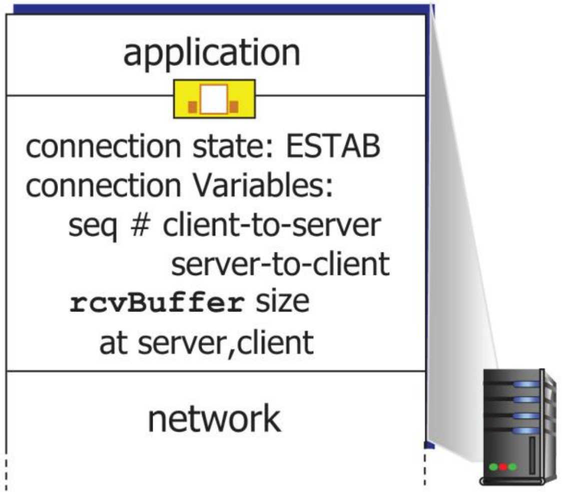
TCP Connection Management

before exchanging data, sender/receiver "handshake":

- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters (e.g., starting seq #s)

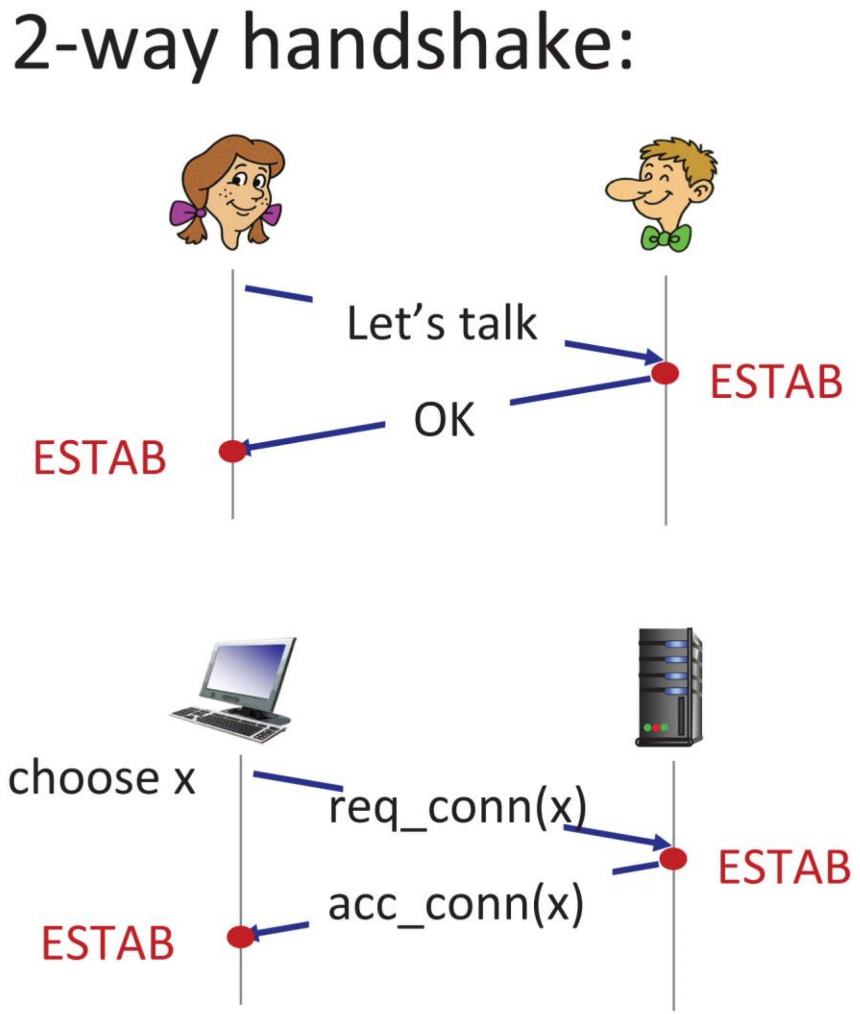



```
Socket clientSocket =
  newSocket("hostname","port number");
```




