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

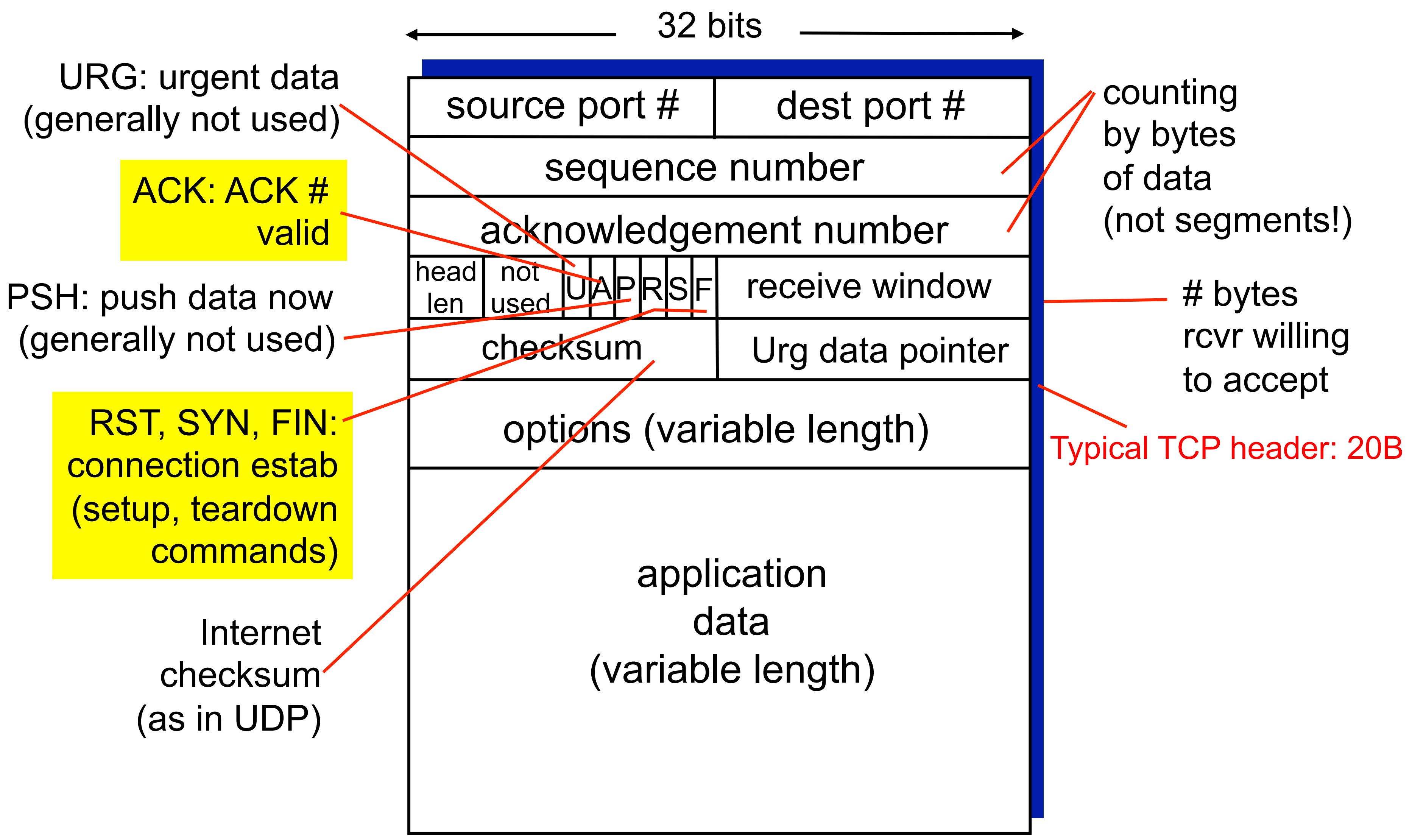
3.5 connection-oriented transport: TCP

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

3.6 principles of congestion control

3.7 TCP congestion control

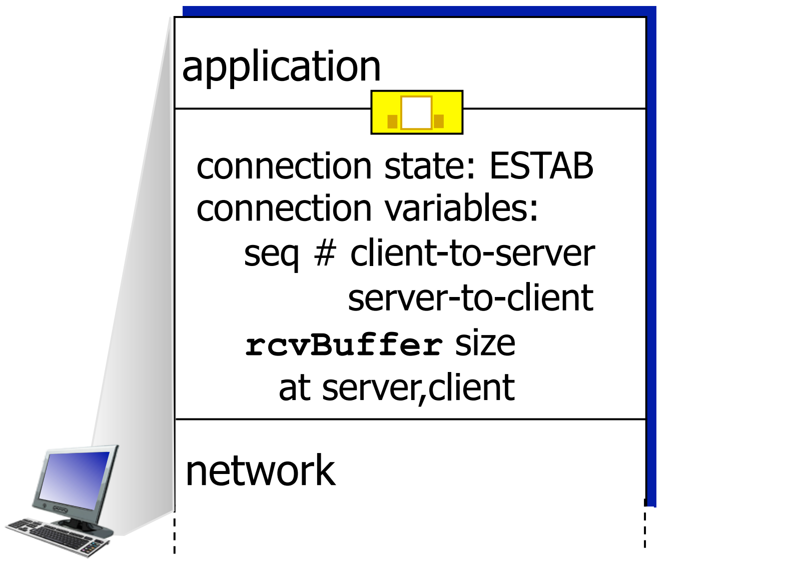
[TCP segment structure]


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
Typical TCP header: 20B

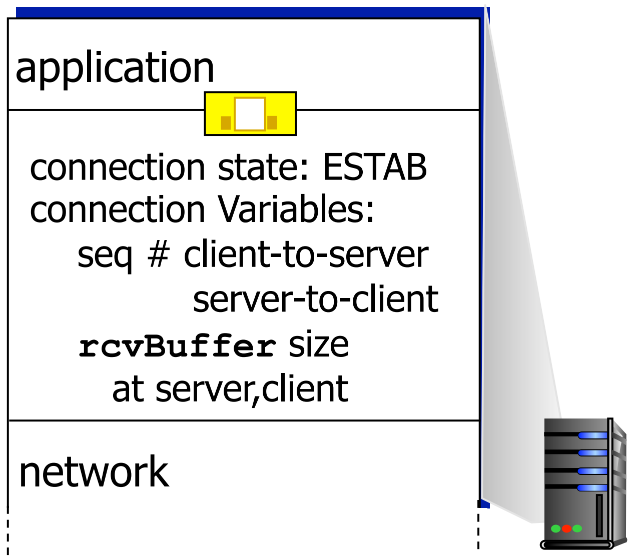
Connection Management

before exchanging data, sender/receiver "handshake":

