50.012 Networks

Lecture 10: TCP Part II

2021 Term 6

Assoc. Prof. CHEN Binbin

SINGAPORE UNIVERSITY OF
TECHNOLOGY AND DESIGN

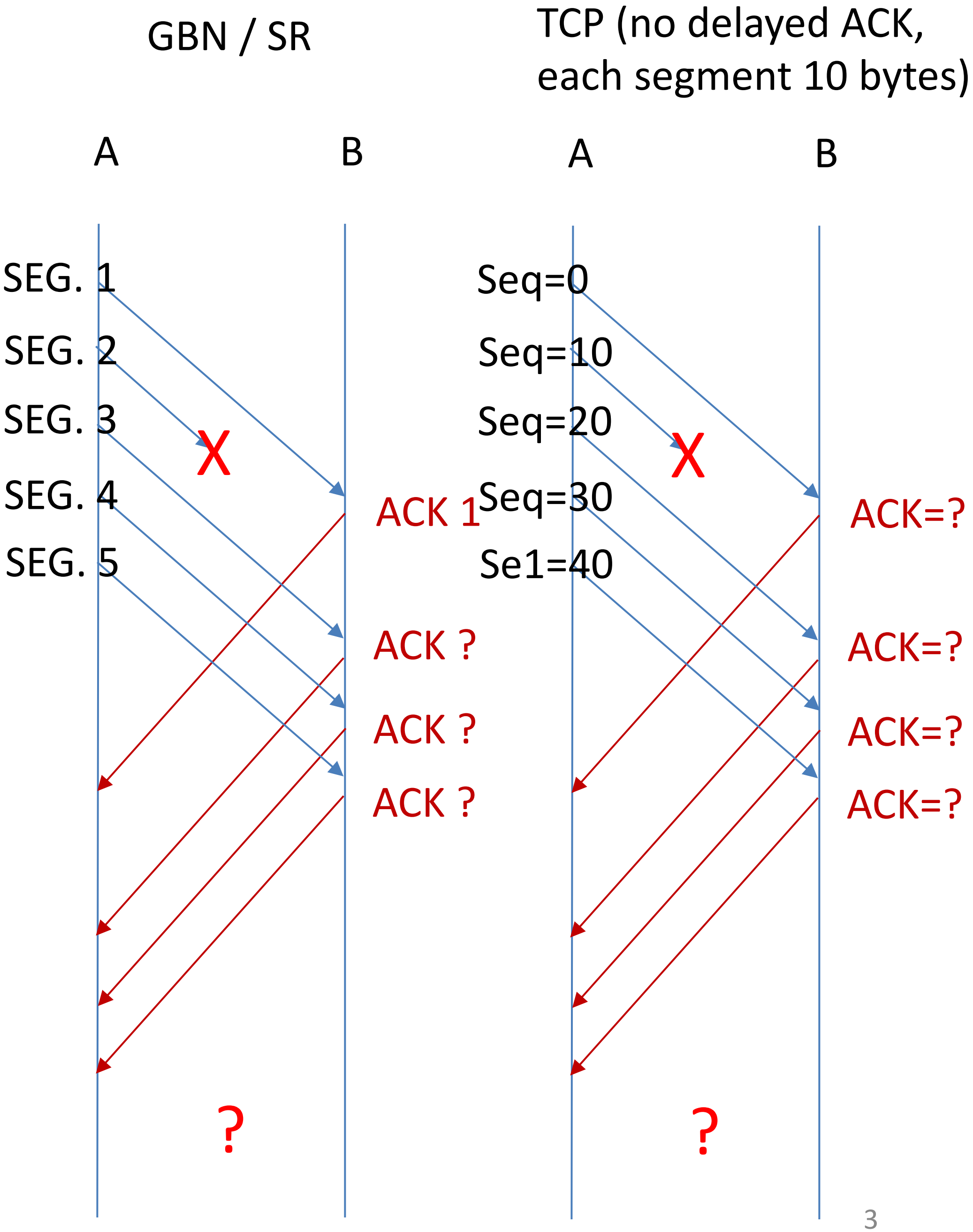
In-class Activity

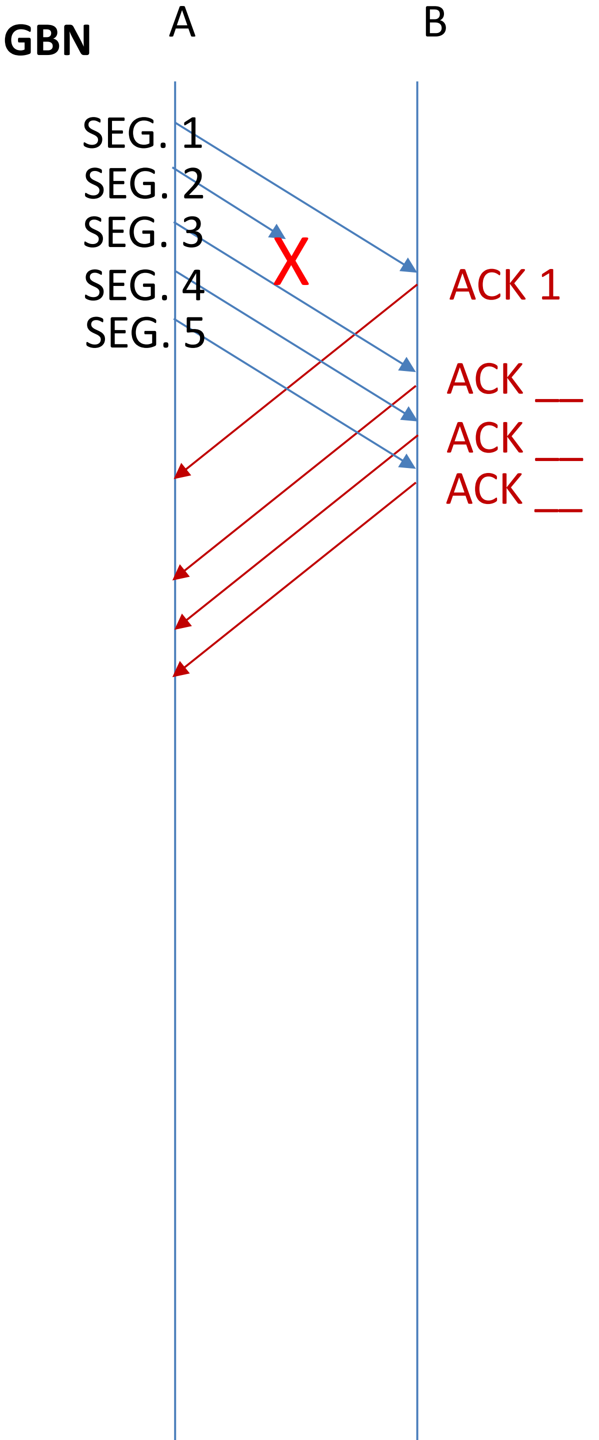
Compare GBN, SR, and TCP (assuming receiver keeps out-of-order segment). Assume the timeout values for all three protocols are sufficiently long such that 5 consecutive data segments and their corresponding ACKs can be received (if not lost).

