DATA COMMUNICATOIN NETWORKING

Instructor: Ouldooz Baghban Karimi

Course Book: Computer Networking, A Top-Down Approach
By: Kurose, Ross

Course Overview

- **Basics of Computer Networks**
  - Internet & Protocol Stack
  - Application Layer
  - Transport Layer
  - Network Layer
  - Data Link Layer

- **Advanced Topics**
  - Case Studies of Computer Networks
  - Internet Applications
  - Network Management
  - Network Security

TCP Sender Events

Data received from App

- Create segment with sequence number
- Sequence number 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`

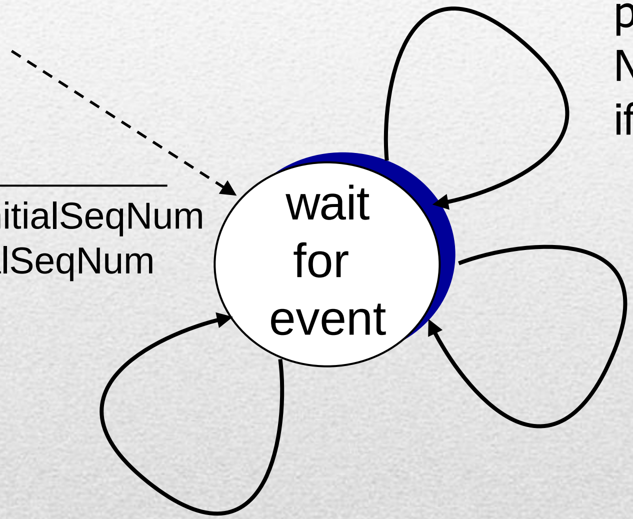
Timeout

- Retransmit segment that caused timeout
- Restart timer

ACK received

- If ACK acknowledges previously unACKed segments
  - Update what is known to be ACKed
  - Start timer if there are still unACKed segments