- agree to establish connection
- agree on connection parameters



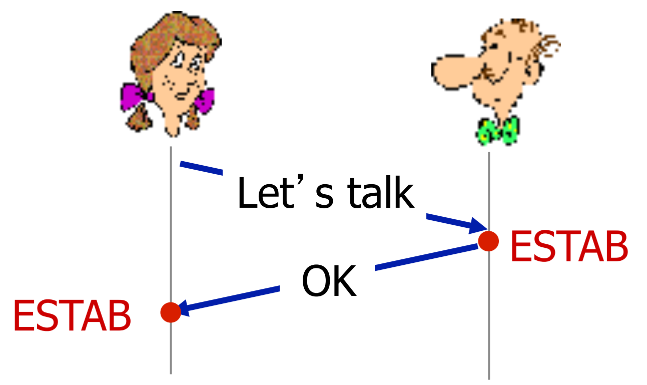
clientSocket.connect((serverName,serverPort))

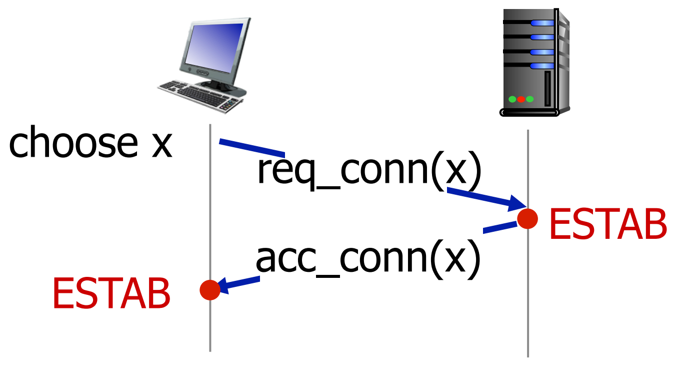


connectionSocket, addr = serverSocket.accept()

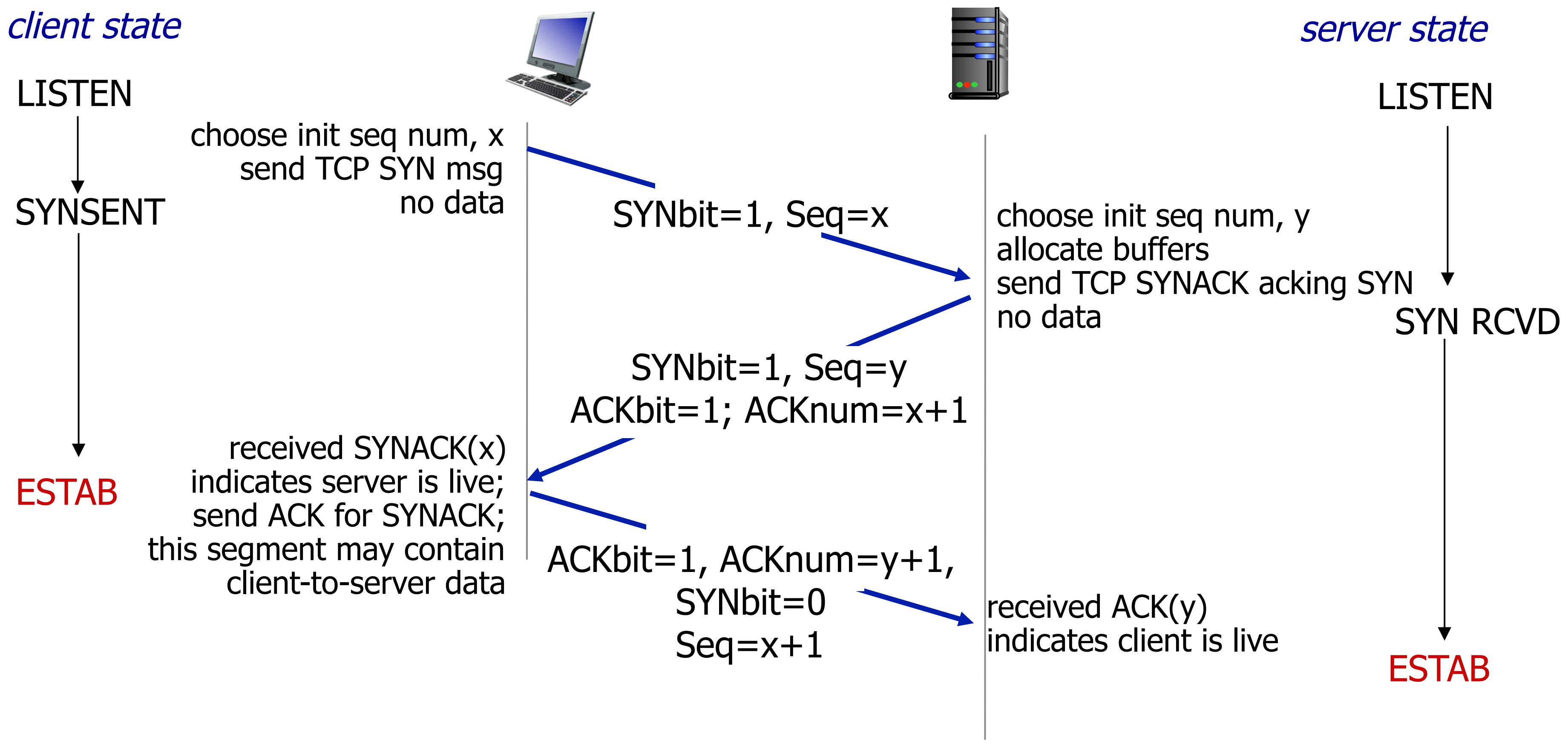
Agreeing to establish a connection

2-way handshake:

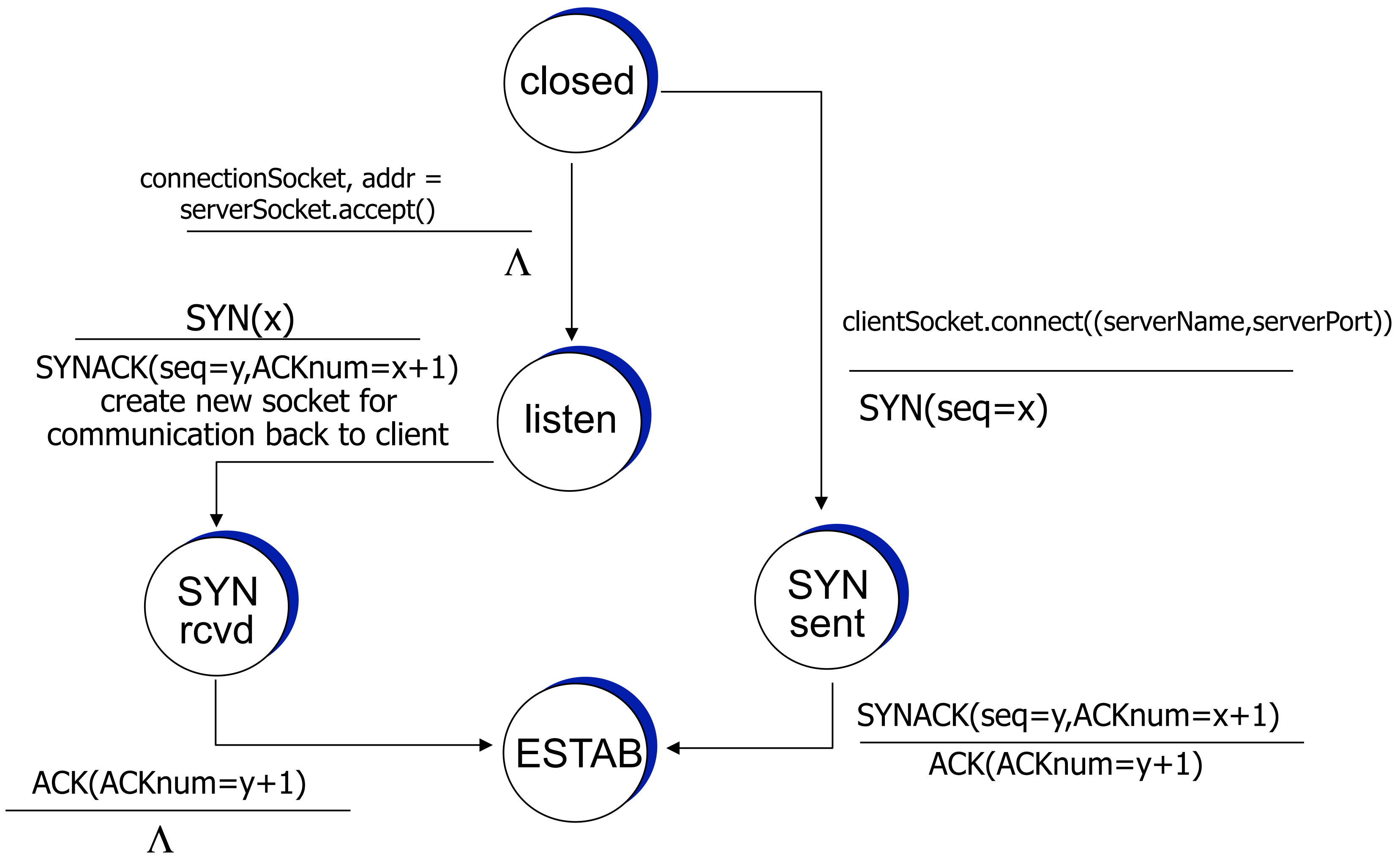




TCP 3-way handshake

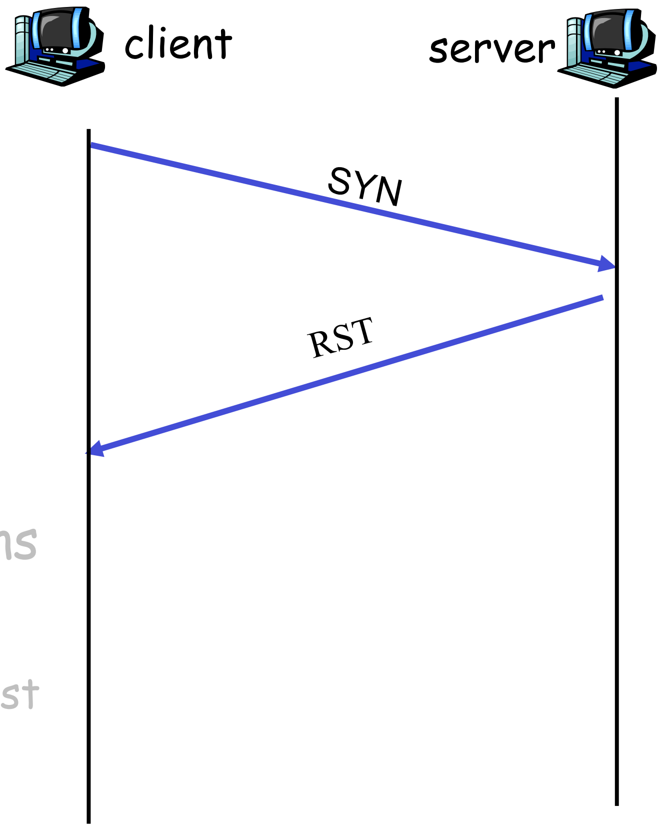


TCP 3-way handshake: FSM


closed
connectionSocket, addr = serverSocket.accept()
Λ
SYN(x)
SYNACK(seq=y,ACKnum=x+1)
create new socket for communication back to client
listen
clientSocket.connect((serverName,serverPort))
SYN(seq=x)
SYN rcvd
SYN sent
ESTAB
SYNACK(seq=y,ACKnum=x+1)
ACK(ACKnum=y+1)
ACK(ACKnum=y+1)
Λ

