Transport Layer (TCP/UDP)

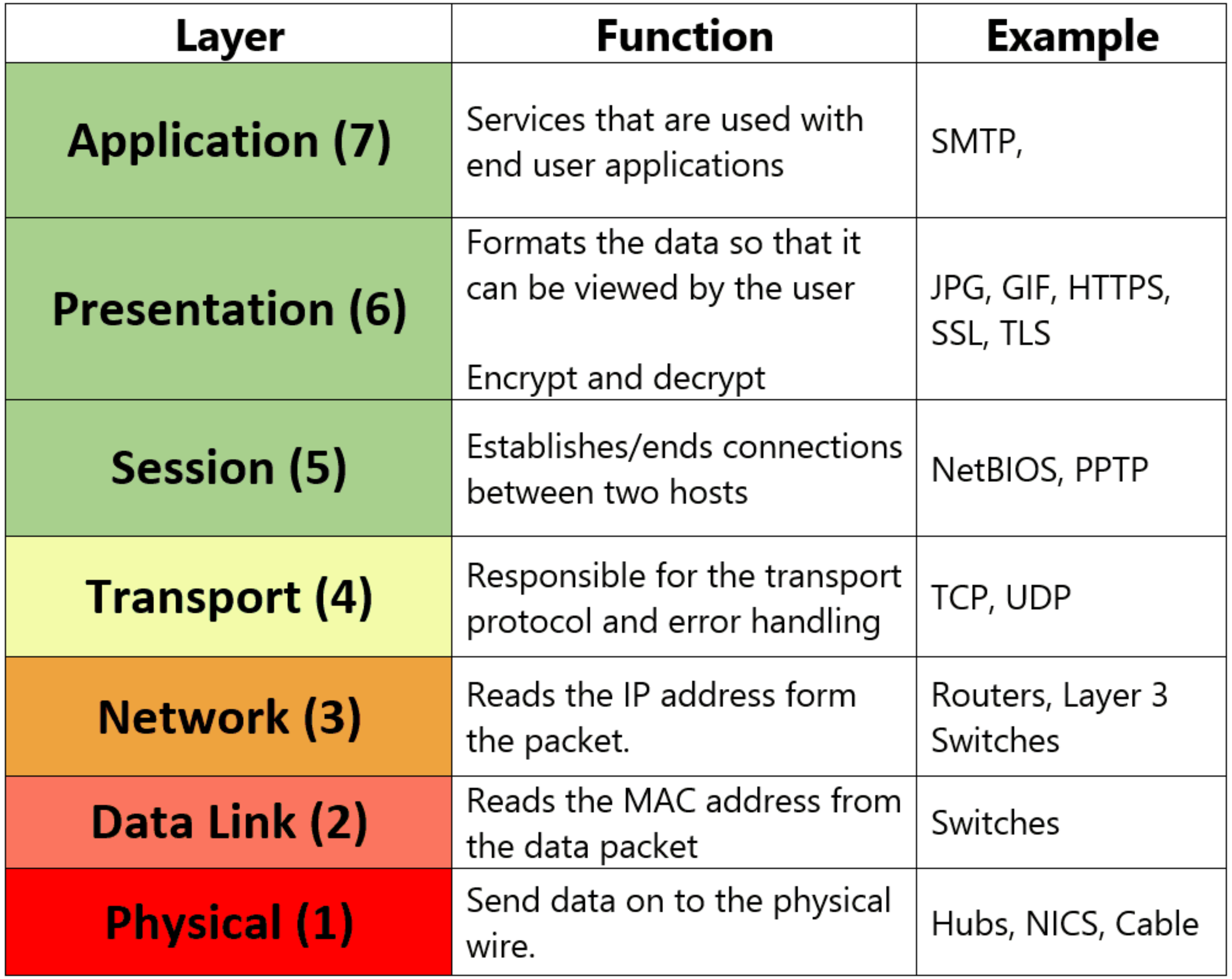
Recall the protocol stack

Organize network functionality into protocols and layers

Higher layer protocols use the services provided by the lower layer

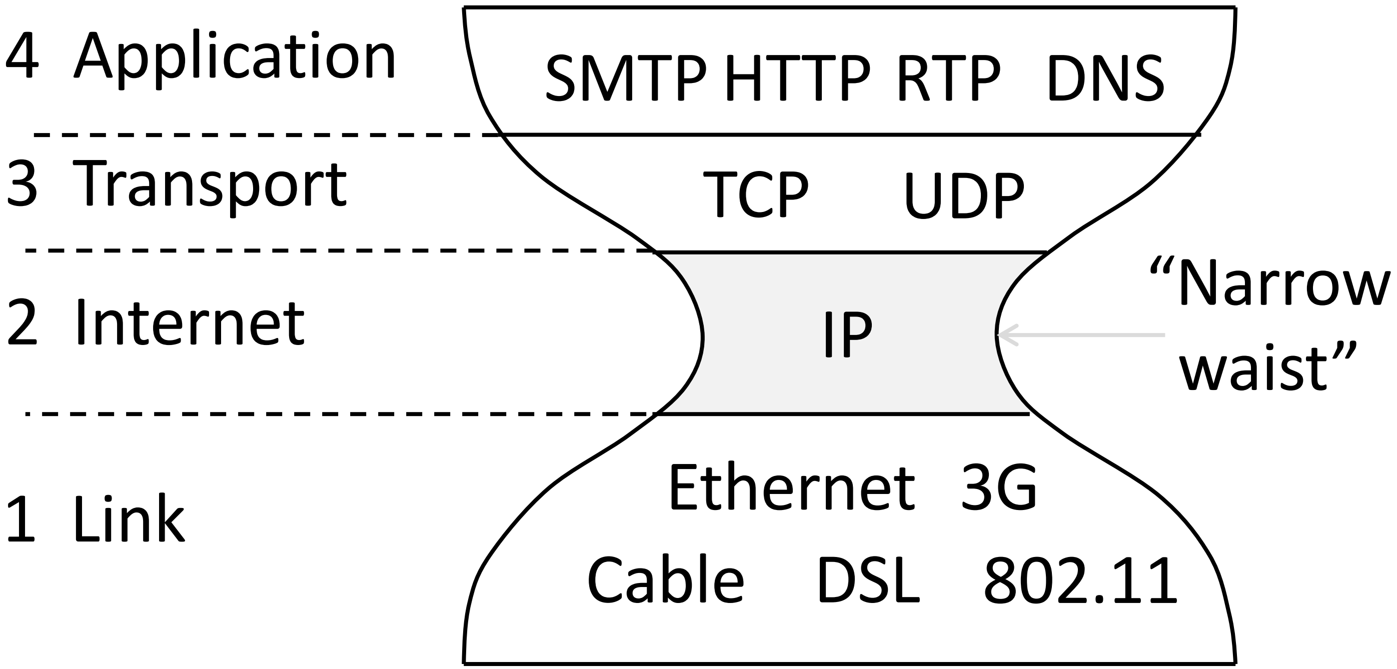
Protocol instances of the same type communicate with each other virtually

OSI Layers



Layer	Function	Example
Application (7)	Services that are used with end user applications	SMTP,
Presentation (6)	Formats the data so that it can be viewed by the user Encrypt and decrypt	JPG, GIF, HTTPS, SSL, TLS
Session (5)	Establishes/ends connections between two hosts	NetBIOS, PPTP
Transport (4)	Responsible for the transport protocol and error handling	TCP, UDP
Network (3)	Reads the IP address form the packet.	Routers, Layer 3 Switches
Data Link (2)	Reads the MAC address from the data packet	Switches
Physical (1)	Send data on to the physical wire.	Hubs, NICS, Cable

Internet layers

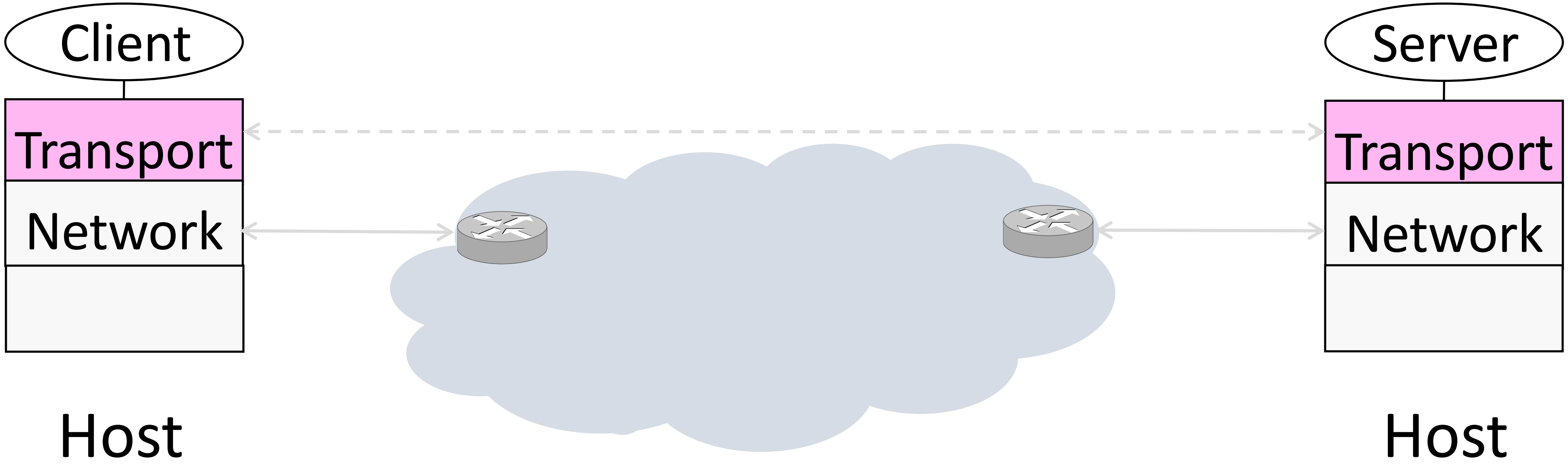


Internet layers

Application	– Programs that use network service
Transport	– Provides end-to-end data delivery
Network	– Send packets over multiple networks
Link	– Send frames over one or more links
Physical	– Send bits using signals

