Network Programming
Transport Layer

Minchen Yu

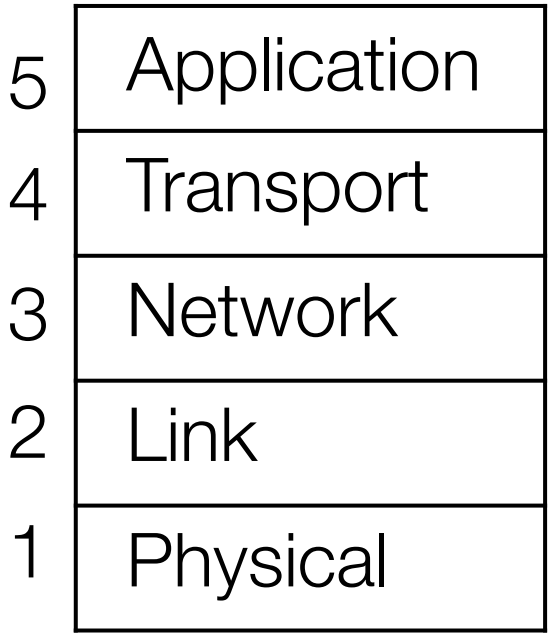
SDS@CUHK-SZ

Spring 2026

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

SCHOOL OF
DATA SCIENCE
數據科學學院

Network model



Programs that use network service

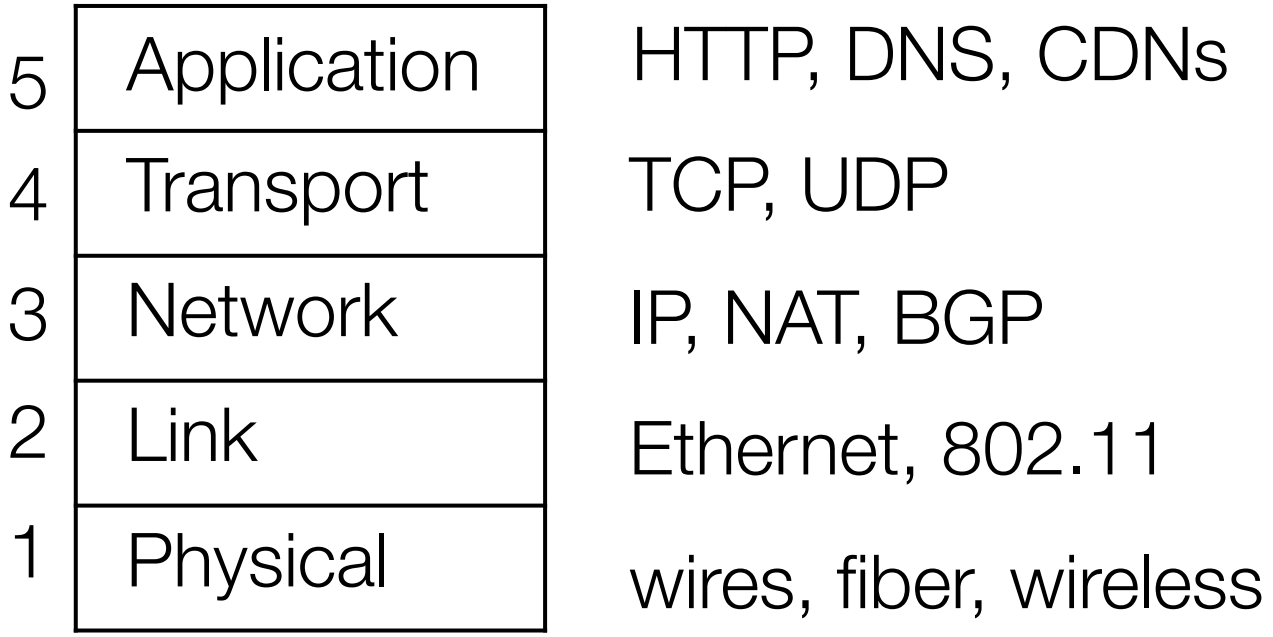
Provides end-to-end data delivery

Send packets over multiple networks

Send frames over one or more links

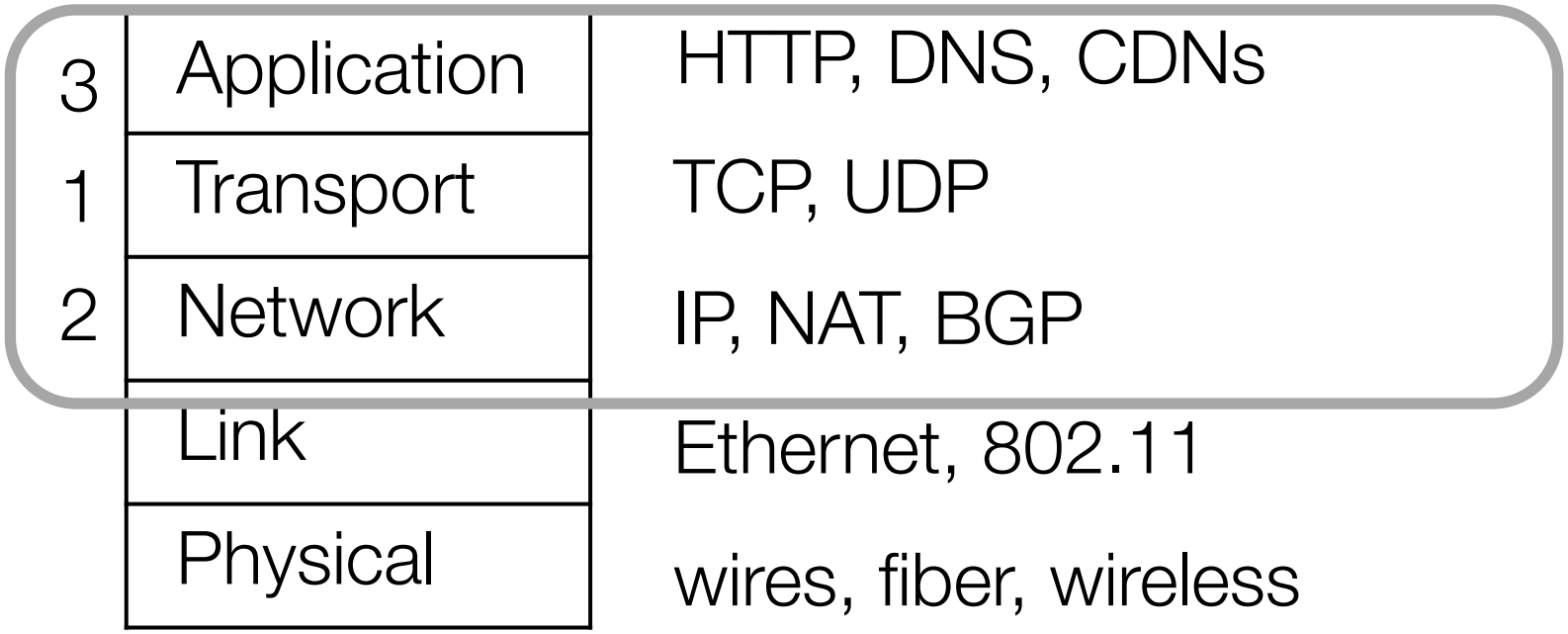
Send bits using signals

Network model



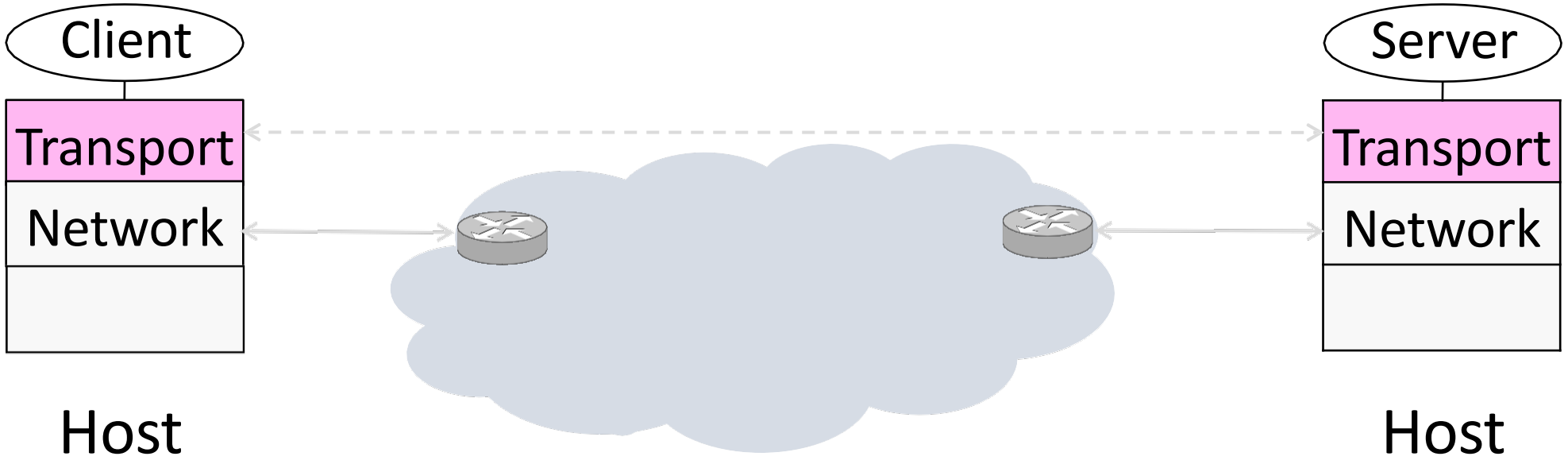
Coverage

Middle (TCP/UDP/IP) -> Top (HTTP)