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

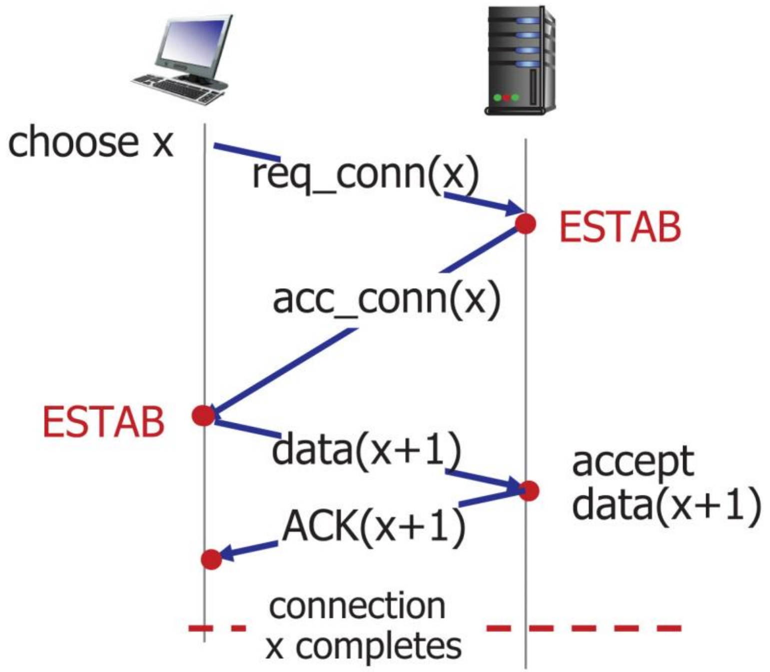
Agreeing to Establish a Connection



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

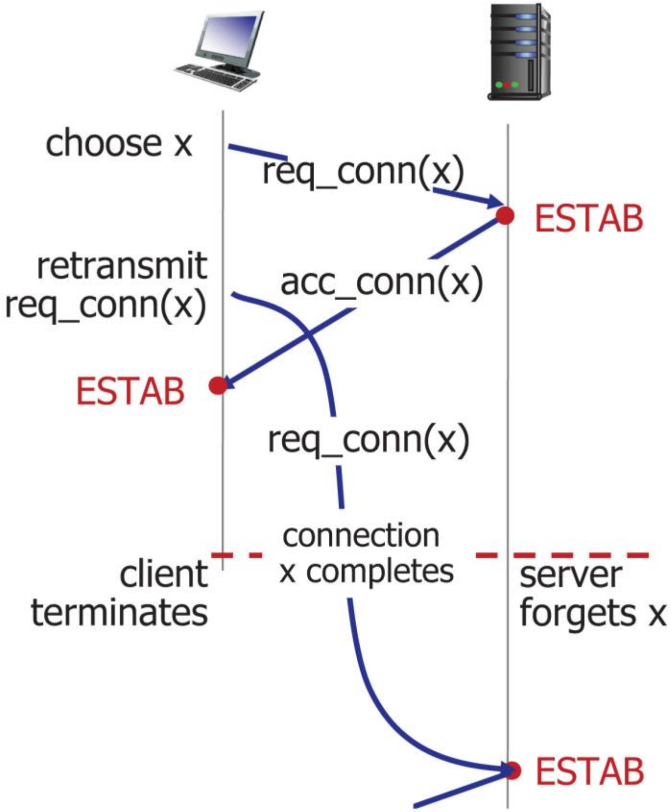
- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

2-way Handshake Scenarios (1 of 3)



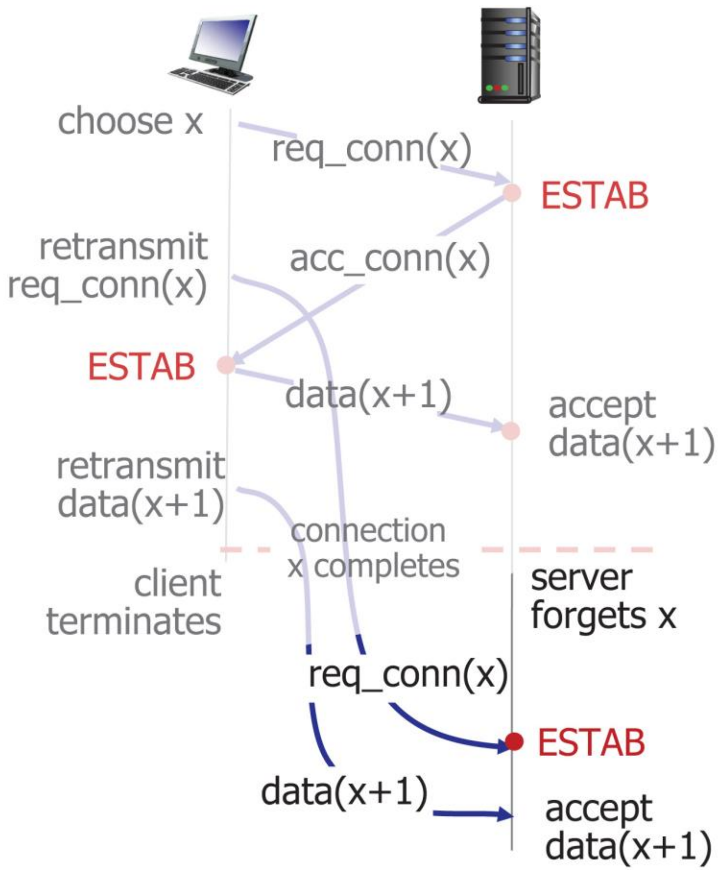
No problem!