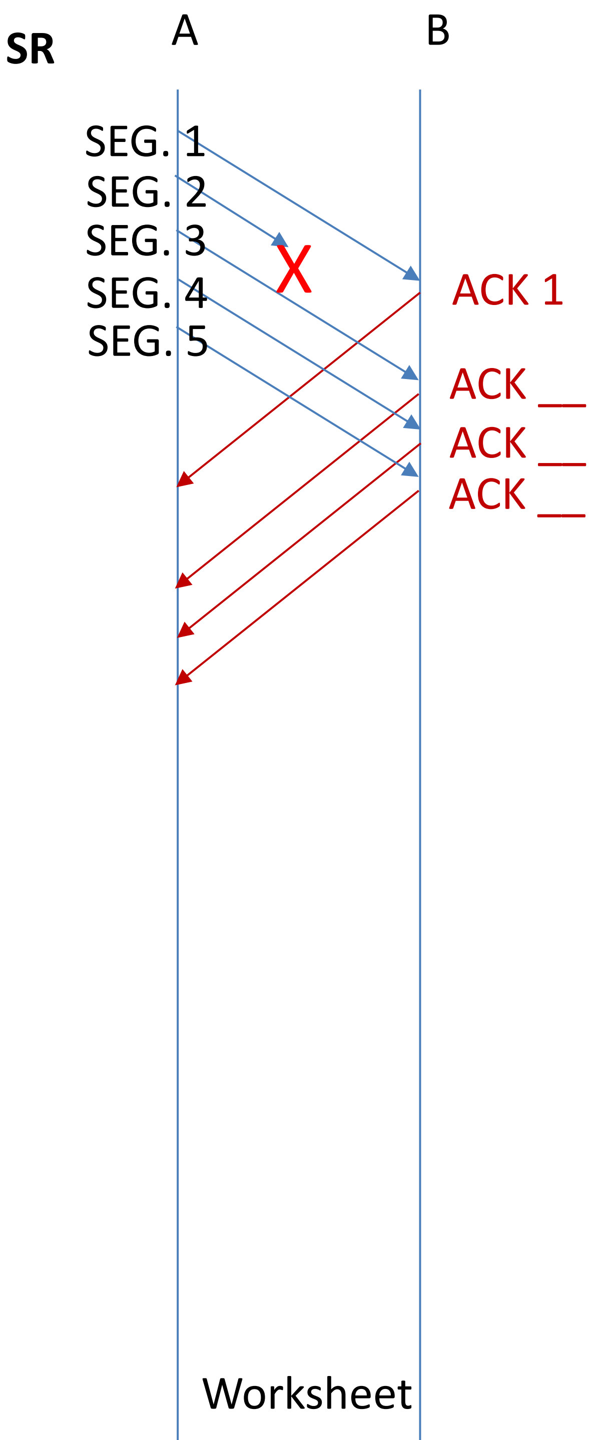
Suppose Host A sends 5 data segments to Host B, and the first transmission of the 2nd segment (sent from A) is the only lost packet during the whole process. In the end, all 5 data segments are correctly received by Host B.

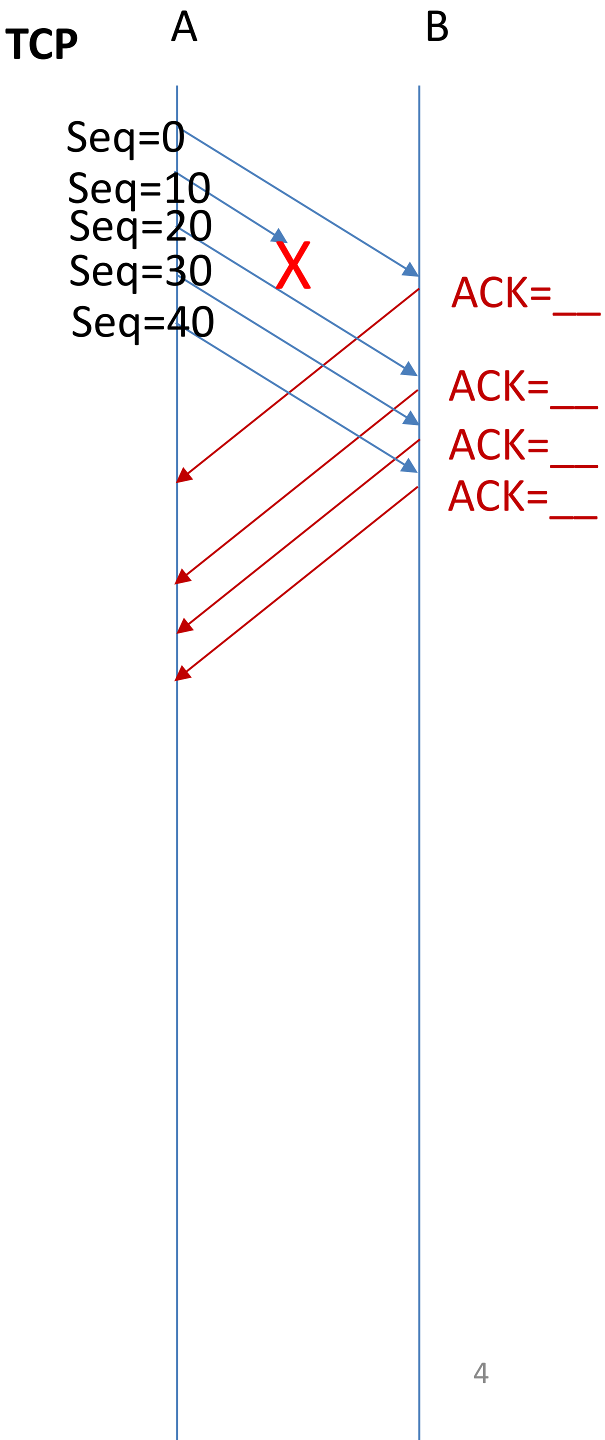
a. How many transmissions does Host A carry out and how many ACKs does Host B send? What are their corresponding sequence & ACK numbers? Complete the S-T diagram for all protocols.

b. If the timeout values for all three protocol are much longer than 5 RTT, which protocol successfully delivers all the five segments in the shortest time interval?