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**
Datagrams (UDP)	Streams (TCP)

Comparison of Internet transports

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

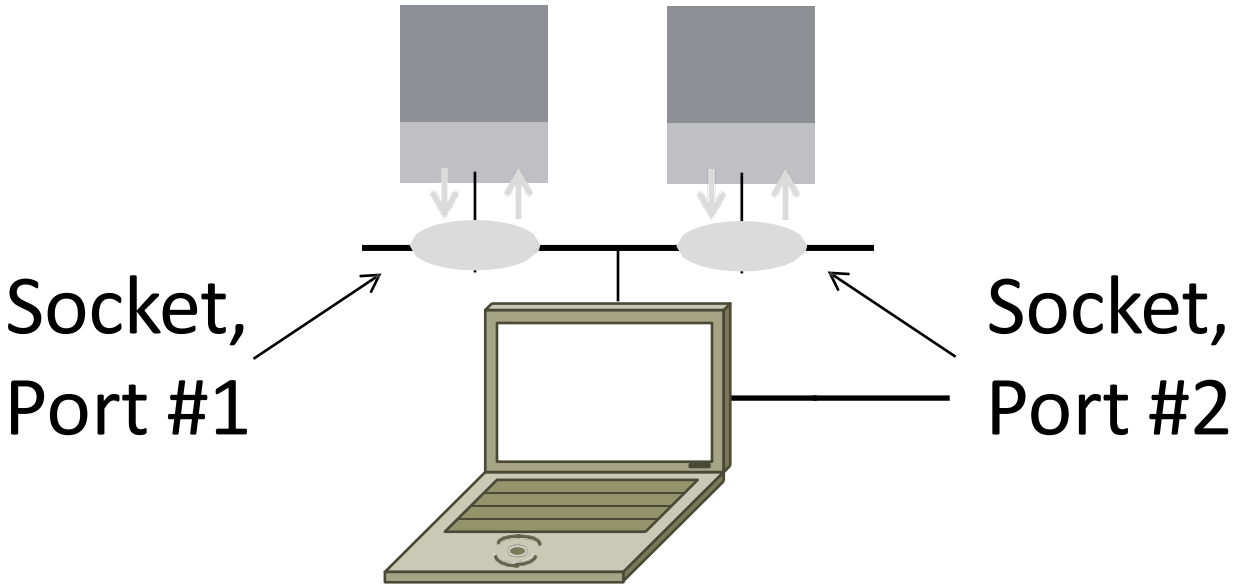
Socket

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

Socket

Sockets let apps attach to the local network at different ports



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”

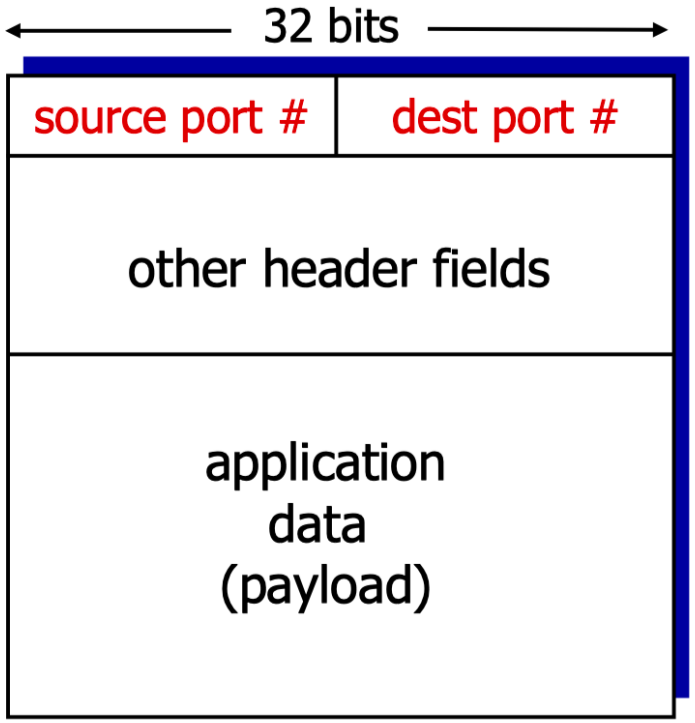
- <1024, require administrative privileges

Clients often assigned “ephemeral” ports

- Chosen by OS, used temporarily

TCP/UDP format

Use IP addresses & port numbers to direct packets to appropriate socket



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

User Datagram Protocol (UDP)

Used by apps that don’t want reliability or bytestreams

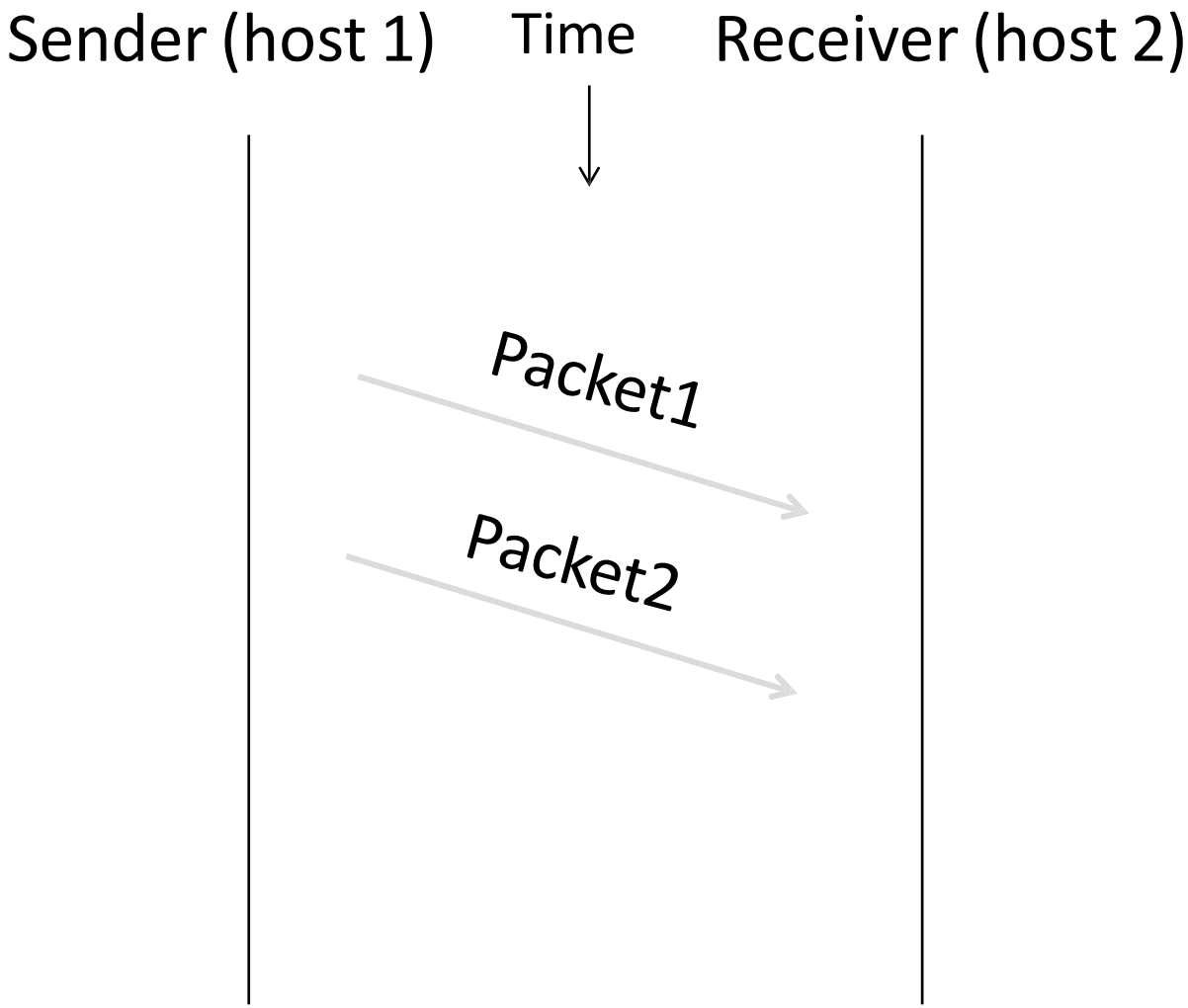
User Datagram Protocol (UDP)