Transport layer

Provides end-to-end connectivity to applications



Transport layer protocols

- Provide different kinds of data delivery across the network to applications

	Unreliable	**Reliable**
Messages	Datagrams (UDP)	
Bytestream		Streams (TCP)

Comparison of Internet transports

- TCP is full-featured, UDP is a glorified packet

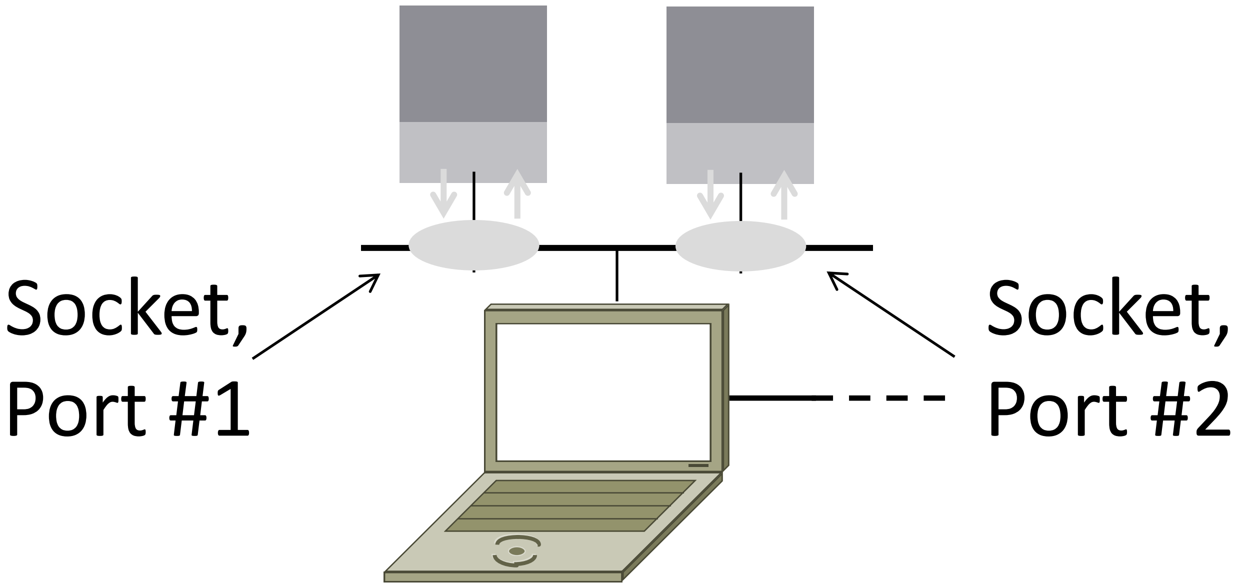
TCP (Streams)	**UDP (Datagrams)**
Connections	Datagrams
Bytes are delivered once, reliably, and in order	Messages may be lost, reordered, duplicated
Arbitrary length content	Limited message size
Flow control matches sender to receiver	Can send regardless of receiver state
Congestion control matches sender to network	Can send regardless of network state

Socket API

- Simple abstraction to use the network
  - The “network” API (really Transport service) used to write all Internet apps
  - Part of all major OSes and languages; originally Berkeley (Unix) ~1983
- Supports both Internet transport services (Streams and Datagrams)

Socket API (2)

- Sockets let apps attach to the local network at different ports



Socket API (3)

- Same API used for Streams and Datagrams

Primitive	Meaning
SOCKET	Create a new communication endpoint
BIND	Associate a local address (port) with a socket
LISTEN	Announce willingness to accept connections
ACCEPT	Passively establish an incoming connection
CONNECT	Actively attempt to establish a connection
SEND(TO)	Send some data over the socket
RECEIVE(FROM)	Receive some data over the socket
CLOSE	Release the socket

Only needed for Streams

To/From for Datagrams

Ports

- Application process is identified by the tuple IP address, transport protocol, and port
  - Ports are 16-bit integers representing local “mailboxes” that a process leases
- Servers often bind to “well-known ports”
  - <1024, require administrative privileges
- Clients often assigned “ephemeral” ports
  - Chosen by OS, used temporarily

Some Well-Known Ports

Port	Protocol	Use
TCP/20, 21	FTP	File transfer
TCP/22	SSH	Remote login, replacement for Telnet
TCP/25	SMTP	Email
TCP/80	HTTP	World Wide Web
TCP/443	HTTPS	Secure Web (HTTP over SSL/TLS)
TCP/3306	MYSQL	MYSQL database access
UDP/53	DNS	Domain name service

Full list: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.txt

Topics

- Service models
  - Socket API and ports
  - Datagrams, Streams
- User Datagram Protocol (UDP)
- Connections (TCP)
- Sliding Window (TCP)
- Flow control (TCP)
- Retransmission timers (TCP)
- Congestion control (TCP)

UDP

User Datagram Protocol (UDP)

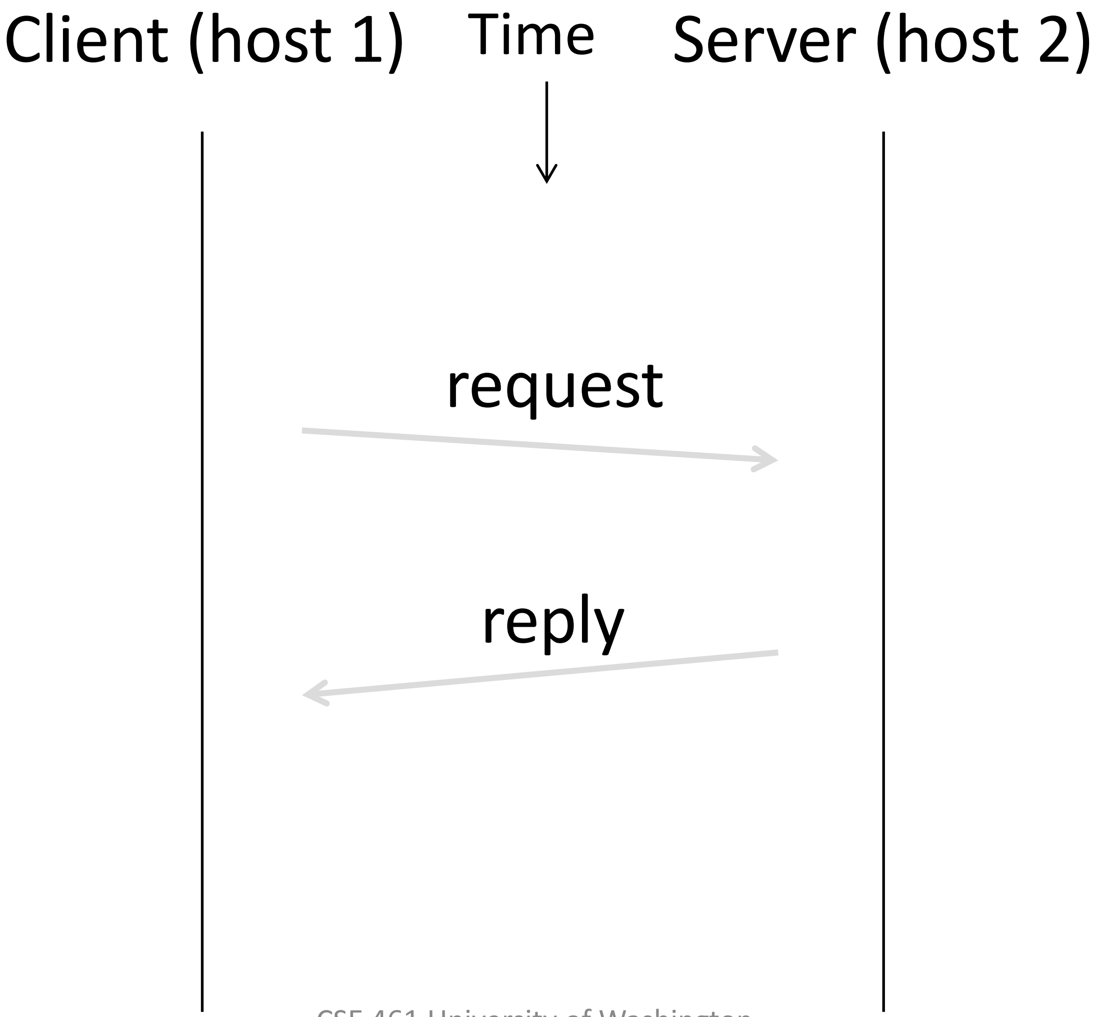
- Used by apps that don’t want reliability or bytestreams
    - Like what?

User Datagram Protocol (UDP)