GBN
A
B
SEG. 1
SEG. 2
SEG. 3
SEG. 4
SEG. 5
X
ACK 1
ACK __
ACK __
ACK __


SR
A
B
SEG. 1
SEG. 2
SEG. 3
SEG. 4
SEG. 5
X
ACK 1
ACK __
ACK __
ACK __

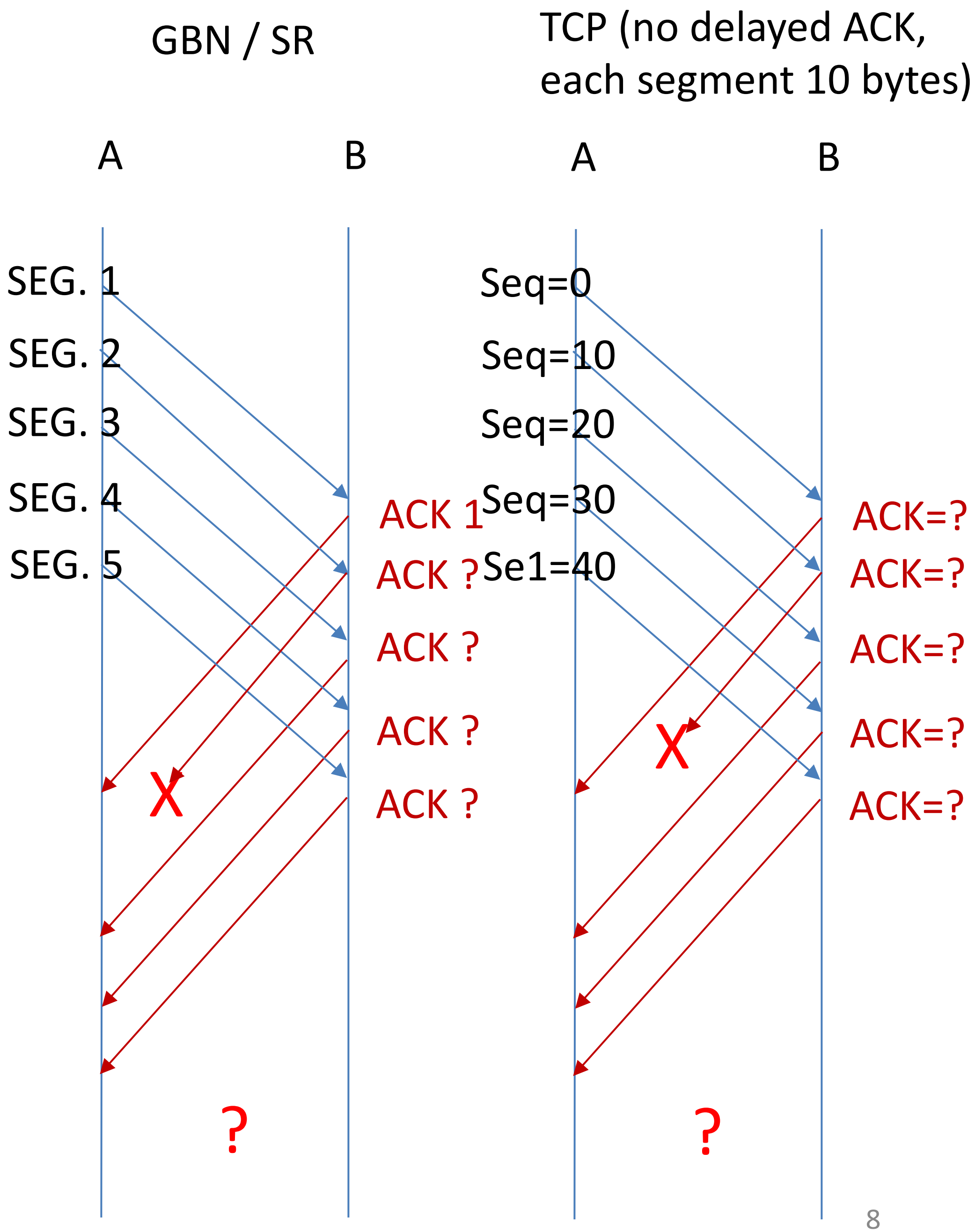

TCP
A
B
Seq=0
Seq=10
Seq=20
Seq=30
Seq=40
X
ACK=__
ACK=__
ACK=__
ACK=__

Worksheet

Compare GBN, SR, and TCP (assuming receiver keeps out-of-order segment). Assume the timeout values for all three protocols are sufficiently long such that 5 consecutive data segments and their corresponding ACKs can be received (if not lost).

Suppose Host A sends 5 data segments to Host B, and the first transmission of the 2nd segment's ACK is the only lost packet during the whole process. In the end, all 5 data segments are correctly received by Host B.

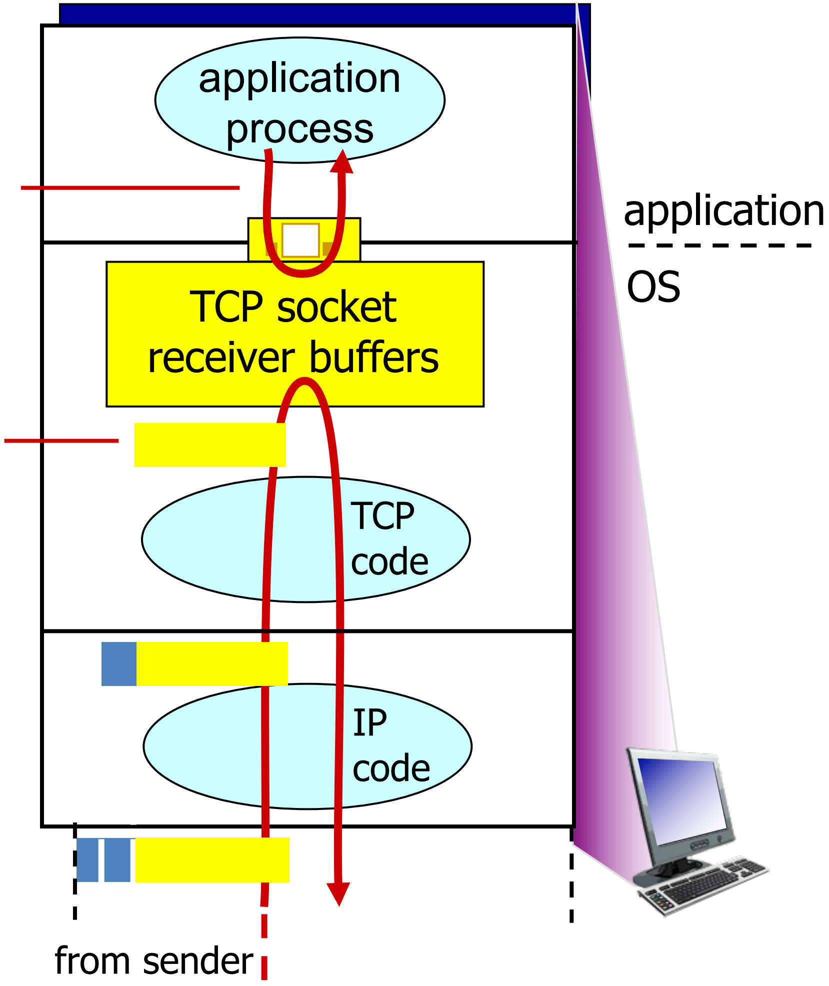
Q: How many transmissions does Host A carry out and how many ACKs does Host B send? What are their corresponding sequence & ACK numbers? Complete the S-T diagram for all protocols.



