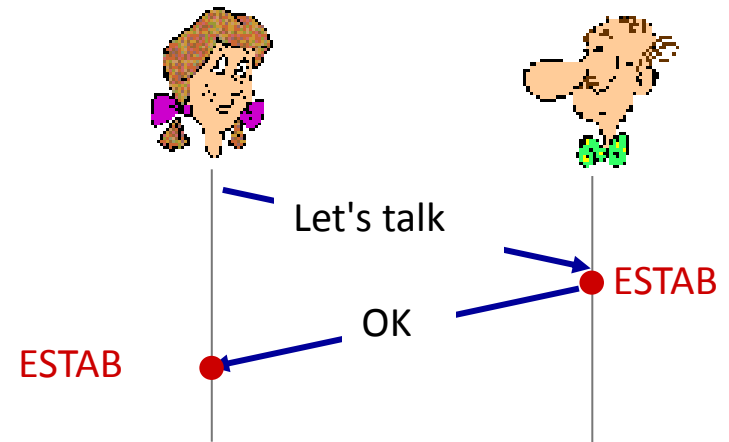
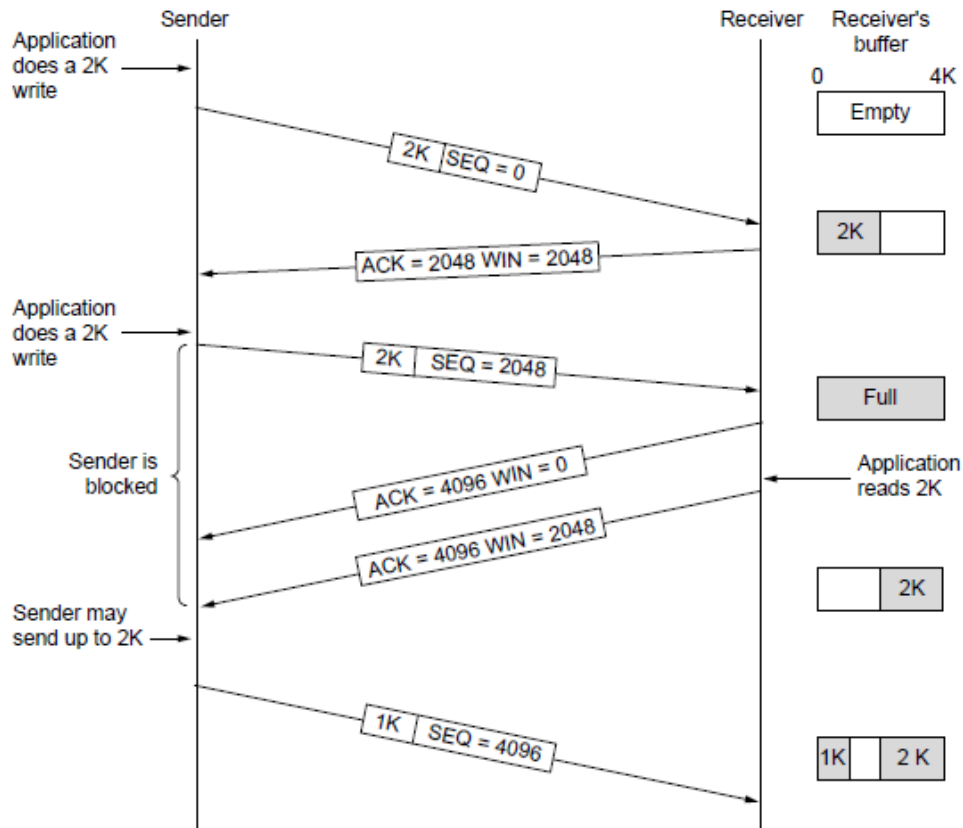


TCP flow control and connection setup



Computer Networking: A Top Down Approach

6th edition

Jim Kurose, Keith Ross

Addison-Wesley

Some materials copyright 1996-2012
J.F Kurose and K.W. Ross, All Rights Reserved



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

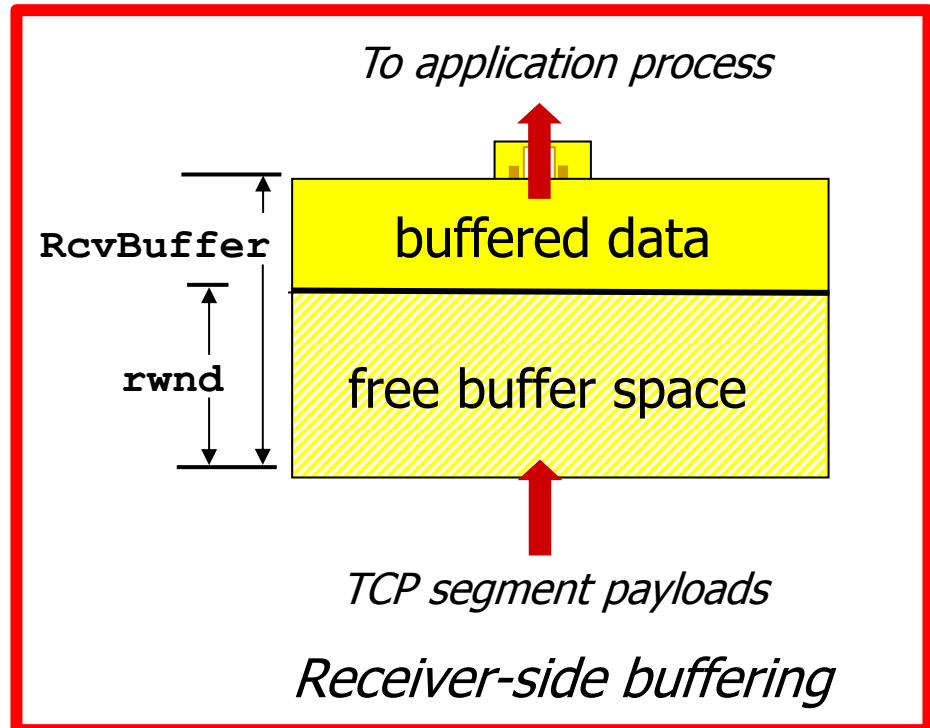
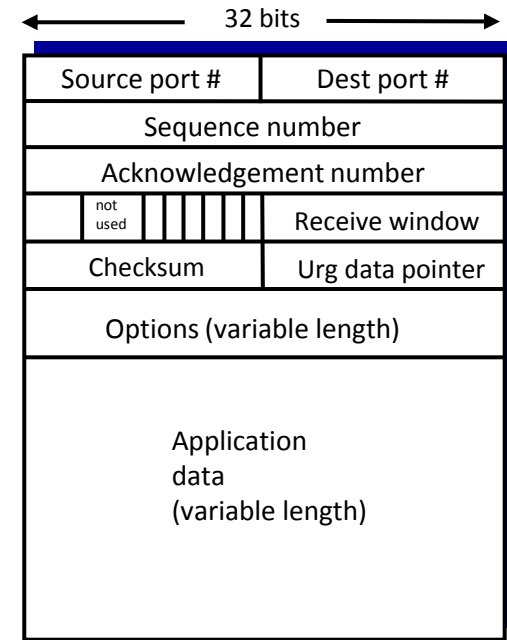
- Connection management

