National University of Computer & Emerging Sciences

CS 3001 - COMPUTER NETWORKS

Lecture 14

Chapter 3

10th October, 2023

Nauman Moazzam Hayat

nauman.moazzam@lhr.nu.edu.pk

Office Hours: 02:30 pm till 06:00 pm (Every Tuesday & Thursday)

TCP Sender (simplified)

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

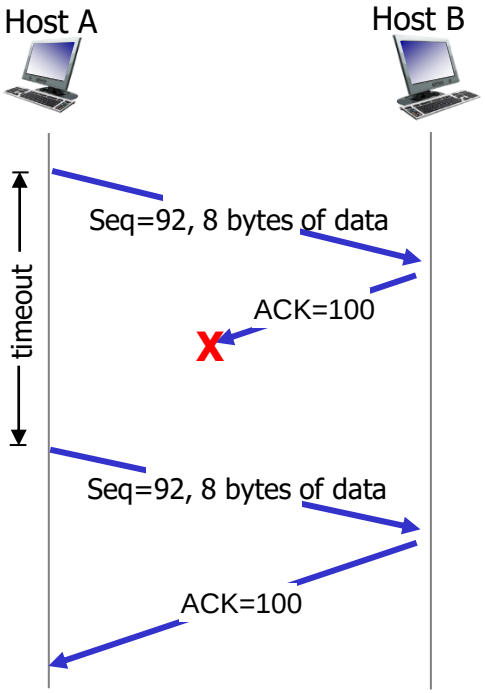
event: ACK received

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

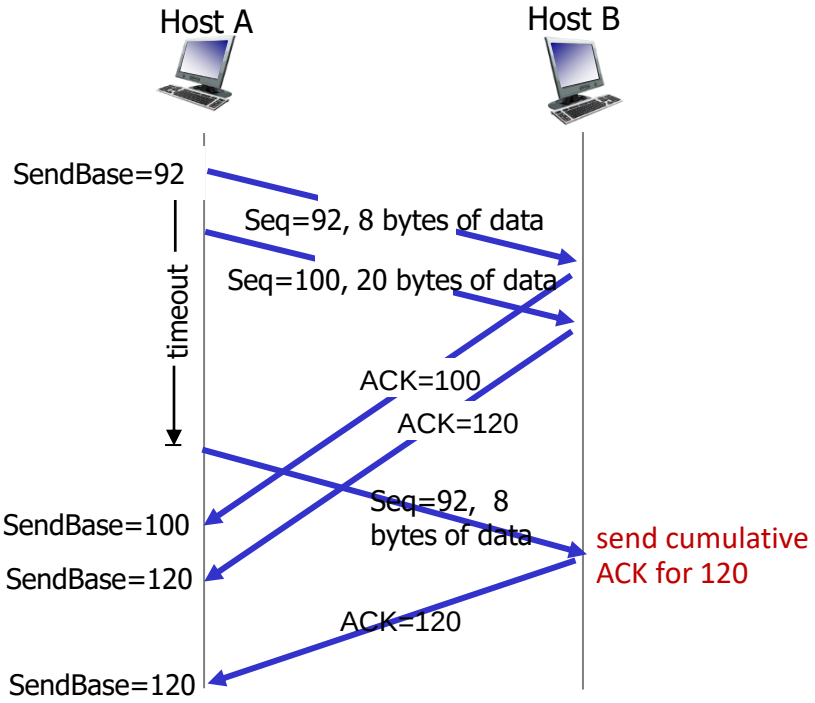
TCP Receiver: ACK generation [RFC 5681]

Event at receiver	*TCP receiver action*
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

