National University of Computer & Emerging Sciences

CS 3001 - COMPUTER NETWORKS

Lecture 14

Chapter 3

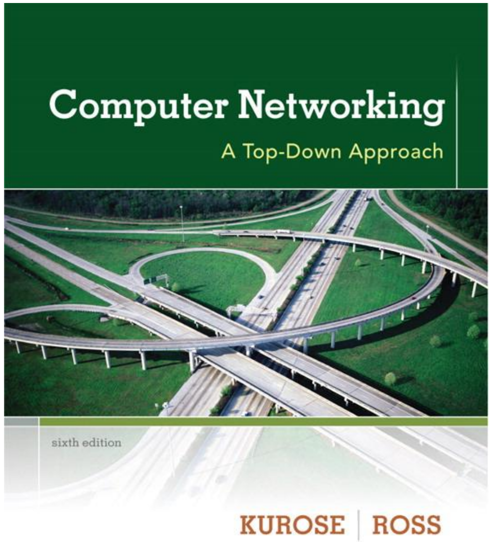
11th October, 2022

Nauman Moazzam Hayat

nauman.moazzam@lhr.nu.edu.pk

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

Chapter 3 Transport Layer



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

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

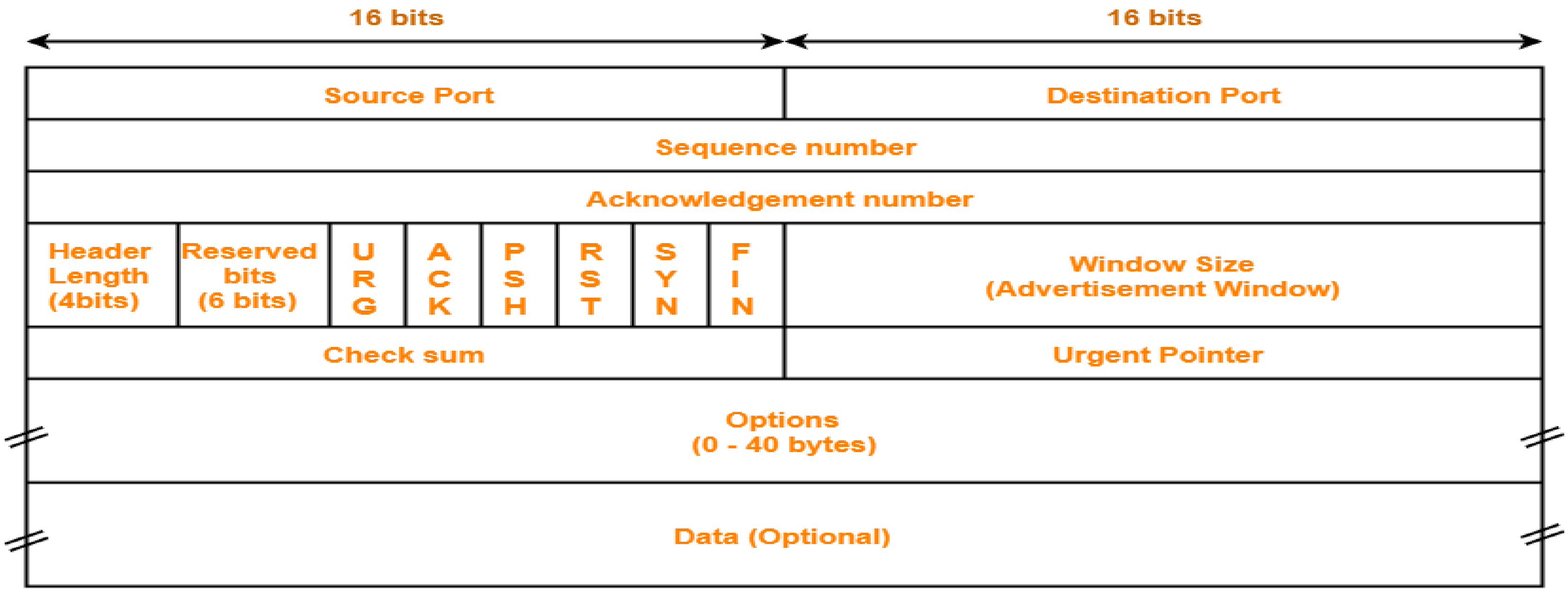
3.5 connection-oriented transport: TCP

- segment structure
- reliable data transfer
- flow control
- connection management

3.6 principles of congestion control

3.7 TCP congestion control

TCP Header (Re-visited)



TCP Header

- Header length is a 4 bit field.
- It contains the length of TCP header.
- It helps in knowing from where the actual data begins.
- Minimum and Maximum Header length

The length of TCP header always lies in the range: [20 bytes (minimum) till 60 bytes (maximum)]

- The initial 5 rows of the TCP header are always used.
- The size of the 6th row representing the Options field vary as it can be used or not used.
- The size of Options field can go from 0 bytes till 40 bytes.
- Header length is a 4-bit field, thus it can have a min value of 0000 (0 in decimal) till a max value of 1111 (15 in decimal)
- But the range of header length is [20, 60].
- So, to represent the header length, we use a scaling factor of 4.
-
- Thus, in general:

Header length = Header length field value x 4 bytes

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 events:

data rcvd from app:

- 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`**

timeout:

- retransmit segment that caused timeout
- restart timer

ack rcvd:

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

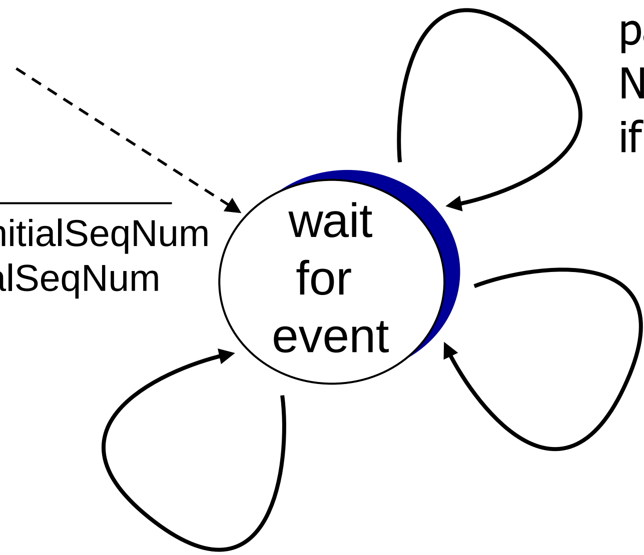
TCP sender (simplified, uses only timeouts)