Used by apps that don't want reliability or bytestreams

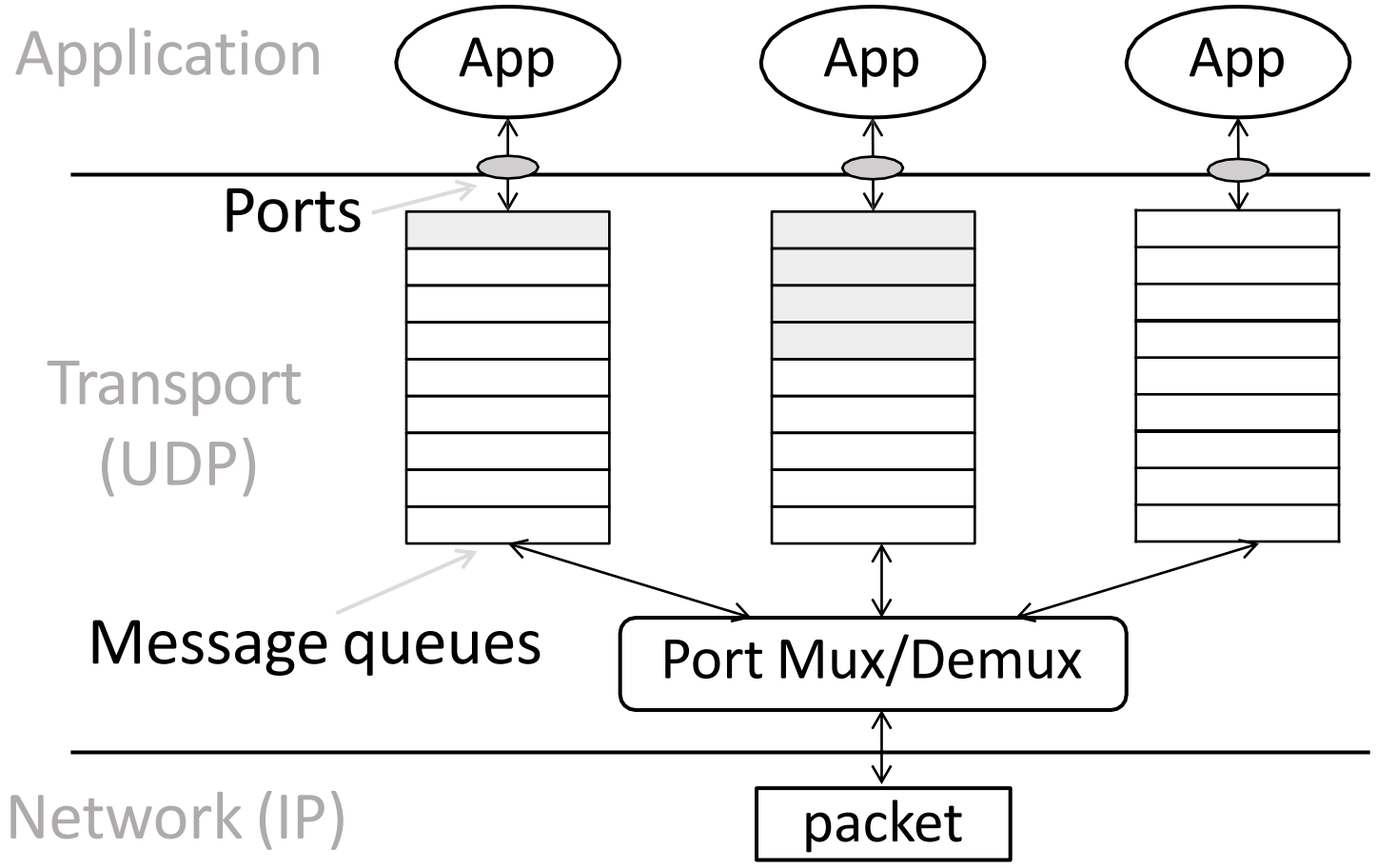
- Voice-over-IP
- DNS
- Games

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

Connectionless

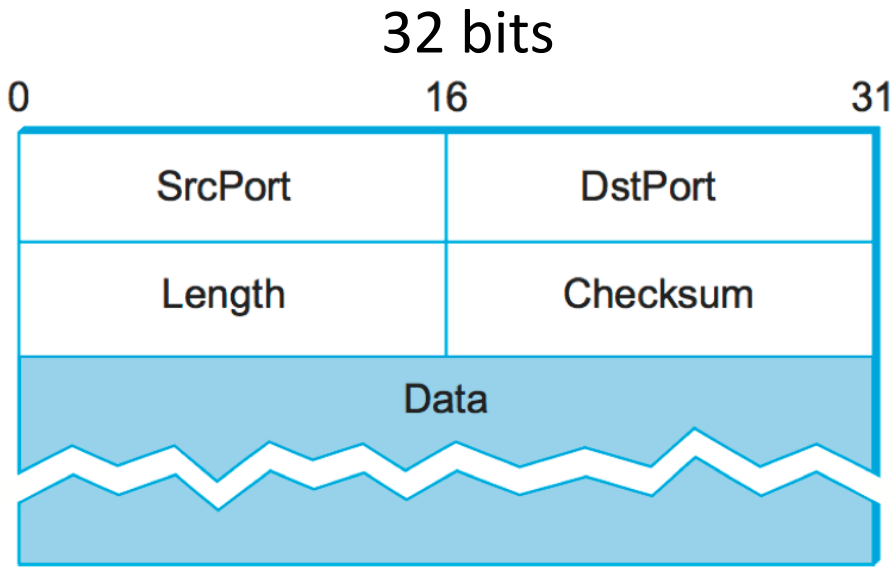


UDP buffering

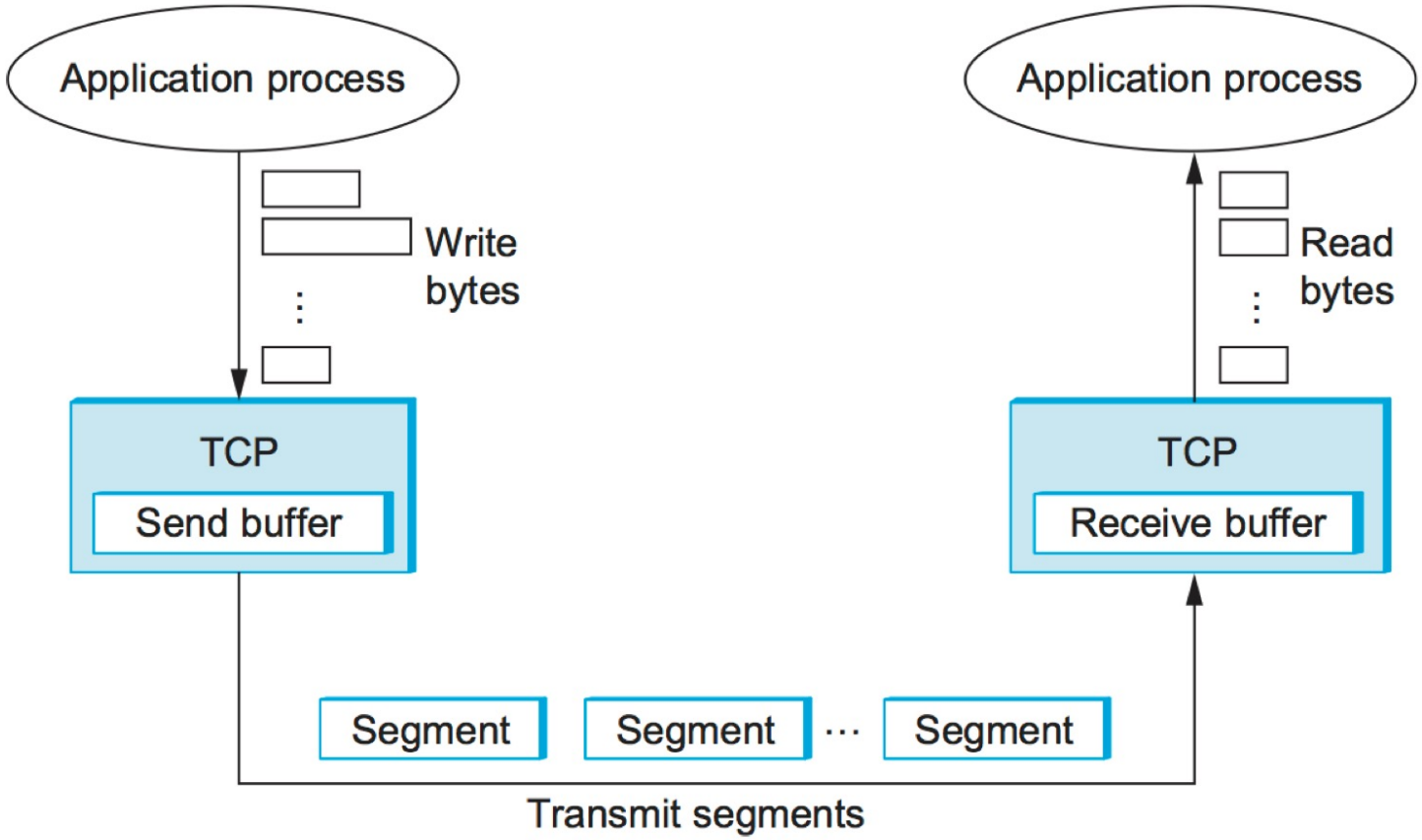


UDP header

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



TCP byte stream



TCP

Consists of 3 primary phases:

