Section 1:

Socket API, Bandwidth & Delay, TCP State Transition

(§1.3.4, 1.5, 5.2.3)

With Tapan and Xieyang

Originally by David Wetherall (djw@)

Outline

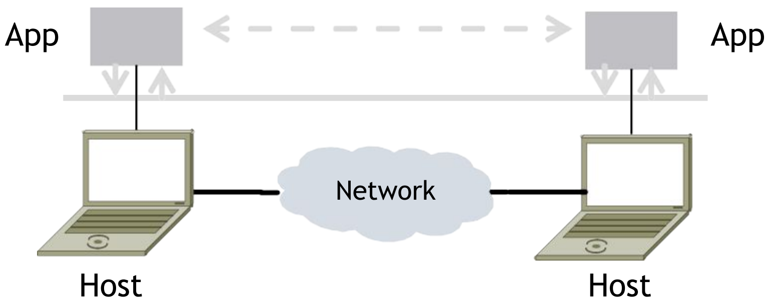
- Administrivia
- Project 1: Socket API
- Hw 1: Bandwidth and delay
- Hw 1: TCP state transition

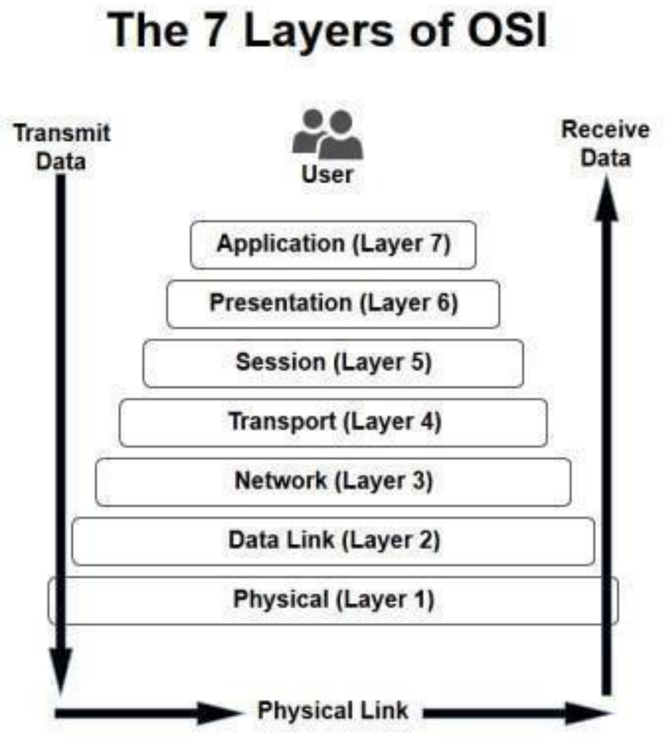
Administrivia

- Different weeks will be led by different TA’s
- Hw 1 due Thursday Apr 13 at 11pm
- Project 1 due Monday Apr 17 at 11pm