2-way Handshake Scenarios (2 of 3)



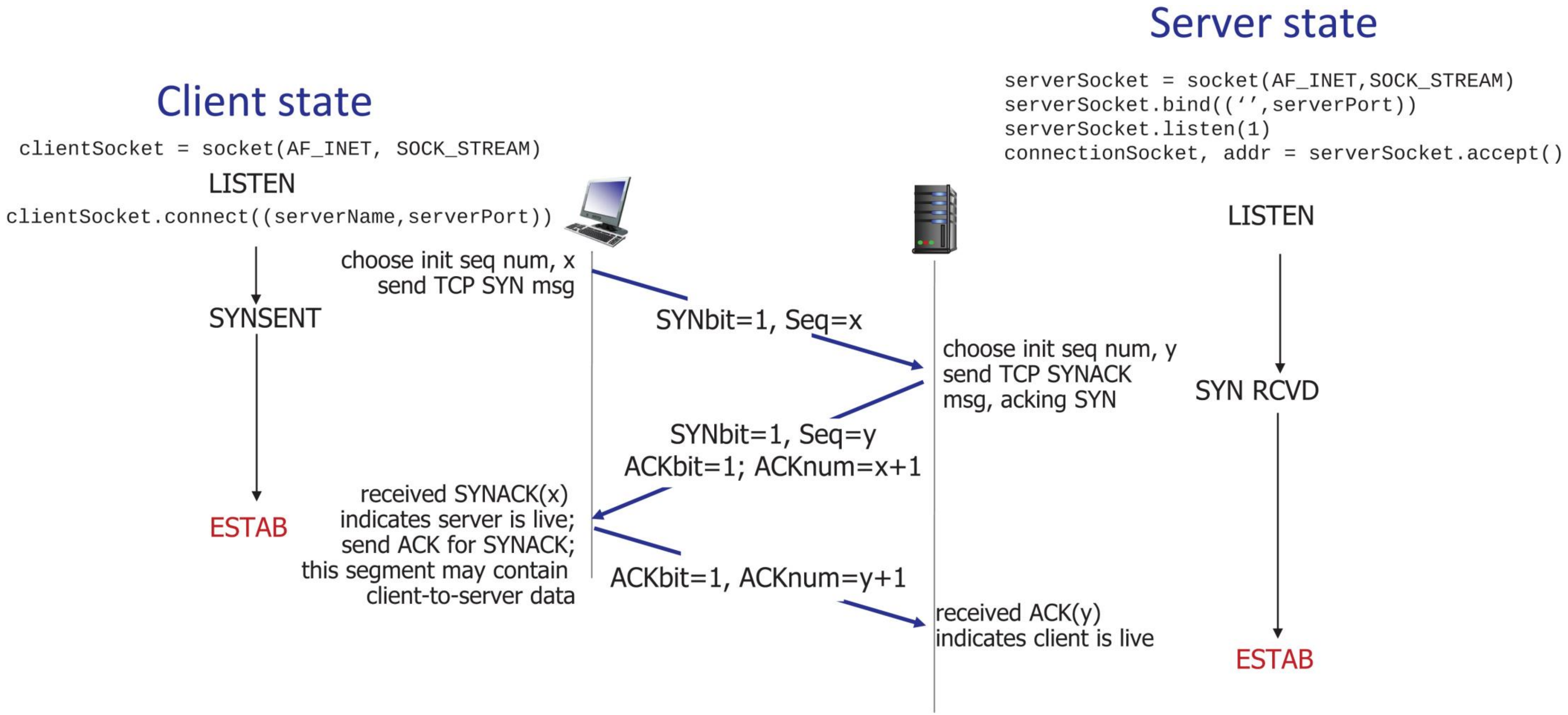
Problem: half open connection! (no client)