3.6 Principles of congestion control

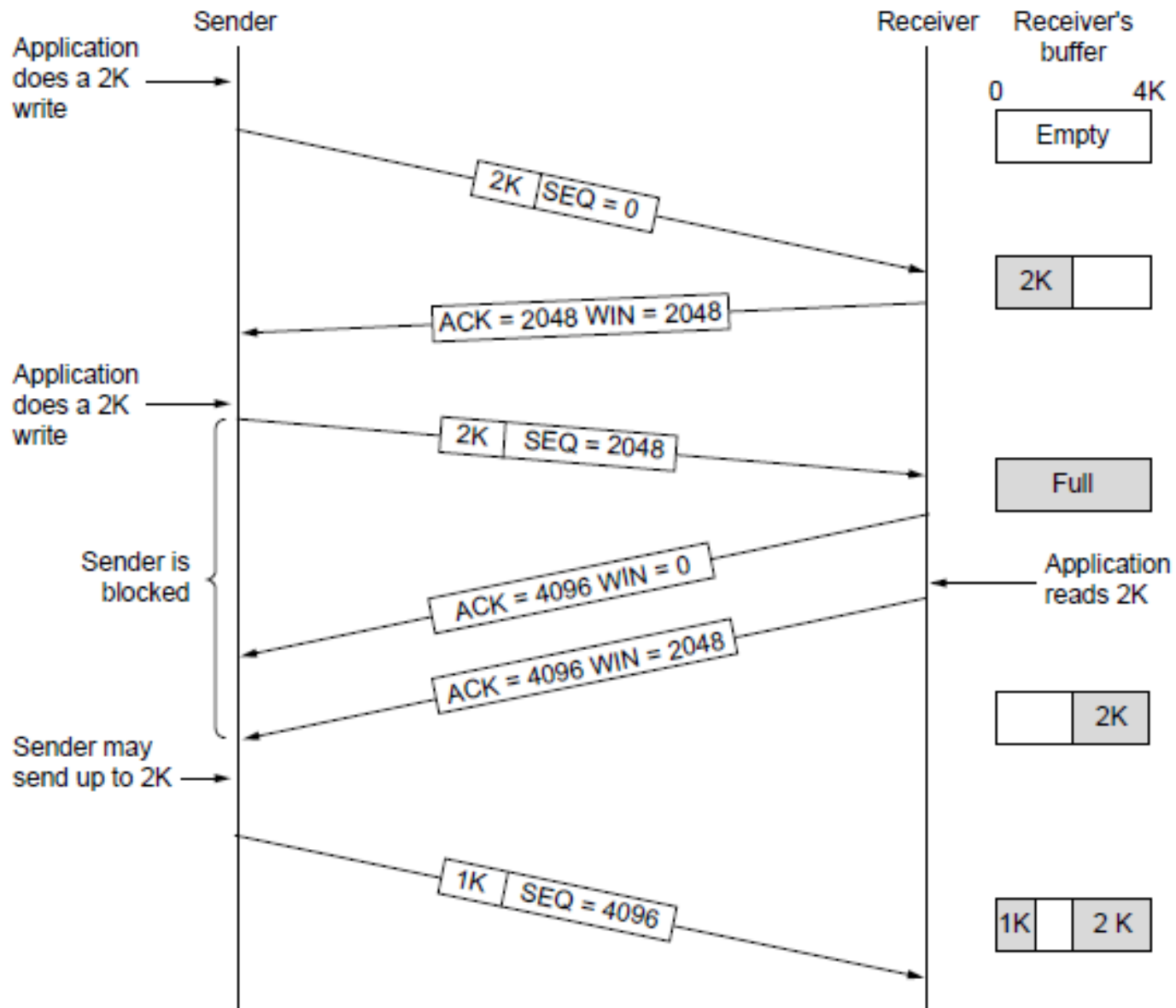
3.7 TCP congestion control

TCP flow control

- Receiver advertises free buffer space by including **rwnd** value in TCP header
 - **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