TCP Sender Events



Λ

```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum
```

wait for event

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
```

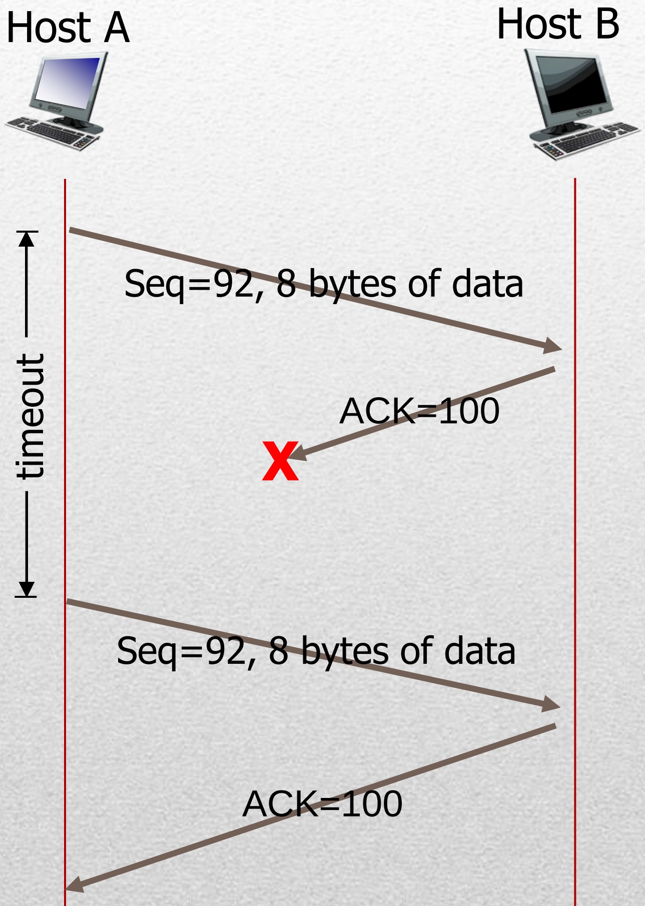
timeout

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

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



Host A

Host B

timeout

Seq=92, 8 bytes of data

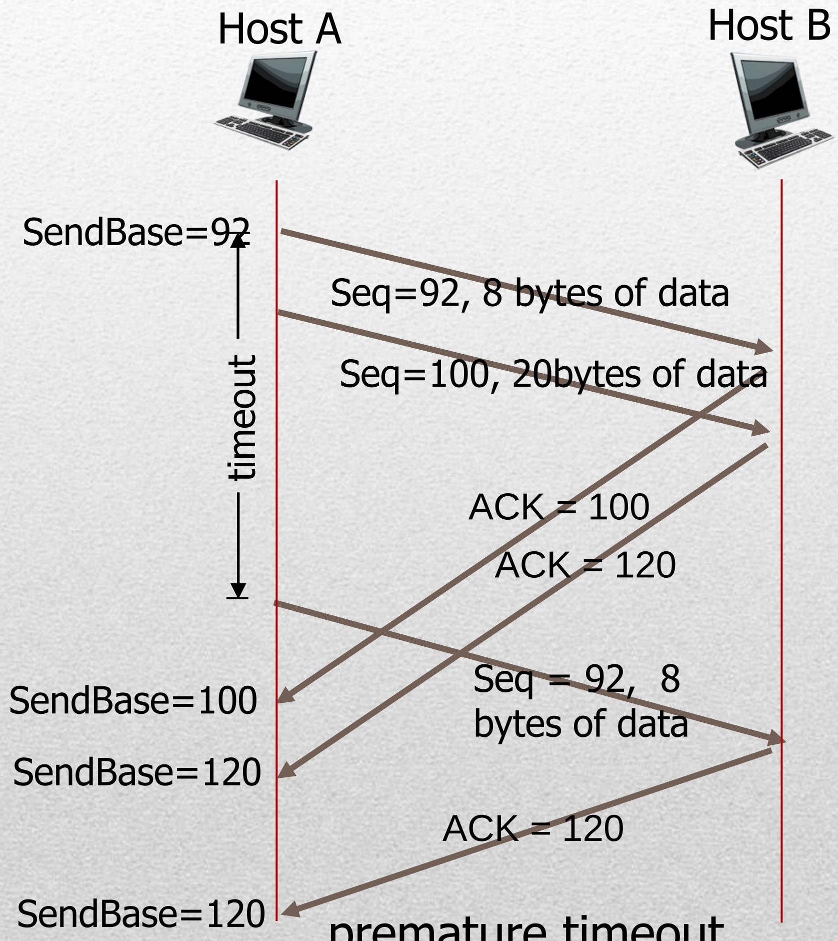
ACK=100

X

Seq=92, 8 bytes of data

ACK=100

lost ACK scenario



Host A

Host B

SendBase=92

timeout

Seq=92, 8 bytes of data

Seq=100, 20bytes of data

ACK = 100

ACK = 120

Seq = 92, 8 bytes of data