Server may not accept the connection

- Why?
  - Server may not be accepting TCP connections to that port
  - Server may be out of resources
- What happens?
  - Sends RST
  - No connection established
  - No resources allocated
- USP servers do not have connections
  - just listen to a socket on a dest port#
  - If server receives a UDP packet with dest port# that does not match an existing UDP socket → Sends ICMP message back

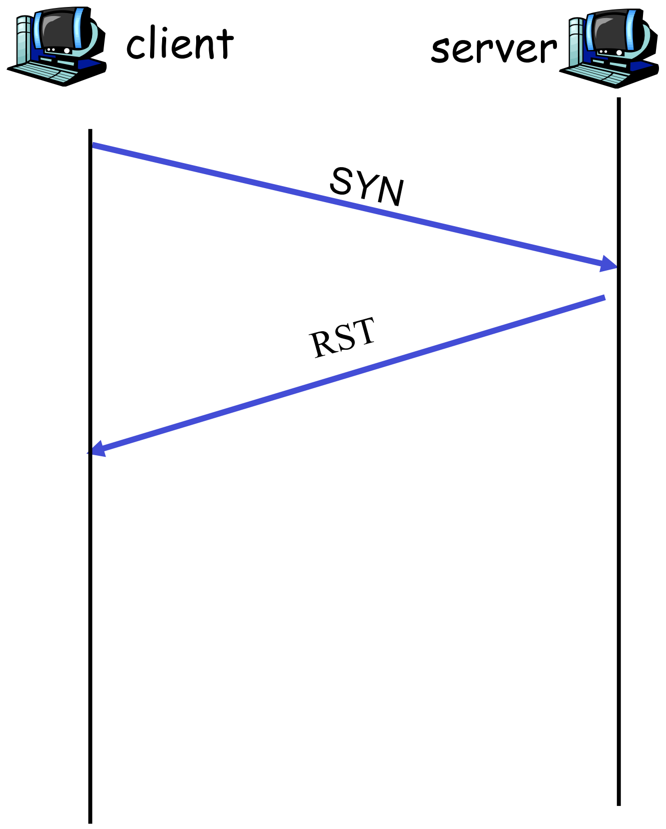


Scanning ports

- www.nmap.org

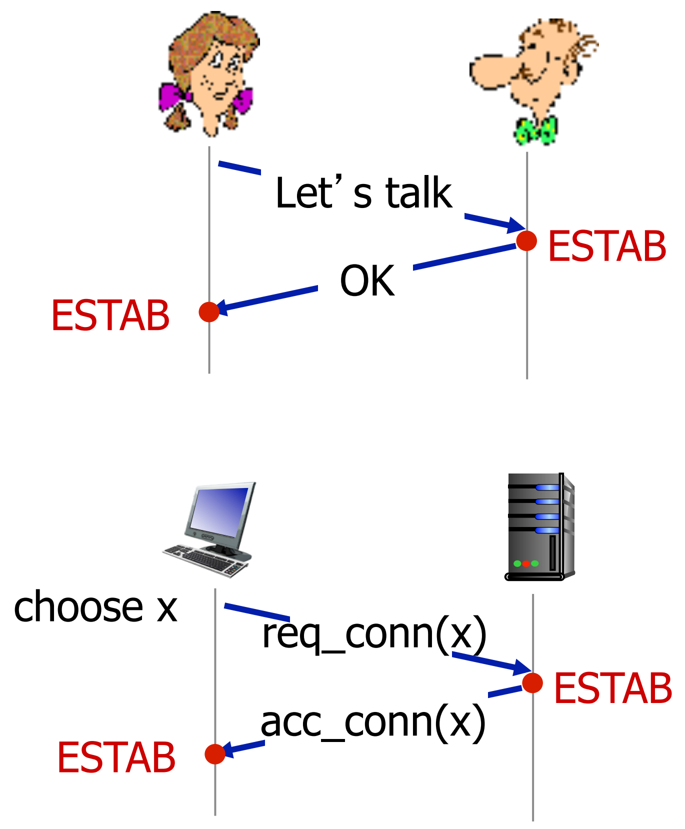
- Scanning TCP ports
  - Send TCP SYN
  - receive SYNACK, RST, nothing

- Scanning UDP ports
  - Receive ICNP messages



[Agreeing to establish a connection]

2-way handshake:

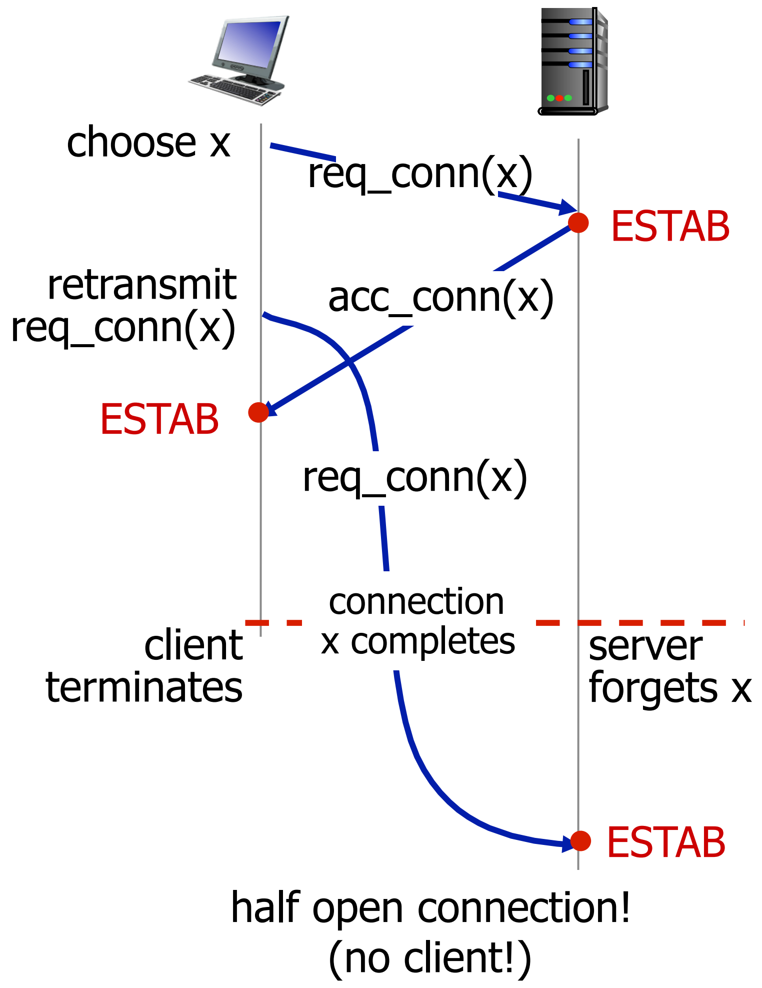


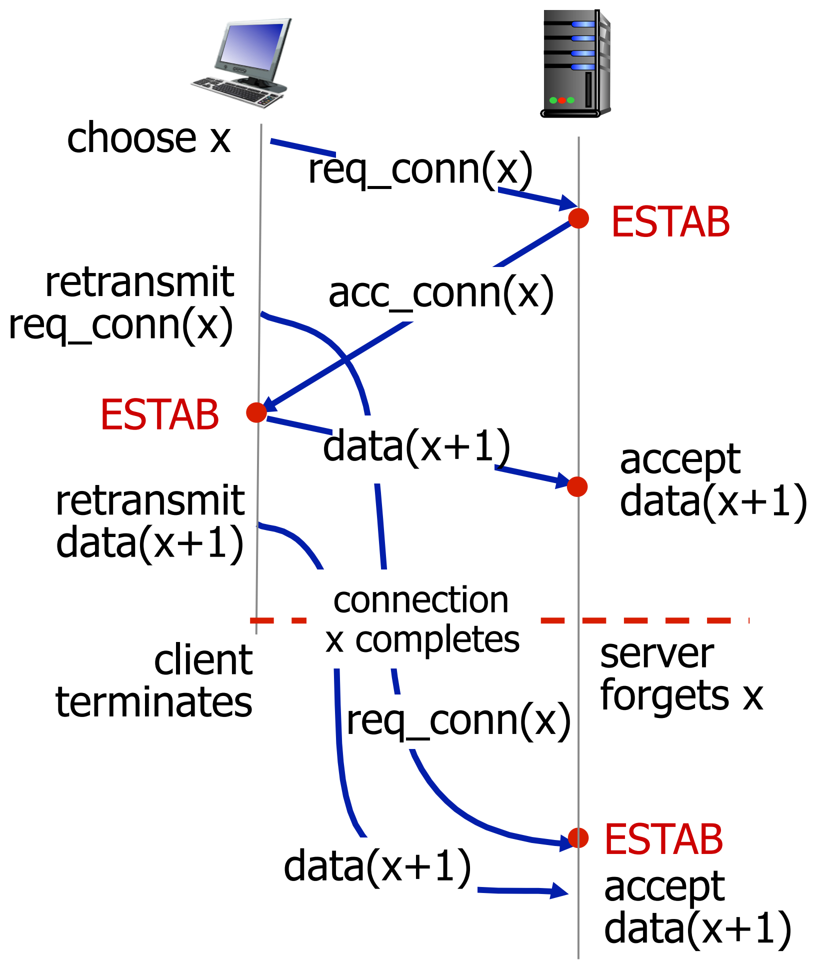
Q: will 2-way handshake always work in network?

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

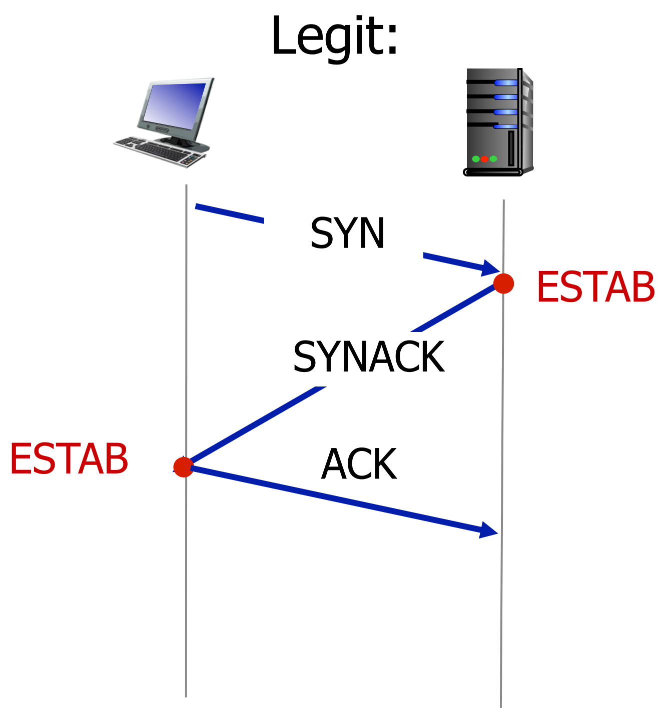
[Why not 2-way connection estalishement?]

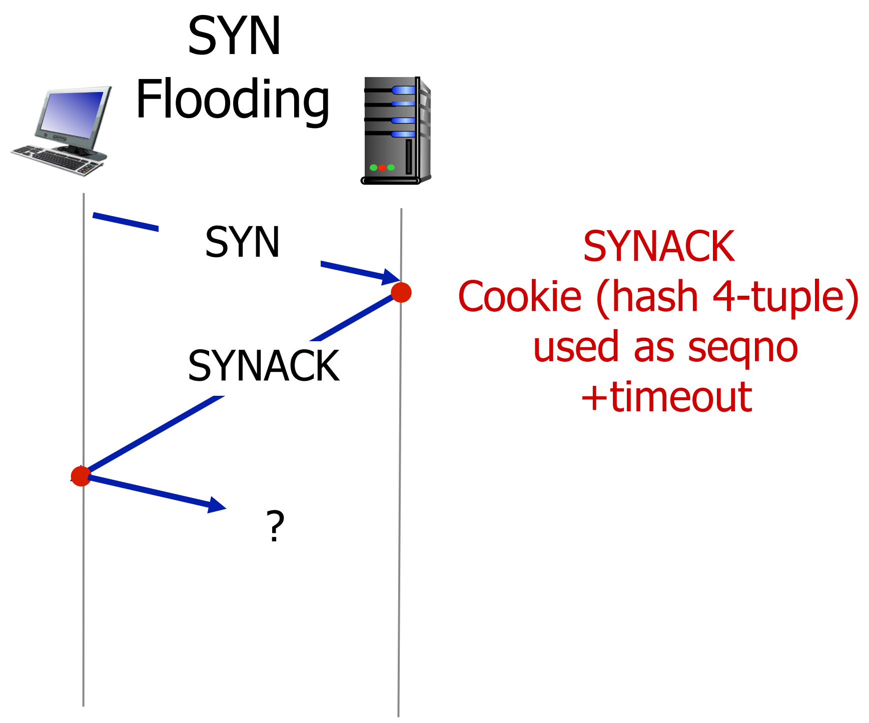
2-way handshake failure scenarios:



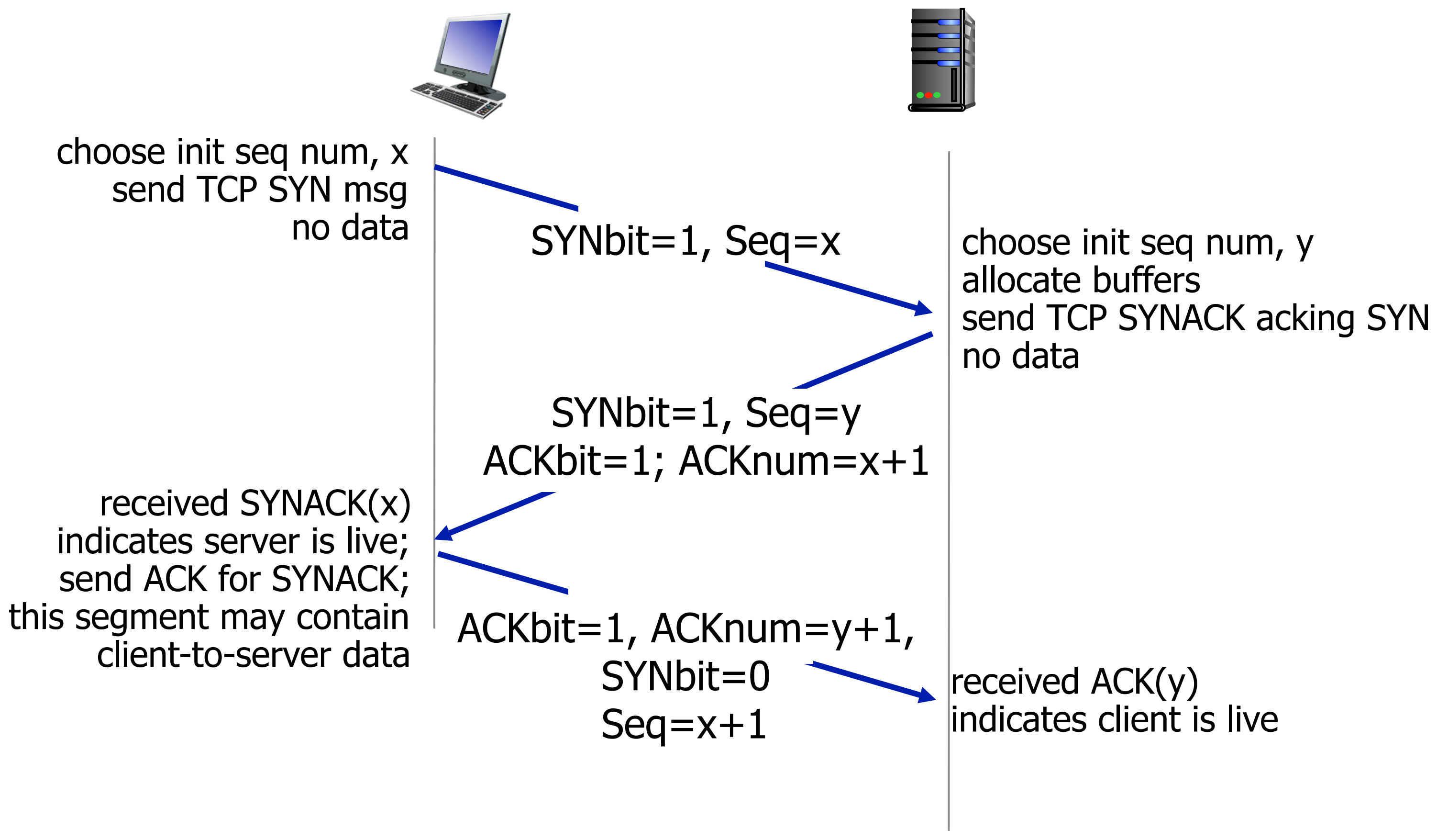


Attack 1: SYN Flooding

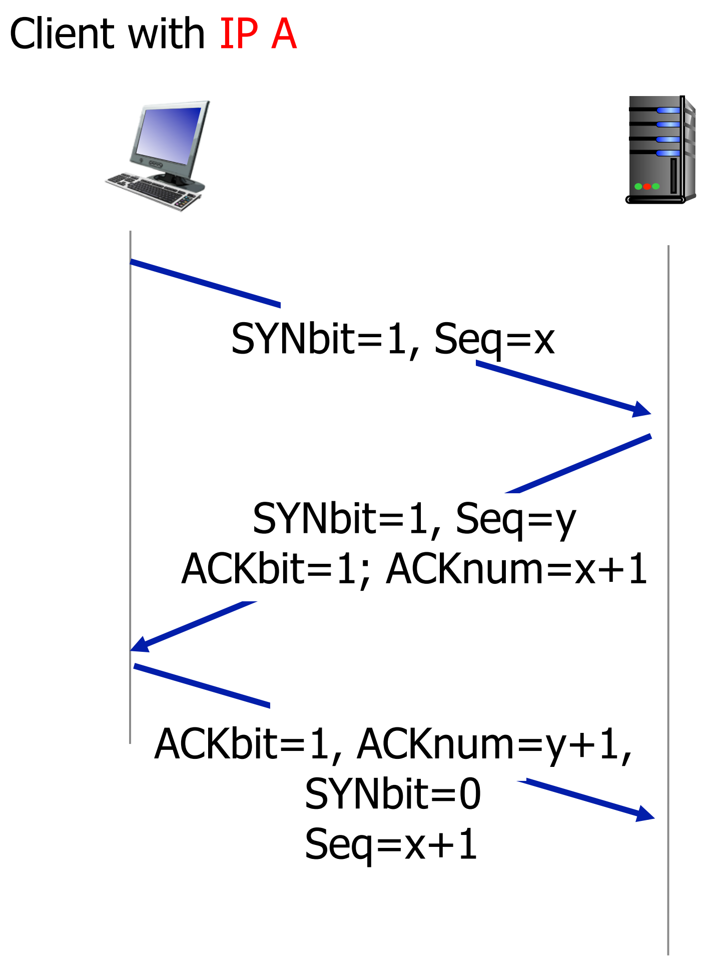




TCP 3-way handshake - revisited