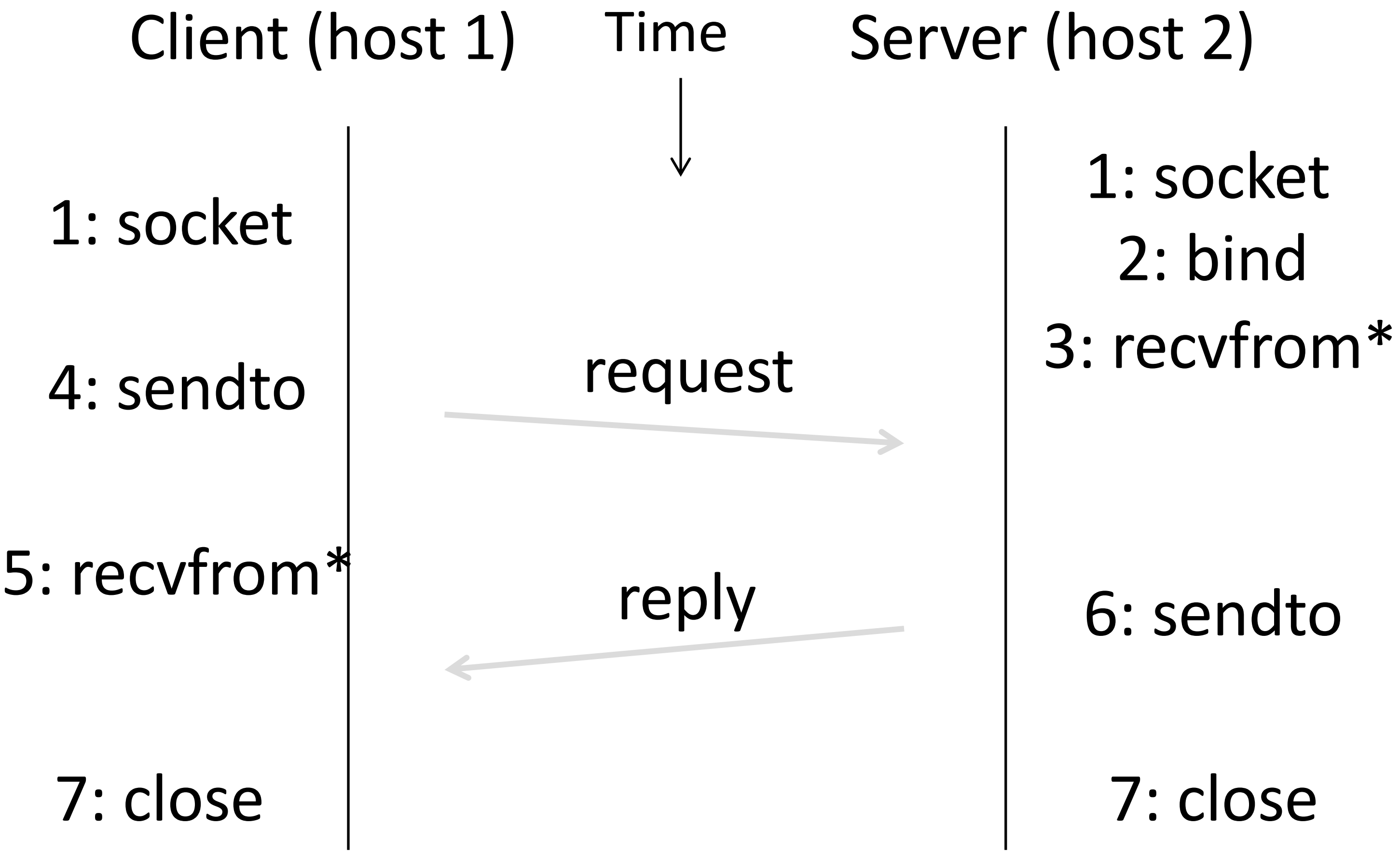
- Used by apps that don’t want reliability or bytestreams
  - Voice-over-IP
  - DNS
  - DHCP
  - Games

(If application wants reliability and messages then it has work to do!)

Datagram Sockets



Datagram Sockets (2)



Client (host 1)

Time

Server (host 2)

1: socket

4: sendto

5: recvfrom*

7: close

request

reply

1: socket

2: bind

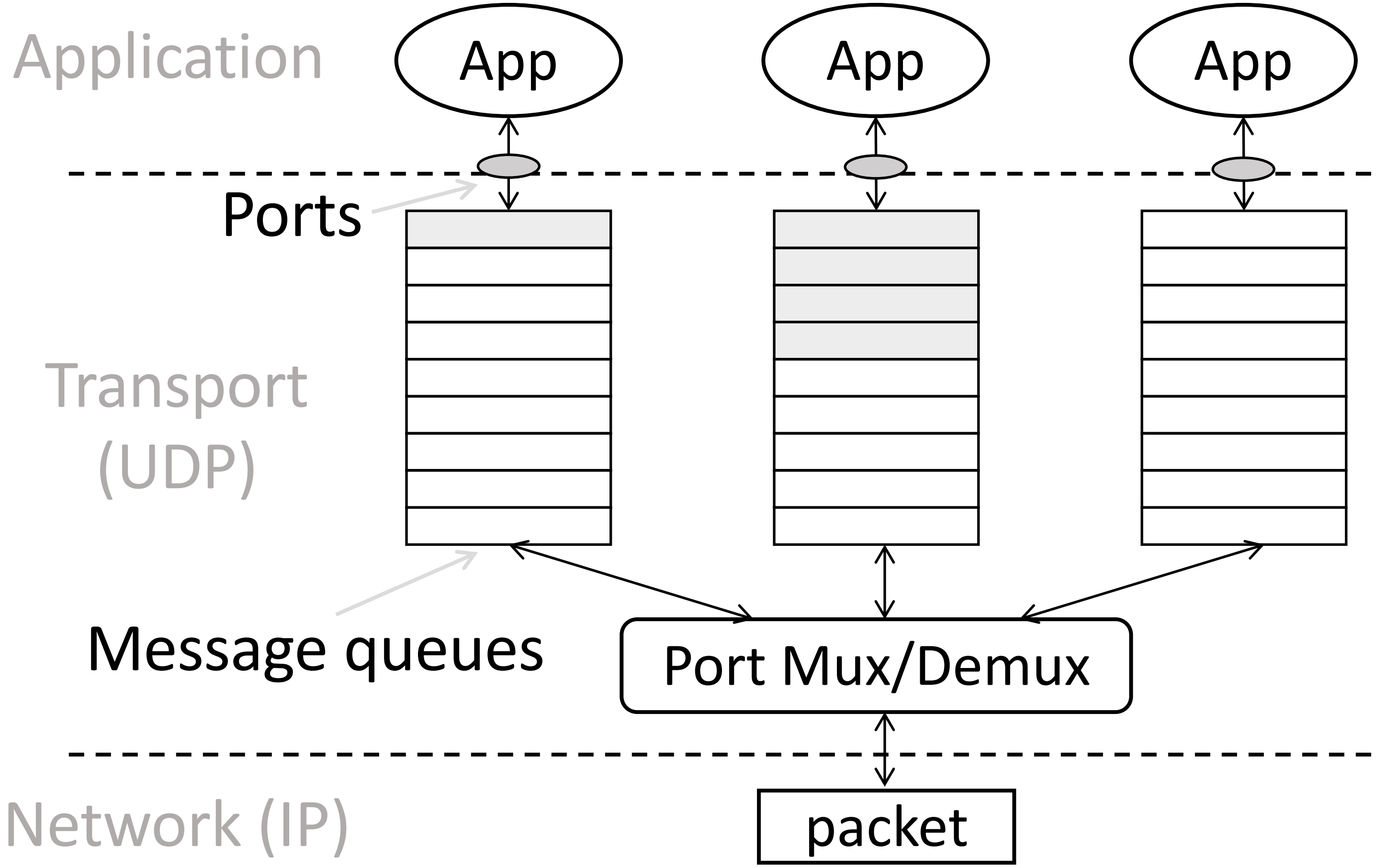
3: recvfrom*

6: sendto

7: close

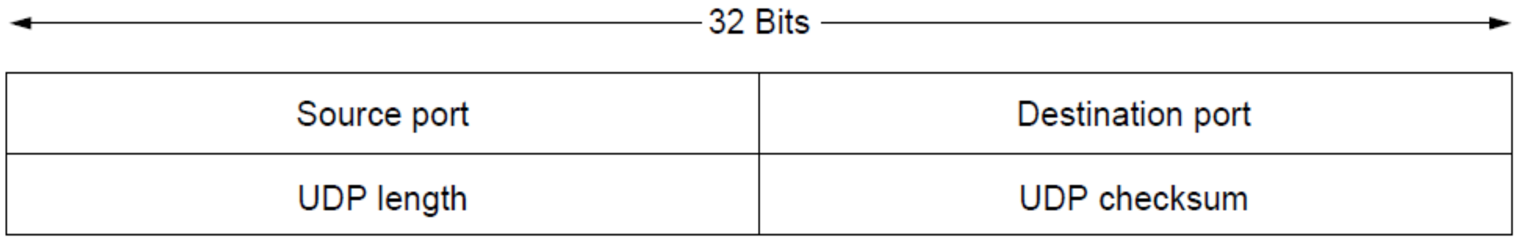
*= call blocks

UDP Buffering



UDP Header

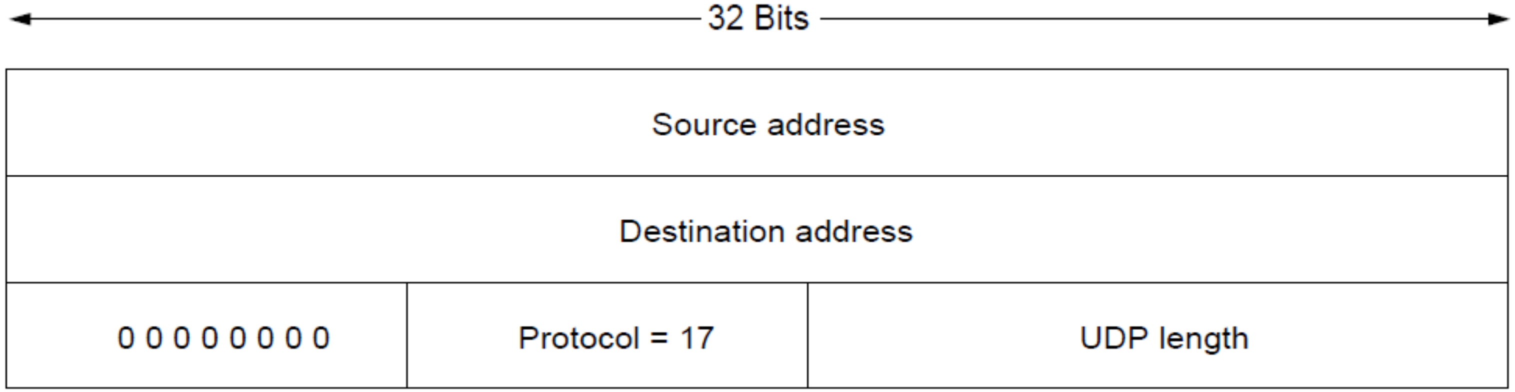
- Uses ports to identify sending and receiving application processes
- Datagram length up to 64K
- Checksum (16 bits) for reliability



Source port	Destination port
UDP length	UDP checksum

UDP Header (2)

- Optional checksum covers UDP segment and IP pseudoheader
  - Checks key IP fields (addresses)
  - Value of zero means “no checksum”