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



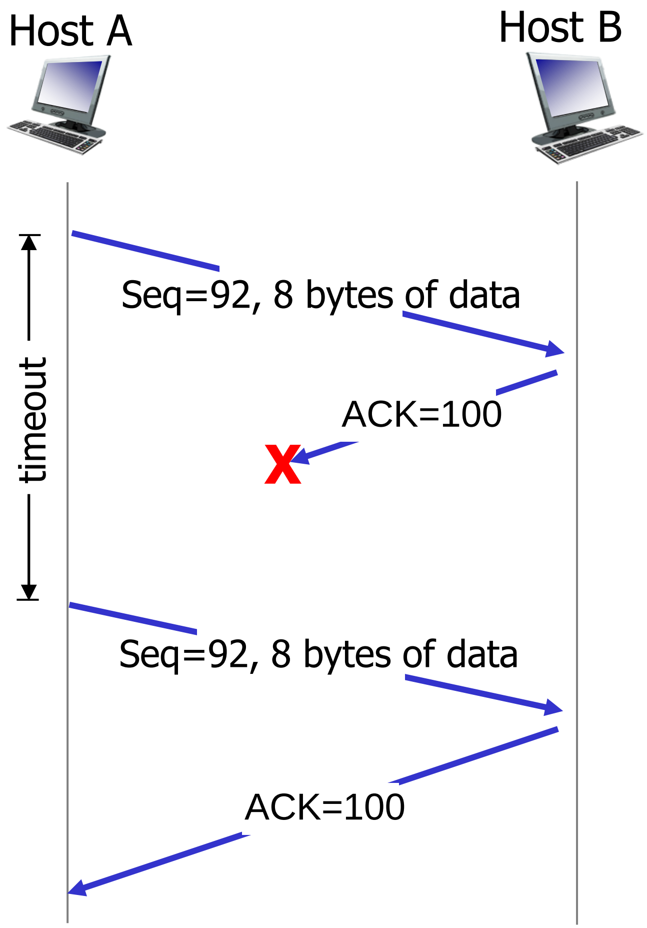
timeout

retransmit not-yet-acked segment
    with smallest seq. #
start timer

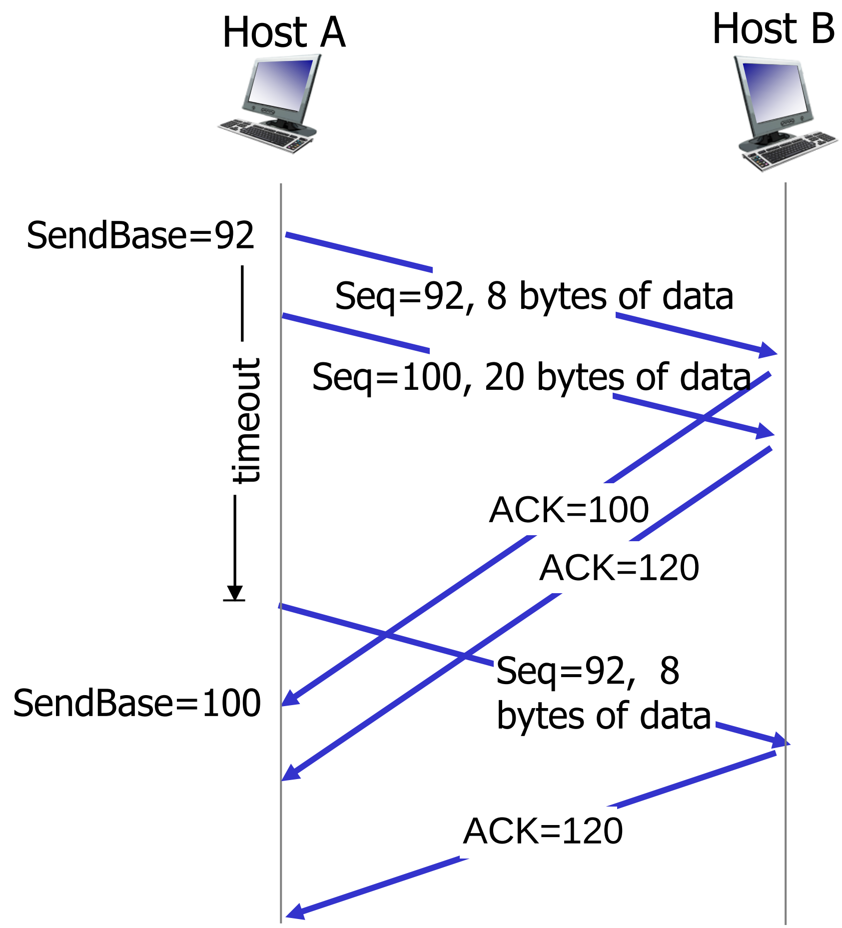
ACK received, with ACK field value y

```
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

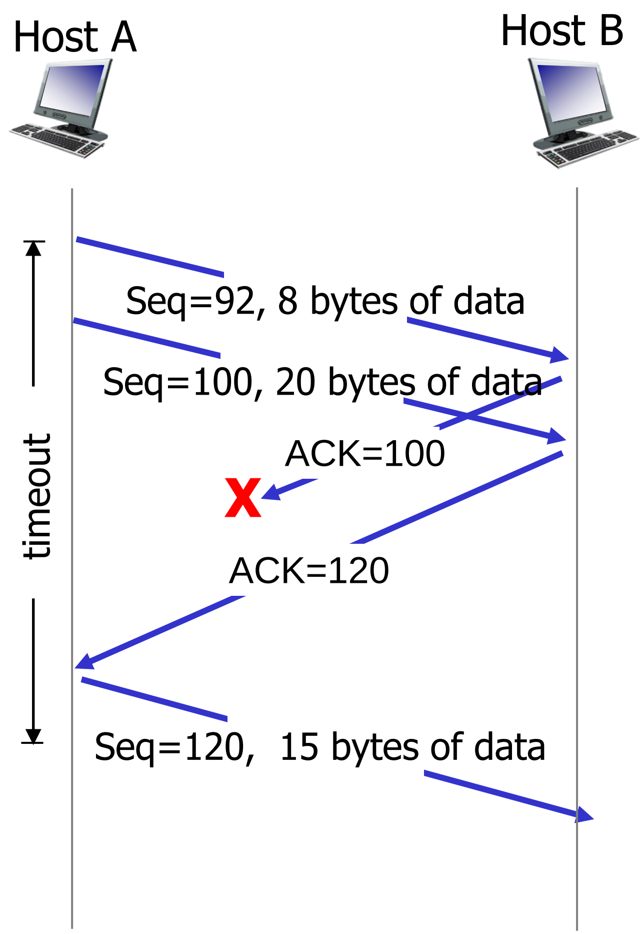


lost ACK scenario



premature timeout
(segment 100 not retransmitted)

TCP: retransmission scenarios