TCP sliding window



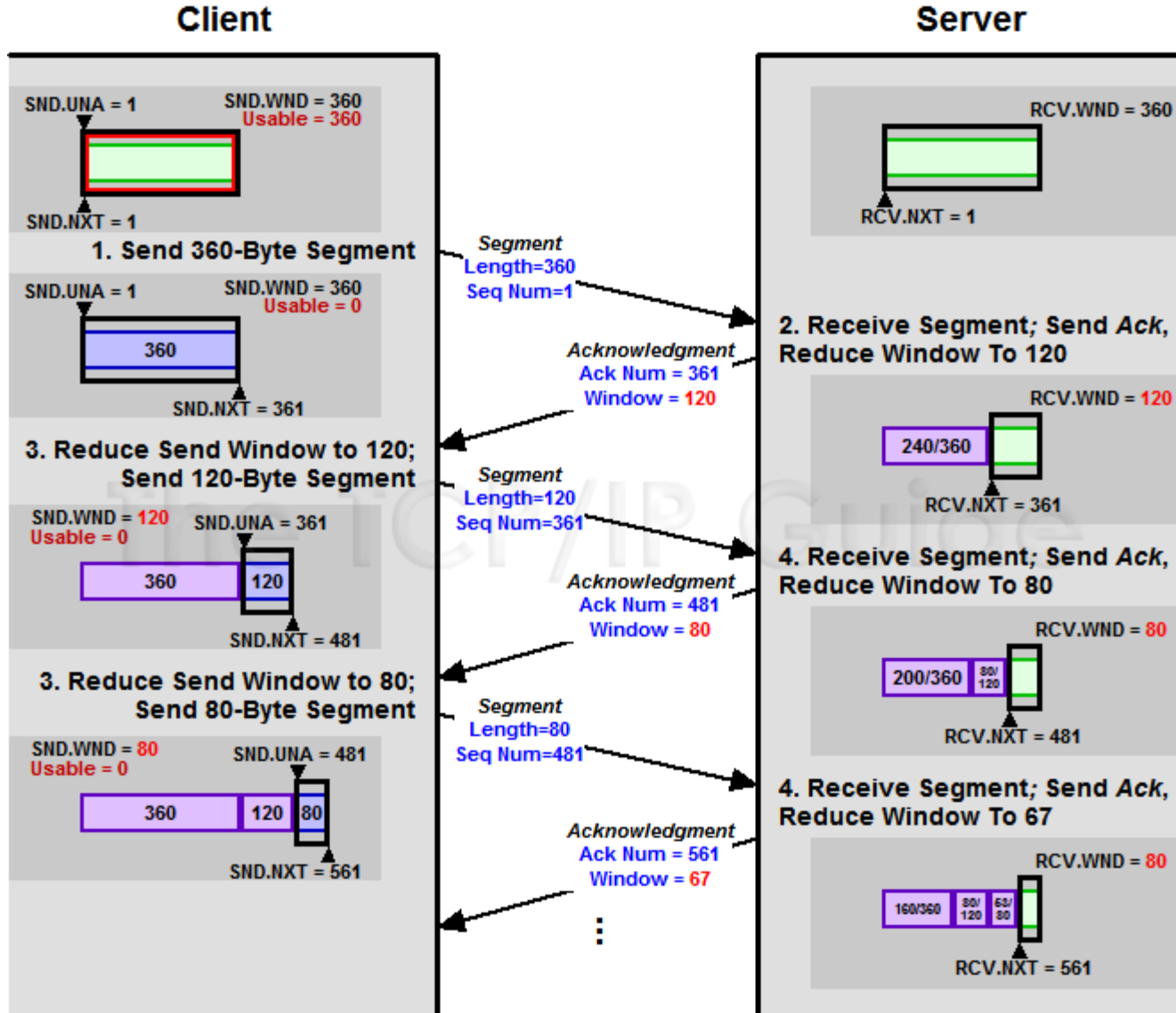
TCP sliding window

- **Windows size = 0**
 - Bytes up to and including ACK # - 1 have been received
 - Receiver has not consumed data so don't send more
 - When ready, receiver issues same ACK # and non-zero window size
 - Provides the flow-control in TCP
- **Sender can still send:**
 - Urgent requests (e.g. kill the process)
 - Periodic **window probe** frames, see if window has opened
 - Prevents deadlock should the receiver's windows update get lost
 - Persistence timer

http://media.pearsoncmg.com/aw/aw_kurose_network_4/applets/flow/FlowControl.htm

<http://www.ccs-labs.org/teaching/rn/animations/flow/index.htm>

Silly Window Syndrome



http://www.tcpiptide.com/free/t_TCPSillyWindowSyndromeandChangesTotheSlidingWindow.htm

Nagle's Algorithm

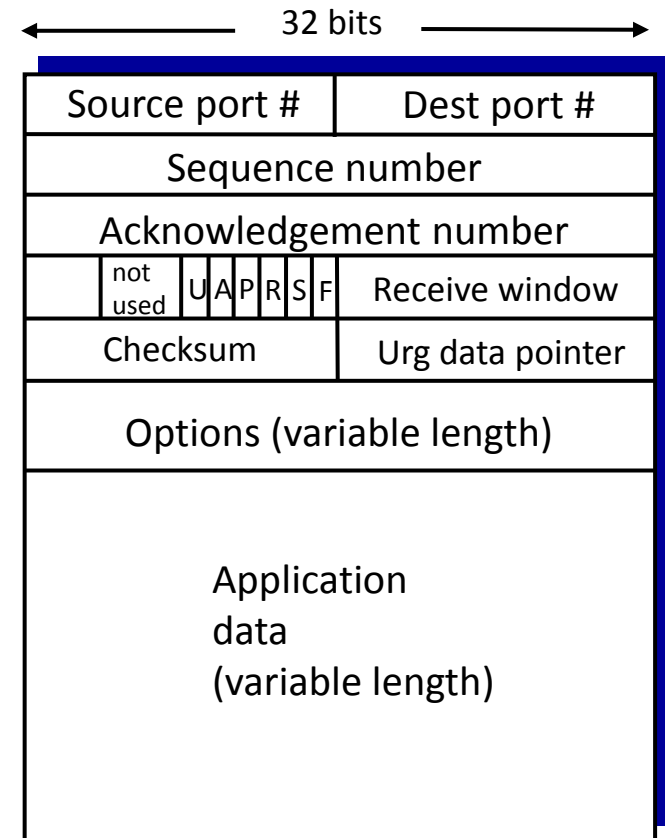
- Sender-side silly window avoidance
- Application produces data to send
 - If \geq MSS, send segment
 - If no segments in flight, send the segment
 - Otherwise queue the data
- Limits to one small segment in network
 - But bad for interactive apps like gaming
 - Especially bad if combined with delayed ACKs
 - write byte, write byte, read byte
 - Can be disabled, TCP_NODELAY option

Clark's solution

- Receiver-side silly window avoidance
- Do not send window size update unless:
 - It can handle full MSS size
 - Half of its buffer is empty