TCP flow control

application may remove data from TCP socket buffers ….

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



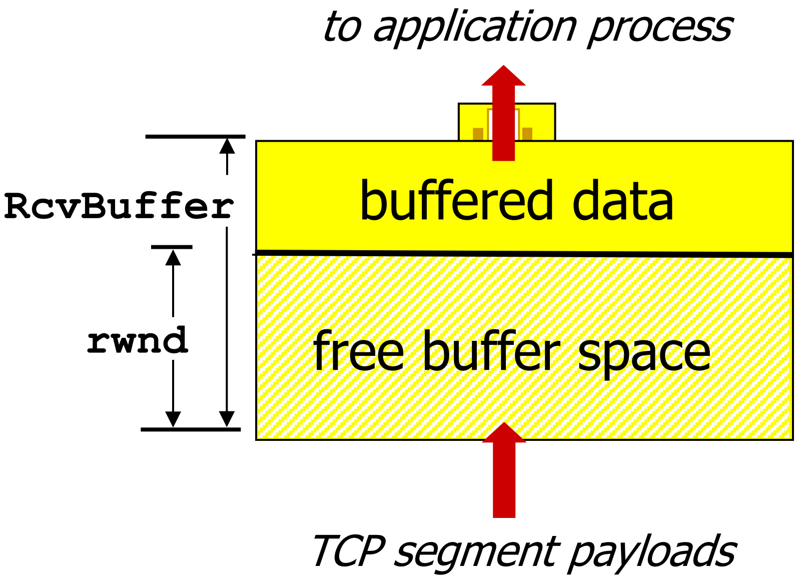
receiver protocol stack

flow control

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

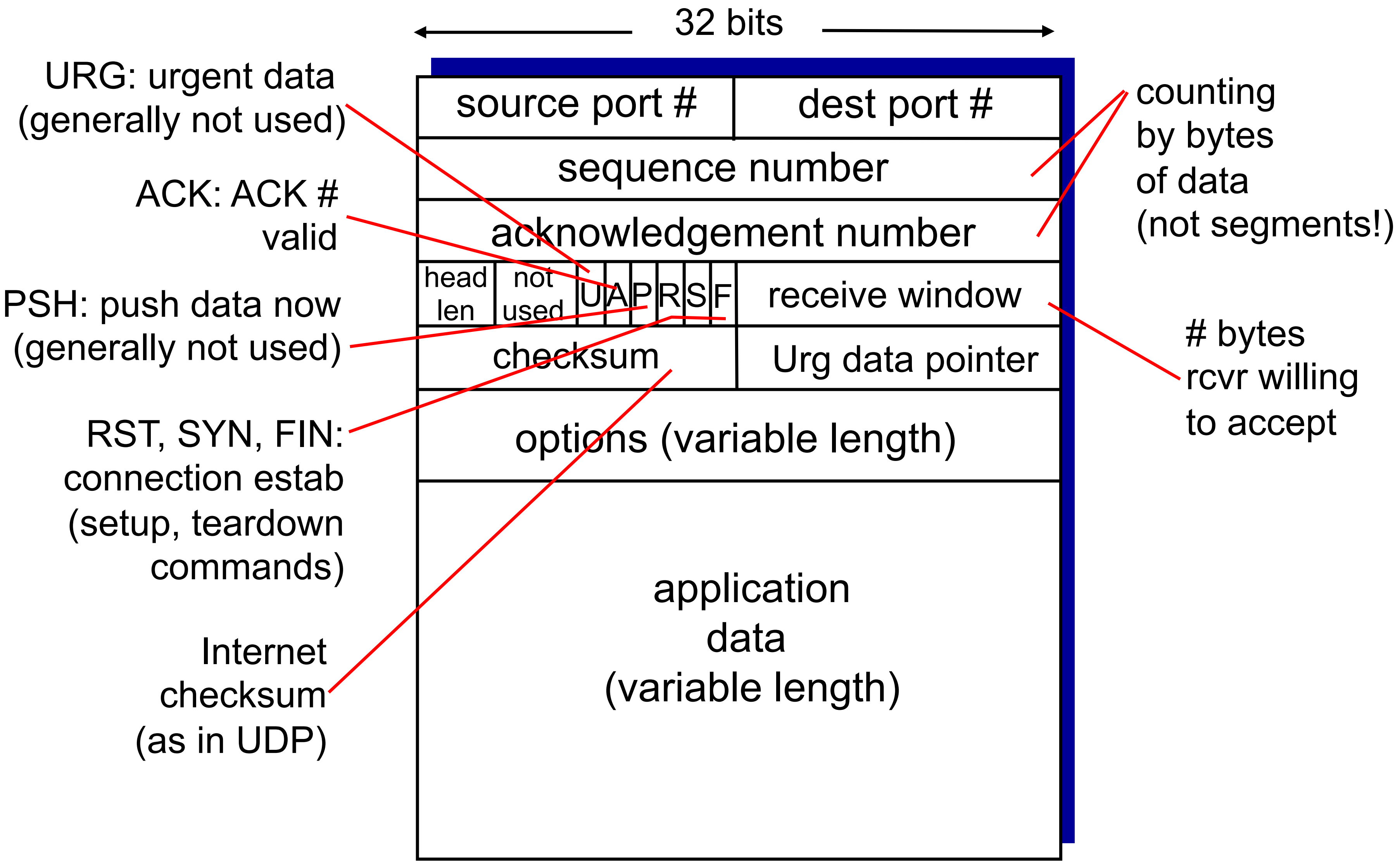
TCP flow control

- 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



receiver-side buffering

TCP segment structure



32 bits

source port #								dest port #
sequence number								
acknowledgement number								
head len	not used	U	A	P	R	S	F	receive window
checksum								Urg data pointer
options (variable length)								
application data (variable length)								

URG: urgent data (generally not used)

ACK: ACK # valid

PSH: push data now (generally not used)