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

TCP 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.

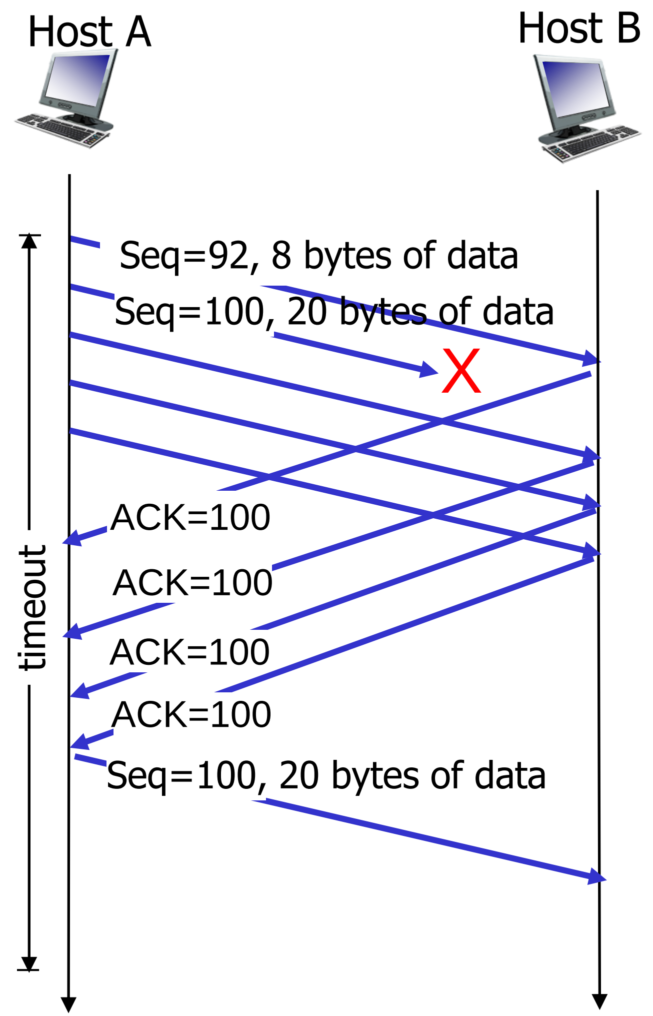
TCP fast retransmit

if sender receives 3 ACKs for same data

("triple duplicate ACKs"), resend unacked segment with smallest seq #

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

TCP 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

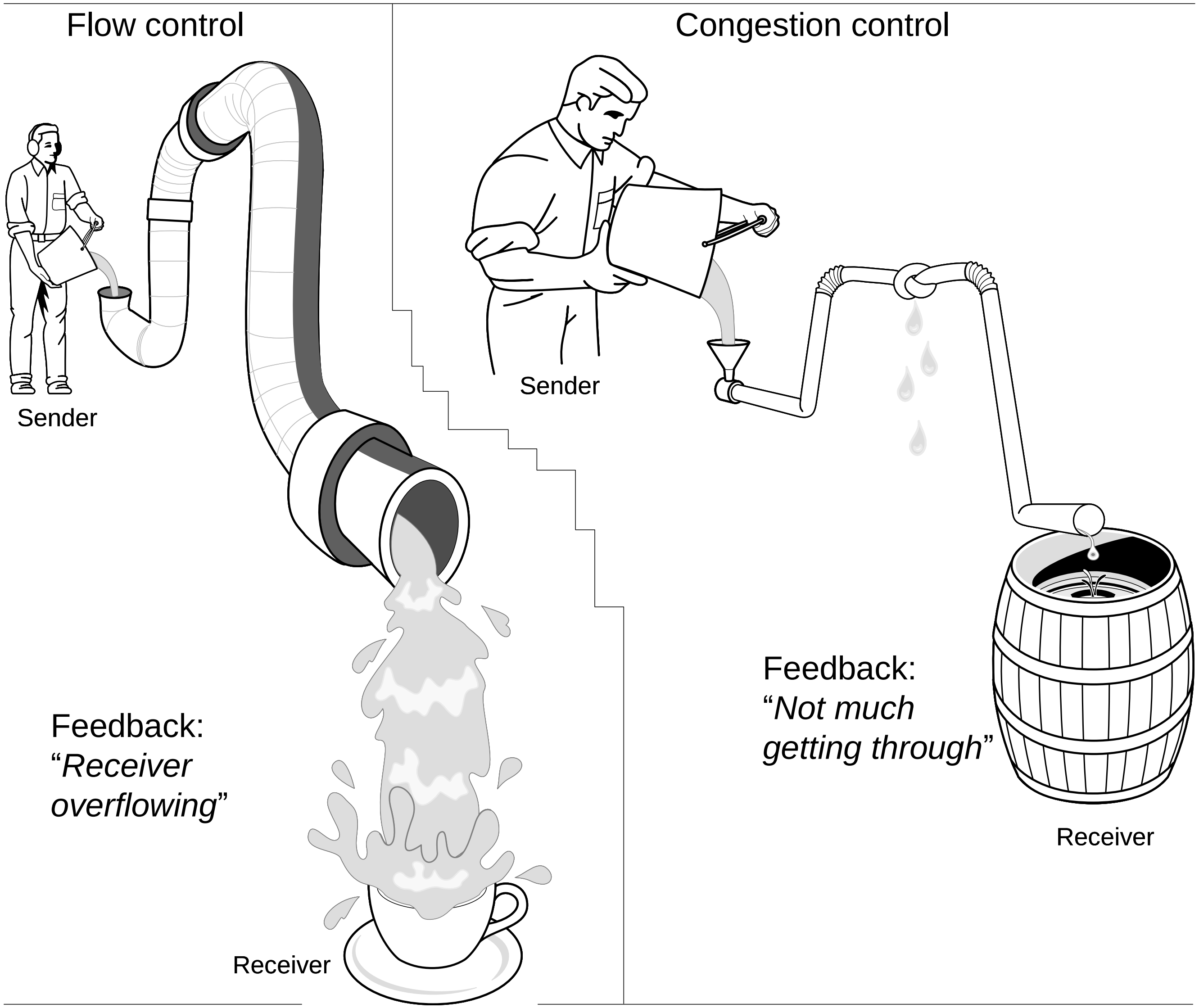