TCP: retransmission scenarios

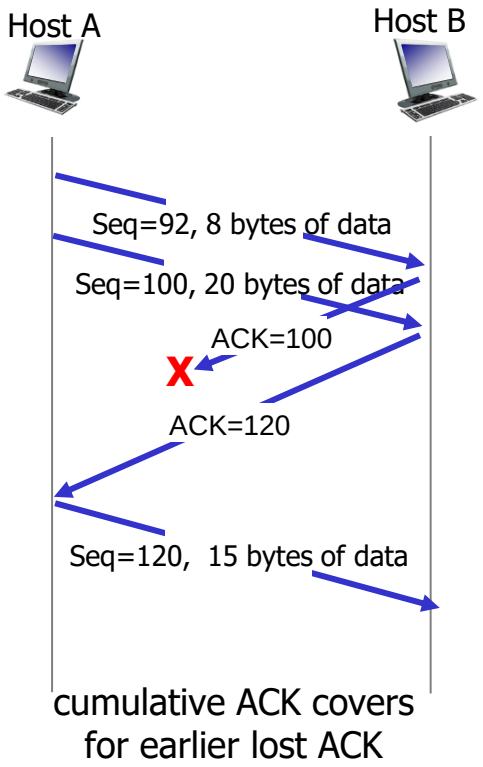


lost ACK scenario



premature timeout

TCP: retransmission scenarios



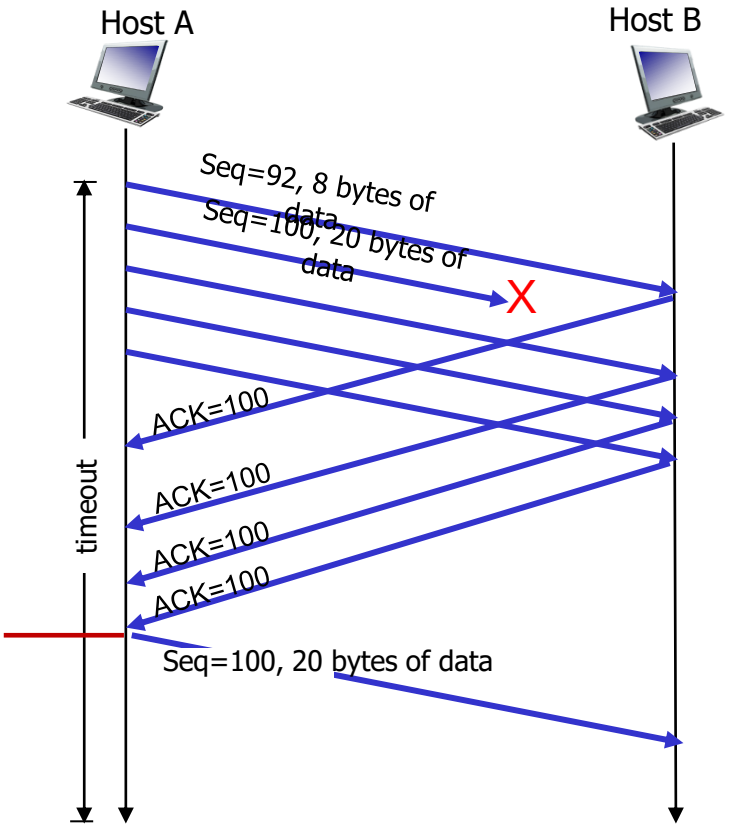
TCP fast retransmit

TCP fast retransmit

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

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