SendBase=100

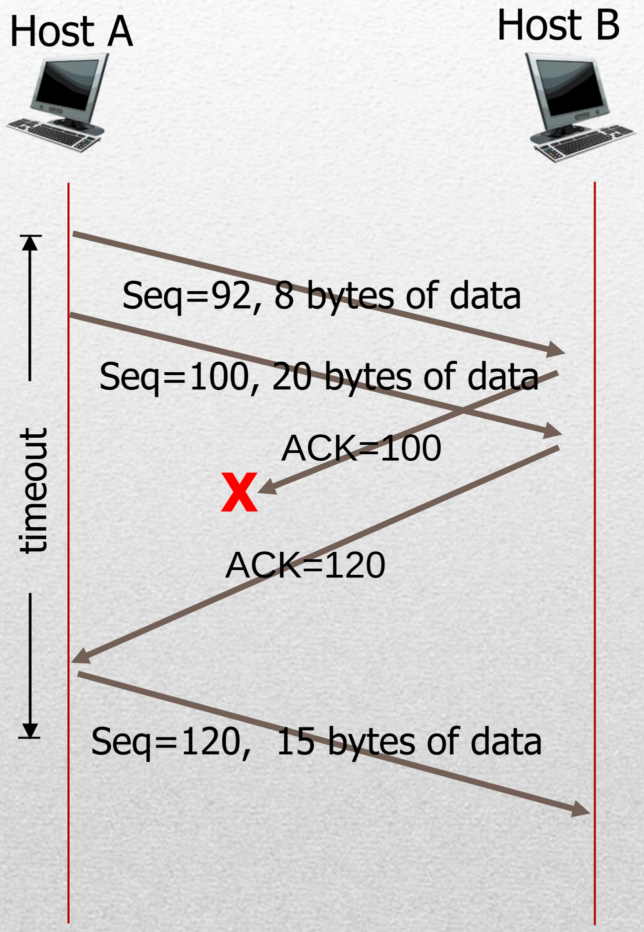
SendBase=120

ACK = 120

SendBase=120

premature timeout

TCP: Retransmission Scenarios



Cumulative ACK

TCP ACK Generation

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

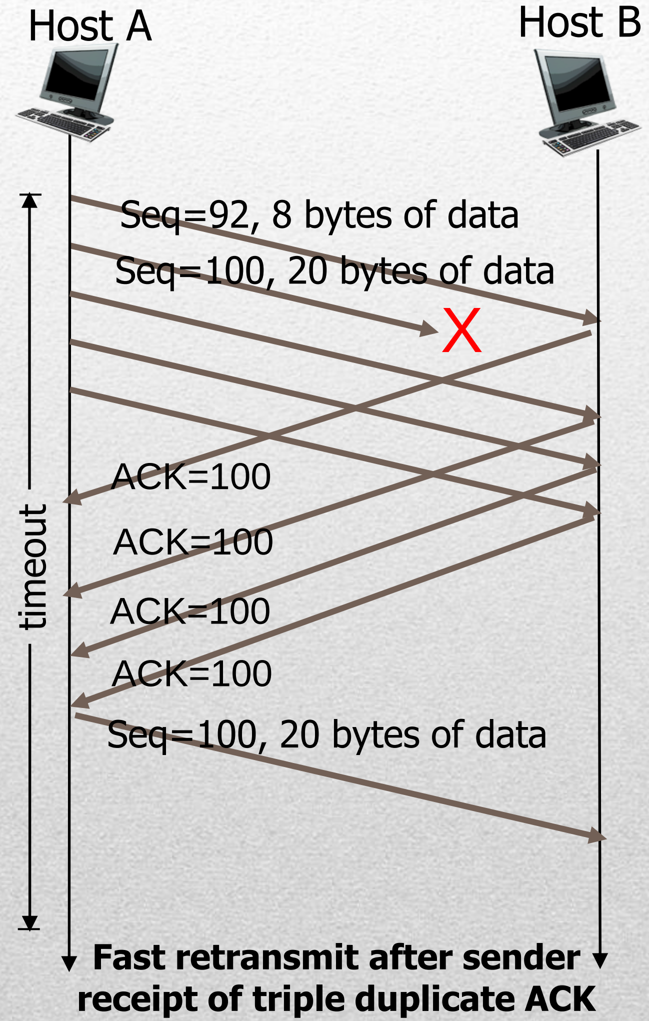
- **Time-out period often relatively long**
  - Long delay before resending lost packet
- **Detect lost segments via duplicate ACKs**
  - Sender often sends many segments back-to-back
  - If segment is lost, there will likely be many duplicate ACKs.

TCP fast retransmit