RST, SYN, FIN: connection estab (setup, teardown commands)

Internet checksum (as in UDP)

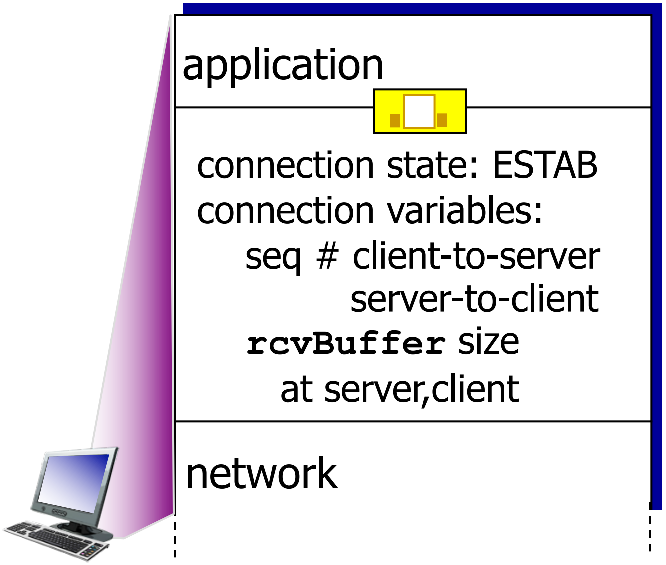
counting by bytes of data (not segments!)

# bytes rcvr willing to accept

Connection Management

before exchanging data, sender/receiver “handshake”:

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

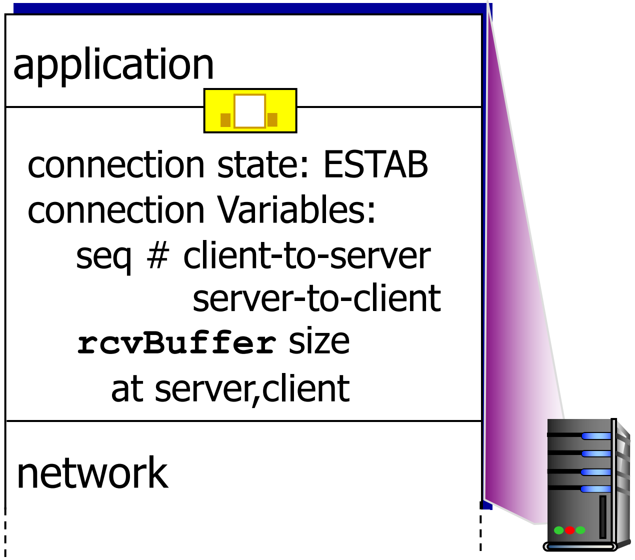


application

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

network

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



application

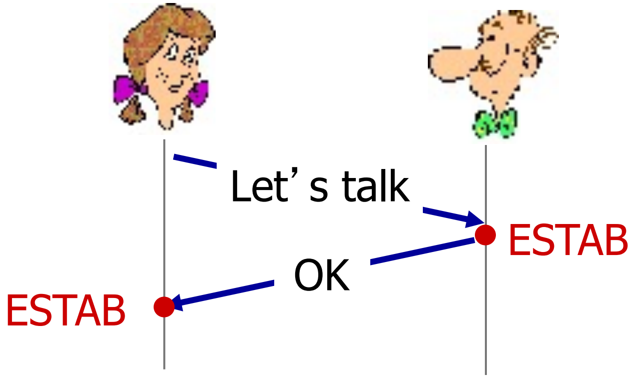
connection state: ESTAB
connection Variables:
seq # client-to-server
server-to-client
rcvBuffer size
at server,client

network

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

Agreeing to establish a connection

2-way handshake:

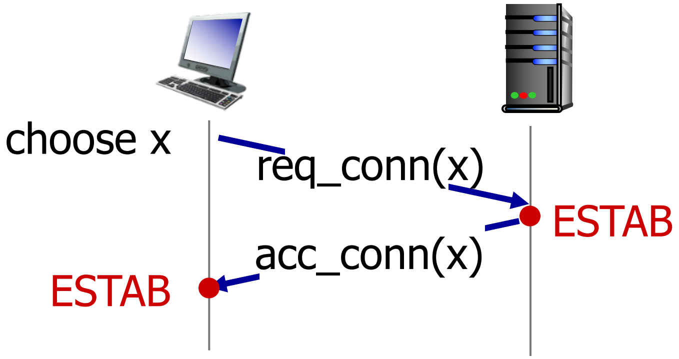


Let’ s talk

ESTAB

OK

ESTAB



choose x

req_conn(x)

ESTAB

acc_conn(x)

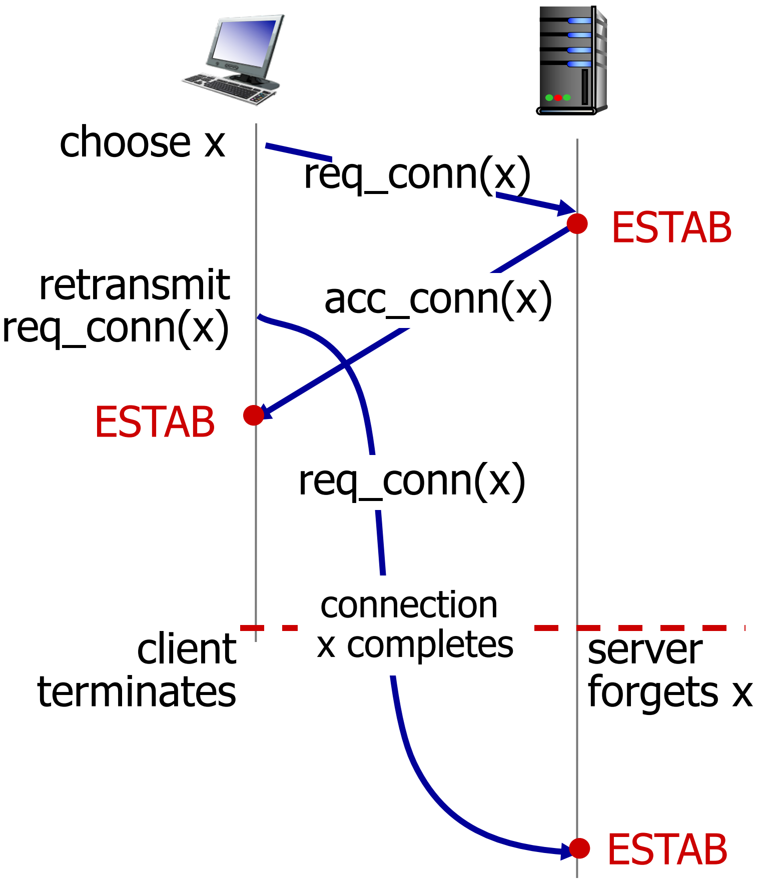
ESTAB

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:



choose x

req_conn(x)