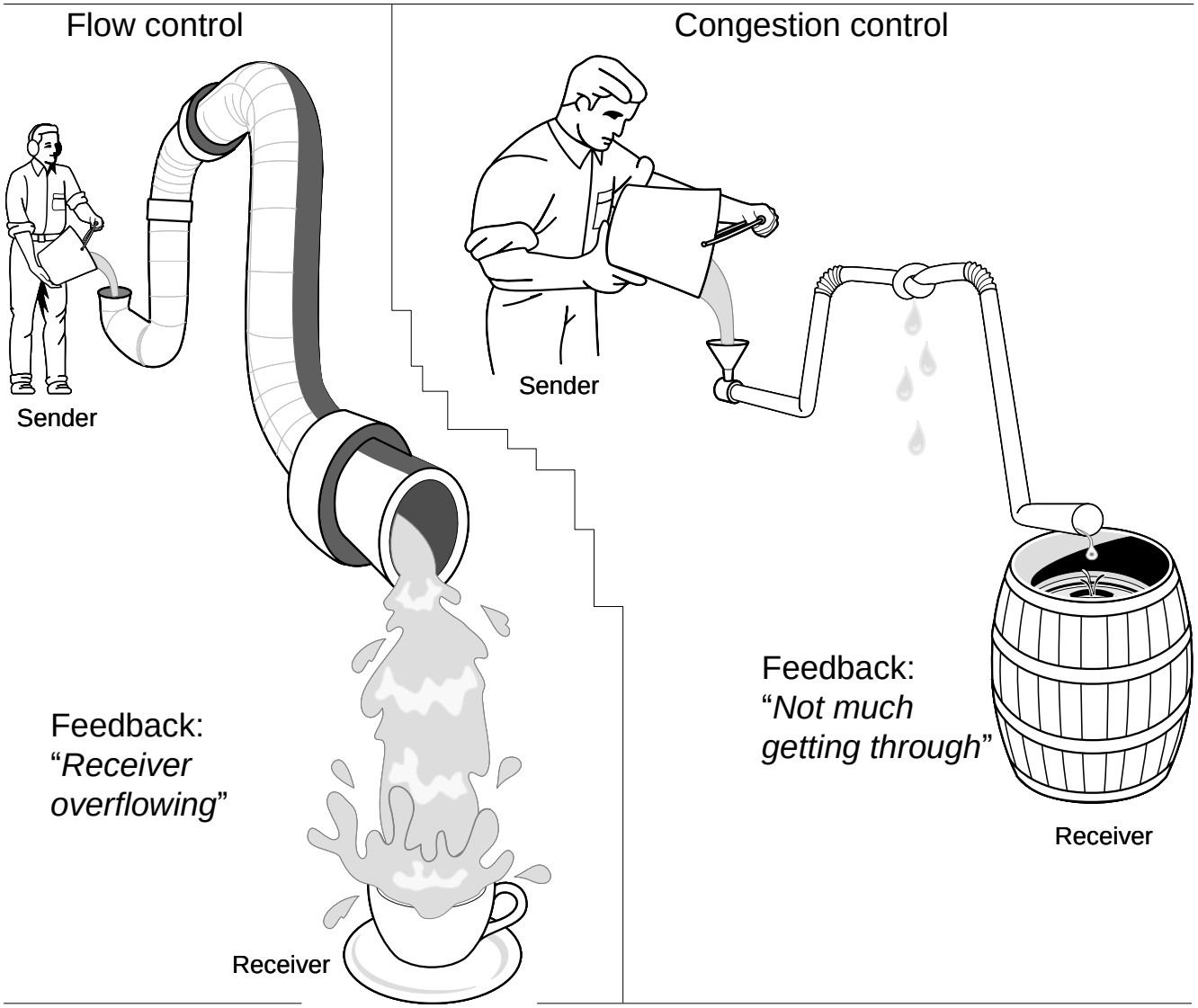
Receipt of three duplicate ACKs indicates 3 segments received after a missing segment – lost segment is likely. So retransmit!



Chapter 3: roadmap

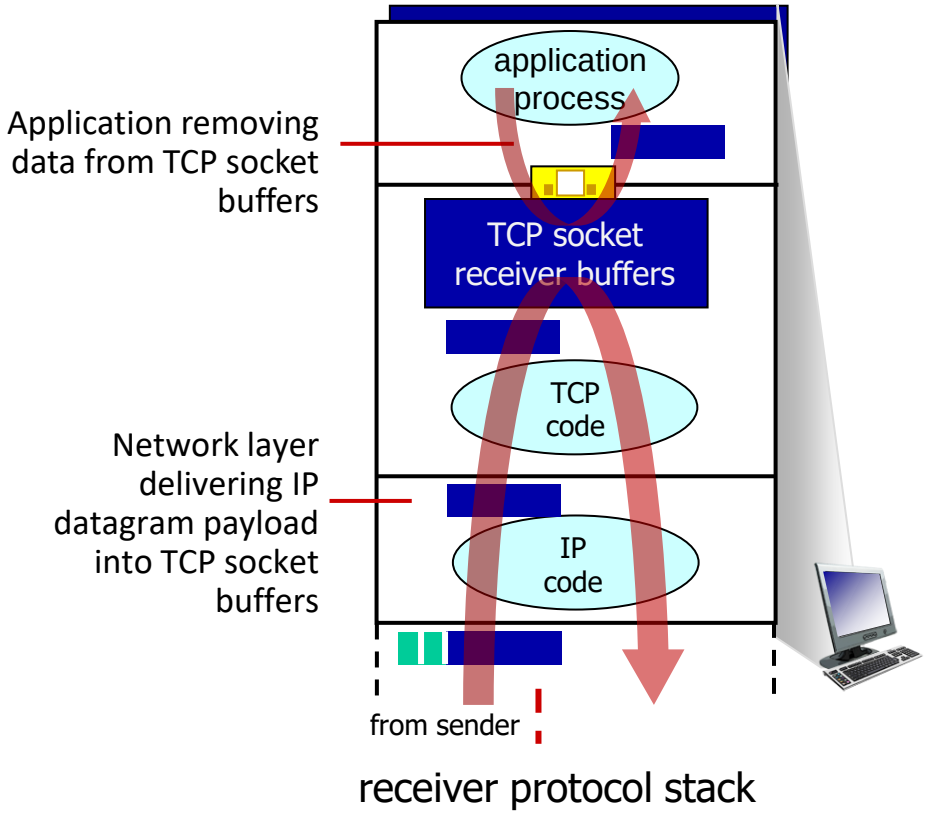
- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- Principles of congestion control
- TCP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