If sender receives 3 ACKs for same data (“triple duplicate ACKs”), resend unACKed segment with smallest sequence number

- Likely that unACKed segment lost, so do not wait for timeout

TCP FAST Retransmit

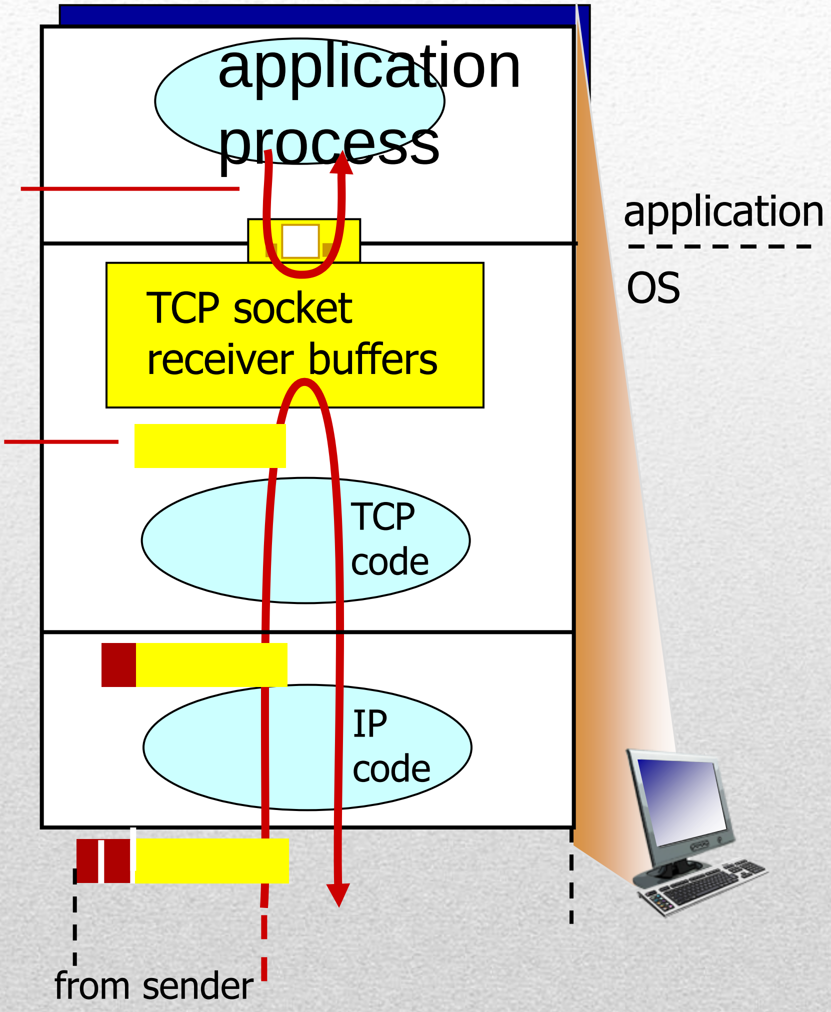


Fast retransmit after sender receipt of triple duplicate ACK

TCP Flow Control

application may remove data from TCP socket buffers ….

… slower than TCP receiver is delivering (sender is sending)



receiver protocol stack

Flow Control

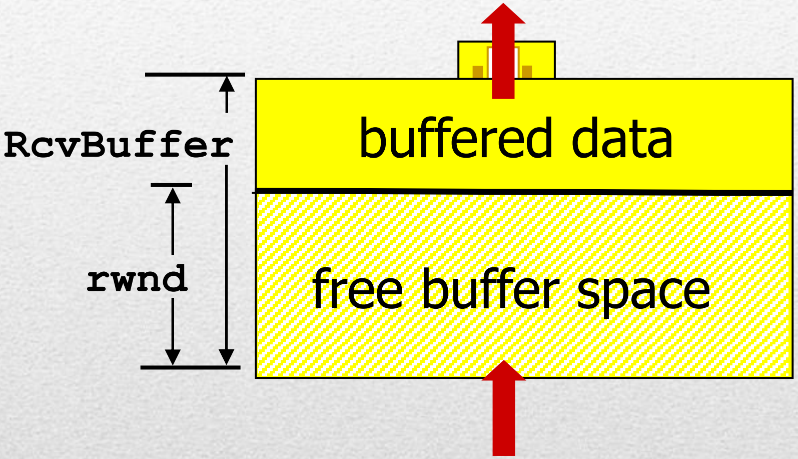
Receiver controls sender, so sender won’t overflow receiver’s buffer by transmitting too much, too fast

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`**