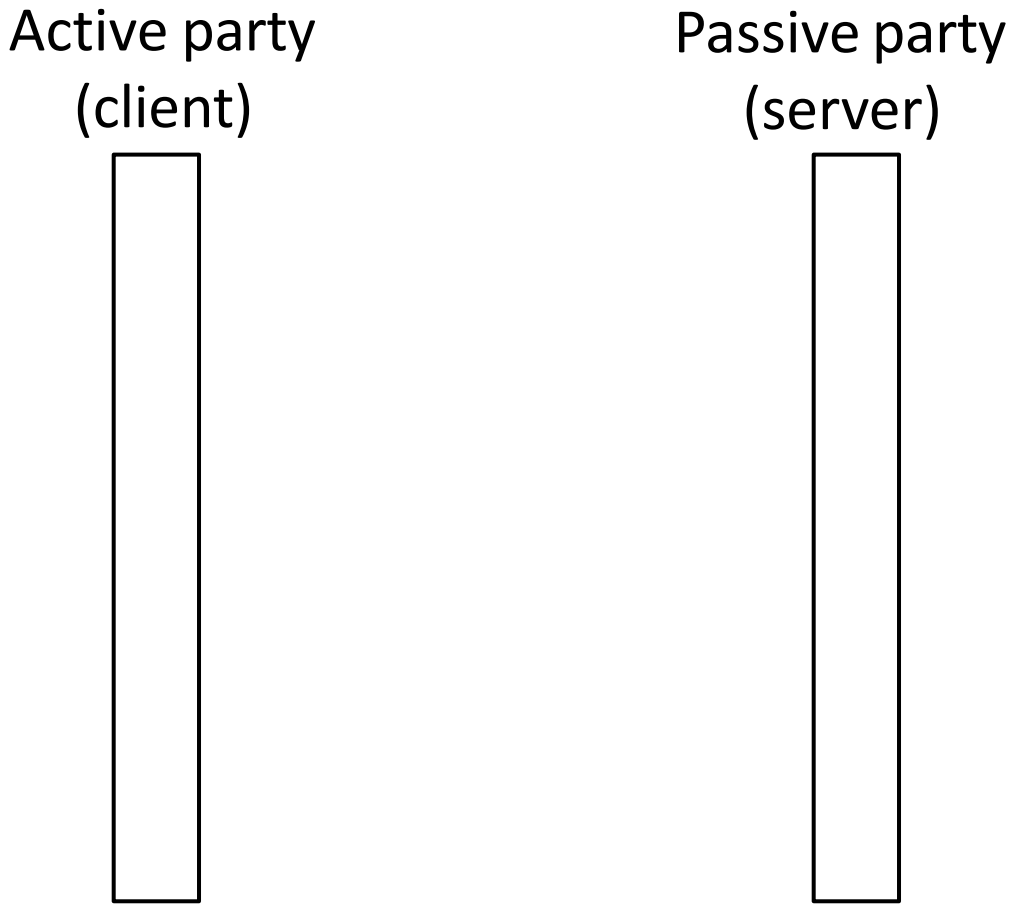
- Connection Establishment (Setup)
- Sliding Windows/Flow Control
- Connection Release (Teardown)