Limits in the TCP header

- Sequence number
 - 32 bits long
- Receive window
 - 16 bits long
 - TCP has satisfied the requirement of the sliding window algorithm that the sequence number space be twice as big as the window size
 - $2^{32} \gg 2 \times 2^{16}$



Protecting against wraparound

- Relevance of the 32-bit sequence number space
 - Sequence number may wraparound
 - A byte with sequence x could be sent, then later time a second byte with the same sequence x could be sent
 - Packets cannot survive in the Internet for longer than the maximum segment lifetime: **MSL = 120s**
 - Sequence number must not wrap around within MSL

| Bandwidth | Time until Wraparound |
|--------------------------|-----------------------|
| T1 (1.5 Mbps) | 6.4 hours |
| Ethernet (10 Mbps) | 57 minutes |
| T3 (45 Mbps) | 13 minutes |
| Fast Ethernet (100 Mbps) | 6 minutes |
| OC-3 (155 Mbps) | 4 minutes |
| OC-12 (622 Mbps) | 55 seconds |
| OC-48 (2.5 Gbps) | 14 seconds |

Keeping the pipe full

- 16-bit receive window must allow sender to keep the pipe full
- If the receiver has enough buffer space
 - Window can be opened to allow a full delay \times bandwidth product's worth of data

| Bandwidth | Delay \times Bandwidth Product |
|--------------------------|----------------------------------|
| T1 (1.5 Mbps) | 18 KB |
| Ethernet (10 Mbps) | 122 KB |
| T3 (45 Mbps) | 549 KB |
| Fast Ethernet (100 Mbps) | 1.2 MB |
| OC-3 (155 Mbps) | 1.8 MB |
| OC-12 (622 Mbps) | 7.4 MB |
| OC-48 (2.5 Gbps) | 29.6 MB |

Required window size for 100 ms RTT

TCP extensions

- **Timestamp option**
 - Timestamp added to segment by the sender
 - Echoed by the receiver
 - Sender can then compute RTT
 - Also can be combined with sequence number
 - Protects against wraparound
- **Large window option**
 - Use a scale factor
 - Left shift window size field by up to 14 bits
 - Windows of up to 2^{30} bytes

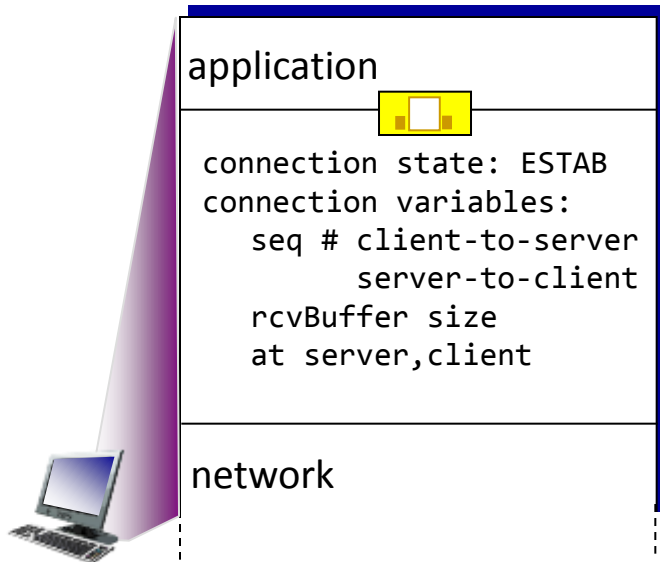
TCP extensions

- **Selective acknowledgements (SACK)**
 - Optional header fields used to acknowledge additional blocks
 - Sender can then resubmit only missing blocks
- **Maximum Segment Size (MSS)**
 - Only valid extension during connection setup
 - Set a non-default value for maximum segment size

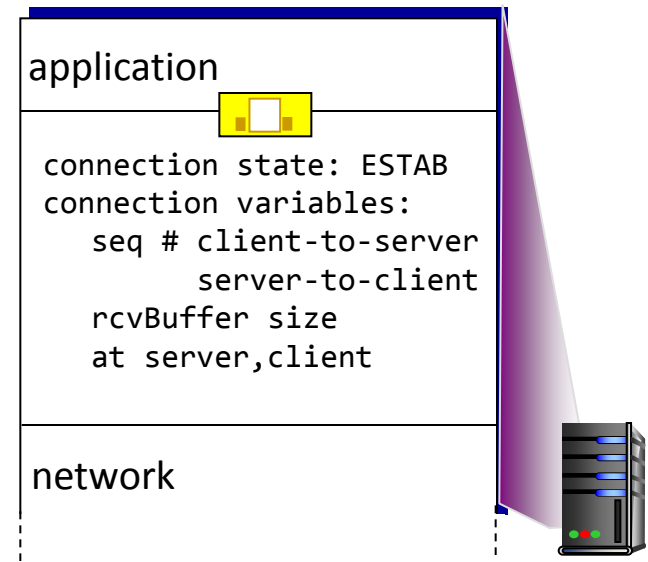
Connection management

Before exchanging data, sender/receiver handshake:

- Agree to establish connection
- Agree on connection parameters



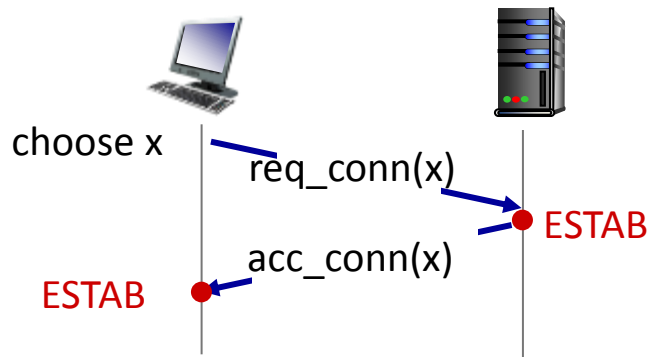
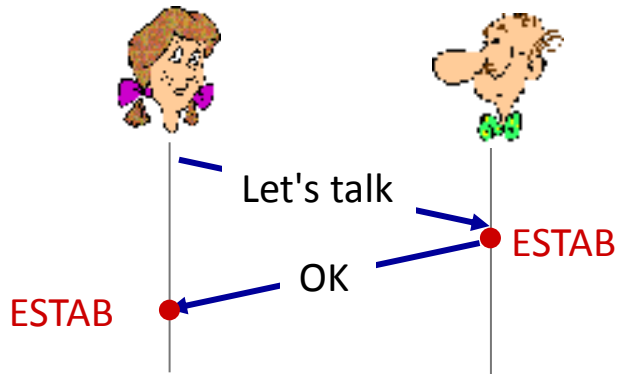
```
Socket clientSocket =  
newSocket("hostname", port);
```