3.4 principles of reliable data transfer

3.5 connection-oriented transport: TCP
- segment structure
- reliable data transfer
- flow control
- connection management

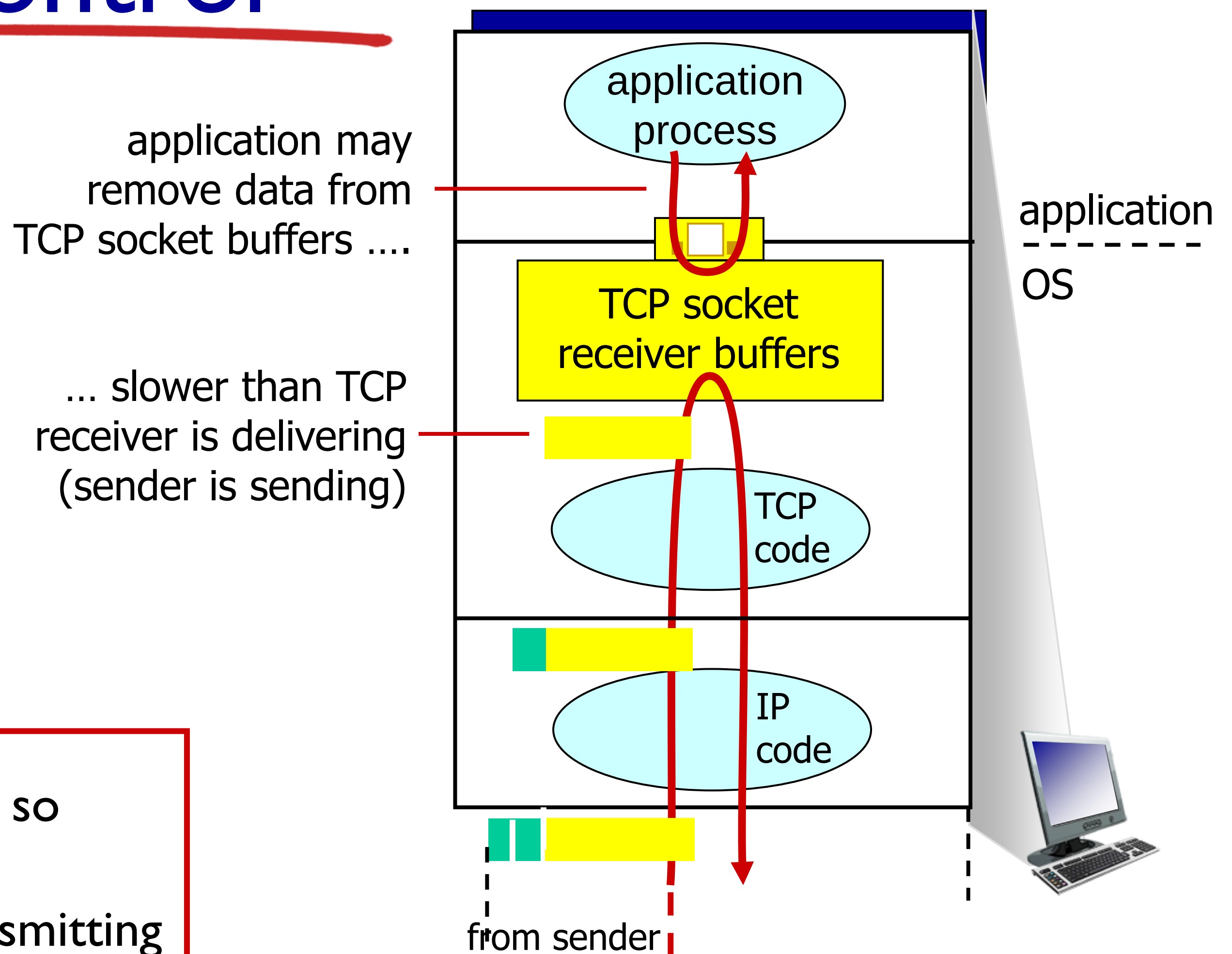
3.6 principles of congestion control

3.7 TCP congestion control

Flow Control vs Congestion Control (Courtesy Rutgers University)



TCP flow control



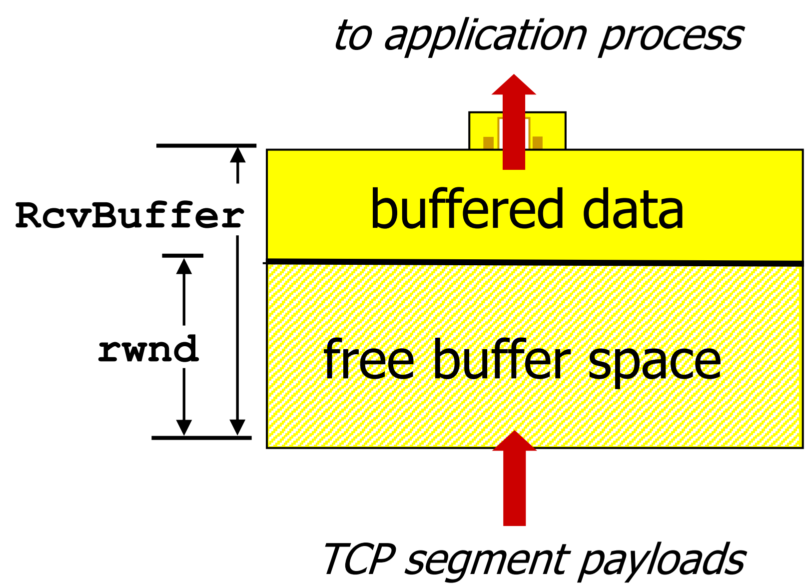
receiver protocol stack

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