2-way Handshake Scenarios (3 of 3)

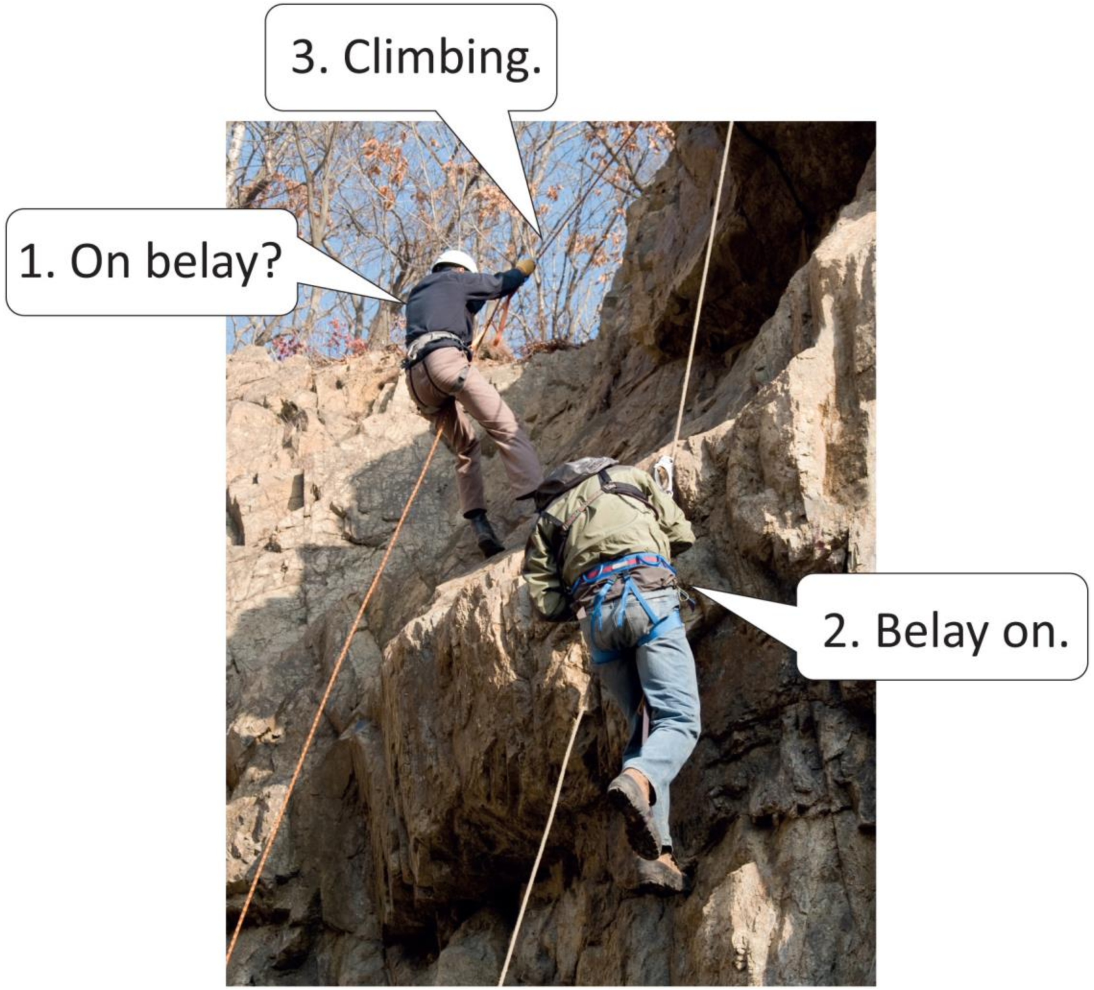


Problem: dup data accepted!

TCP 3-Way Handshake



A Human 3-Way Handshake Protocol



Closing a TCP Connection (1 of 2)

- 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

Closing a TCP Connection (2 of 2)