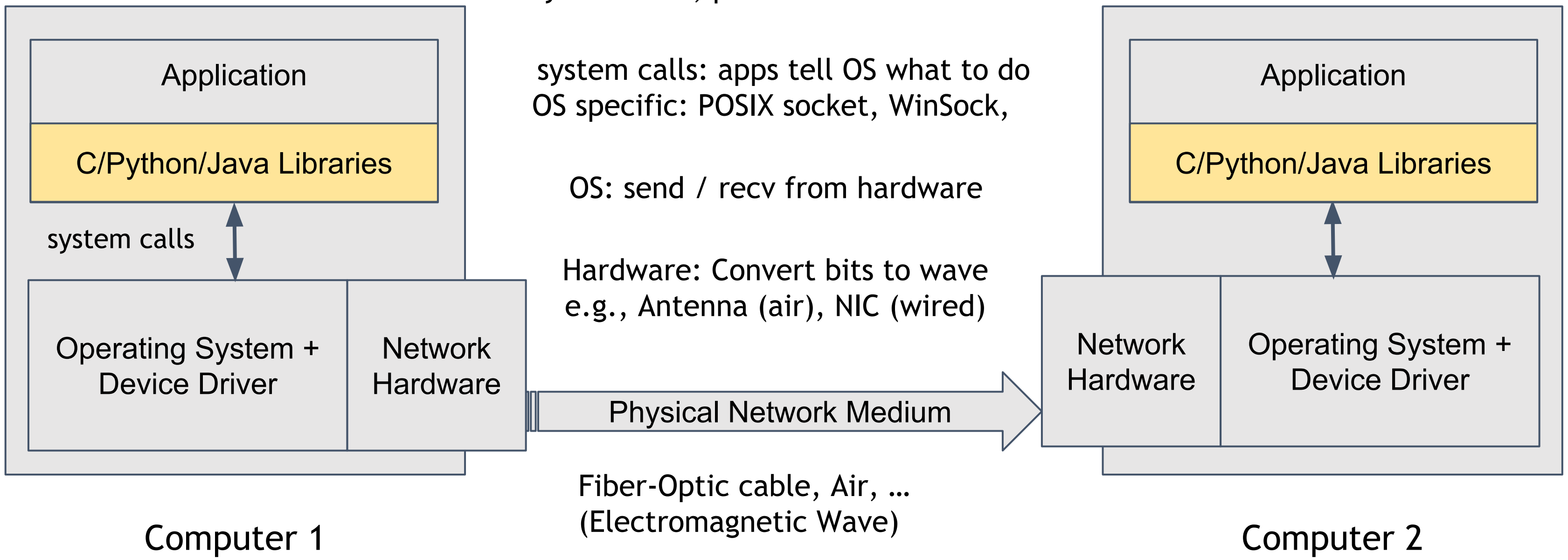
Network-Application Interface

- Application Layer APIs
  - Defines how apps use the network
  - Lets apps talk to each other
  - Hides the other layers of the network





How to build network applications?



libraries: allow application to use
system calls, portable across OSes

system calls: apps tell OS what to do
OS specific: POSIX socket, WinSock,

OS: send / recv from hardware

Hardware: Convert bits to wave
e.g., Antenna (air), NIC (wired)

Fiber-Optic cable, Air, …
(Electromagnetic Wave)

Project 1

- Simple Client
  - Send requests to `attu` server
  - Wait for a reply
  - Extract the information from the reply
  - Continue…
- Simple Server
  - Server handles the Client requests
  - Multi-threaded

Project 1

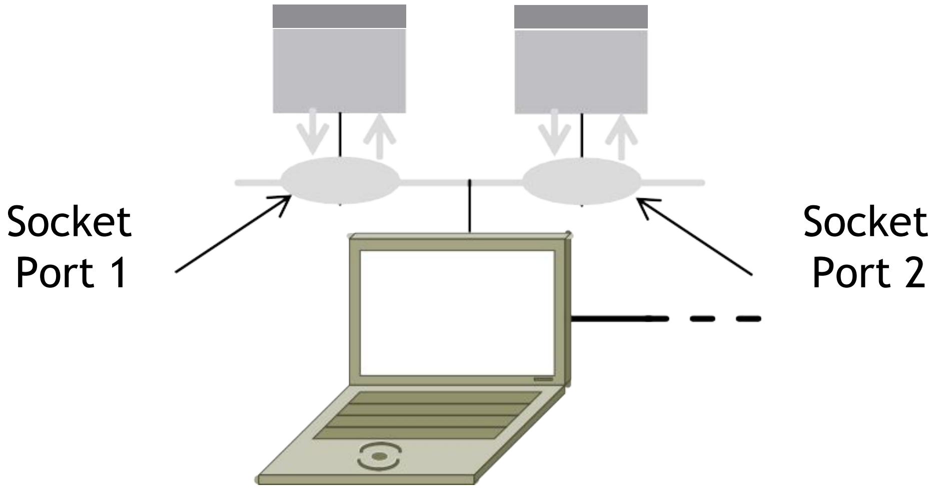
- This is the basis for many apps!
  - File transfer: send name, get file (§6.1.4)
  - Web browsing: send URL, get page
  - Echo: send message, get it back
- Let's see how to write this app …

Socket API

- Simple application-layer abstractions (APIs) to use the network
  - The network service API used to write all Internet applications
  - Part of all major OSes and languages; originally Berkeley (Unix) ~1983
- Two kinds of sockets
  - Streams (TCP): reliably send a stream of bytes
  - Datagrams (UDP): unreliably send separate messages

Socket API (2)

- Sockets let apps attach to the local network at different ports
- Ports are used by OS to distinguish services/apps using internet