ESTAB

retransmit req_conn(x)

acc_conn(x)

ESTAB

req_conn(x)

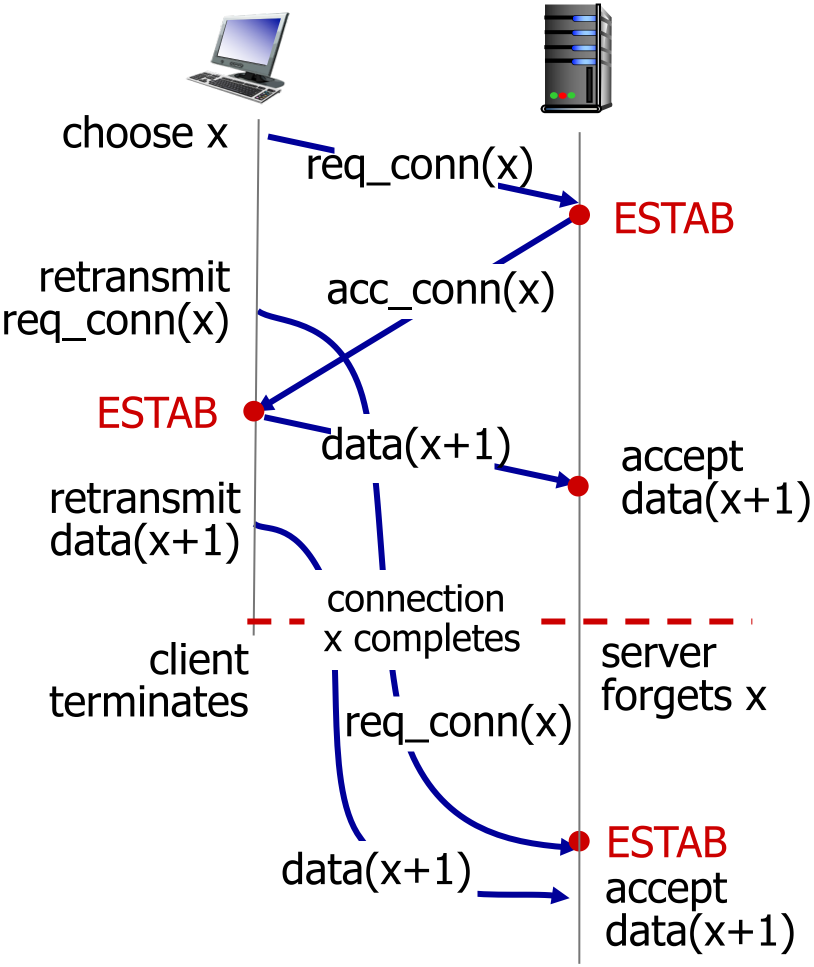
client terminates

connection x completes

server forgets x

ESTAB

half open connection!
(no client!)



choose x

req_conn(x)

ESTAB

retransmit req_conn(x)

acc_conn(x)

ESTAB

data(x+1)

accept data(x+1)

retransmit data(x+1)

client terminates

connection x completes

server forgets x

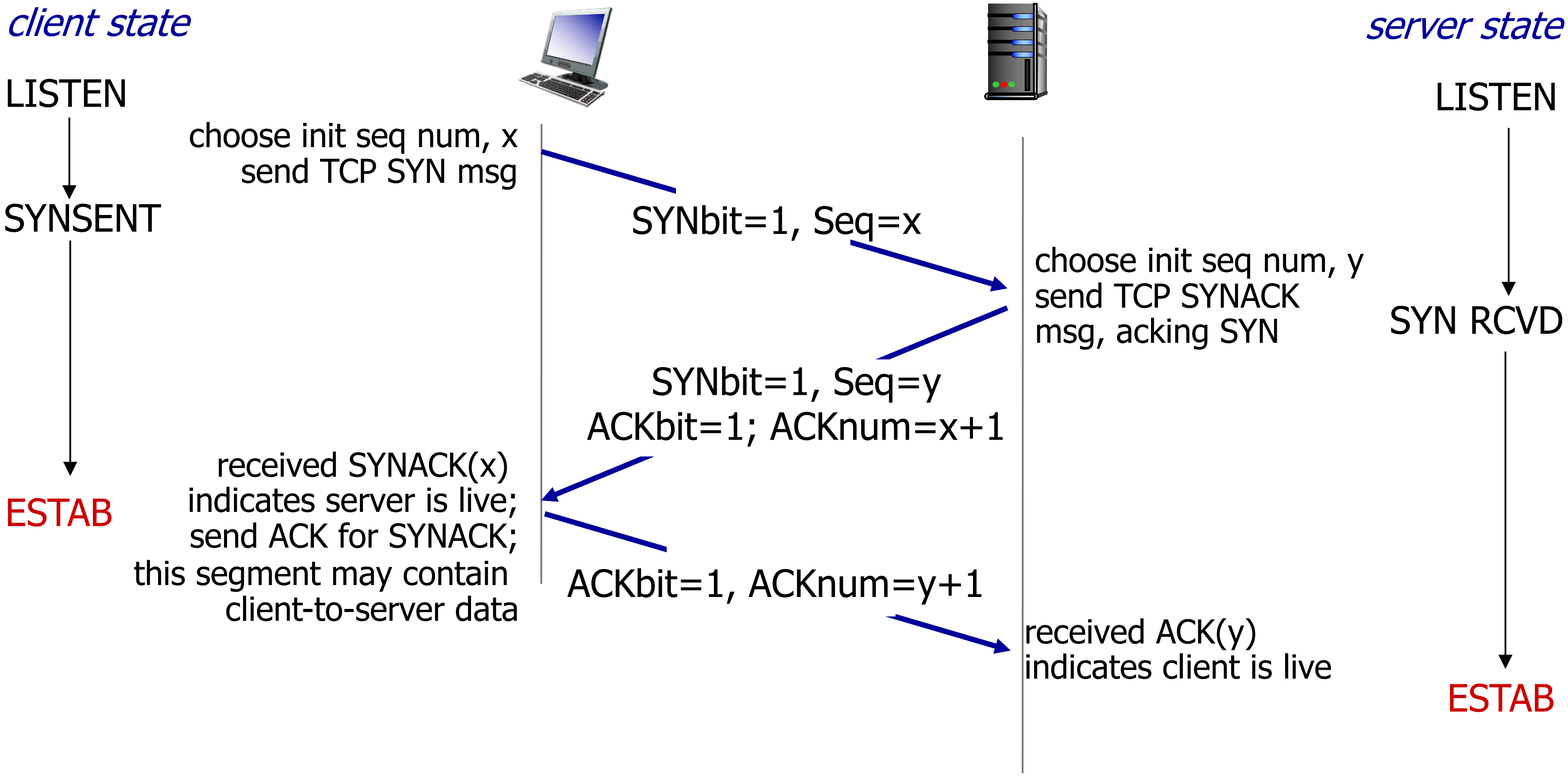
req_conn(x)