- Sender limits amount of unACKed ("in-flight") data to receiver's **`rwnd`** value
- Guarantees receive buffer will not overflow

to application process



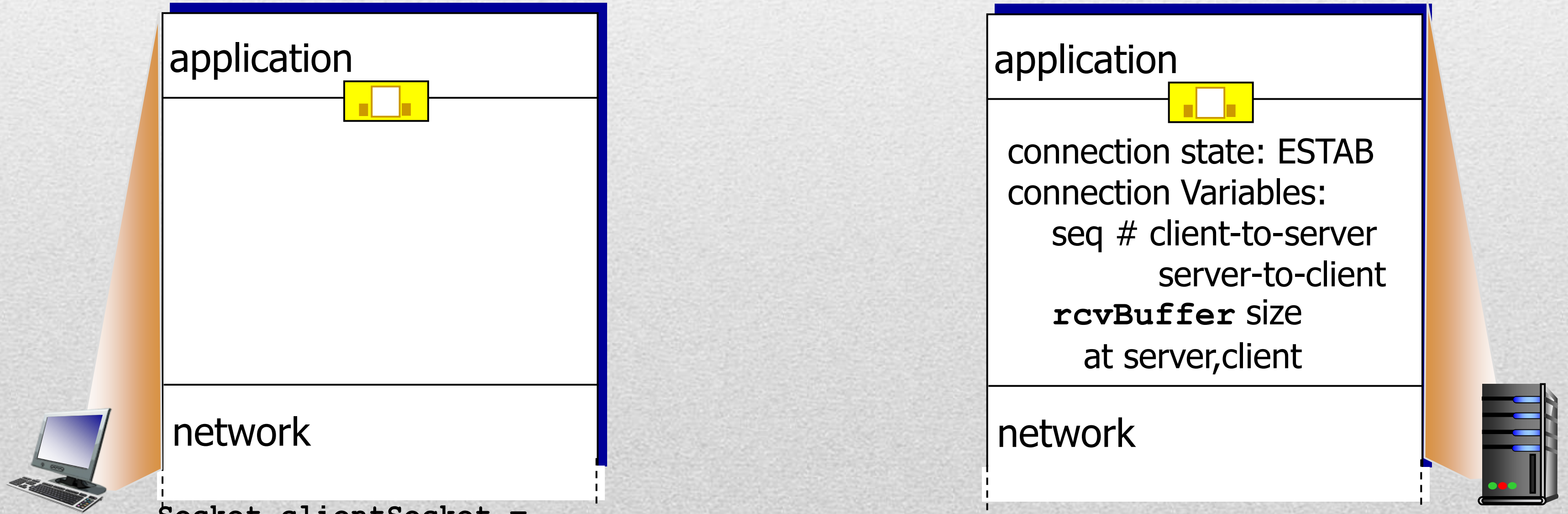
TCP segment payloads

receiver-side buffering

Connection Management

Before exchanging data, sender/receiver "handshake":

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



application

network

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

application

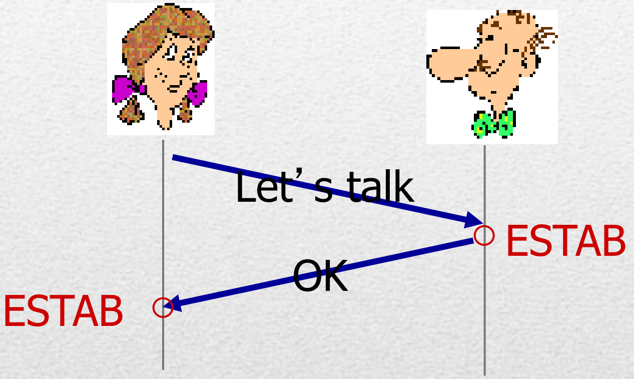
connection state: ESTAB
connection Variables:
    seq # client-to-server