Flow Control vs Congestion Control (Courtesy Rutgers University)



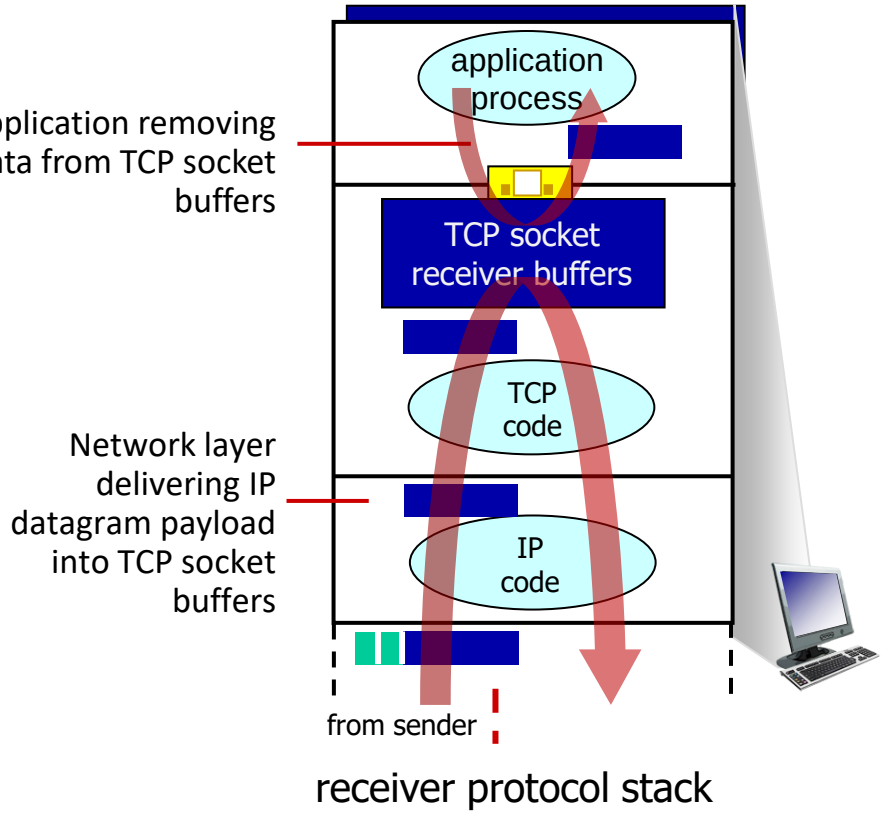
TCP flow control

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



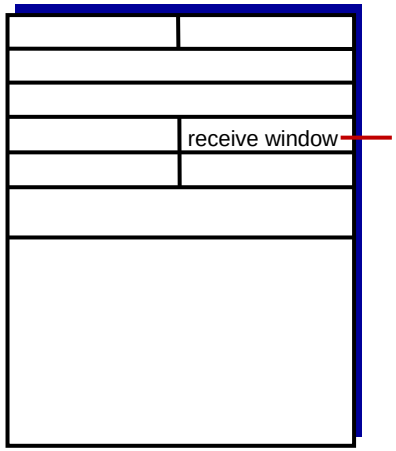
TCP flow control

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

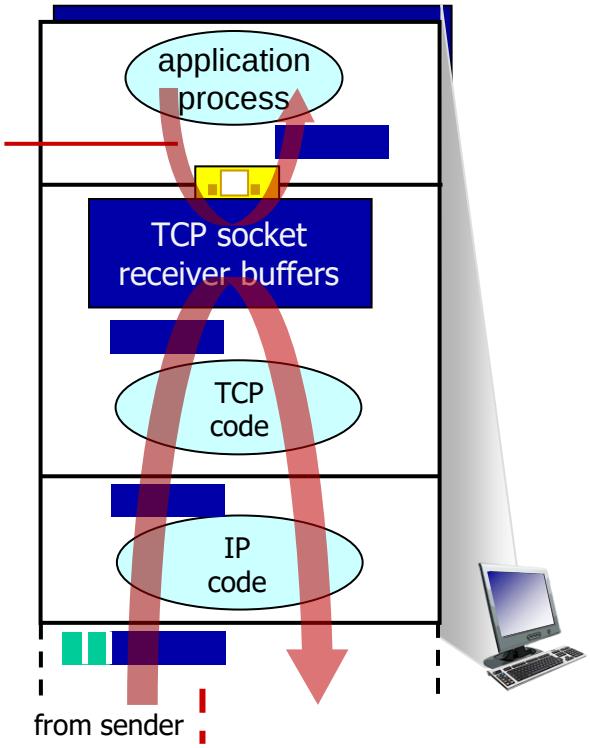


TCP flow control

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



flow control: # bytes receiver willing to accept



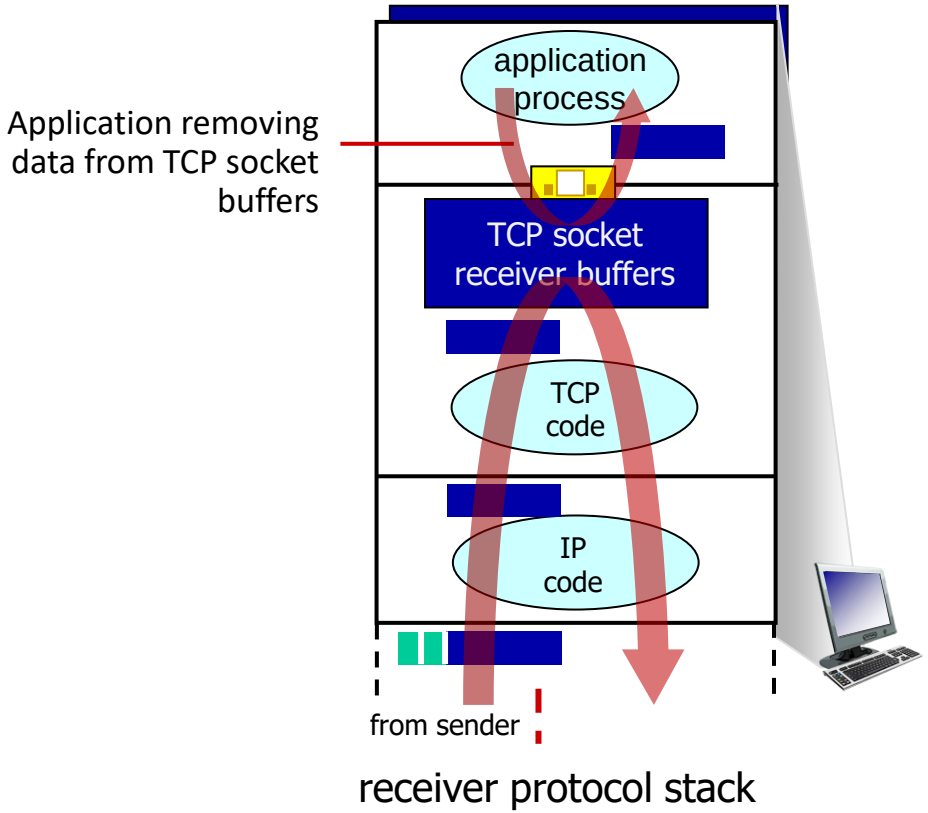
receiver protocol stack

TCP flow control

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

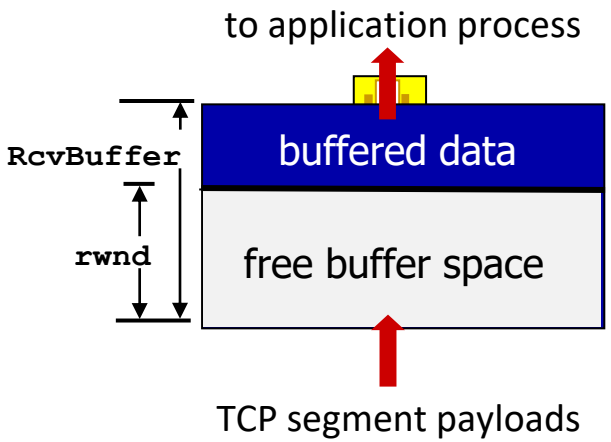
flow control

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



TCP flow control

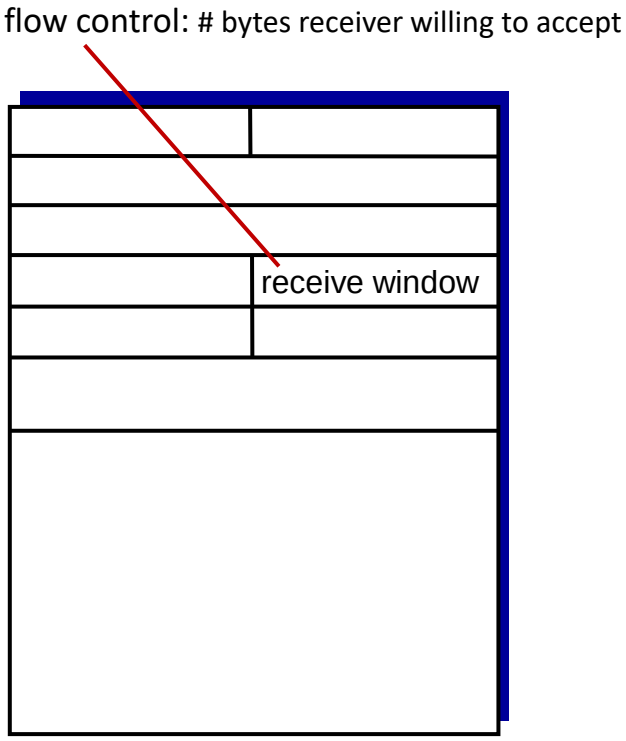
- TCP receiver "advertises" free buffer space in **`rwnd`** field in TCP header
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - many operating systems auto-adjust **`RcvBuffer`**
- sender limits amount of unACKed ("in-flight") data to received **`rwnd`**