32 Bits

Source address		
Destination address		
0 0 0 0 0 0 0 0	Protocol = 17	UDP length

TCP

TCP

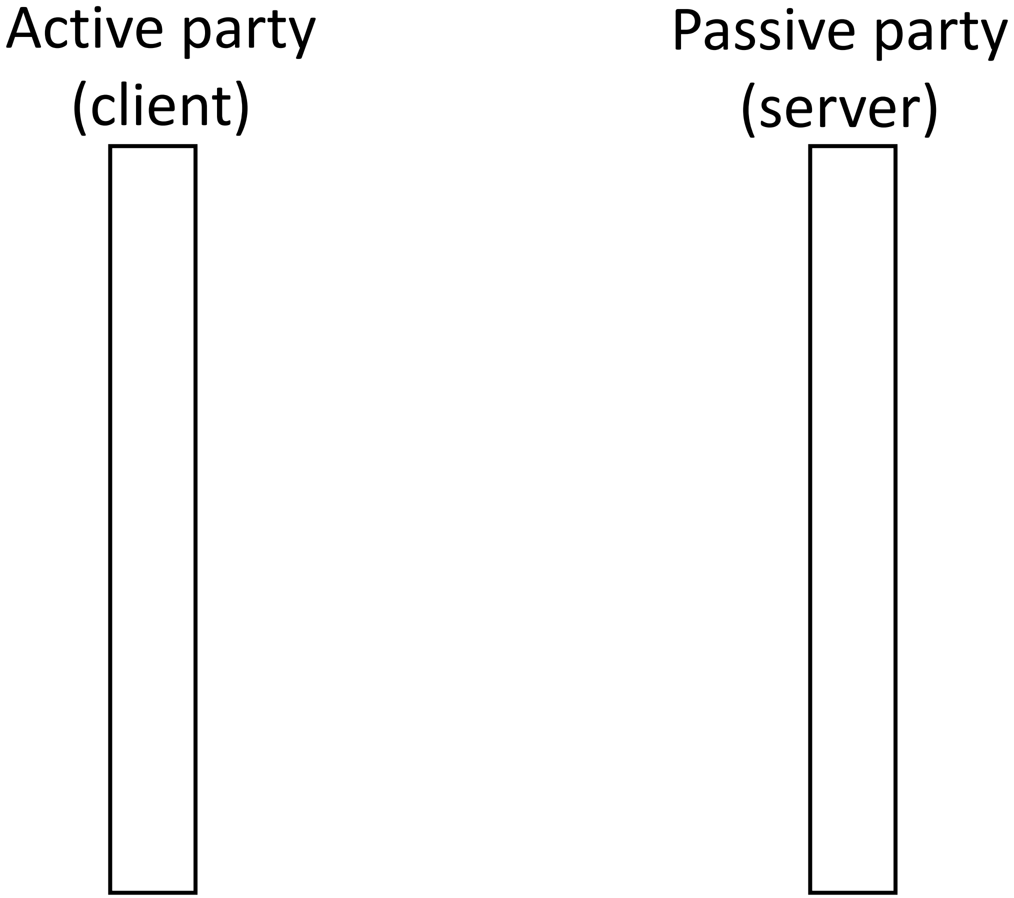
- TCP Consists of 3 primary phases:
  - Connection Establishment (Setup)
  - Sliding Windows/Flow Control
  - Connection Release (Teardown)

Connection Establishment

- Both sender and receiver must be ready before we start the transfer of data
  - Need to agree on a set of parameters
  - e.g., the Maximum Segment Size (MSS)
- This is signaling
  - It sets up state at the endpoints
  - Like “dialing” for a telephone call

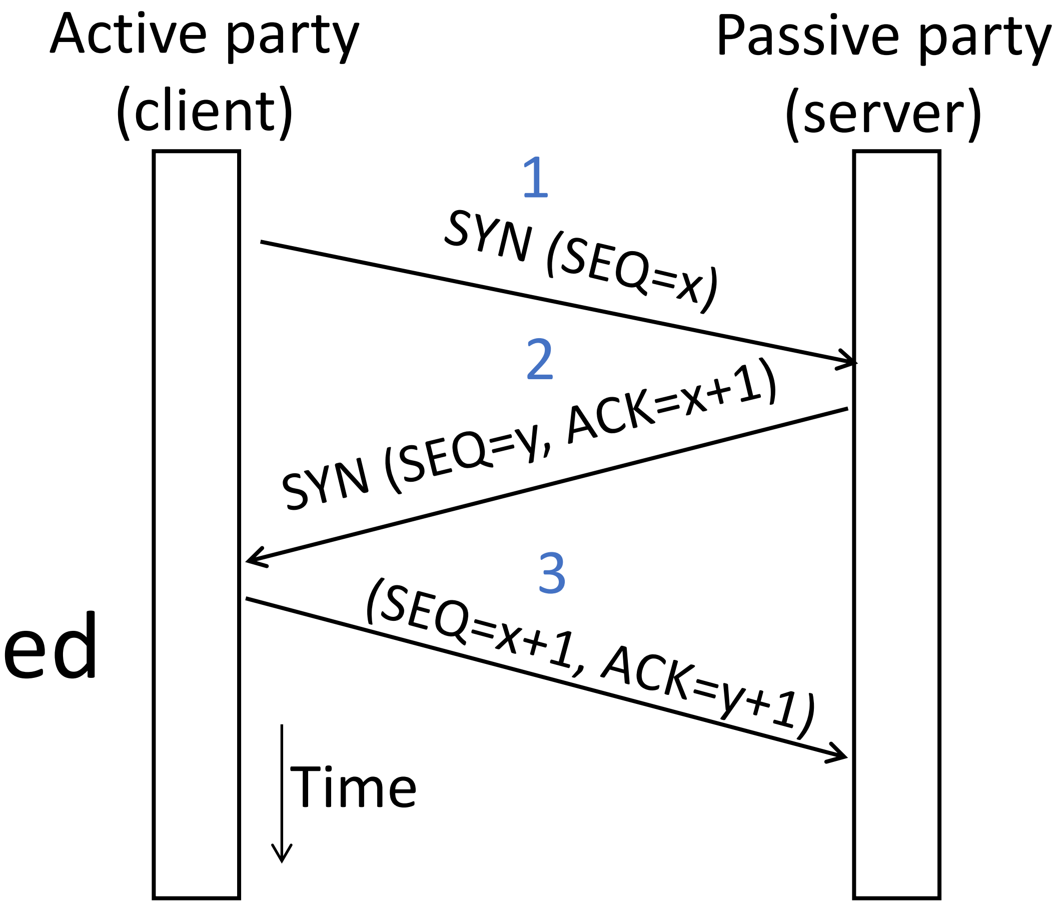
Three-Way Handshake

- Used in TCP; opens connection for data in both directions
- Each side probes the other with a fresh Initial Sequence Number (ISN)
  - Sends on a SYNchronize segment
  - Echo on an ACKnowledge segment
- Chosen to be robust even against delayed duplicates



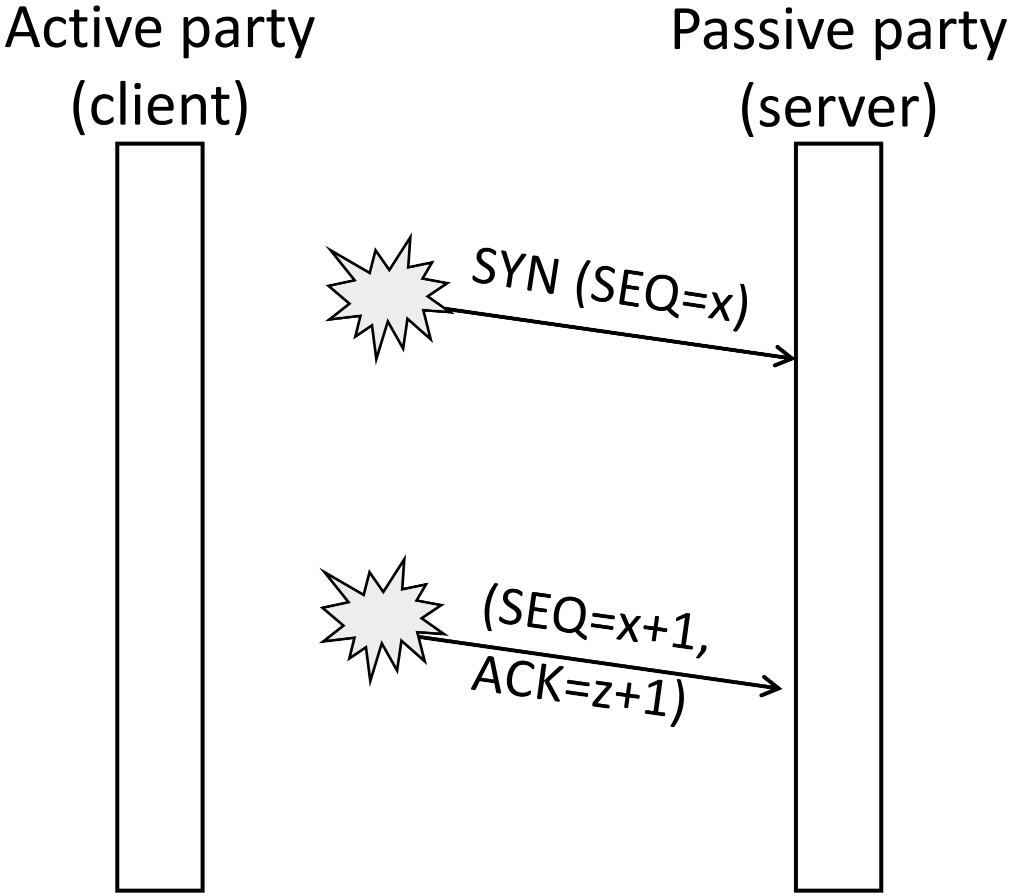
Three-Way Handshake (2)

- Three steps:
  - Client sends SYN(x)
  - Server replies with SYN(y)ACK(x+1)
  - Client replies with ACK(y+1)
  - SYNs are retransmitted if lost
- Sequence and ack numbers carried on further segments



Three-Way Handshake (3)