              server-to-client
    rcvBuffer size
      at server,client

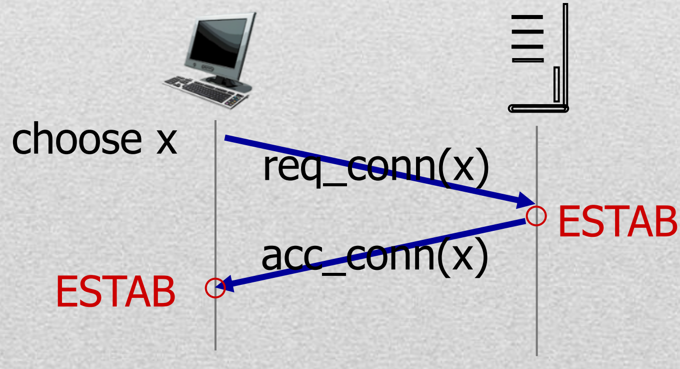
network

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

Agreeing to Establish a Connection

Two-way handshake

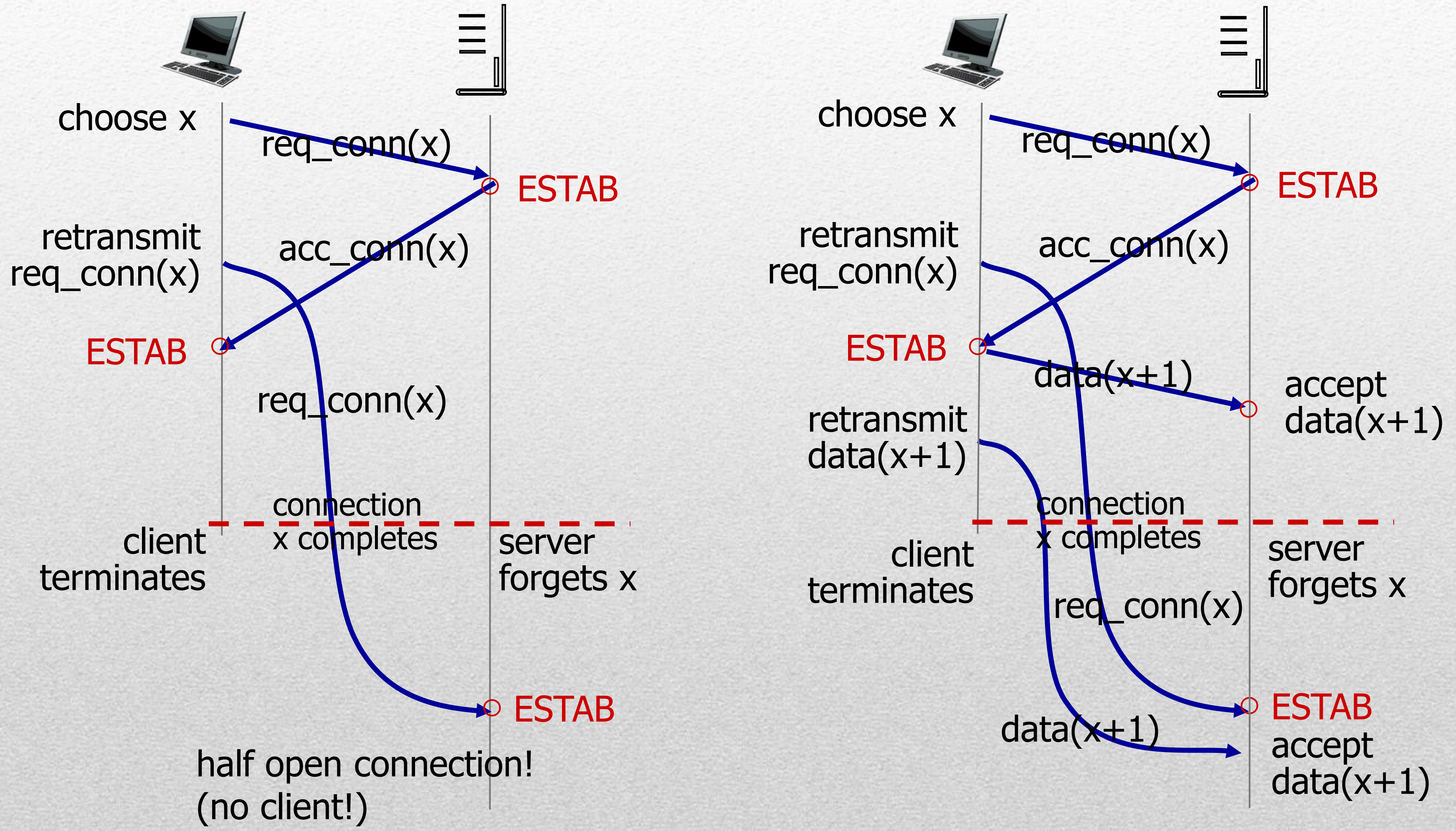




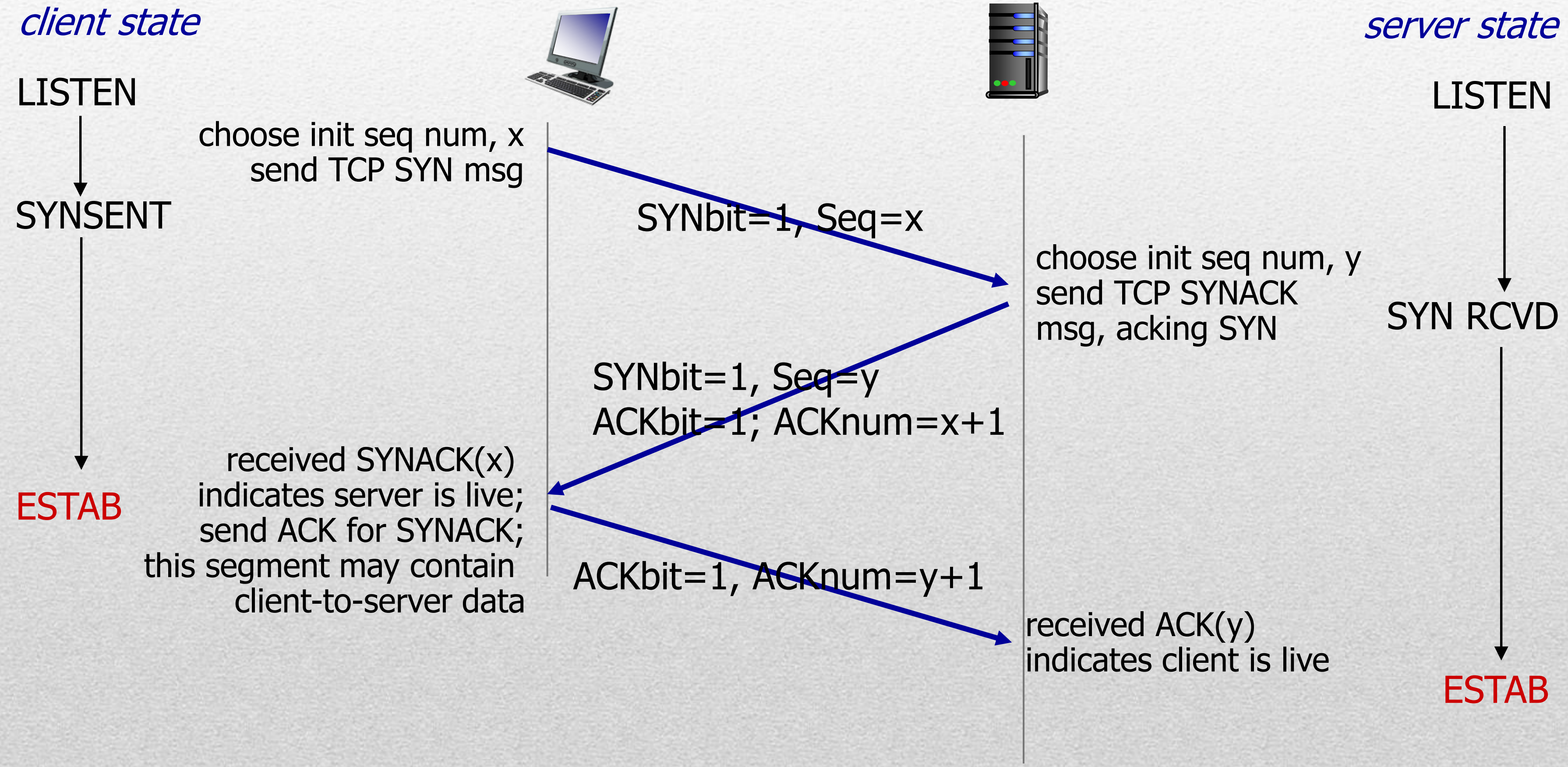
Q: Will 2-way handshake always work in network?

- Variable delays
- Retransmitted messages (e.g. req_conn(x)) due to message loss
- Message reordering
- Cannot "see" other side

Two-way Handshake Failure



TCP Three-way Handshake



client state

LISTEN

SYNSENT

ESTAB

choose init seq num, x