receiver-side buffering

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.

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

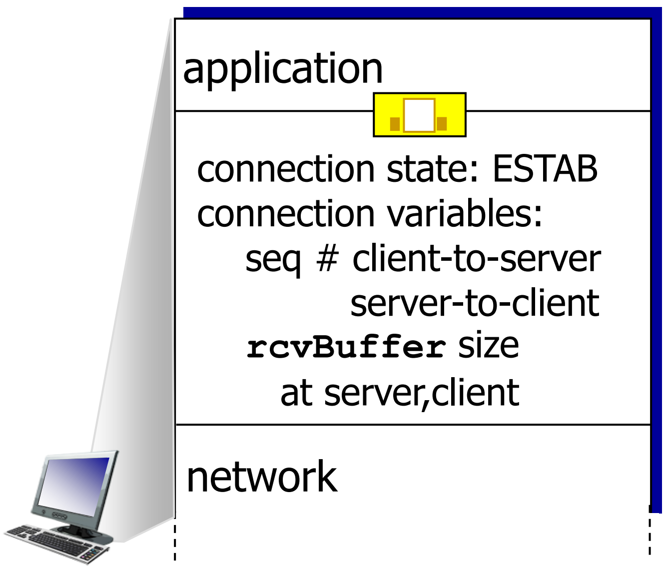
3.6 principles of congestion control

3.7 TCP congestion control

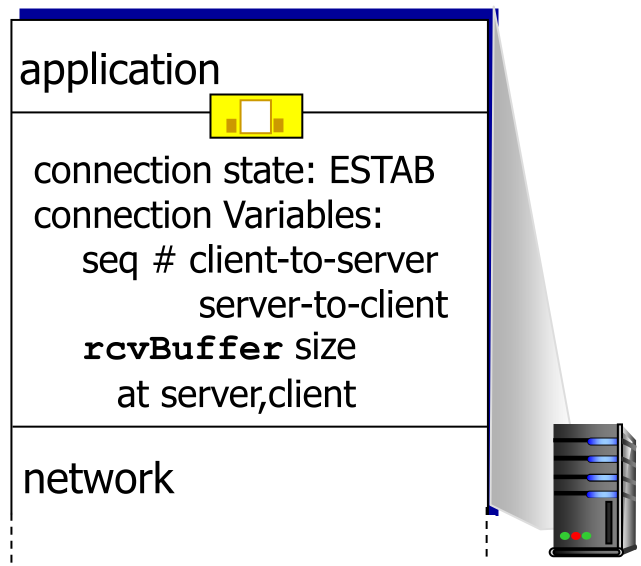
Connection Management

before exchanging data, sender/receiver "handshake":

- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters



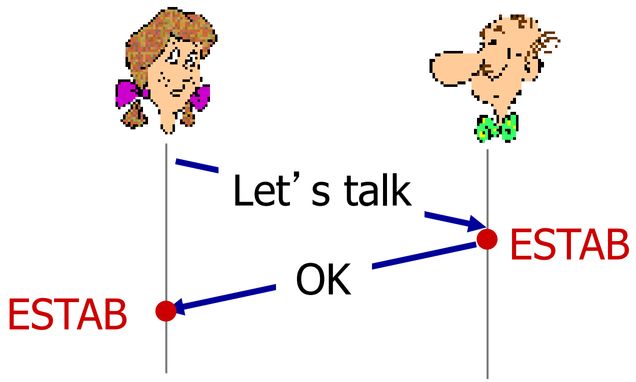