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

Agreeing to establish a connection

2-way handshake:

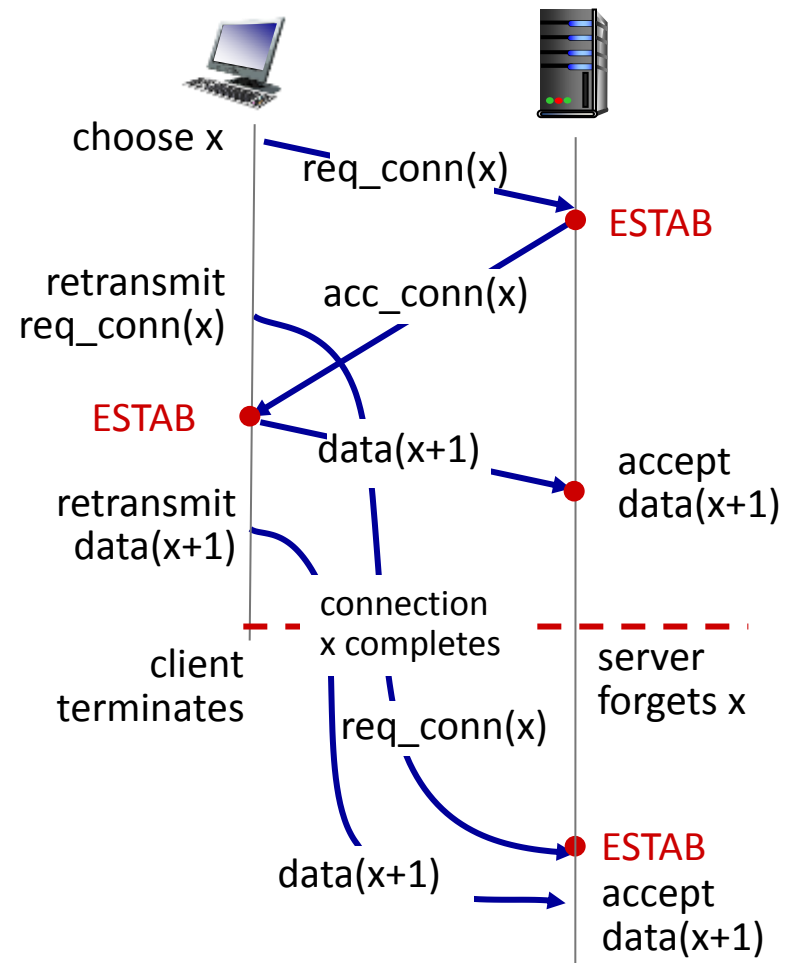
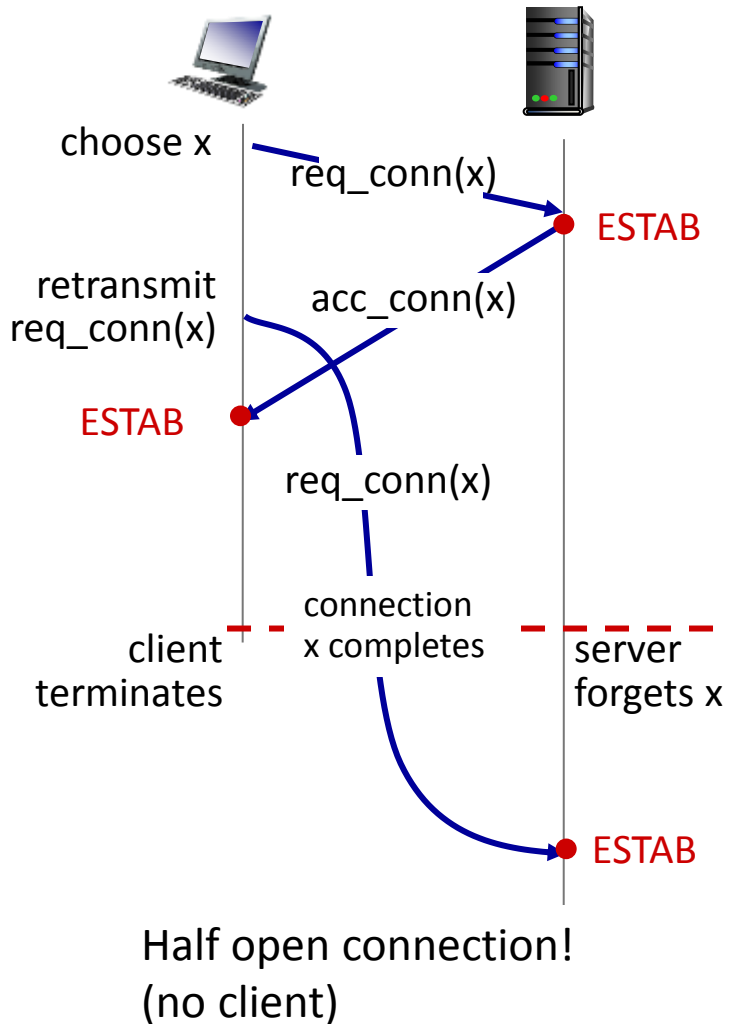


Q: Will 2-way handshake always work in a network?