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

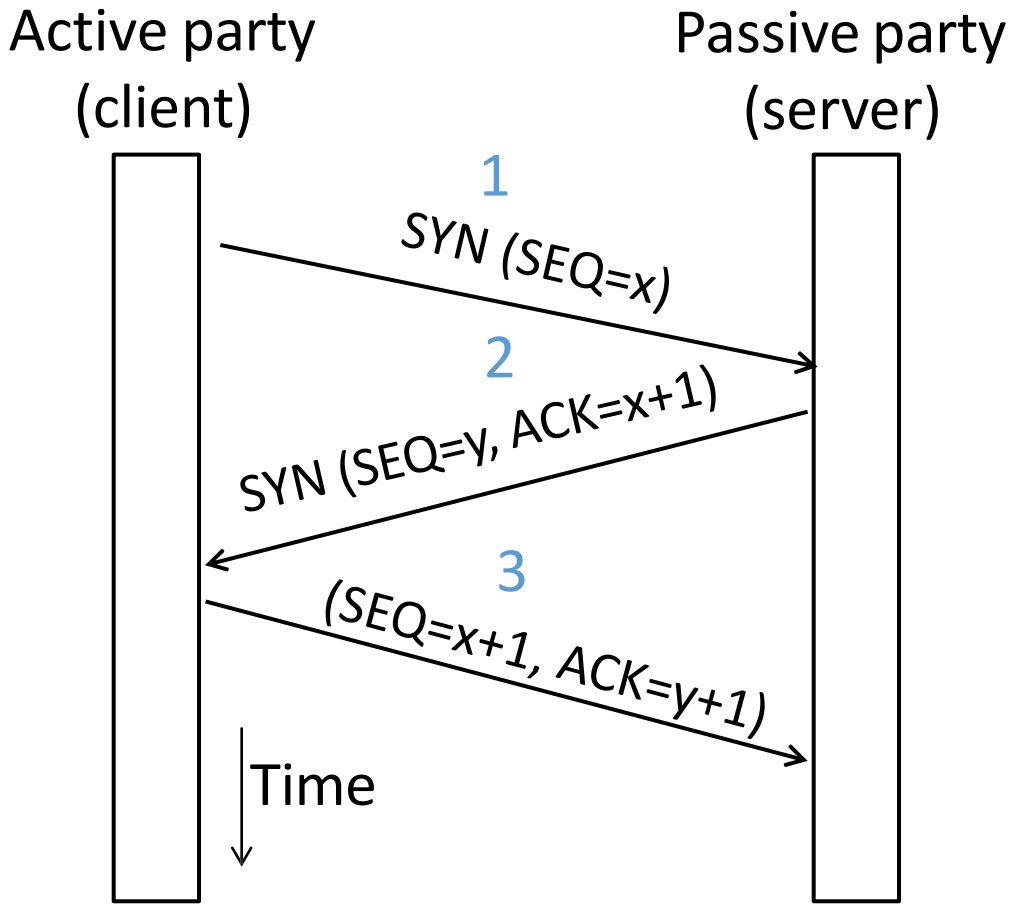
Three-Way Handshake

- 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



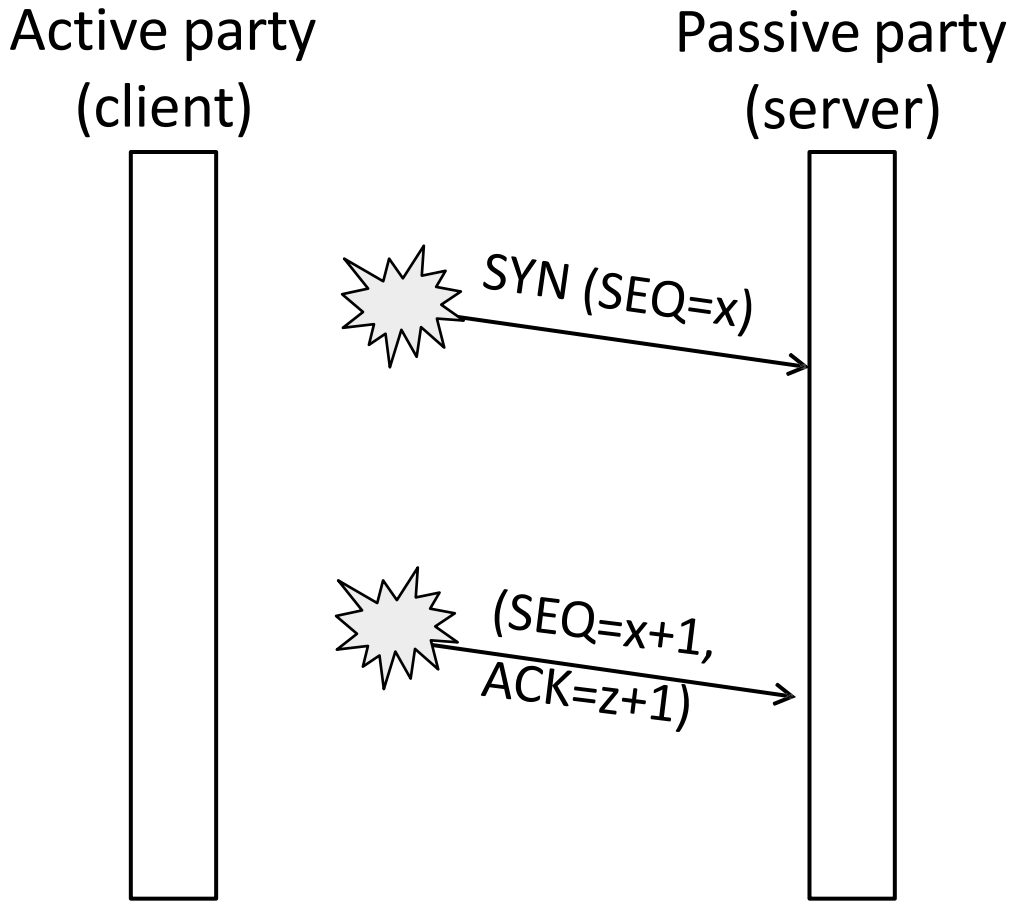
Three-Way Handshake

- 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



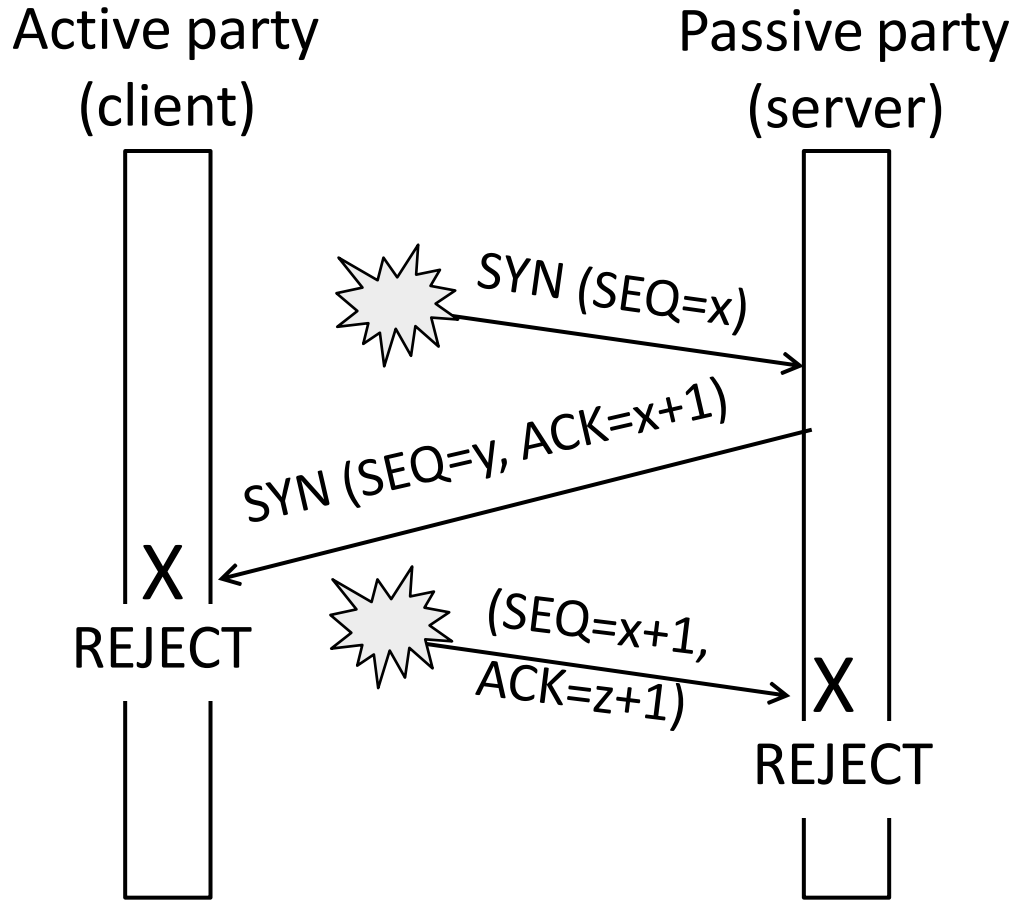
Three-Way Handshake

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



Three-Way Handshake

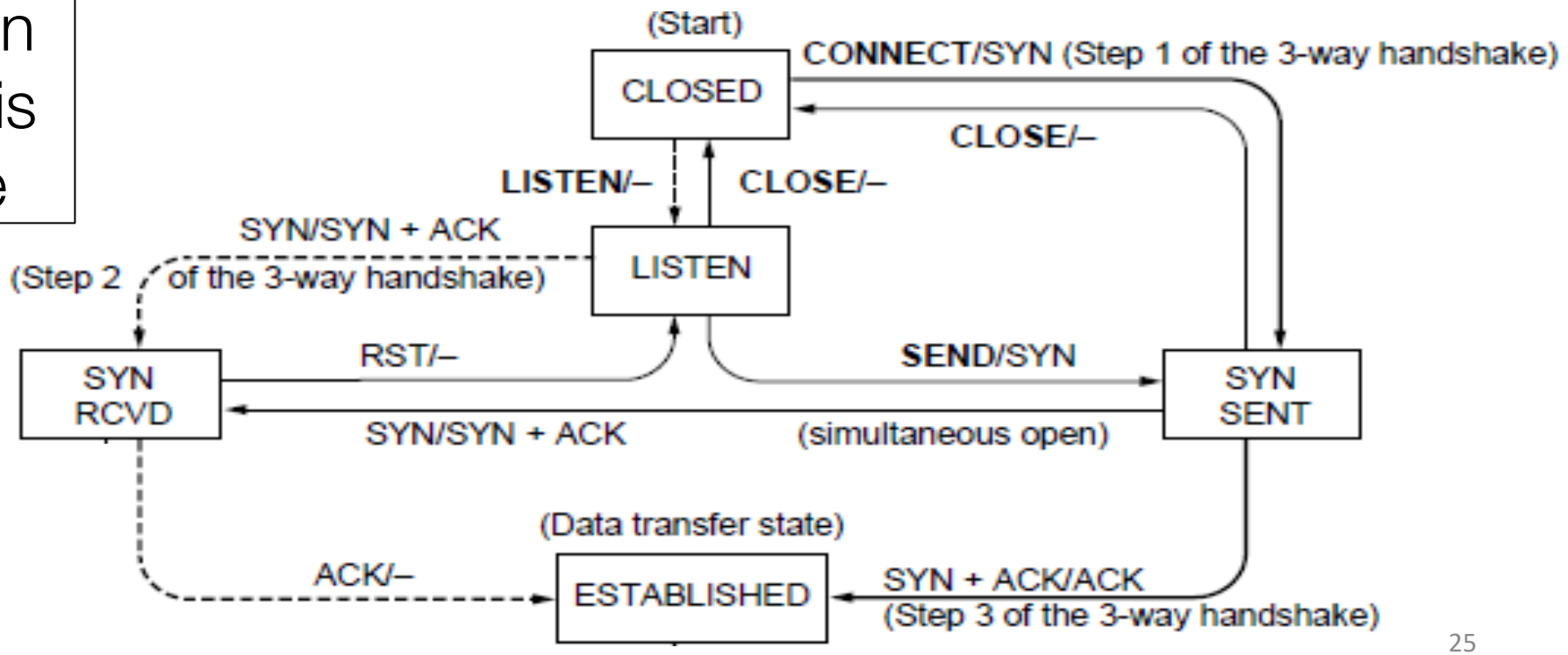
- Suppose delayed, duplicate copies of the SYN and ACK arrive at the server!
  - Improbable, but anyhow …
- Connection will be cleanly rejected on both sides



TCP Connection State Machine

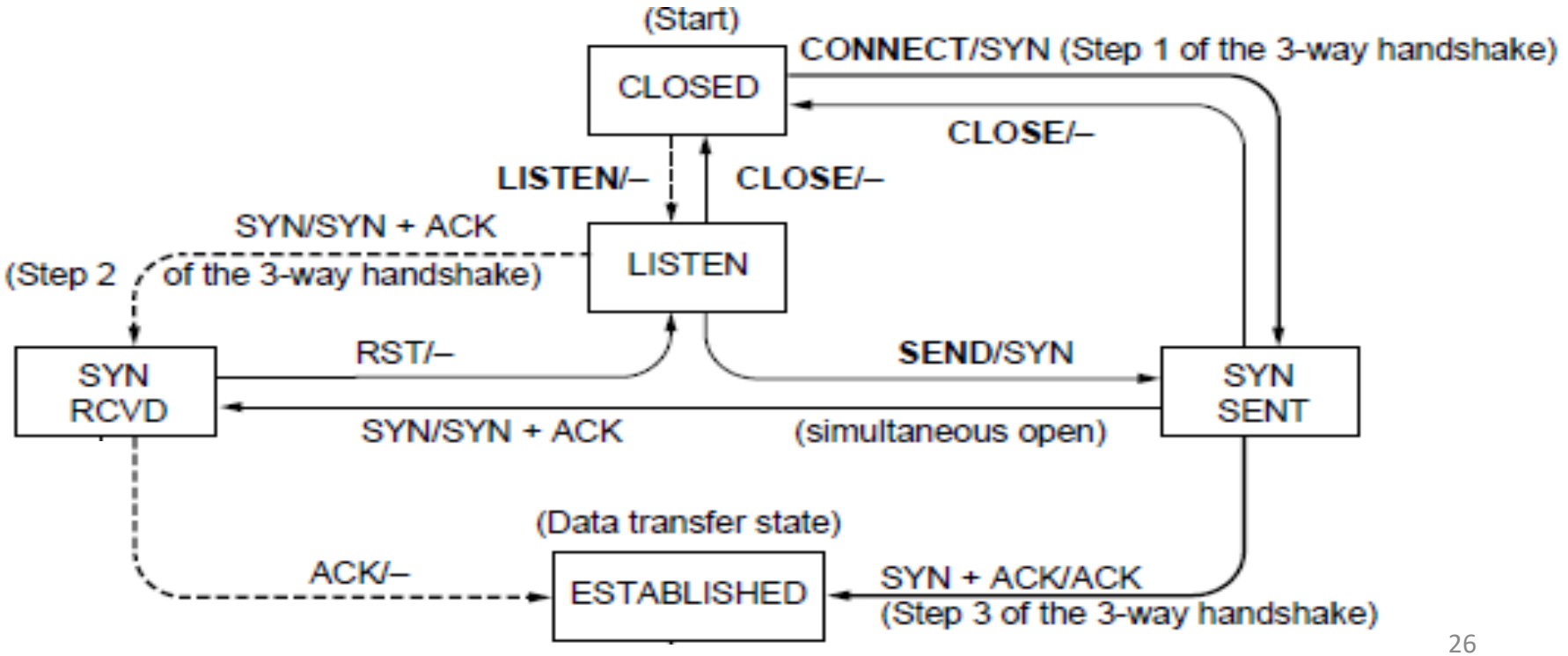
- Captures the states ([]) and transitions (->)
  - A/B means event A triggers the transition, with action B