ESTAB

data(x+1)

accept data(x+1)

TCP 3-way handshake



client state

LISTEN

choose init seq num, x
send TCP SYN msg

SYNSENT

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

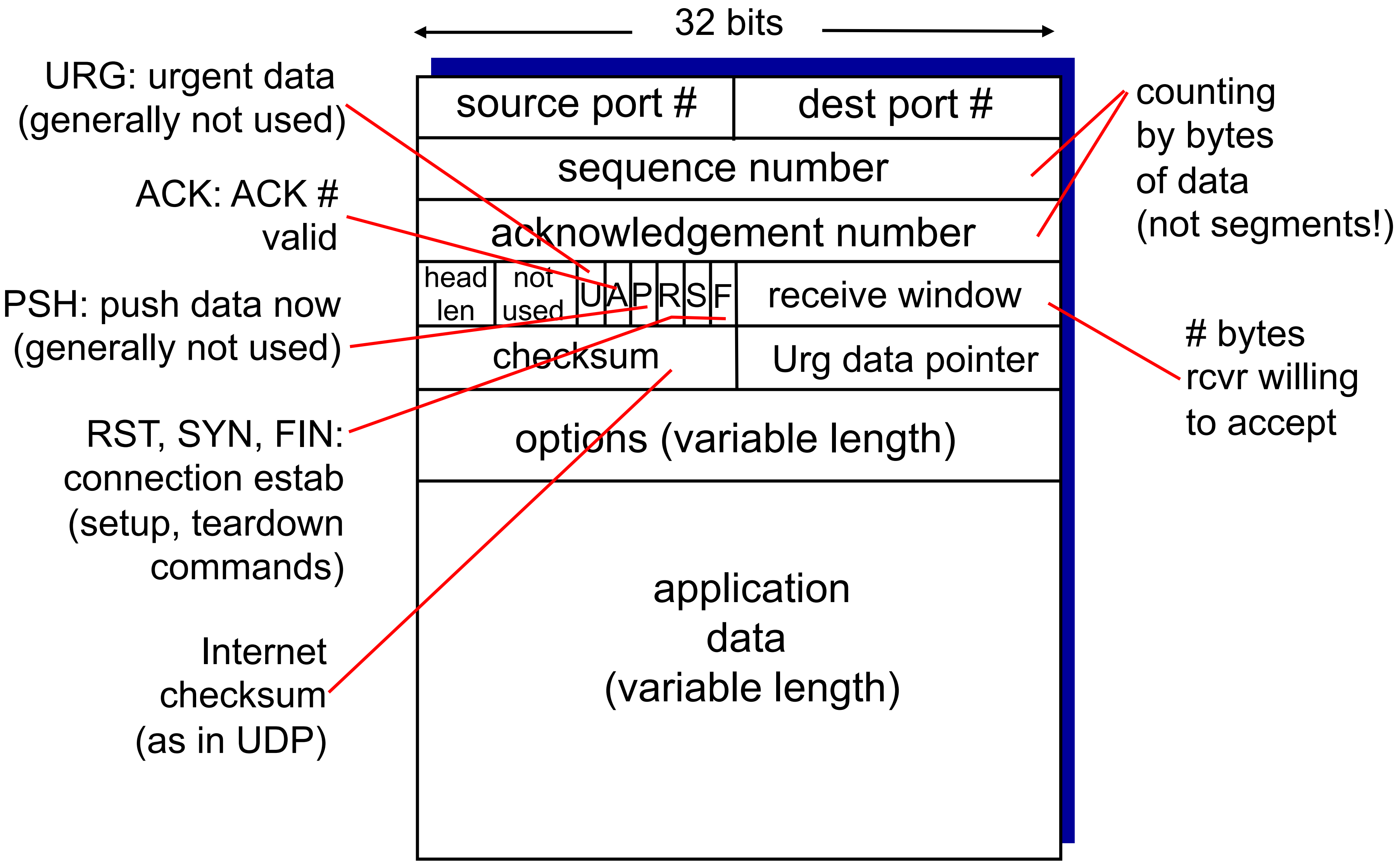
ESTAB

ACKbit=1, ACKnum=y+1

received ACK(y)
indicates client is live

ESTAB

TCP segment structure



32 bits

source port #							dest port #	
sequence number								
acknowledgement number								
head len	not used	U	A	P	R	S	F	receive window
checksum							Urg data pointer	
options (variable length)								
application data (variable length)								

URG: urgent data (generally not used)

ACK: ACK # valid

PSH: push data now (generally not used)