- Variable delays
- Retransmitted messages (e.g. req_conn(x)) due to message loss
- Message reordering
- One side can't see other side

Agreeing to establish a connection

2-way handshake failure scenarios:



TCP 3-way handshake

Client state

LISTEN
↓
SYNSENT
↓
ESTAB

Choose init seq num, x
Send TCP SYN msg



SYNbit=1, Seq= x



Server state

LISTEN
↓
SYN RCVD
↓
ESTAB

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;

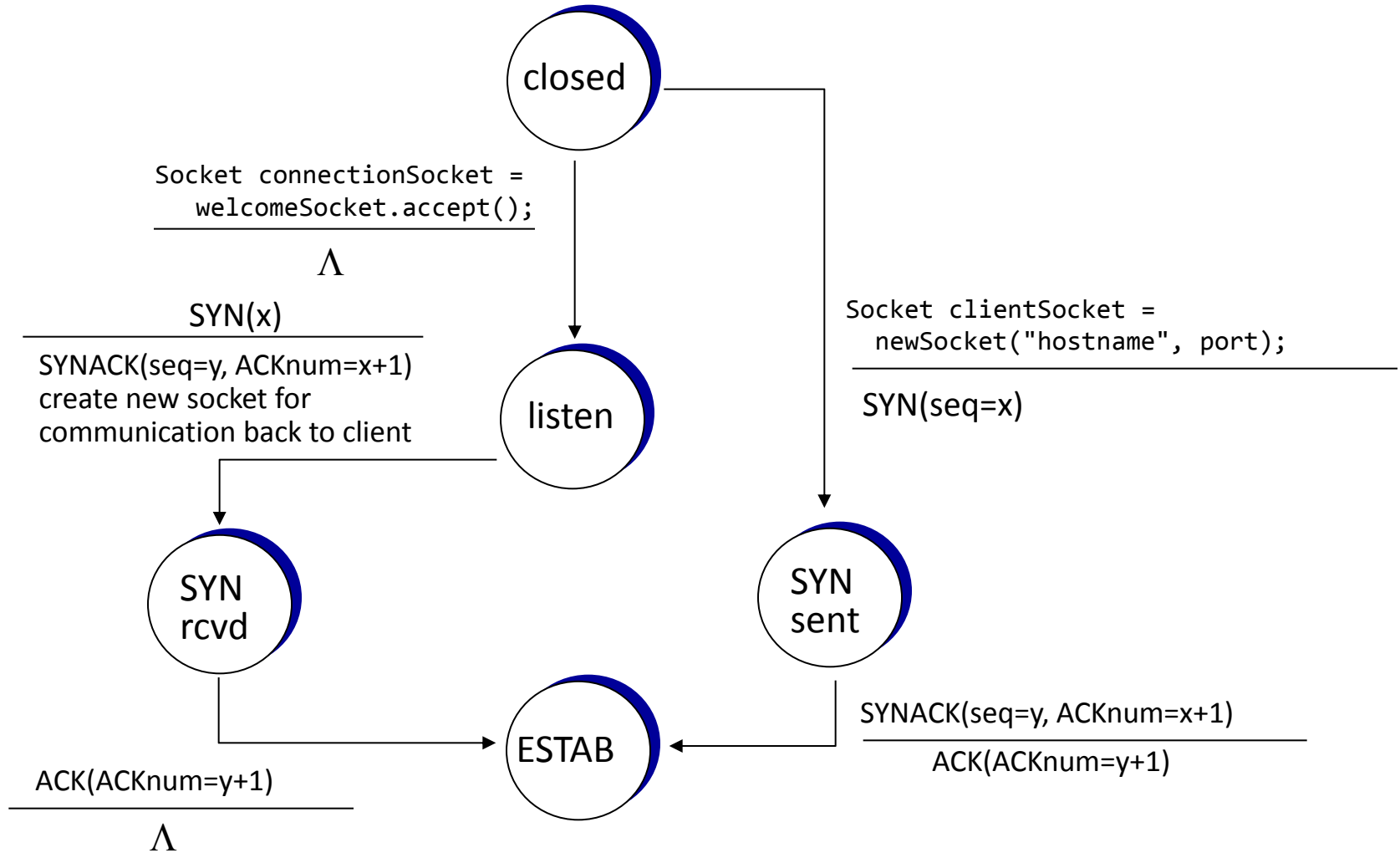
Send ACK for SYNACK;

This segment may contain
client-to-server data

ACKbit=1, ACKnum= $y+1$

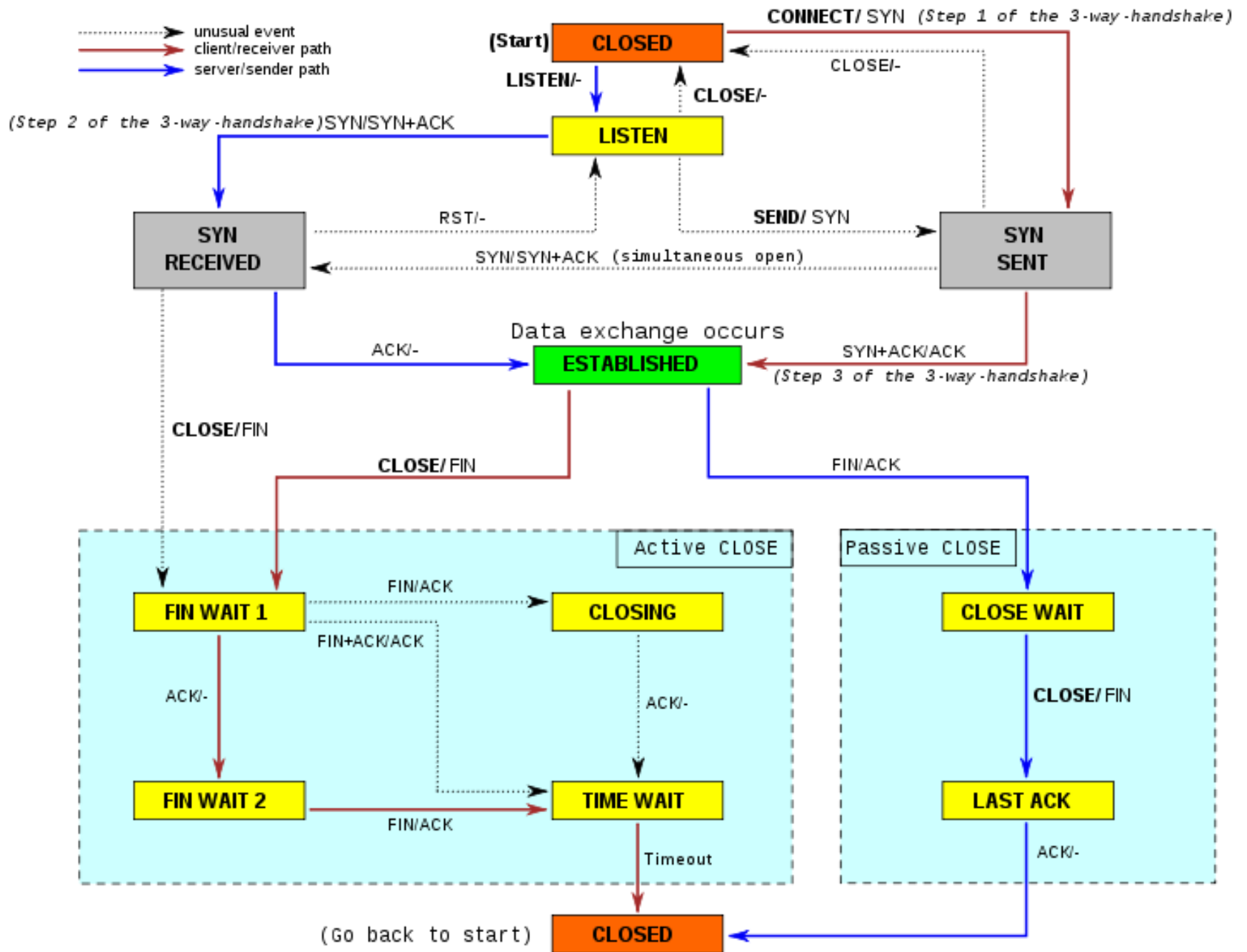
Received ACK(y)
indicates client is live