Both parties run instances of this state machine



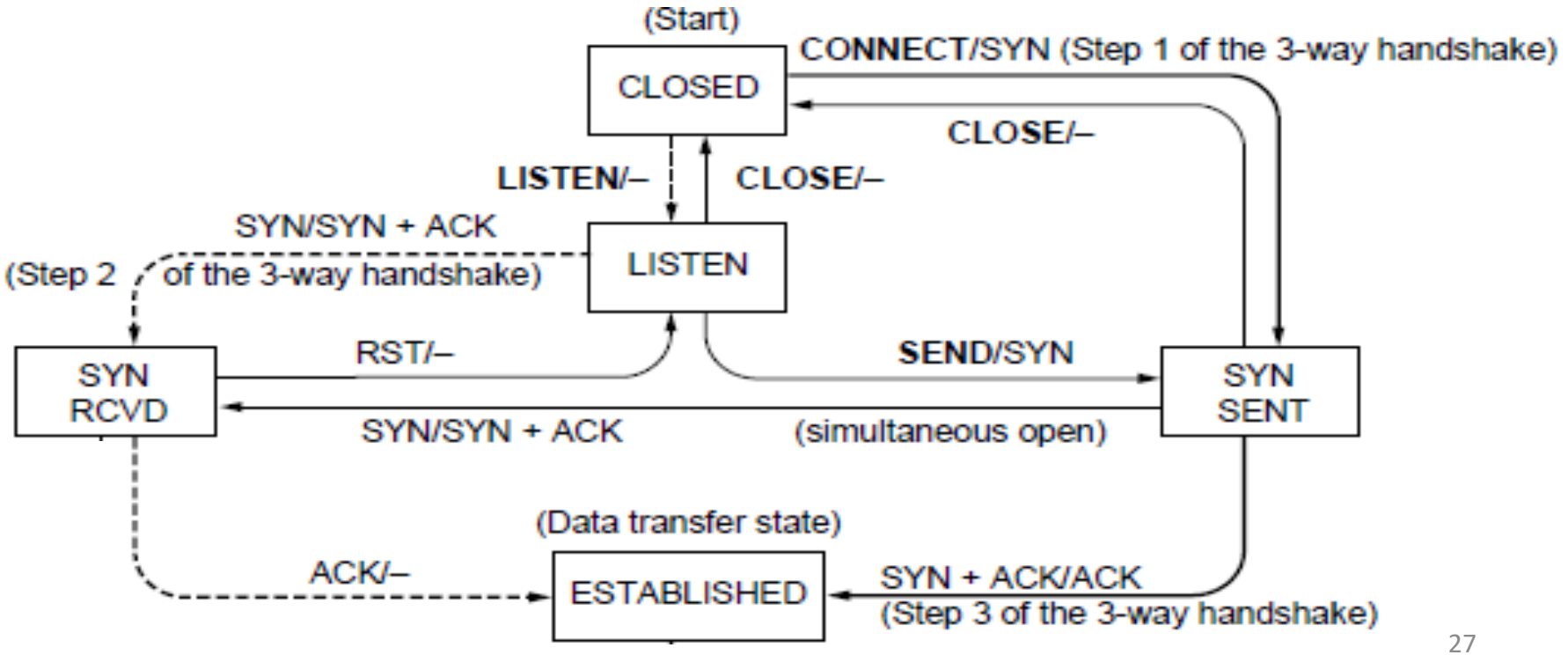
TCP connections

- Follow the path of the client:



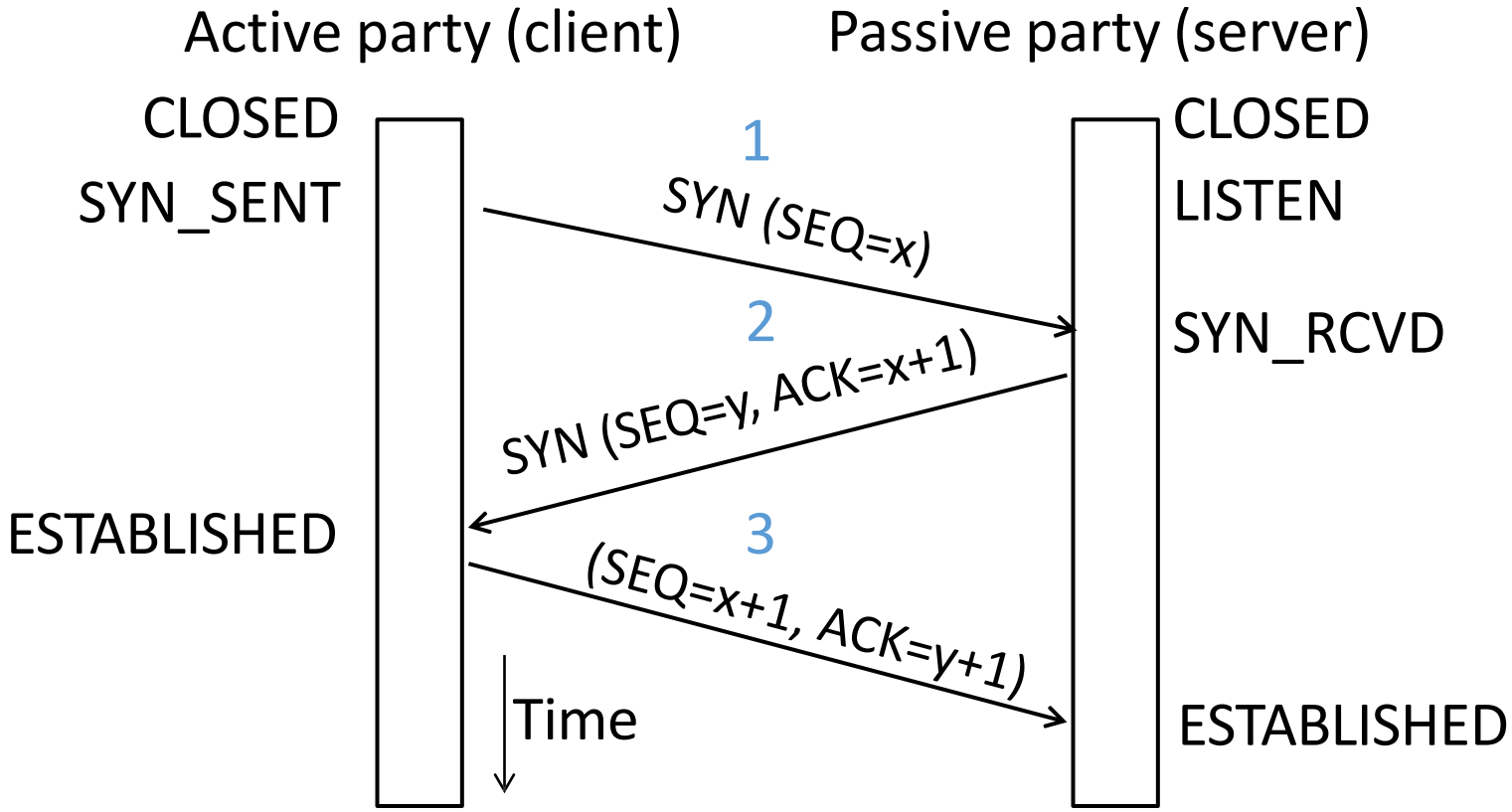
TCP connections

- And the path of the server:



TCP connections

- Again, with states …



TCP connections

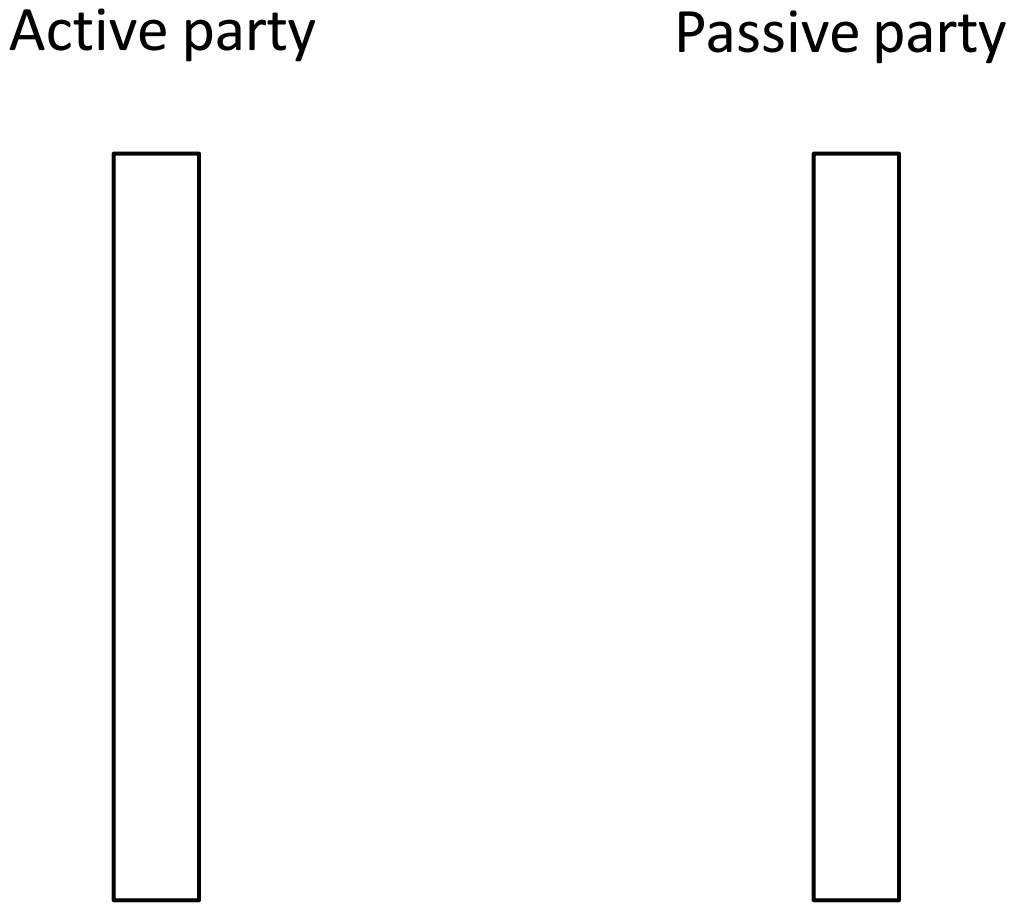
- Finite state machines are a useful tool to specify and check the handling of all cases that may occur
- TCP allows for simultaneous open
  - i.e., both sides open instead of the client-server pattern
  - Try at home to confirm it works