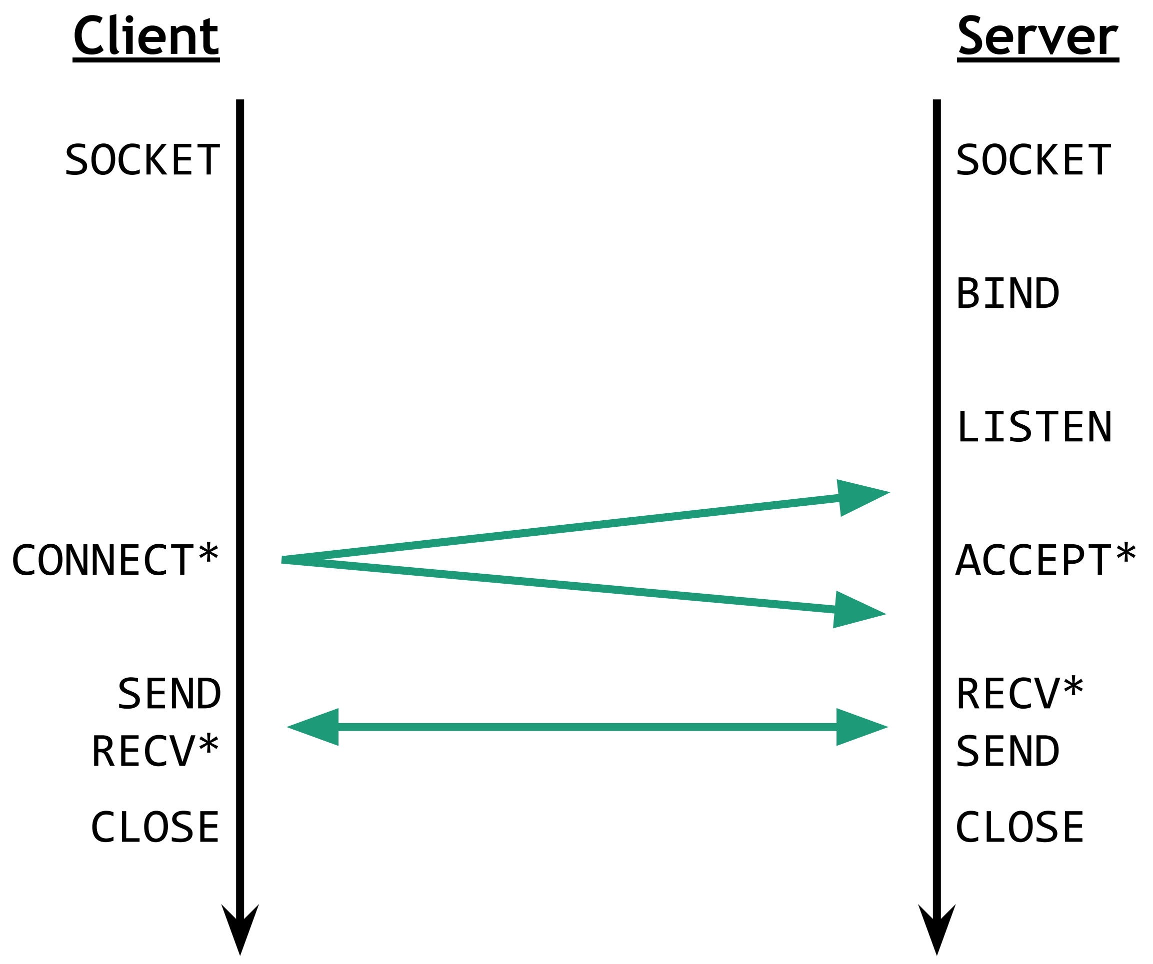


Socket API (3)

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

https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html
https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html

Using Sockets



* Denotes a blocking call

- Waits until action is done
- Use threads to avoid blocking

Client Program (Outline)

```
socket(); // create socket
getaddrinfo();   // server and port name
   // www.example.com:80
connect();   // connect to a server [blocking]
…
send();   // send data
recv();   // await reply [blocking]
… // process reply
close()   // done, disconnect
```

Server Program (Outline)

```
socket(); // create socket
getaddrinfo();  // get info for port on this host
bind();   // associate port with socket
listen(); // start accepting connections
while (true) {
  accept();  // wait for a connection [blocking]
   // returns a new socket
  {// spawn a new thread for each connection
    recv();  // wait for request [blocking]
    … // process reply
    send();  // send reply
    close(); // close connection with client
  }
}
close();  // close the server socket
```

Java Tips

- `ServerSocket` for TCP server socket
- `Socket` for TCP client socket
- `DatagramSocket` for UDP server/client socket

Some other useful utils:

- `ByteBuffer` to manipulate bytes

Python Tips

- `socket.socket(socket.AF_INET, socket.SOCK_DGRAM)` for UDP
- `socket.socket(socket.AF_INET, socket.SOCK_STREAM)` for TCP

Might be useful:

- `socketserver`
- `struct.pack()` and `struct.unpack()` to manipulate bytes

Some guidelines

- Make sure your code runs on `attu`.
  - Python users can only use packages that are available on `attu` (no `pip` unfortunately)
- Small portions of the grade will be awarded to robustness of your server
  - Your server should accept clients outside localhost
  - Close connection when client sends faulty packets or timeout.
    - Padding and payload length; Number of packets; Correct content; etc.
  - Multithreaded?