send TCP SYN msg

SYNbit=1, Seq=x

server state

LISTEN

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

SYN RCVD

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

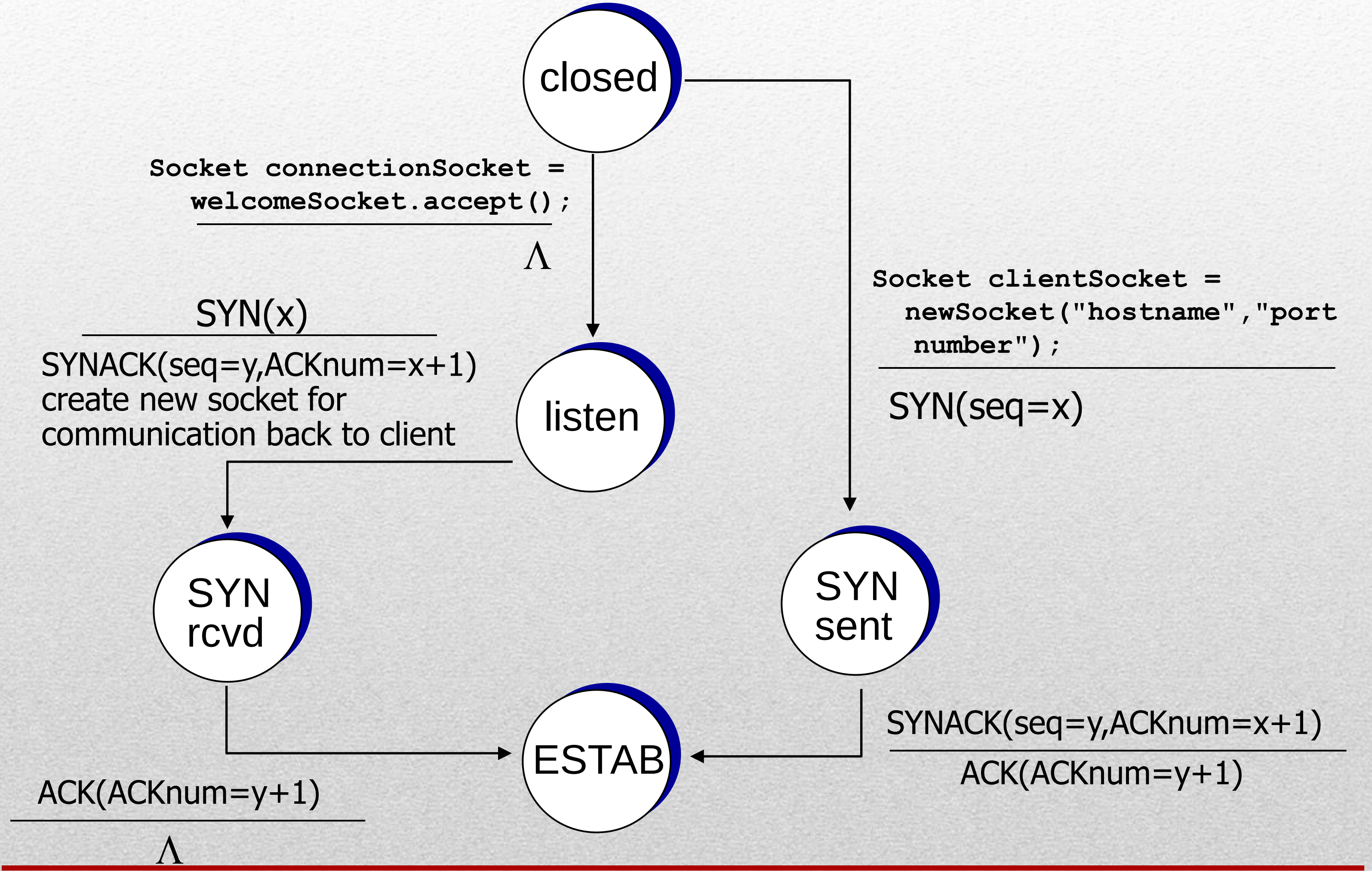
received SYNACK(x)
indicates server is live;
send ACK for SYNACK;
this segment may contain
client-to-server data

ACKbit=1, ACKnum=y+1

received ACK(y)
indicates client is live

ESTAB

TCP Three-way Handshake



closed

Socket connectionSocket =
welcomeSocket.accept();

Λ

listen

SYN(x)

SYNACK(seq=y,ACKnum=x+1)
create new socket for
communication back to client

SYN
rcvd

ACK(ACKnum=y+1)

Λ