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

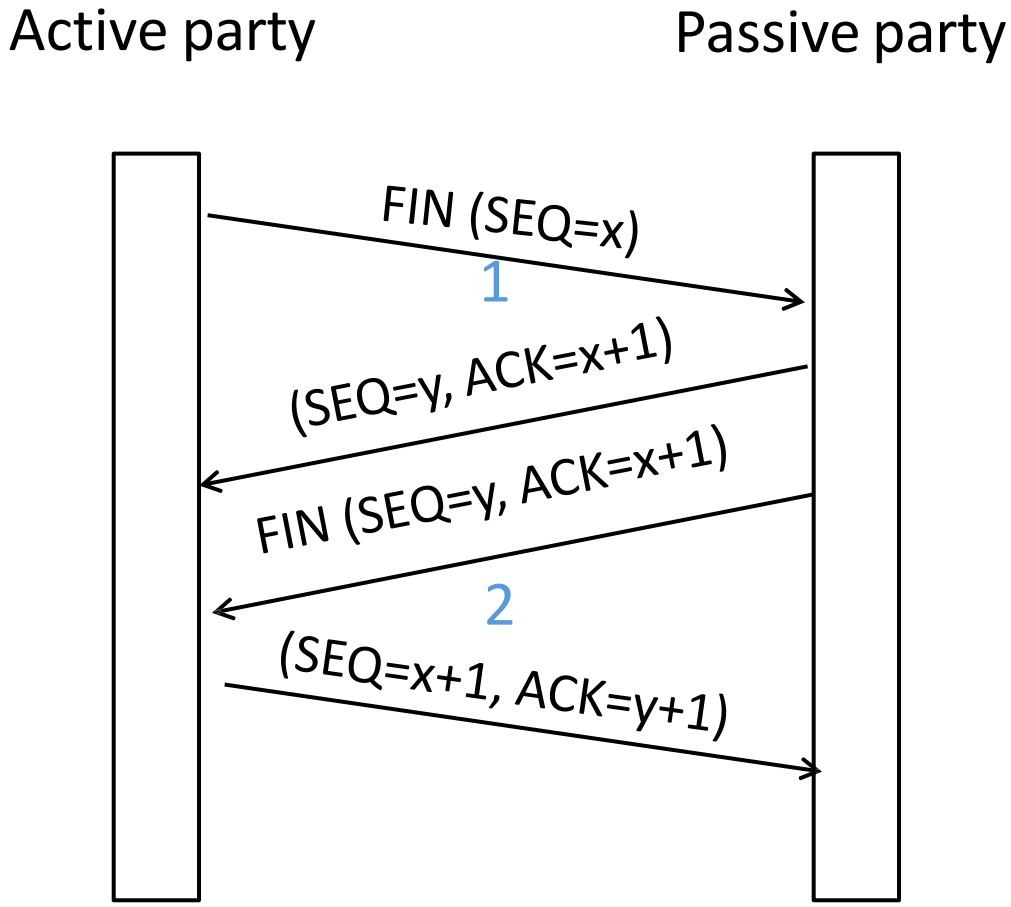
Connection release

- 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



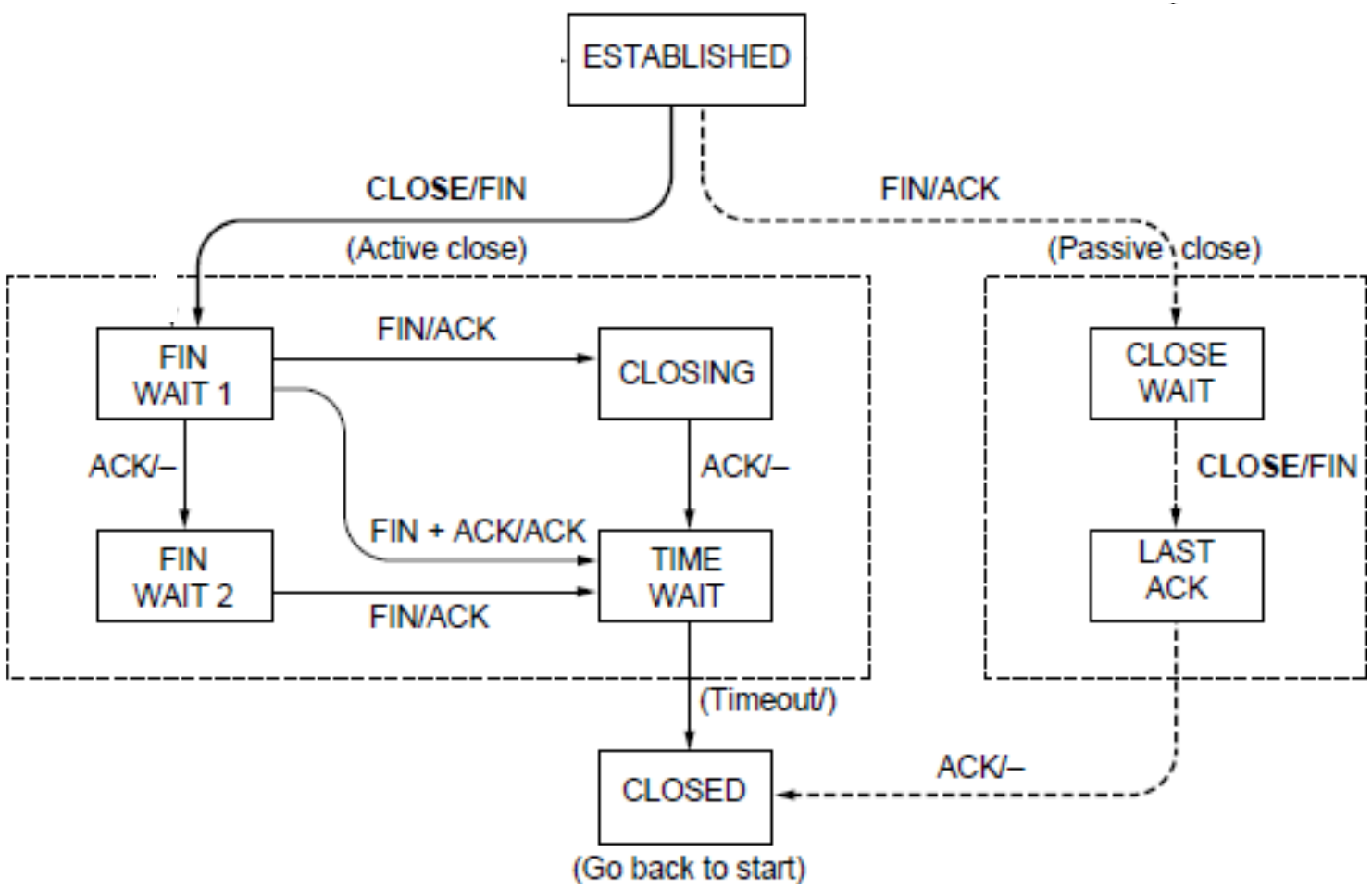
Connection release

- 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



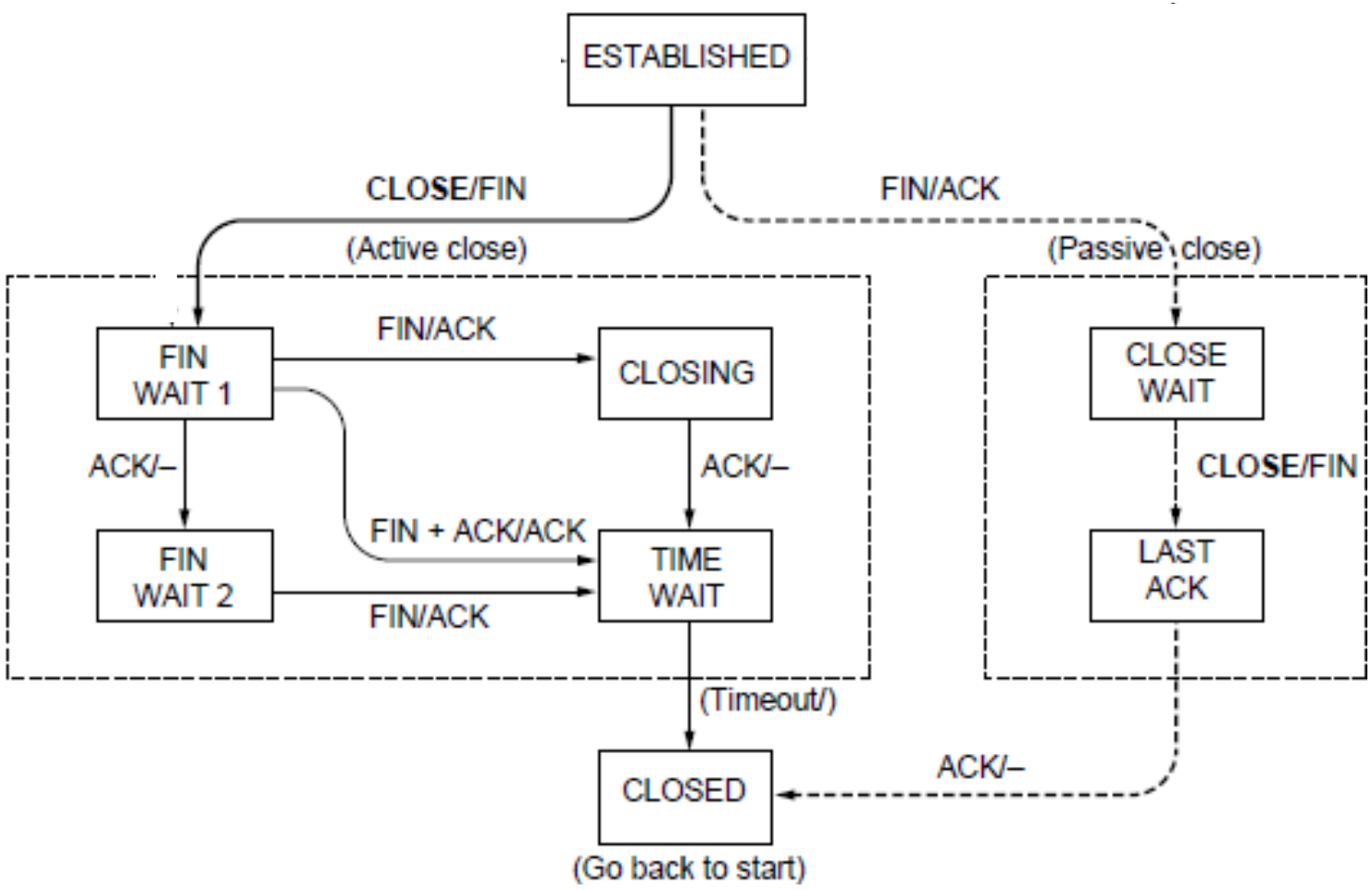
TCP Connection State Machine