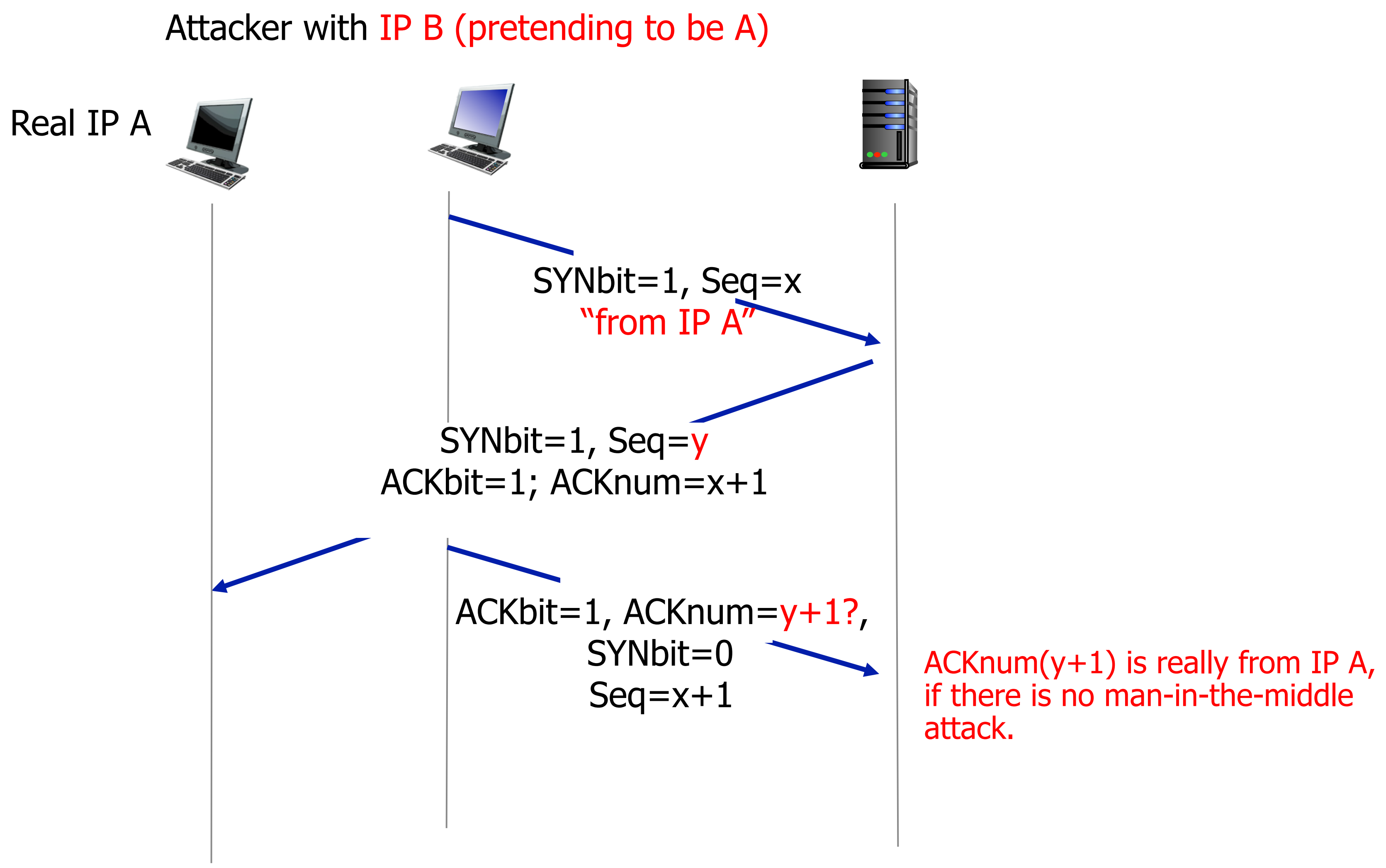


Attack II: Spoofing



Can the server be sure that this is really A, and not B pretending to be A?