TCP 3-way handshake: FSM



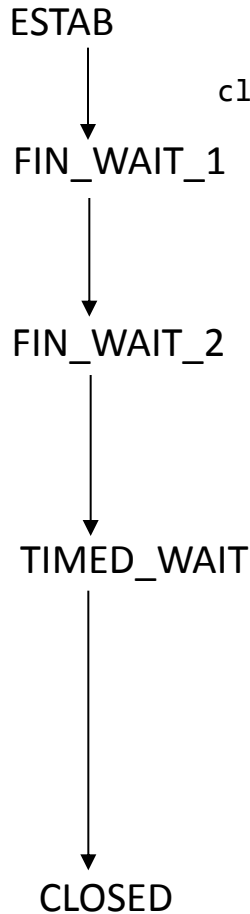
TCP: closing a connection

- ❖ 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

Client state



`clientSocket.close()`

Can no longer send but can receive data

Wait for server close

Timed wait for $2 * \text{max segment lifetime}$



FINbit=1, seq=x

ACKbit=1; ACKnum=x+1

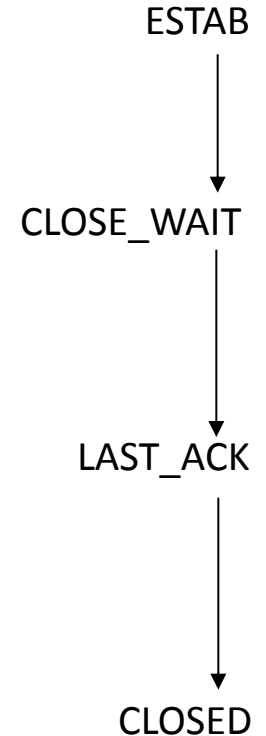
FINbit=1, seq=y

ACKbit=1; ACKnum=y+1

Can still send data

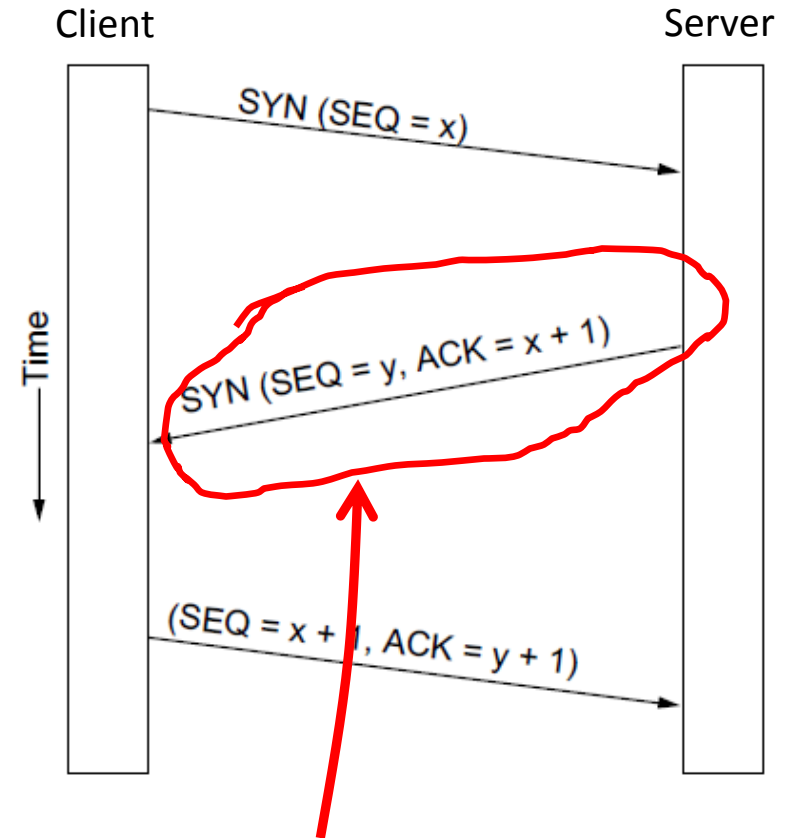
Can no longer send data

Server state



Connection: three-way handshake

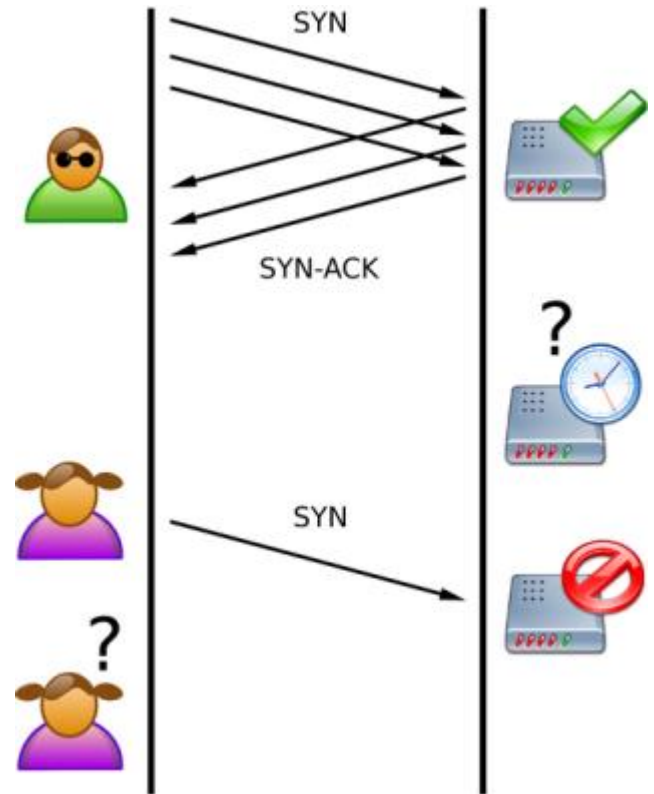
| Client | Server |
|--|--|
| | LISTEN, ACCEPT Passively waits for incoming connection |
| CONNECT Sends TCP segment to (IP, port) with SYN bit on, ACK bit off | |
| | Receives segment. OS hands off to process that has done LISTEN on port. If process accepts, send TCP with SYN and ACK bit set. |



Server has to remember its sequence number in step 2

SYN flooding

- SYN flooding
 - Denial-of-service attack
 - Attacker sends large number of SYN requests
 - Never responds or spoofs source IP address
 - Server runs out of resources
 - Server has to track assigned sequence number
 - Fills with half-open connections

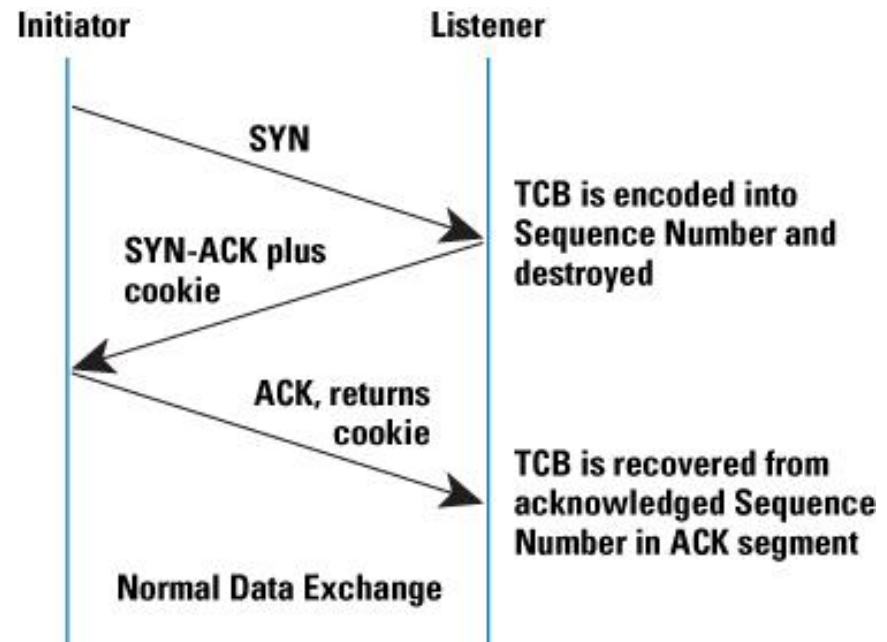


SYN cookies

- **Server generates sequence number**
 - Uses cryptographic hash function
 - Combine counter, MSS requested, and secret generated from client/server IP and ports