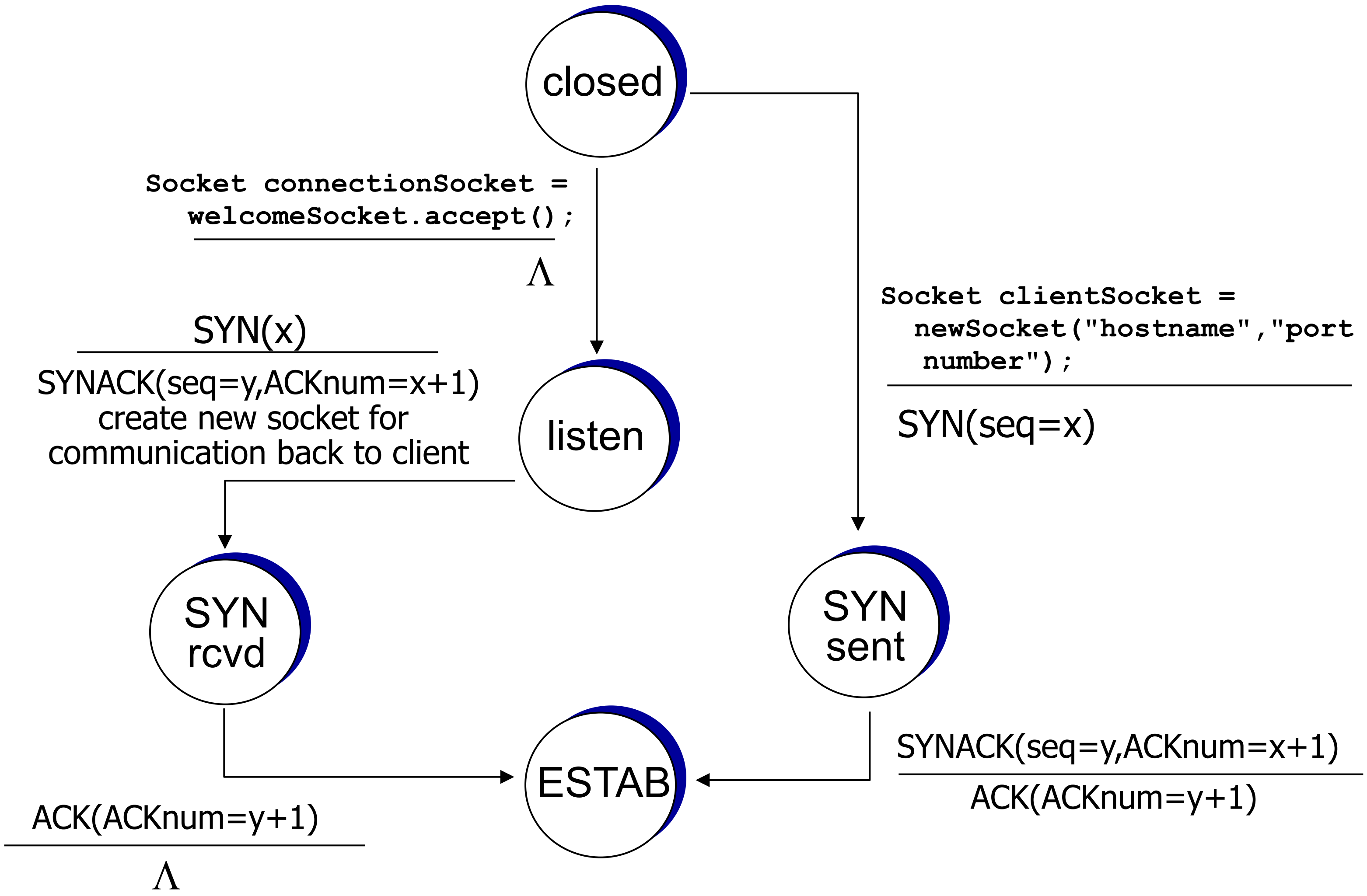
RST, SYN, FIN: connection estab (setup, teardown commands)

Internet checksum (as in UDP)

counting by bytes of data (not segments!)

# bytes rcvr willing to accept

TCP 3-way handshake: FSM



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

Λ

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

SYN(seq=x)

SYN(x)

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

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

ACK(ACKnum=y+1)

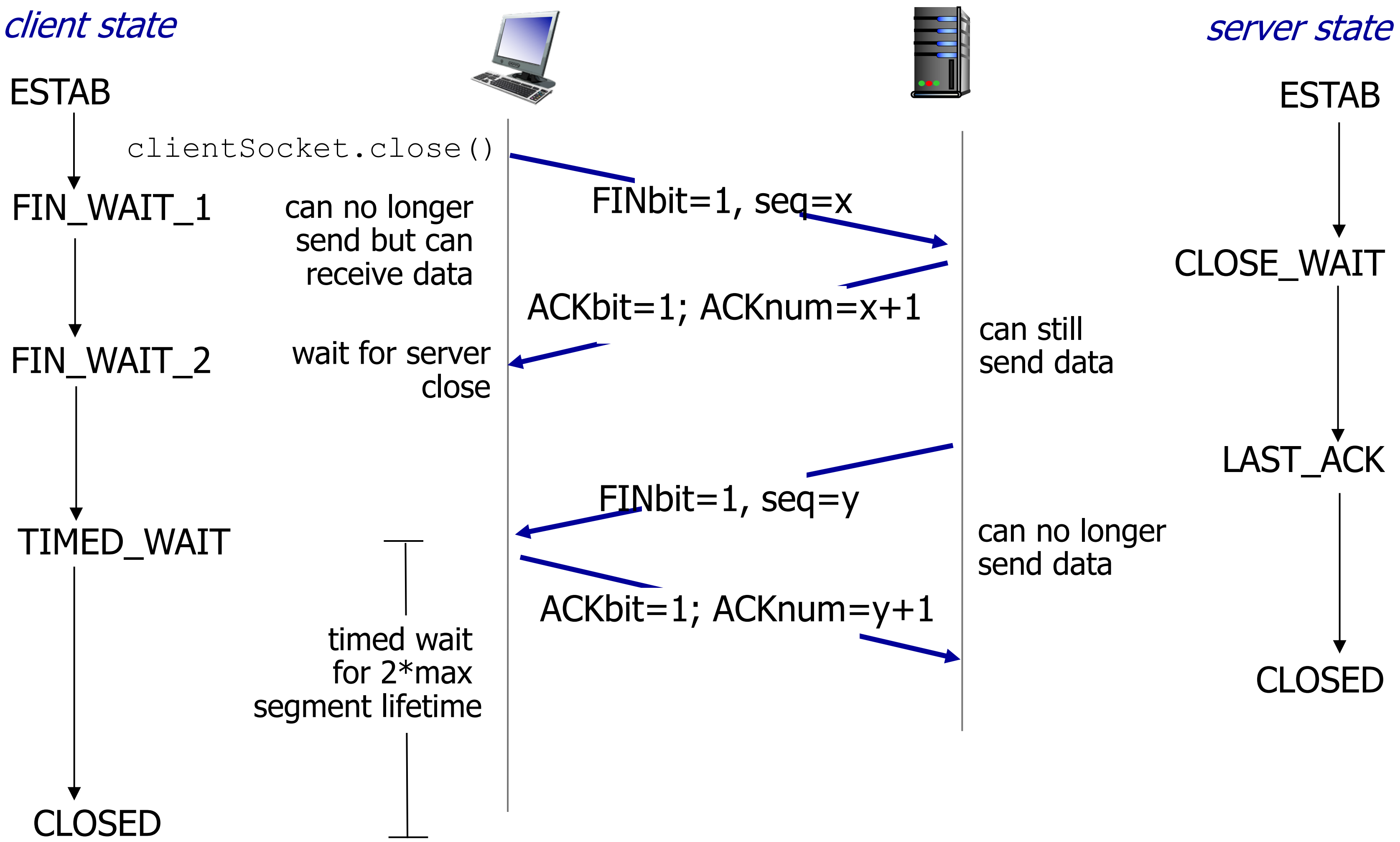
ACK(ACKnum=y+1)

Λ

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



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
can still send data
FIN_WAIT_2
wait for server close
LAST_ACK
FINbit=1, seq=y
TIMED_WAIT
can no longer send data
ACKbit=1; ACKnum=y+1
timed wait for 2*max segment lifetime
CLOSED
CLOSED