```
Socket clientSocket =
  newSocket("hostname","port
  number");
```

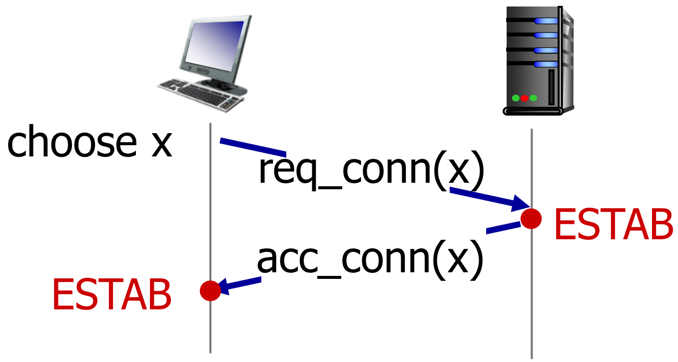



```
Socket connectionSocket =
  welcomeSocket.accept();
```

Agreeing to establish a connection

2-way handshake:



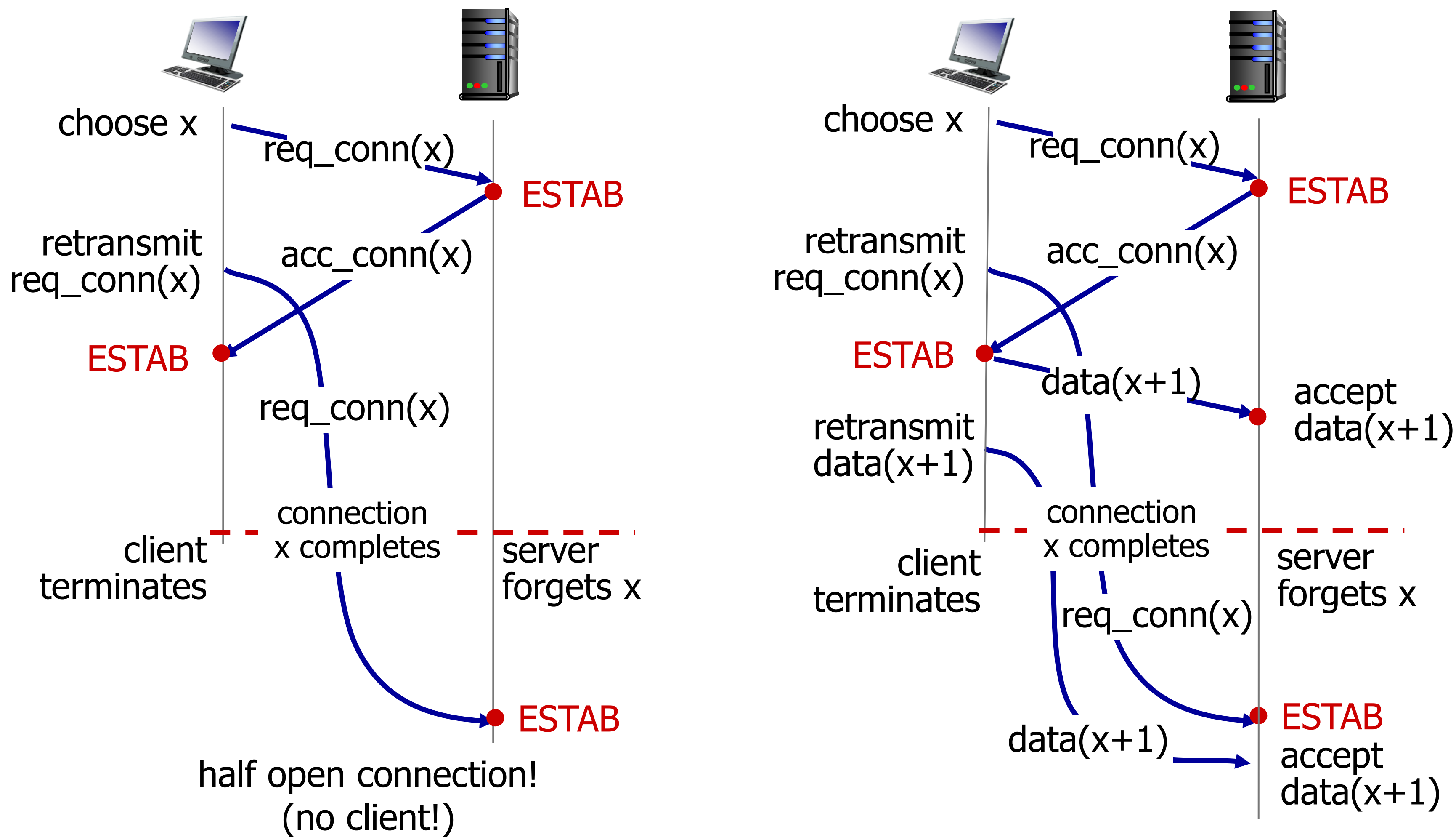


Q: will 2-way handshake always work in network?

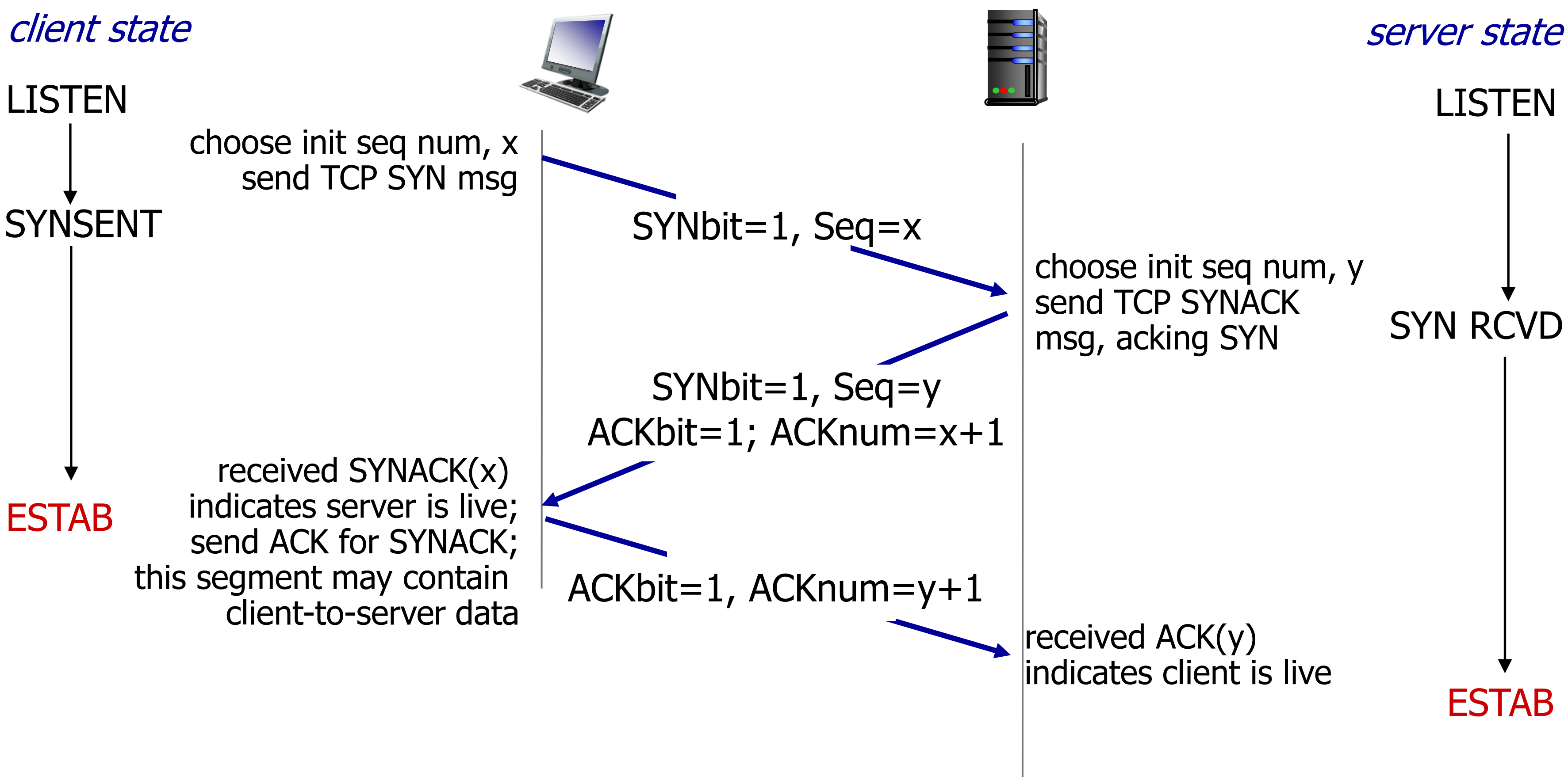
- variable delays
- retransmitted messages (e.g. req_conn(x)) due to message loss