- guarantees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



TCP receiver-side buffering

TCP flow control

- 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`** **`(i.e. dynamic)`**
- sender limits amount of unACKed ("in-flight") data to received **`rwnd`**
- guarantees receive buffer will not overflow
- (Assumption: TCP receiver discards out-of-order segments)



TCP segment format

TCP Flow Control - Implementation

- **TCP provides flow control by having the sender maintain a dynamic variable called the receive window (rwnd)**

(Since TCP is not permitted to overflow the allocated buffer, we must have)

LastByteRcvd – LastByteRead <= RcvBuffer *(where:*

- RcvBuffer is the size of the buffer allocated at the receiver for this connection,
- LastByteRcvd is the number of the last byte in the data stream received from the network and placed in the receive buffer, &
- LastByteRead is the number of the last byte in the data stream read by the application process from the receive buffer), **thus**

rwnd = RcvBuffer – [LastByteRcvd – LastByteRead]

- **While the receiver is keeping track of many variables as seen above, the sender is keeping track of primarily two variables, (**i.e. LastByteSent & LastByteAcked**)**

LastByteSent – LastByteAcked <= rwnd (i.e. the **UnAckedData**.)

- By maintaining this throughout the connection's life, i.e. keeping the UnAckedData less than or equal to the rwnd, the sender ensures it doesn't overflow the buffer at the receiver

TCP Flow Control - Issue

Issue: One minor technical problem with this scheme.

- To see this, suppose Host B's (receiver) receive buffer becomes full so that rwnd = 0.
- After advertising rwnd = 0 to Host A (sender), also suppose that B has nothing to send to A.
- As the application process at B empties the buffer, TCP does not send new segments with new rwnd values to Host A (indeed, TCP sends a segment to Host A only if it has data to send or if it has an acknowledgment to send)

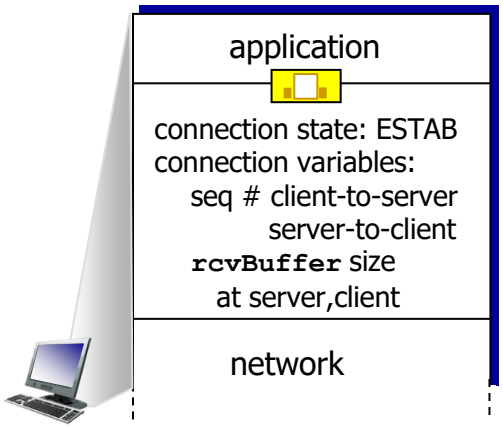
Therefore, Host A is never informed that some space has opened up in Host B's receive buffer (i.e. **Host A is blocked and can transmit no more data!**)