 - Fires off response, forgetting number
 - Recover original sequence number if client responds

<http://cr.yip.to/syncookies.html>

http://nmap.org/nmap_doc.html



nmap versus Tokyo

```
$ nmap -v -A 106.187.54.31 -p 1-65535
...
Scanned at 2012-10-16 16:31:38 MDT for 1391s
Not shown: 65526 closed ports
PORT      STATE      SERVICE      VERSION
22/tcp    open      ssh          OpenSSH 5.9p1 Debian 5ubuntu1 (protocol 2.0)
| ssh-hostkey: 1024 c5:ea:eb:88:a3:f1:d1:2d:5f:ed:63:c2:a8:54:bf:33 (DSA)
|_2048 d1:b3:75:95:ed:2a:13:90:27:89:1b:f4:f5:2b:b8:7c (RSA)
25/tcp    filtered  smtp
53/tcp    filtered  domain
67/tcp    filtered  dhcp
68/tcp    filtered  dhcp
80/tcp    open      http         Apache httpd 2.2.22 ((Ubuntu))
|_http-methods: No Allow or Public header in OPTIONS response (status code 200)
|_http-title: Site doesn't have a title (text/html).
1433/tcp  filtered  ms-sql-s
1434/tcp  filtered  ms-sql-m
8080/tcp  open      http-proxy  Squid http proxy 3.1.19
|_http-methods: No Allow or Public header in OPTIONS response (status code 400)
Service Info: OS: Linux; CPE: cpe:/o:linux:kernel
Final times for host: srtt: 188947 rttvar: 3251  to: 201951
...
```

sudo lsof -i:22 -n

Summary

- TCP flow control
 - Each side informs other of available buffer space
 - Other side never places more unACKed in-flight
- TCP connection setup
 - Three-way handshake
 - Each side chooses random sequence number
 - Can be exploited:
 - SYN flood attack
 - Port scanning (NMAP)