Both parties run instances of this state machine



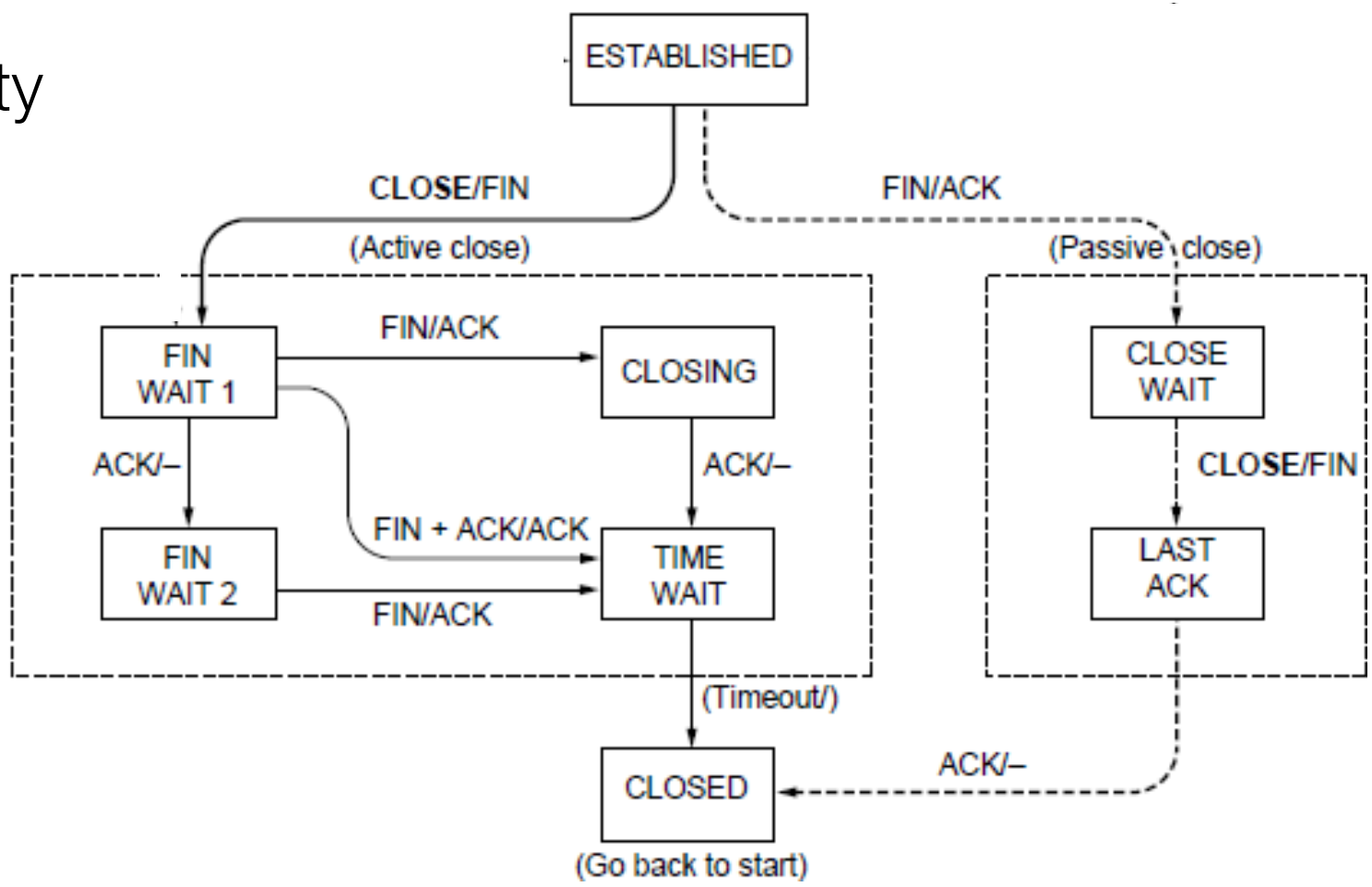
TCP release

- Follow the active party



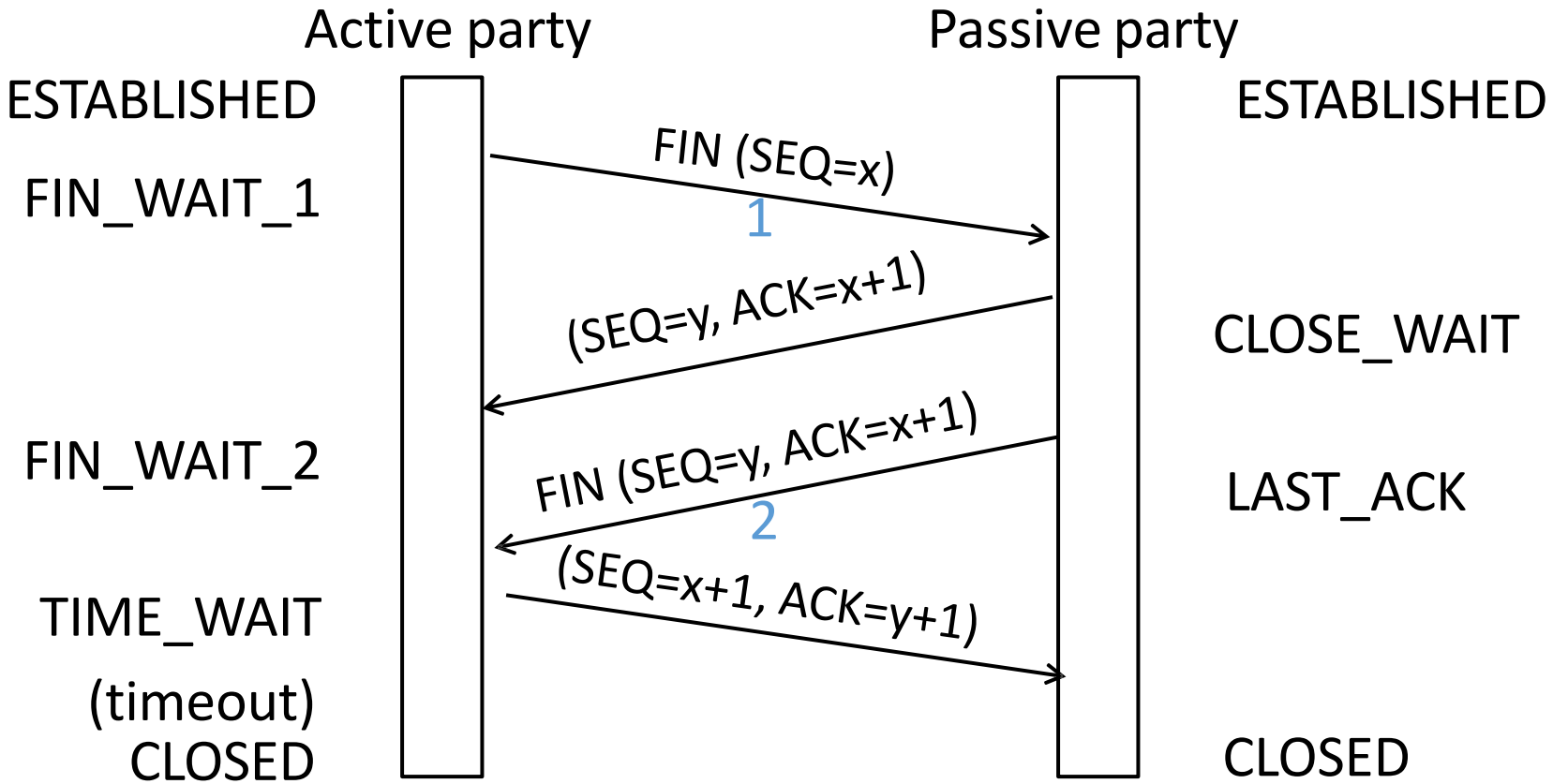
TCP release

- Follow the passive party



TCP release

- 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

Credits

- Some slides are adapted from course slides of CSE 461 in UW