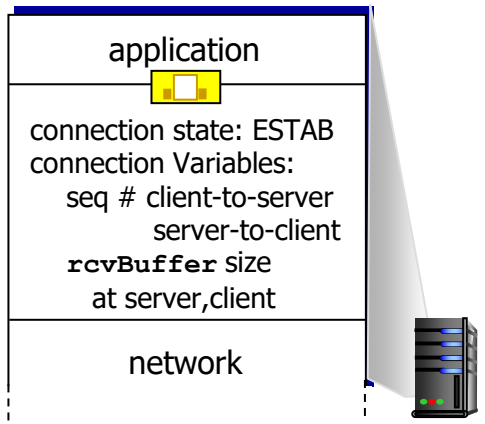
- To solve this problem, the TCP specification requires Host A to continue to send segments with one data byte when B's receive window is zero.
- These segments will be acknowledged by the receiver.
- Eventually the buffer will begin to empty and the acknowledgments will contain a non-zero rwnd value.

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)



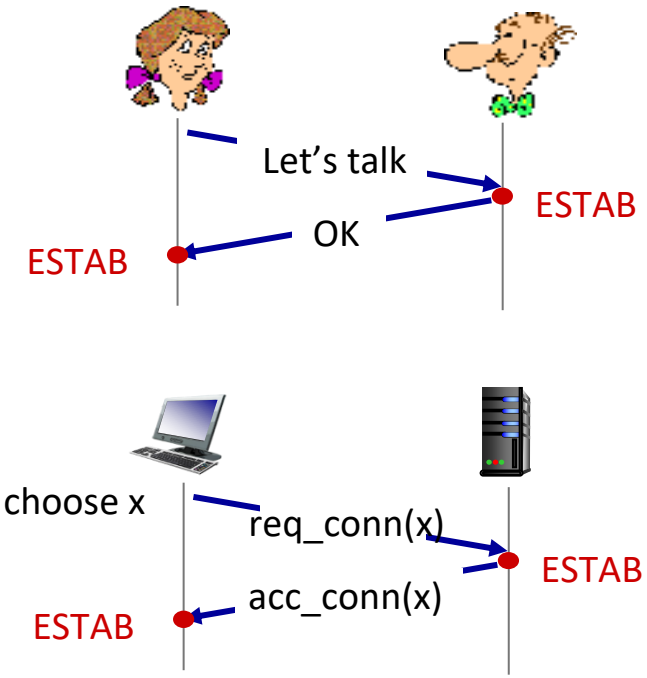



```
Socket clientSocket =
  newSocket("hostname","port number");
```

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

Agreeing to establish a connection

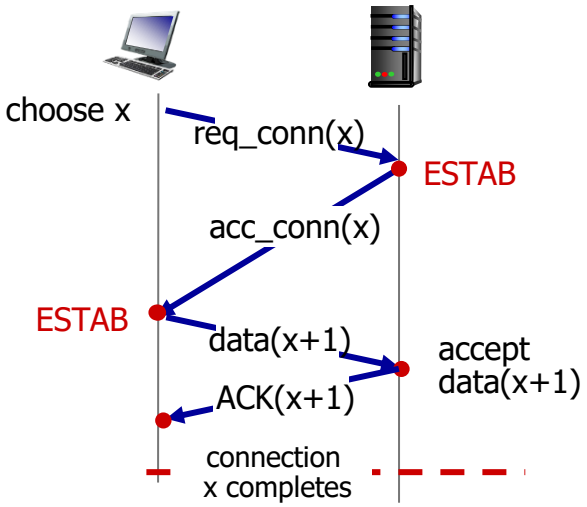
2-way handshake:



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

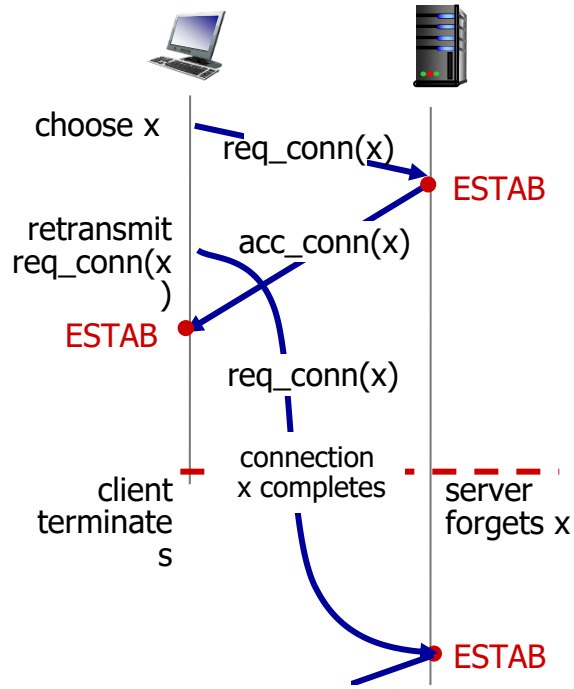
- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

2-way handshake scenarios



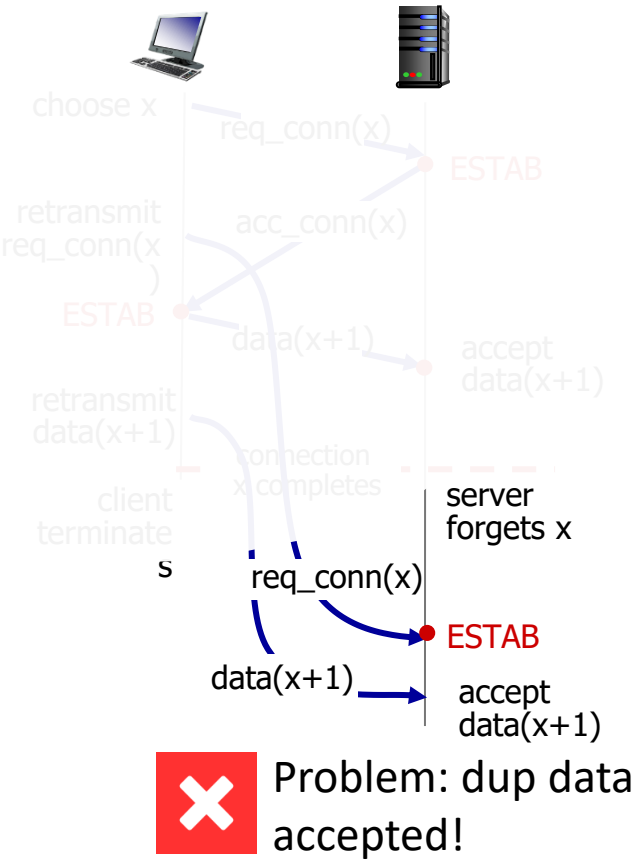
No problem!

2-way handshake scenarios



Problem: half open connection! (no client)

2-way handshake scenarios



TCP 3-way handshake

Client state

```
clientSocket = socket(AF_INET, SOCK_STREAM)
```

LISTEN

```
clientSocket.connect((serverName,serverPort))
```

choose init seq num, x
send TCP SYN msg

SYNSENT

SYNbit=1, Seq=x

choose init seq num, y
send TCP SYNACK
msg, acking SYN

SYNbit=1, Seq=y
ACKbit=1; ACKnum=x+1

received SYNACK(x)
indicates server is live;
send ACK for SYNACK;
this segment may contain
client-to-server data

ESTAB

ACKbit=1, ACKnum=y+1

received ACK(y)
indicates client is live

Server state

```
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
connectionSocket, addr = serverSocket.accept()
```

LISTEN

SYN RCVD

ESTAB

A human 3-way handshake protocol

Closing a TCP 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

Assignement # 3 (Chapter – 3)

- *3rd Assignment will be uploaded on Google Classroom on Thursday, 12th October, 2023, in the Stream - Announcement Section*

- *Due Date: Tuesday, 17th October, 2023 (Handwritten solutions to be submitted during the lecture)*

- *Please read all the instructions carefully in the uploaded Assignment document, follow & submit accordingly*

Quiz # 3 (Chapter – 3)

- *On: Thursday, 19th October, 2023 (During the lecture)*

- *Quiz to be taken during own section class only*