- message reordering
- can't "see" other side

Agreeing to establish a connection

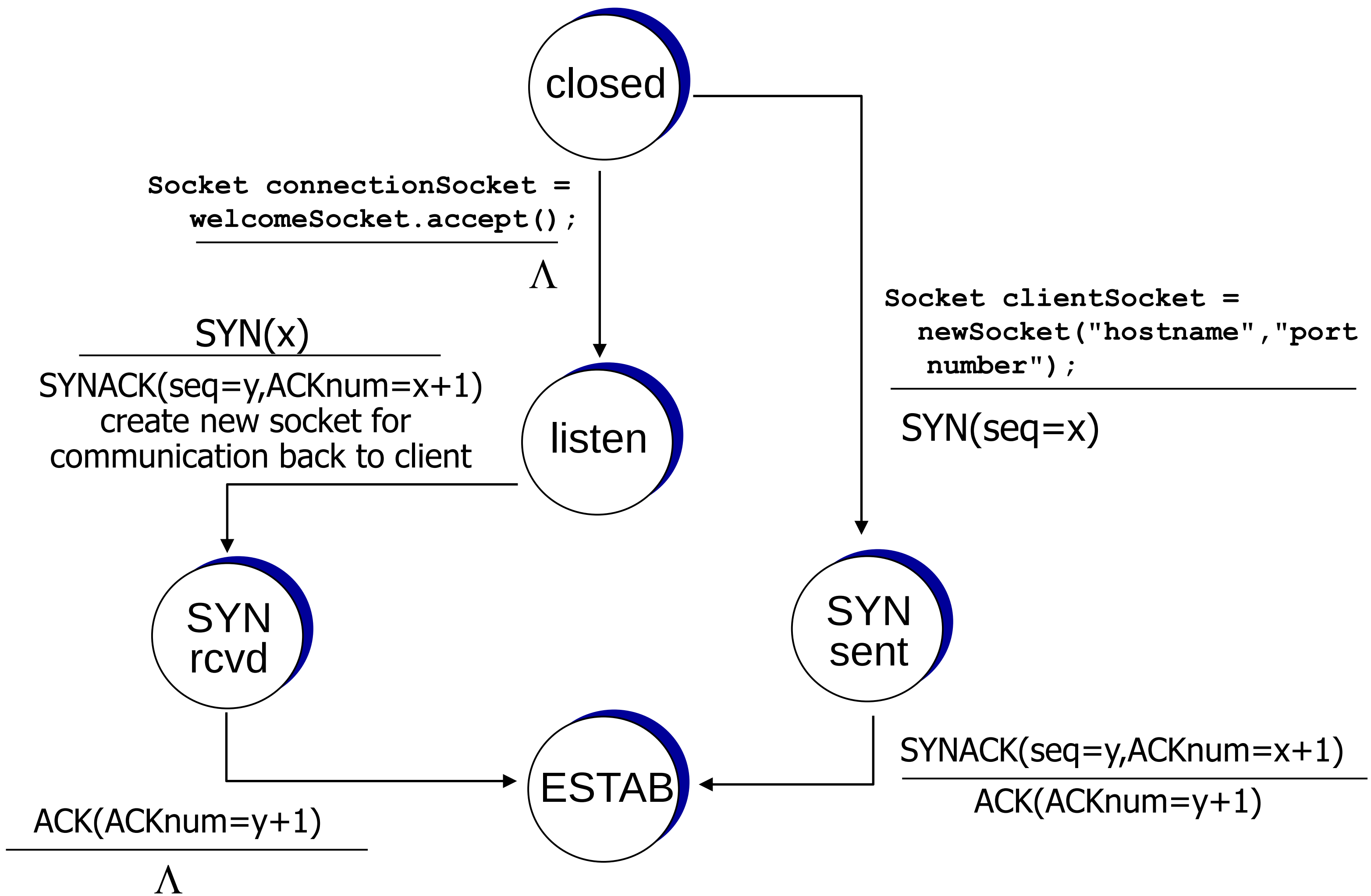
2-way handshake failure scenarios:



TCP 3-way handshake



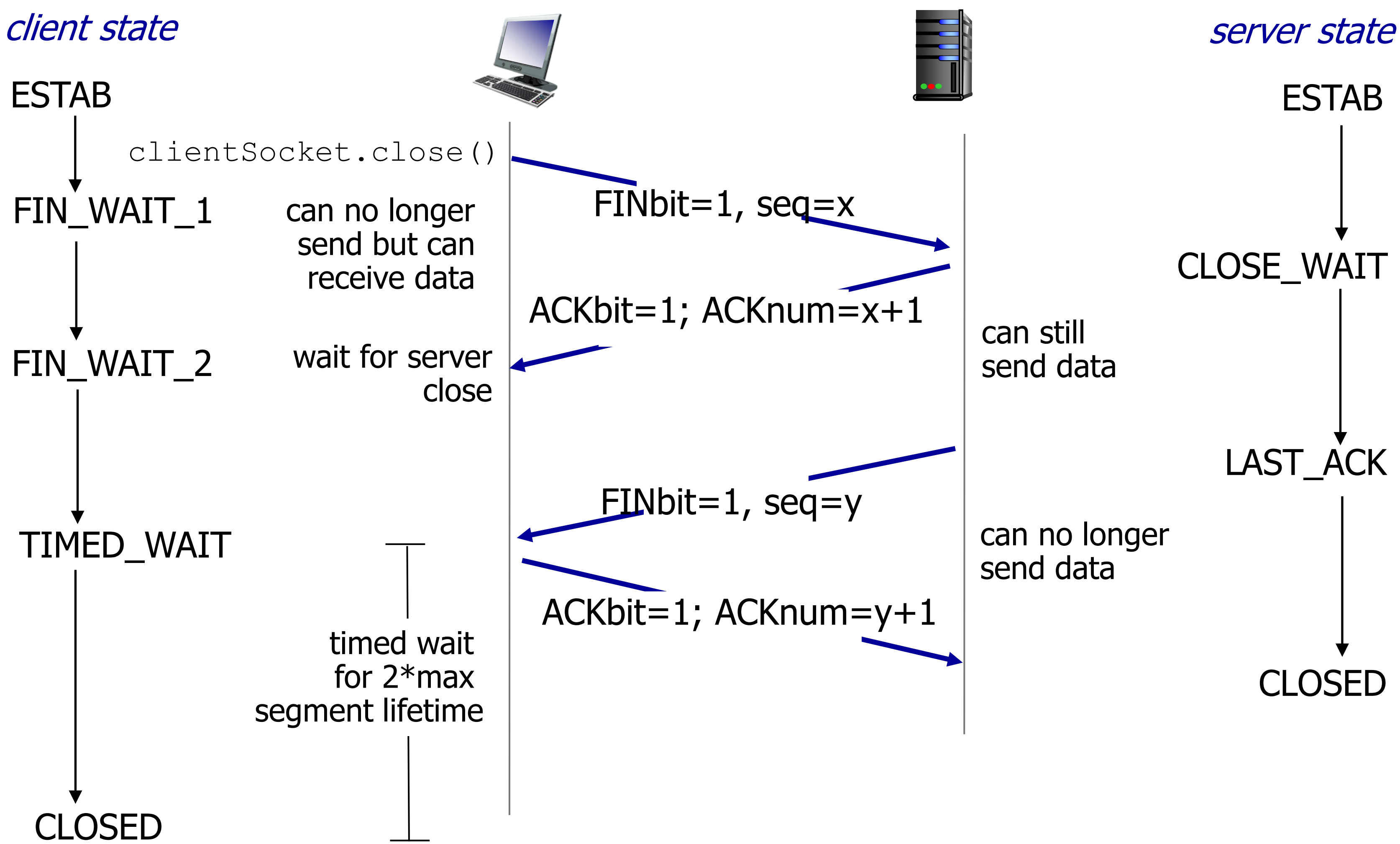
TCP 3-way handshake: FSM



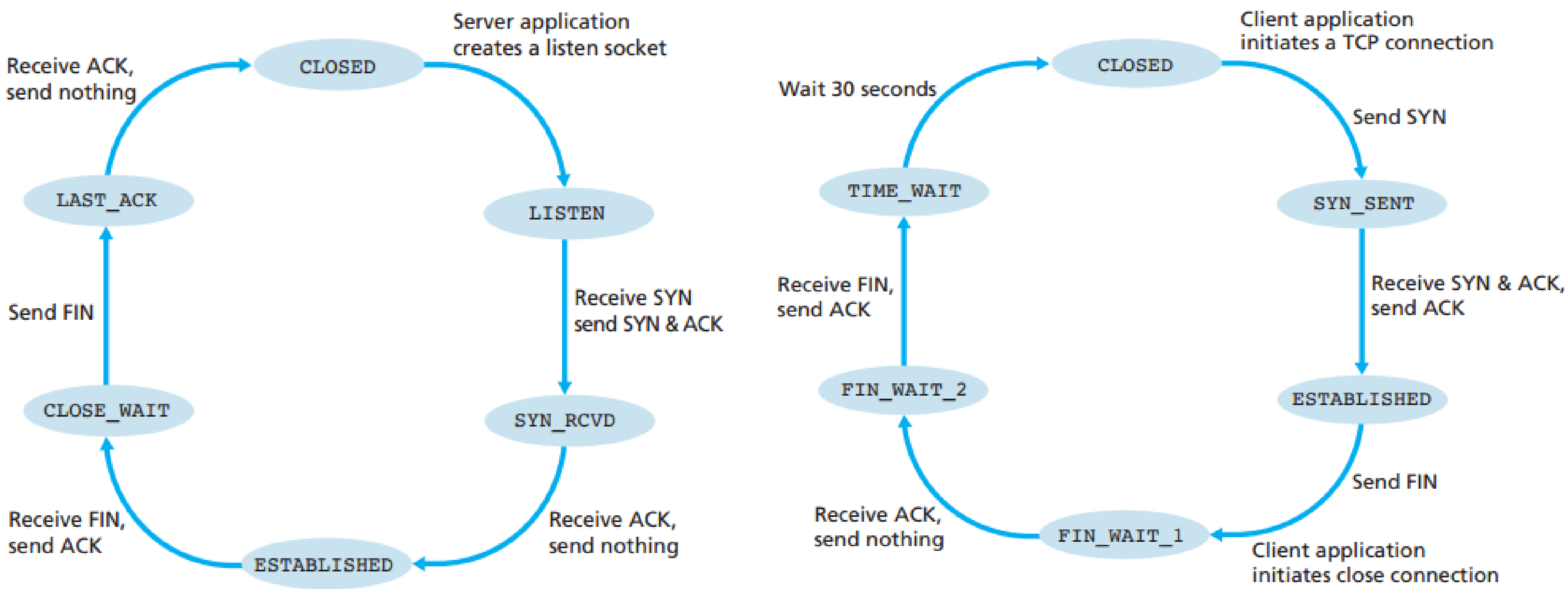
TCP: closing a connection

- client, server each close their side of 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



TCP: connection summary



Server

Client

Assignement # 3 (Chapter – 3)

- *3rd Assignment will be uploaded on Google Classroom on Thursday, 13th October, 2022, in the Stream - Announcement Section (not the Classwork Section)*

- *Due Date: Tuesday, 18th October, 2022 (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, 20th October, 2022 (During the lecture)*

- *Quiz to be taken during <u>own section class only</u>*