Attack II: Spoofing

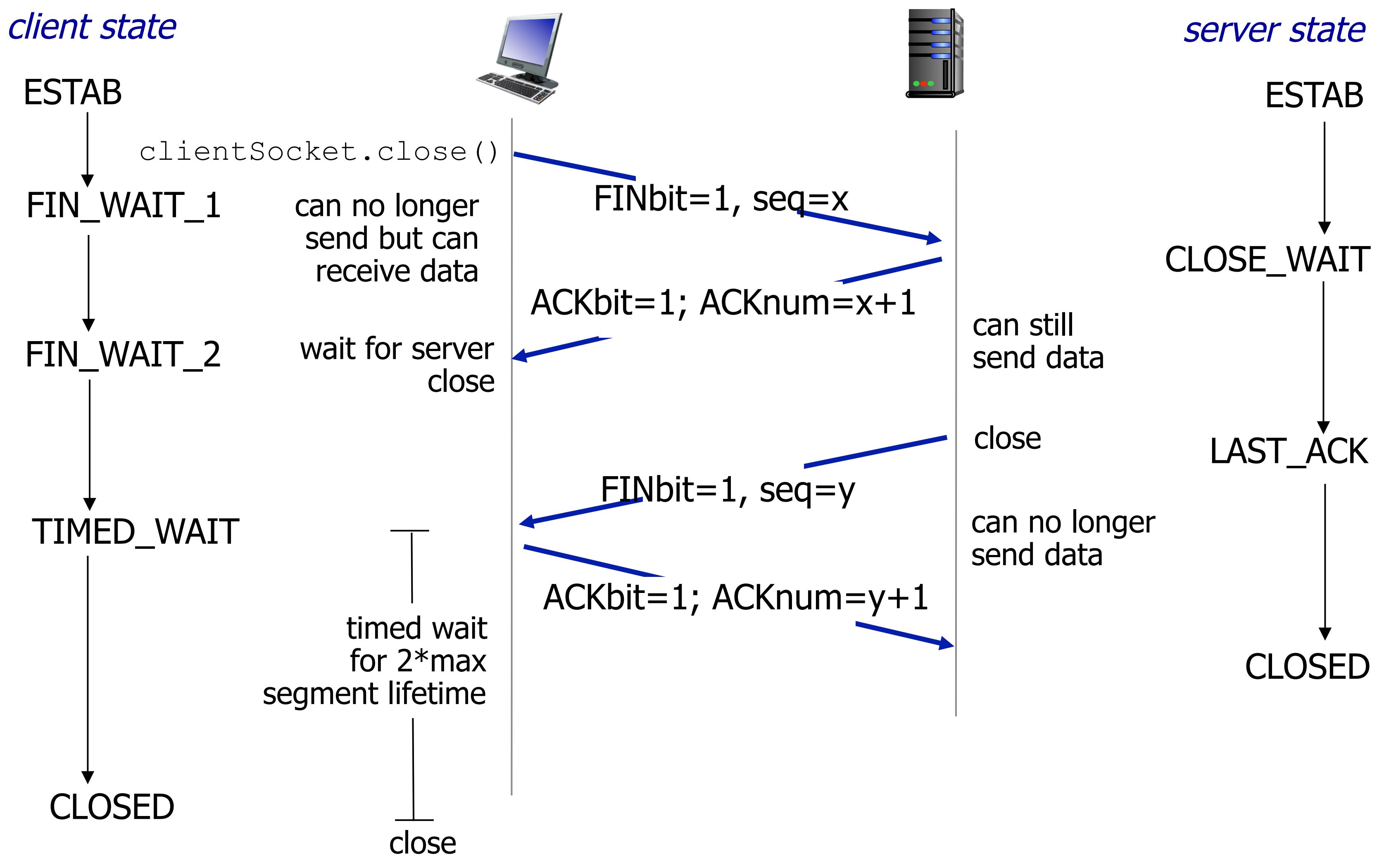


Attacker with IP B (pretending to be A)
Real IP A
SYNbit=1, Seq=x
"from IP A"
SYNbit=1, Seq=y
ACKbit=1; ACKnum=x+1
ACKbit=1, ACKnum=y+1?,
SYNbit=0
Seq=x+1
ACKnum(y+1) is really from IP A, if there is no man-in-the-middle attack.

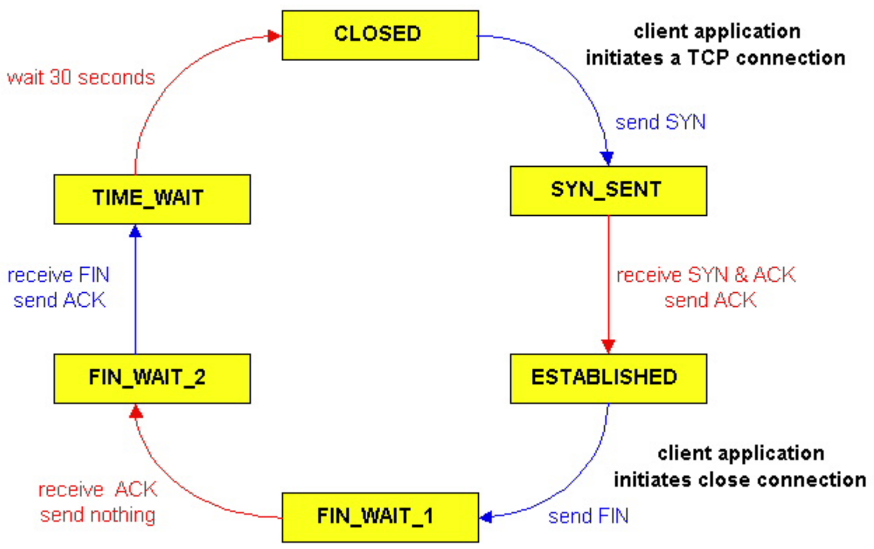
TCP: closing a connection

- remember: this is a duplex connection
- client, server each close their side of connection
  - send TCP segment with FIN bit = 1
  - either of the two can initiate the closing
- 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

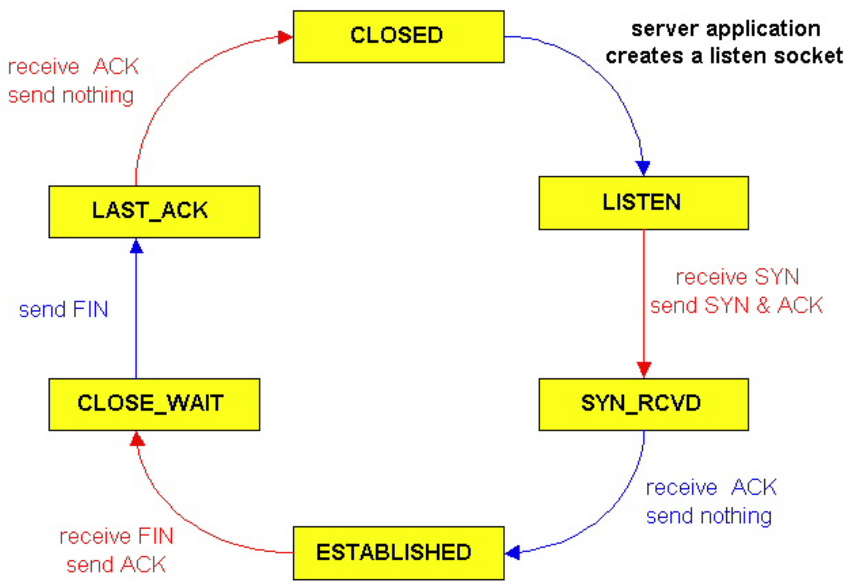


[TCP Connection Management (cont)]



TCP client lifecycle

TCP server lifecycle


CLOSED
server application creates a listen socket
LISTEN
receive SYN
send SYN & ACK
SYN_RCVD
receive ACK
send nothing
ESTABLISHED
receive FIN
send ACK
CLOSE_WAIT
send FIN
LAST_ACK
receive ACK
send nothing