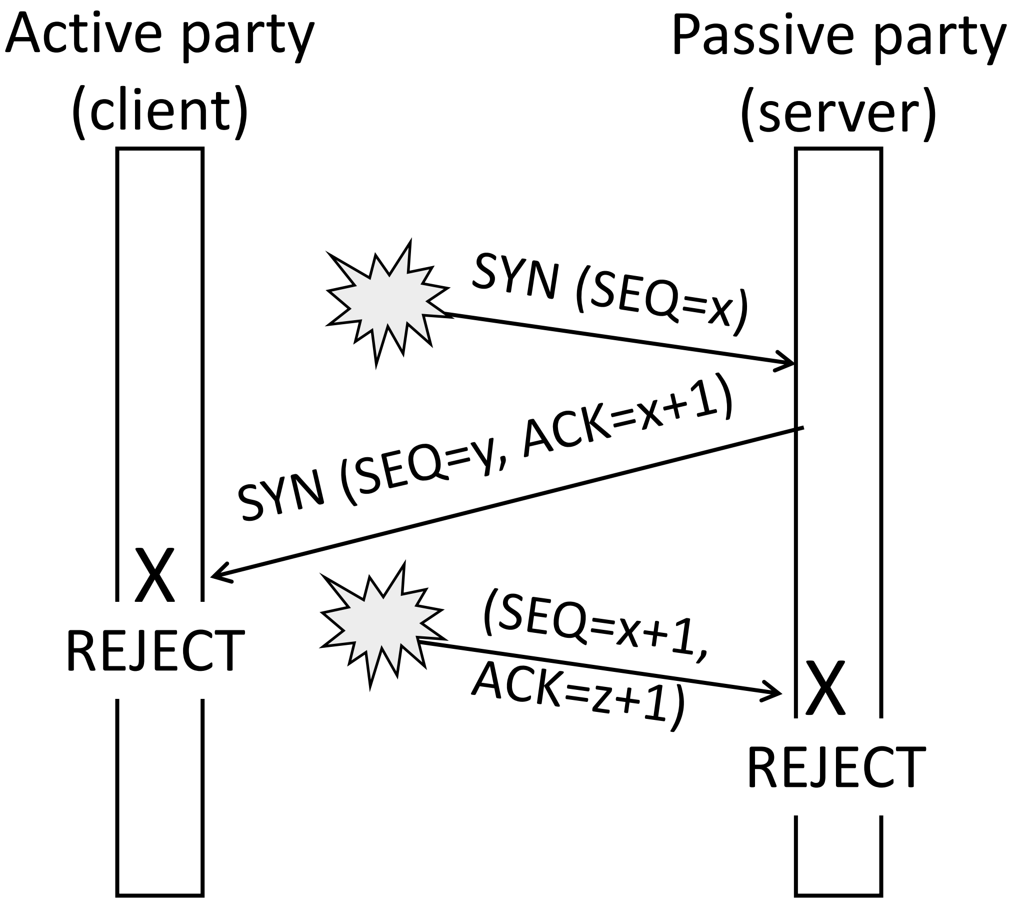
- Suppose delayed, duplicate copies of the SYN and ACK arrive at the server!
  - Improbable, but anyhow …



Three-Way Handshake (4)

- Suppose delayed, duplicate copies of the SYN and ACK arrive at the server!
  - Improbable, but anyhow …

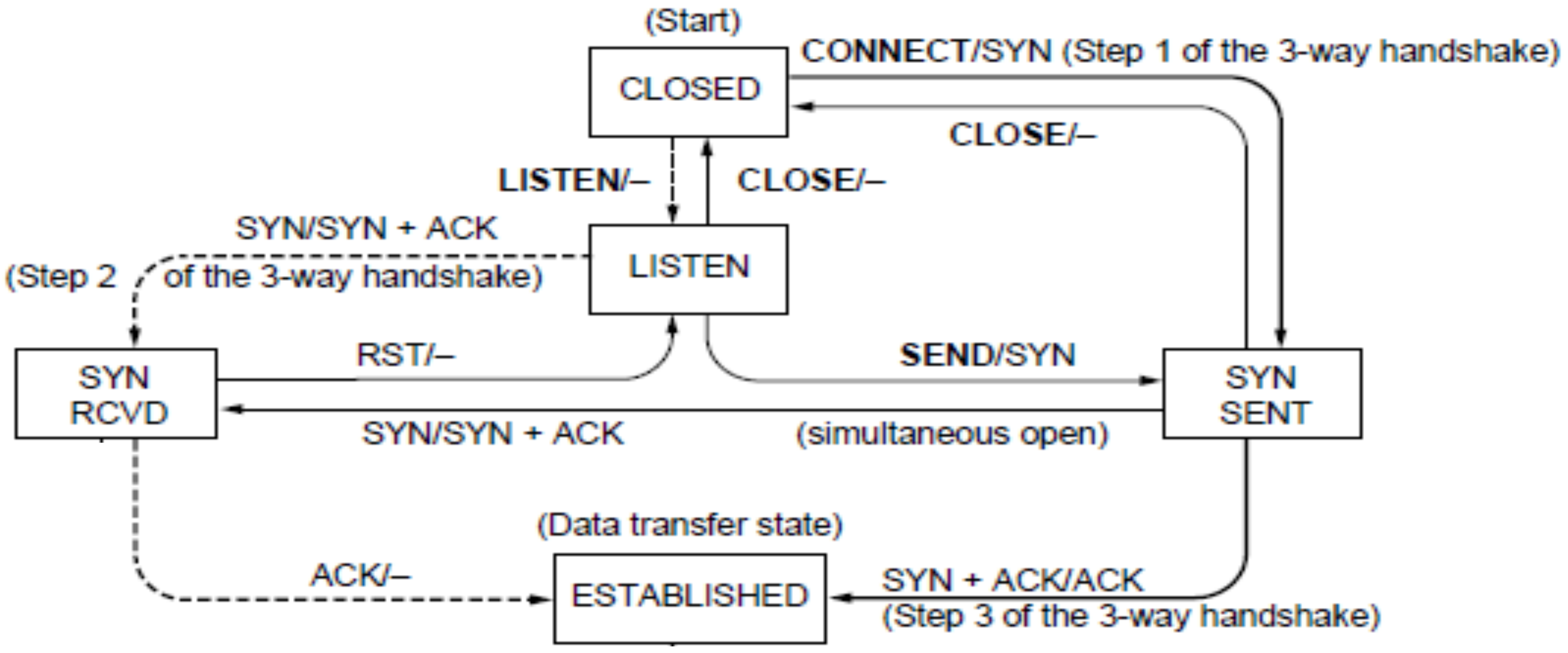
- Connection will be cleanly rejected on both sides ☺



TCP Connection State Machine

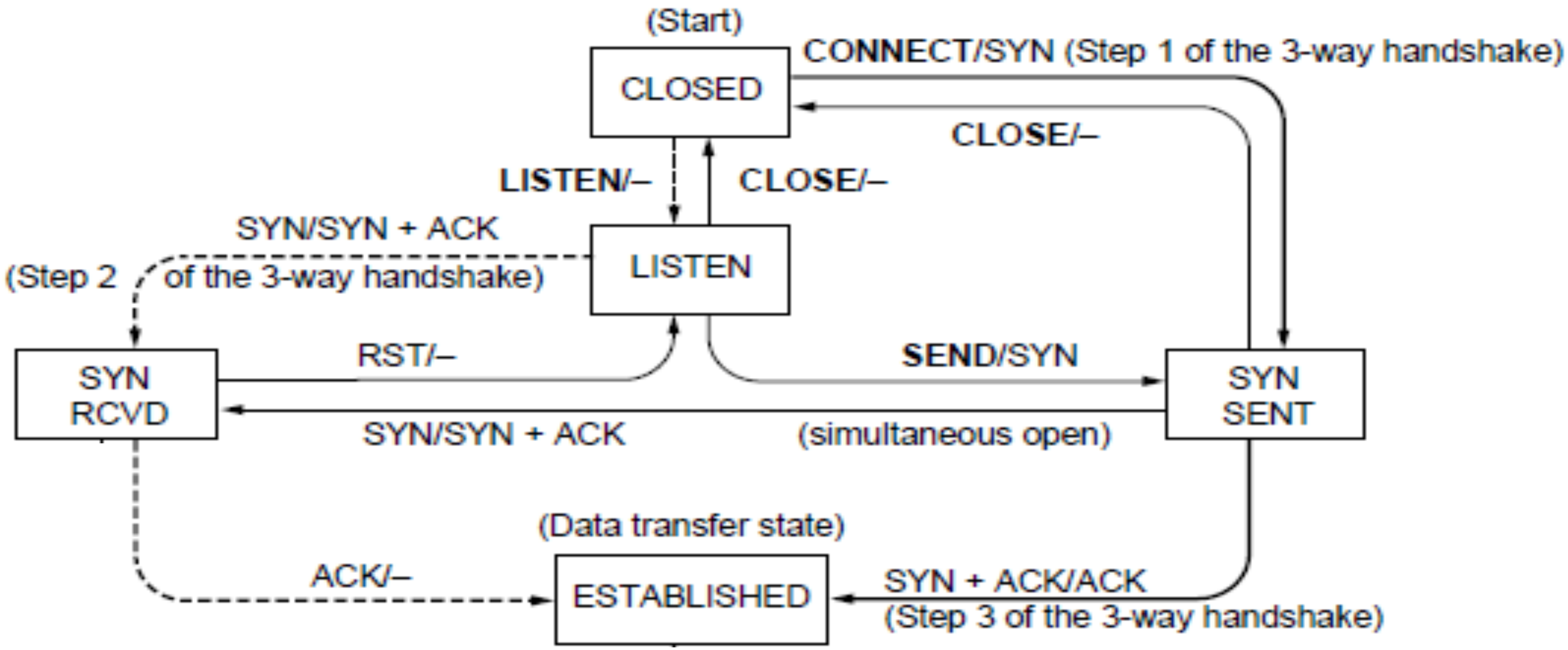
- Captures the states ([]) and transitions (->)
  - A/B means event A triggers the transition, with action B