Outline

- Administrivia
- Project 1: Socket API
- Hw 1: Bandwidth and delay
- Hw 1: TCP state transition

Network performance

- How is network performance evaluated? Try fast.com

Network performance

- How is network performance evaluated? Try fast.com
- Two fundamental metrics
  - Speed: how many bits can be transmitted in a certain period of time
  - Latency: how long it takes for a message to travel one-way/round-trip

Terms explained, all at once

- Bandwidth? Bit rate? Throughput? Goodput?
- Can you sort them in descending order

Terms explained, all at once

- Bandwidth = Bit rate >= Throughput >= Goodput
- All metrics are used to measure network speed, all are in bits/second

- Bandwidth vs Throughput
  - The **capacity** of the network vs the **utilized capacity** of the network
  - E.g., an Ethernet link can transmit at 1 Gbps (bandwidth). If the sender sleeps for 1 ms after each sending period of 1 ms, the average throughput will be 0.5 Gbps.

Terms explained, all at once

- Bandwidth = Bit rate >= Throughput >= Goodput
- All metrics are used to measure network speed, all are in bits/second

- Throughput vs Goodput
  - **All data** (regardless of useful or not) counts vs **only useful data** counts.
  - E.g., a TCP connection last 1 second, during which a 10 MB file was transmitted. The total amount of data transmitted was 15 MB (including packet headers, retransmissions, etc.) What is the throughput and what is the goodput?

Terms explained, all at once

- Latency/Delay? Round trip time(RTT)? Propagation Delay?

Terms explained, all at once

- Latency/Delay: how long it takes for a message to travel one-way
  - From the sender starts sending the first bit, to the receiver gets the **last** bit.
- Round-trip time (RTT): how long it takes for a message from one end of the network to the other and get back.
- Propagation delay: how long it takes for a **signal** to propagate from one end of a link to the other.

Breaking down the latency

- t0 = 0, the sender starts sending the first bit
- t1 = size/bandwidth, the sender finishes sending the last bit
- t2 = t1 + propagation delay + queueing delay, the last bit gets to receiver

Eq1 (Important to your hw 1):

```
Latency = Propagation + Transmit + Queue
Propagation =  Distance/SpeedOfLight
Transmit = Size/Bandwidth
```

Breaking down the latency

- Tips:
  - Queue=0 if not mentioned
  - Propagation=RTT/2 if no info other than RTT is provided
  - Propagation delay limited if Propagation > Transmit, otherwise throughput limited

Eq1 (Important to your hw 1):

```
Latency = Propagation + Transmit + Queue
Propagation =  Distance/SpeedOfLight
Transmit = Size/Bandwidth
```

Exercise

Calculate the total time required to transfer a 1000-KB file, assuming:

- RTT=60 ms
- Bandwidth=8 Mbps
- Needs 2*RTT of handshake time before sending the data

Exercise

Calculate the total time required to transfer a 1000-KB file, assuming:

- RTT=60 ms
- Bandwidth=8 Mbps
- Needs 2*RTT of handshake time before sending the data

2*RTT + 1000*8*10^3 bits / (8*10^6 bits/second) + 0.5*RTT = 1.15 seconds

Outline

- Administrivia
- Project 1: Socket API
- Hw 1: Bandwidth and delay
- Hw 1: TCP state transition

Finite-state machine

- Is a mathematical model of computation.
  - Can be in exactly one of a finite number of **states** at any given time.
  - Can change from one state to another in response to some inputs; the change from one state to another is called a **transition**.

The TCP FSM

- Each box denotes a state that one end of a TCP connection can find itself in.
  a. Both ends have independent states!
- Each arc denotes events that can trigger a state transition.
  a. A segment received from the peer (SYN/ACK/FIN…)
  b. An operation of the local application (*open/close…*)