ESTAB

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

SYN(seq=x)

SYN
sent

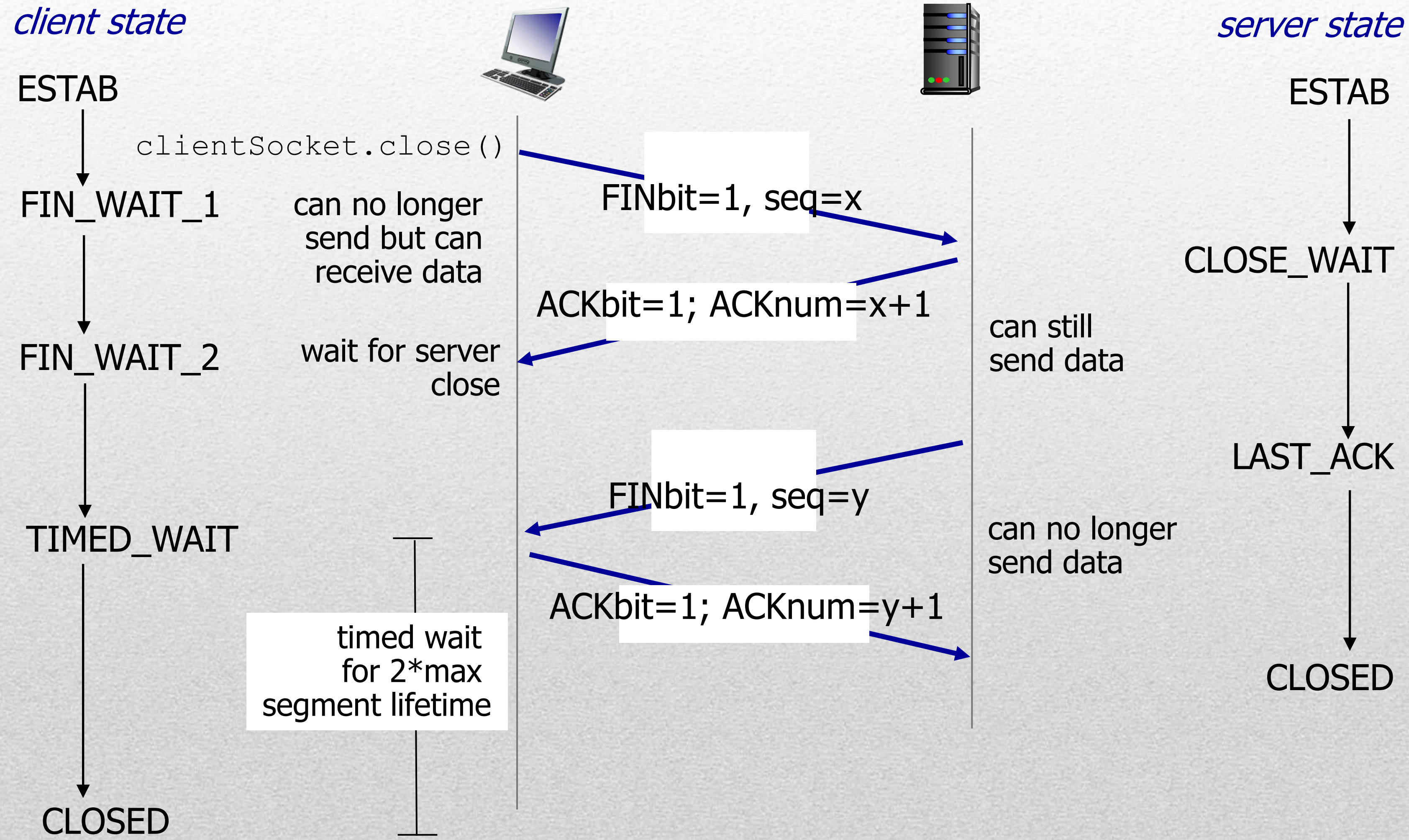
SYNACK(seq=y,ACKnum=x+1)

ACK(ACKnum=y+1)

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

Closing a TCP Connection



client state

server state

ESTAB

ESTAB

`clientSocket.close()`

FIN_WAIT_1

can no longer send but can receive data

FINbit=1, seq=x

CLOSE_WAIT

ACKbit=1; ACKnum=x+1

FIN_WAIT_2

wait for server close

can still send data

LAST_ACK

FINbit=1, seq=y

can no longer send data

TIMED_WAIT

ACKbit=1; ACKnum=y+1

timed wait for 2*max segment lifetime

CLOSED

CLOSED