Both parties run instances of this state machine



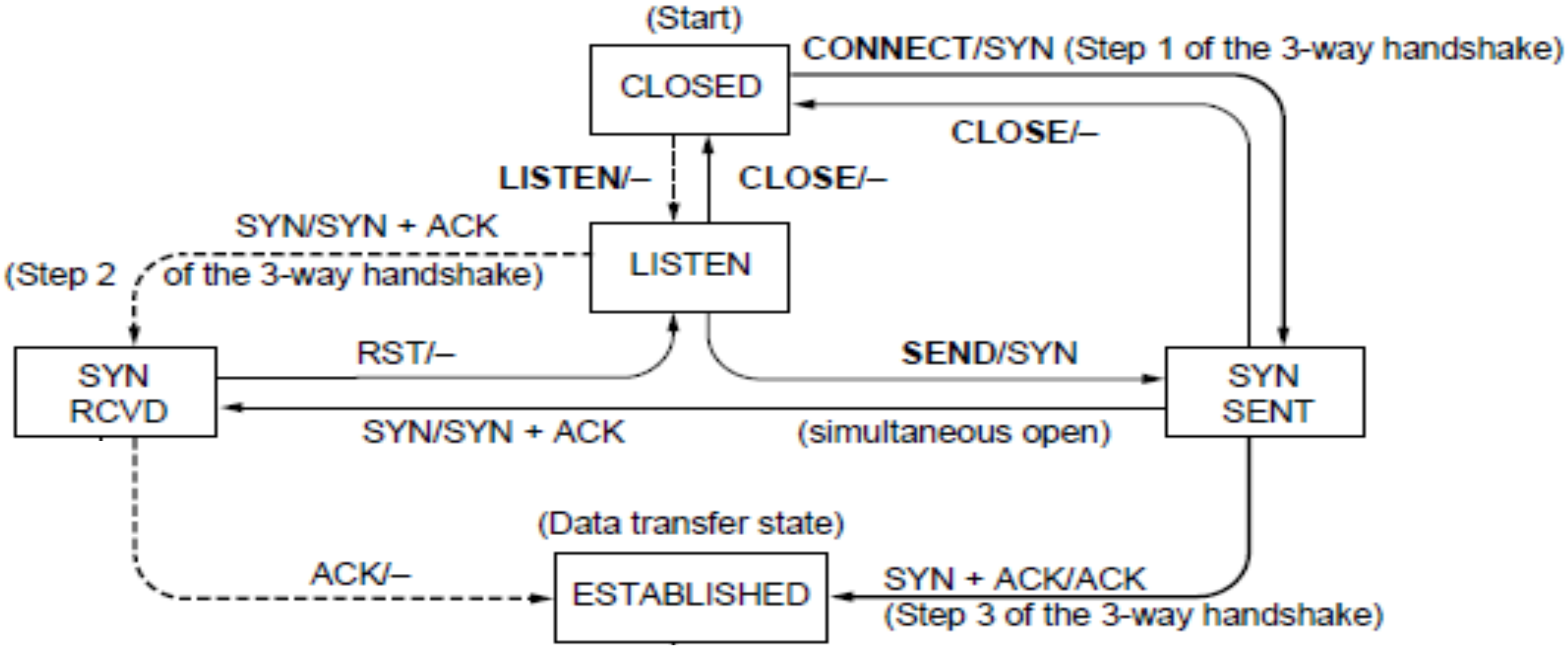
TCP Connections (2)

- Follow the path of the client:



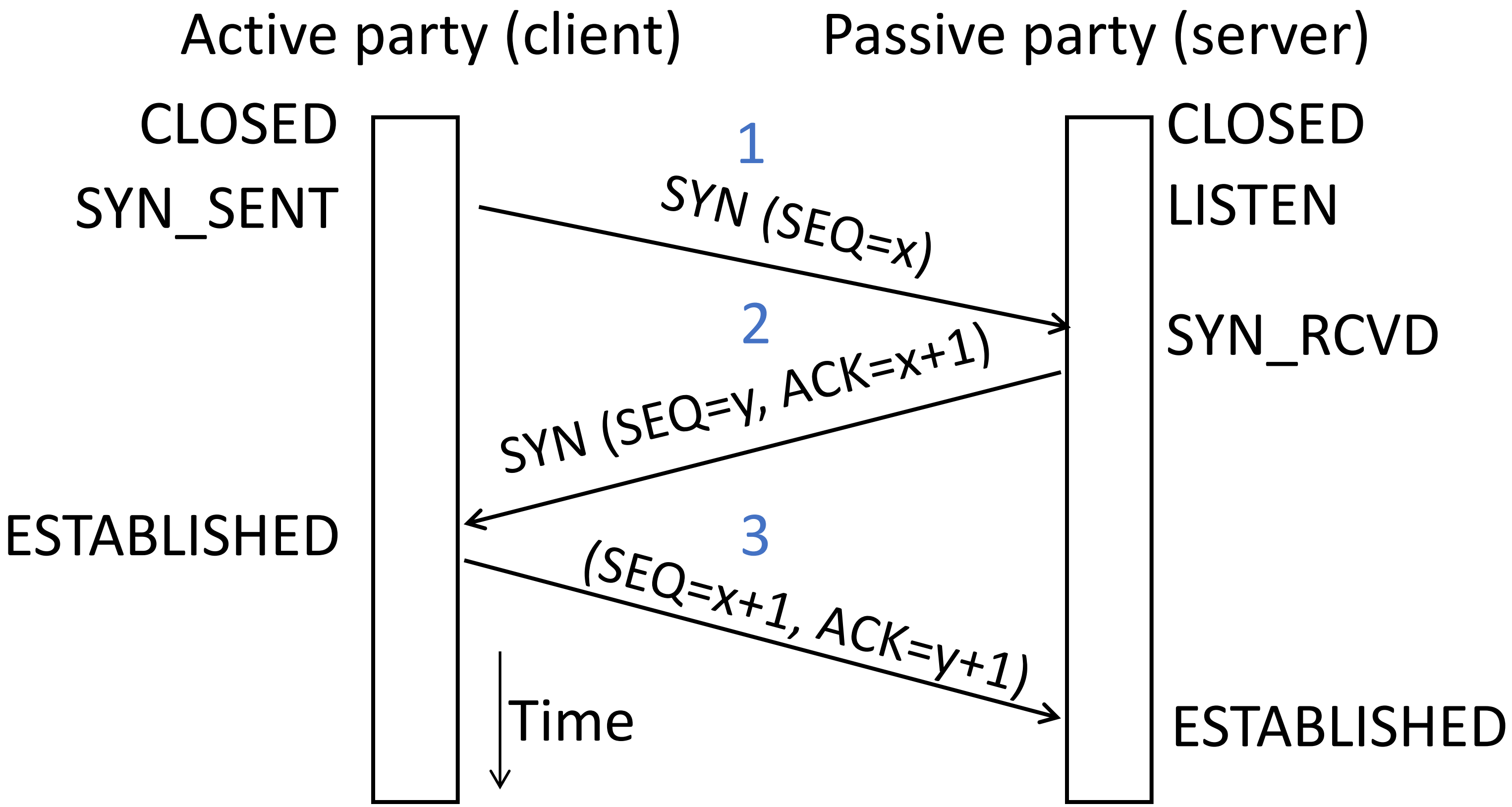
TCP Connections (3)

- And the path of the server:



TCP Connections (4)

- Again, with states …



TCP Connections (5)

- Finite state machines are a useful tool to specify and check the handling of all cases that may occur
- TCP allows for simultaneous open
  - i.e., both sides open instead of the client-server pattern
  - Try at home to confirm it works ☺

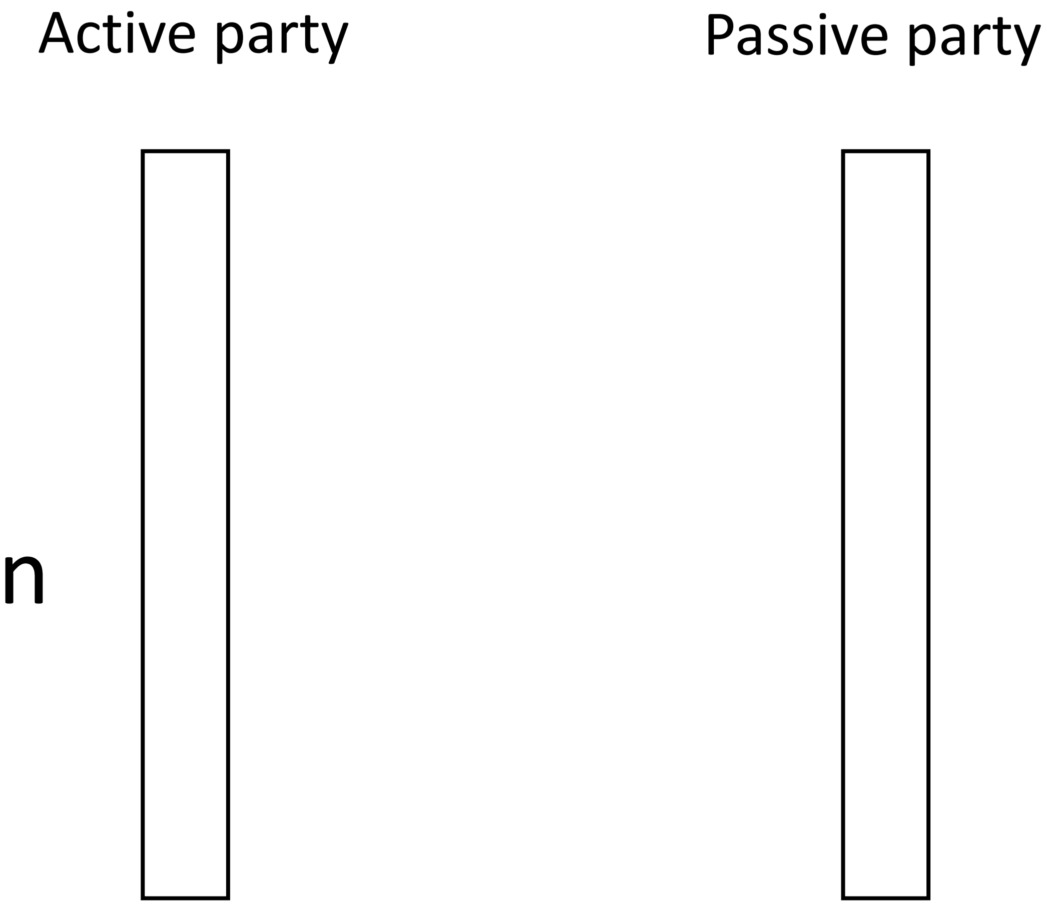
Connection Release

- Orderly release by both parties when done
  - Delivers all pending data and “hangs up”
  - Cleans up state in sender and receiver
- Key problem is to provide reliability while releasing
  - TCP uses a “symmetric” close in which both sides shutdown independently

TCP Connection Release

- Two steps:
  - Active sends FIN(x), passive ACKs
  - Passive sends FIN(y), active ACKs
  - FINs are retransmitted if lost

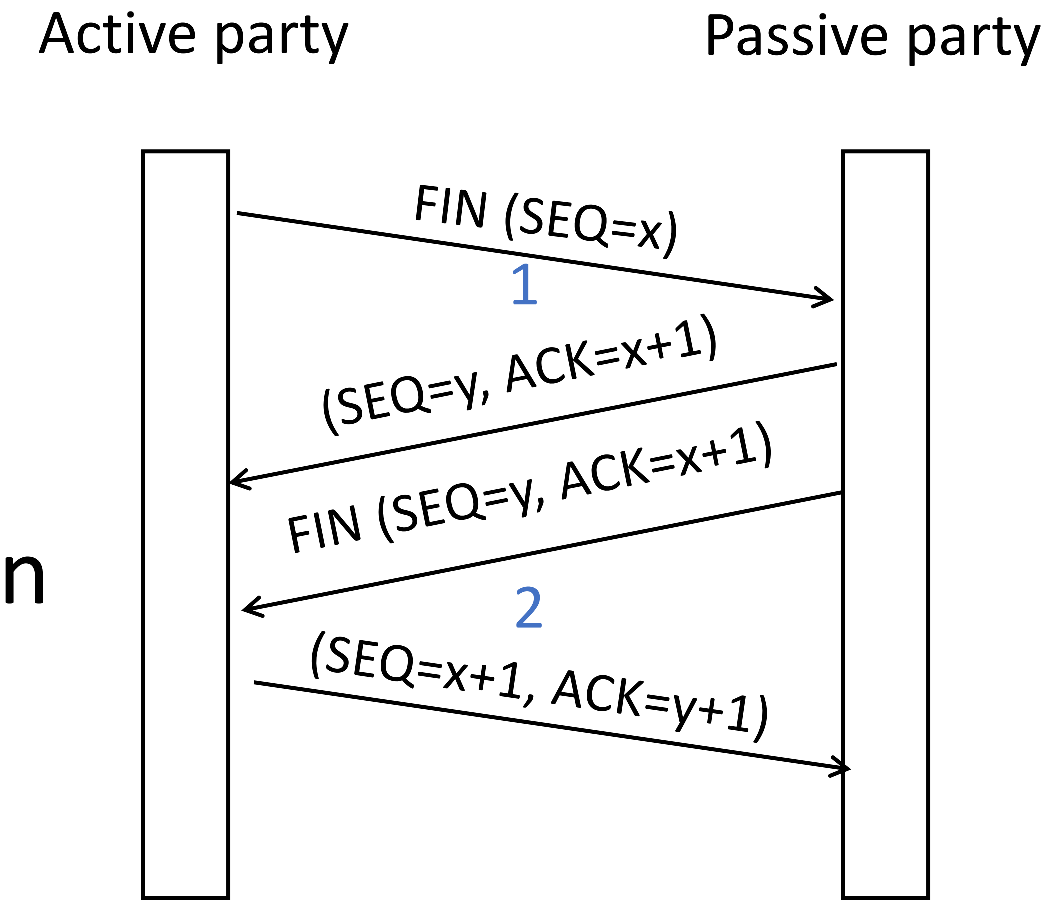
- Each FIN/ACK closes one direction of data transfer



TCP Connection Release (2)

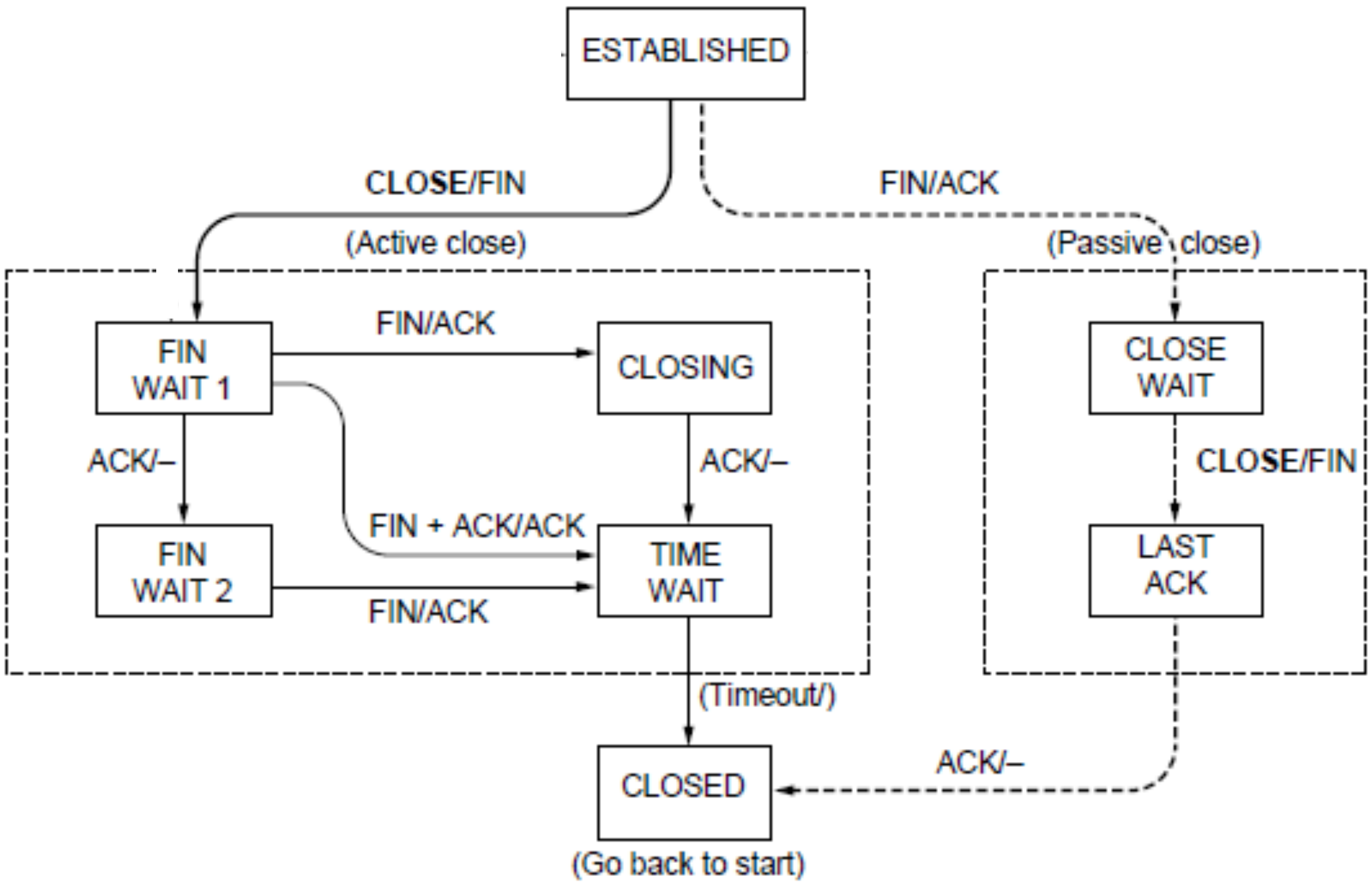
- Two steps:
  - Active sends FIN(x), passive ACKs
  - Passive sends FIN(y), active ACKs
  - FINs are retransmitted if lost

- Each FIN/ACK closes one direction of data transfer



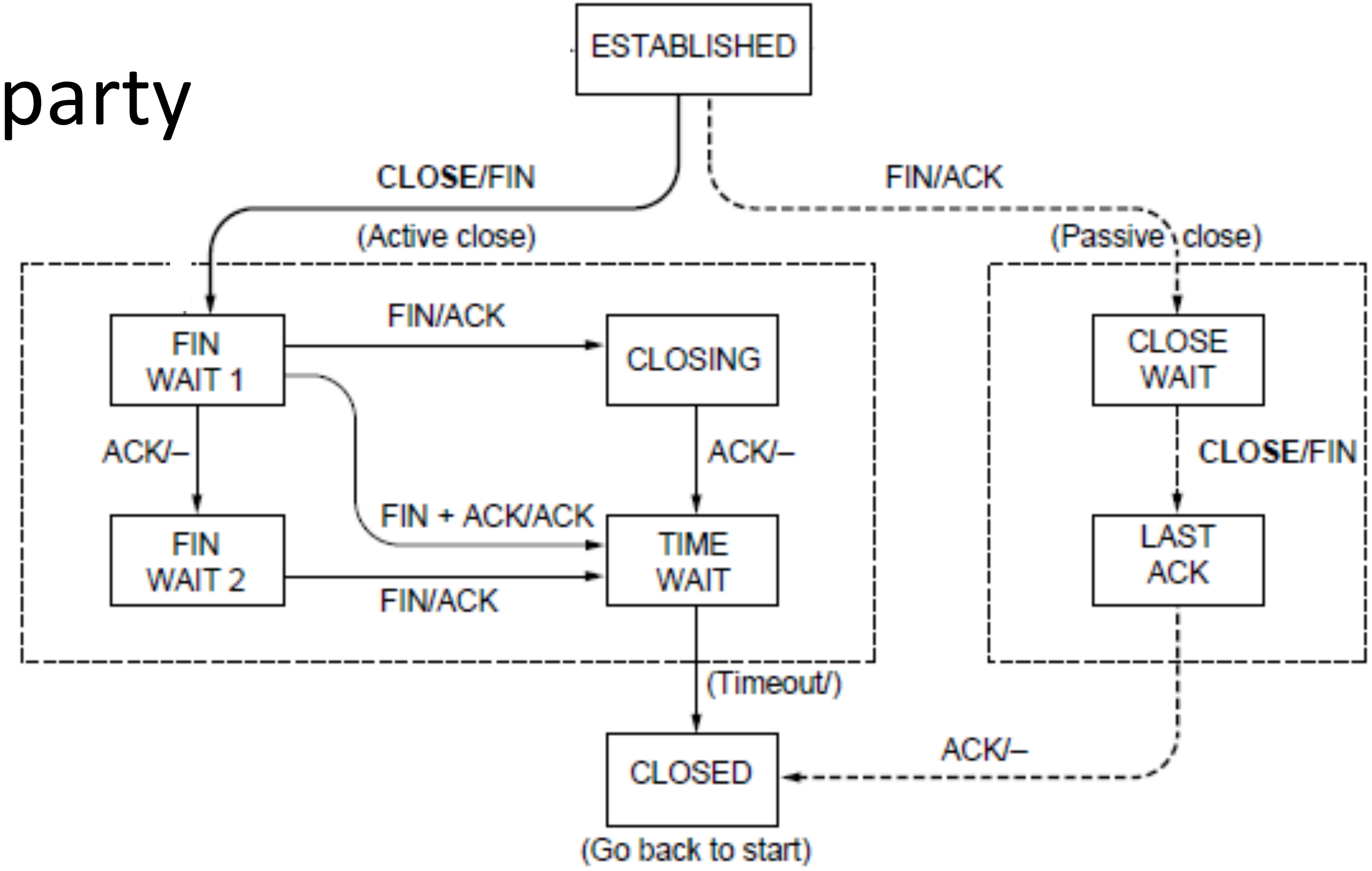
TCP Connection State Machine

Both parties run instances of this state machine



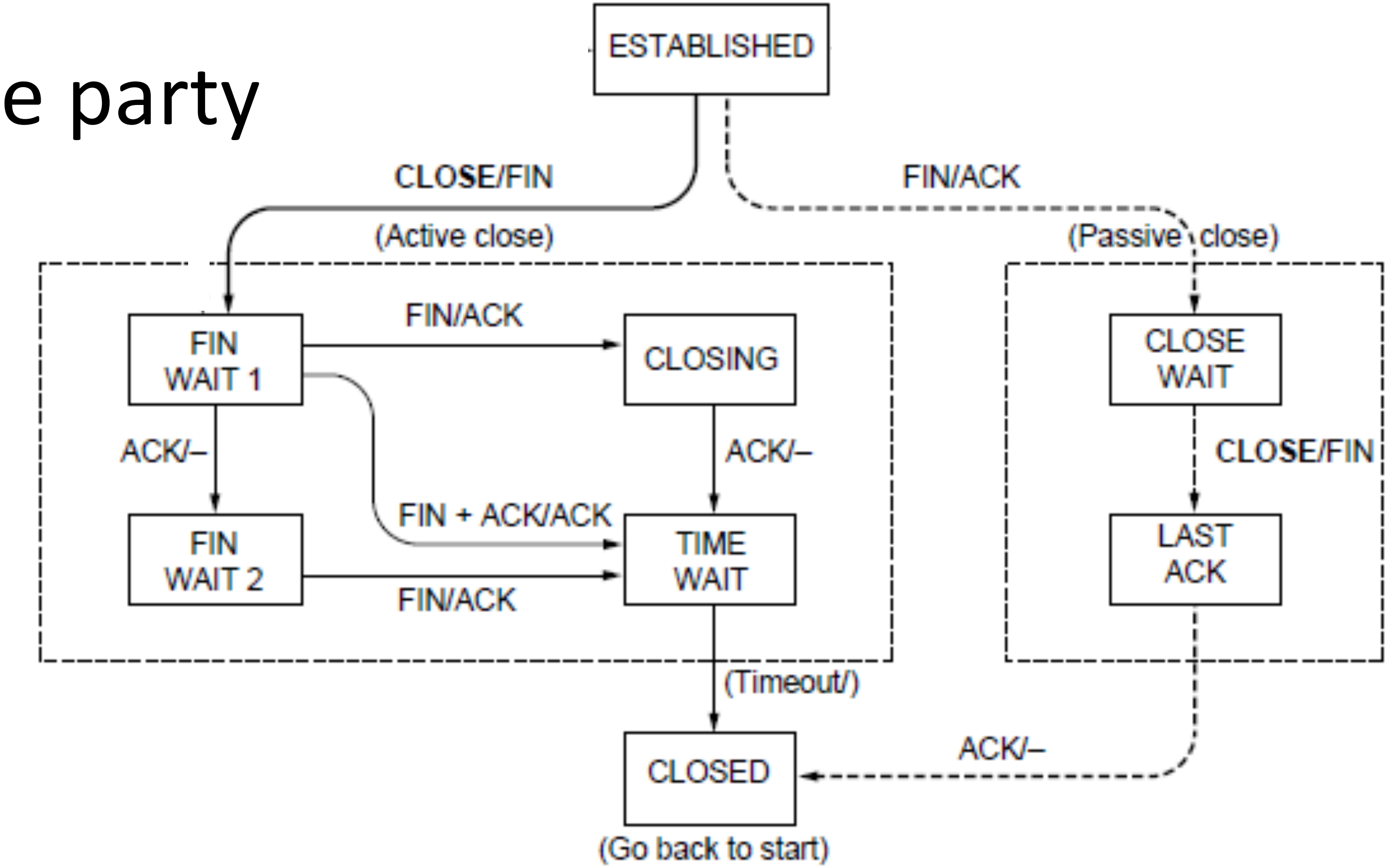
TCP Release

- Follow the active party



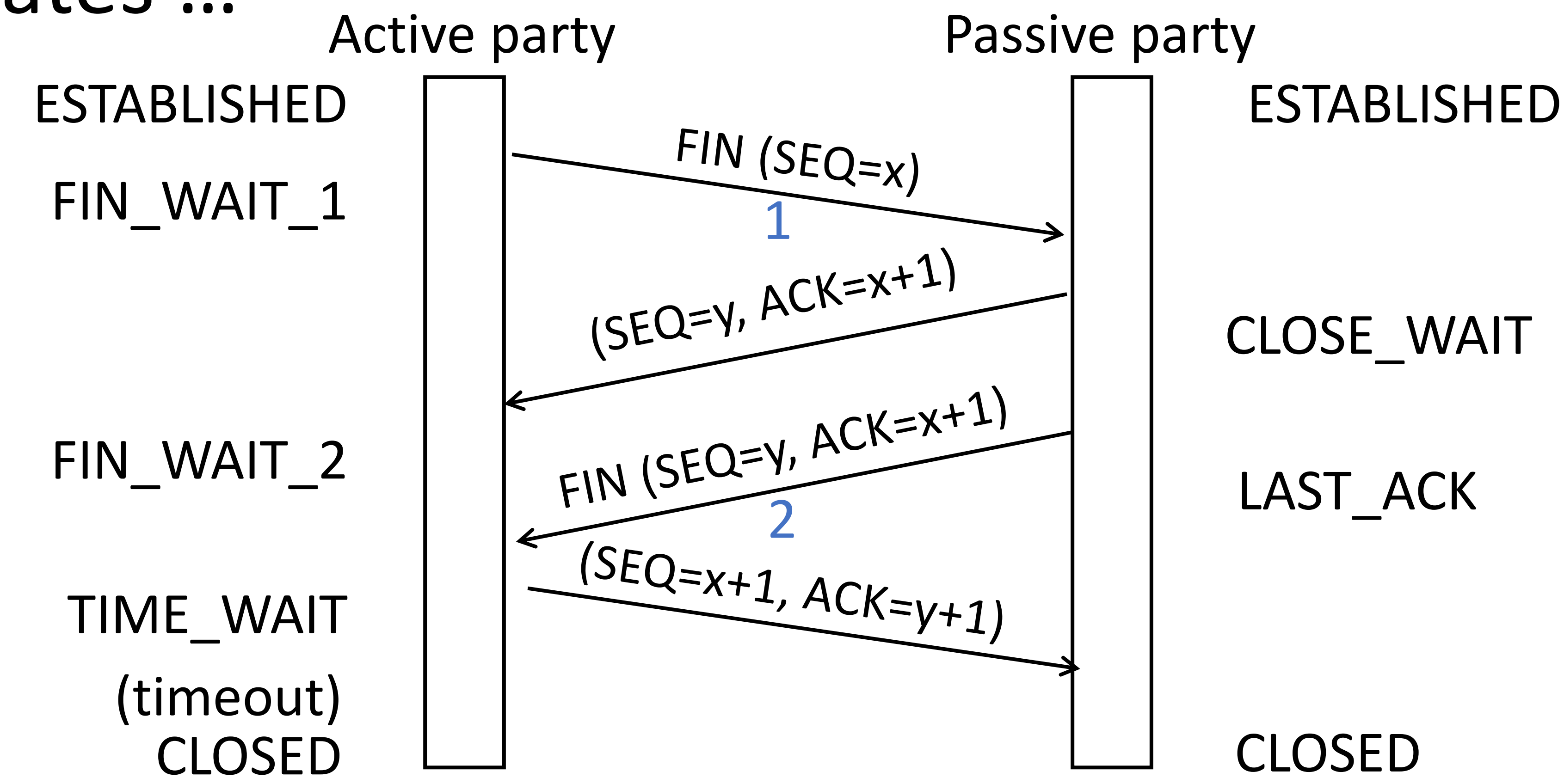
TCP Release (2)

- Follow the passive party



TCP Release (3)

- Again, with states …



TIME_WAIT State

- Wait a long time after sending all segments and before completing the close
    - Two times the maximum segment lifetime of 60 seconds
- Why?

TIME_WAIT State

- Wait a long time after sending all segments and before completing the close
  - Two times the maximum segment lifetime of 60 seconds
- Why?
  - ACK might have been lost, in which case FIN will be resent for an orderly close
  - Could otherwise interfere with a subsequent connection