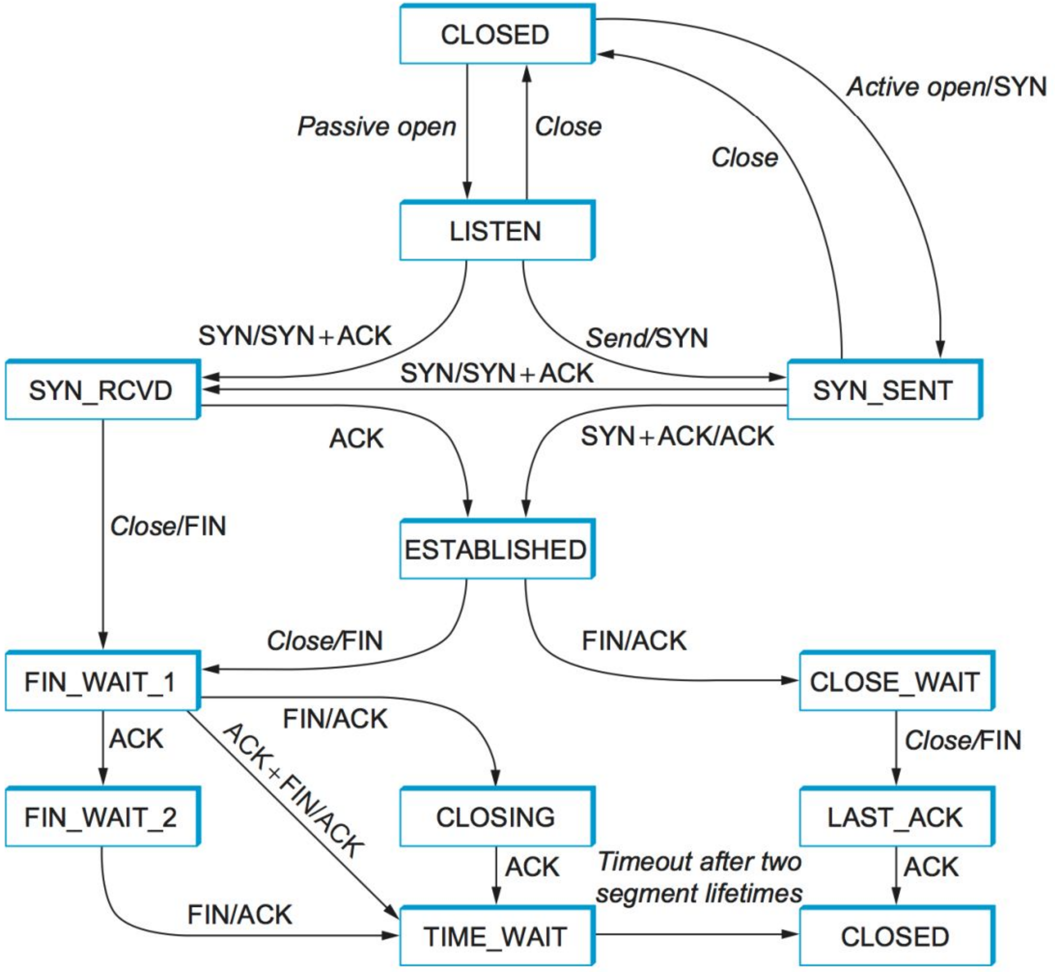
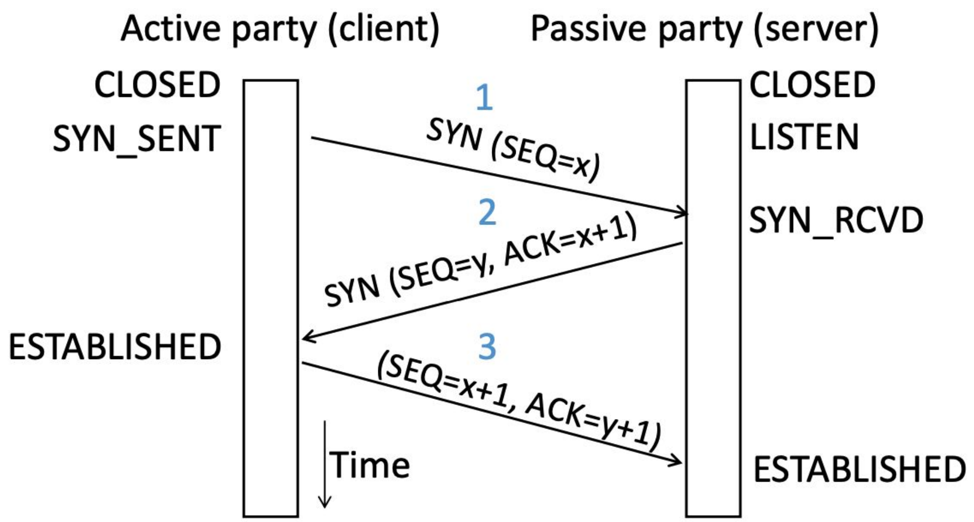
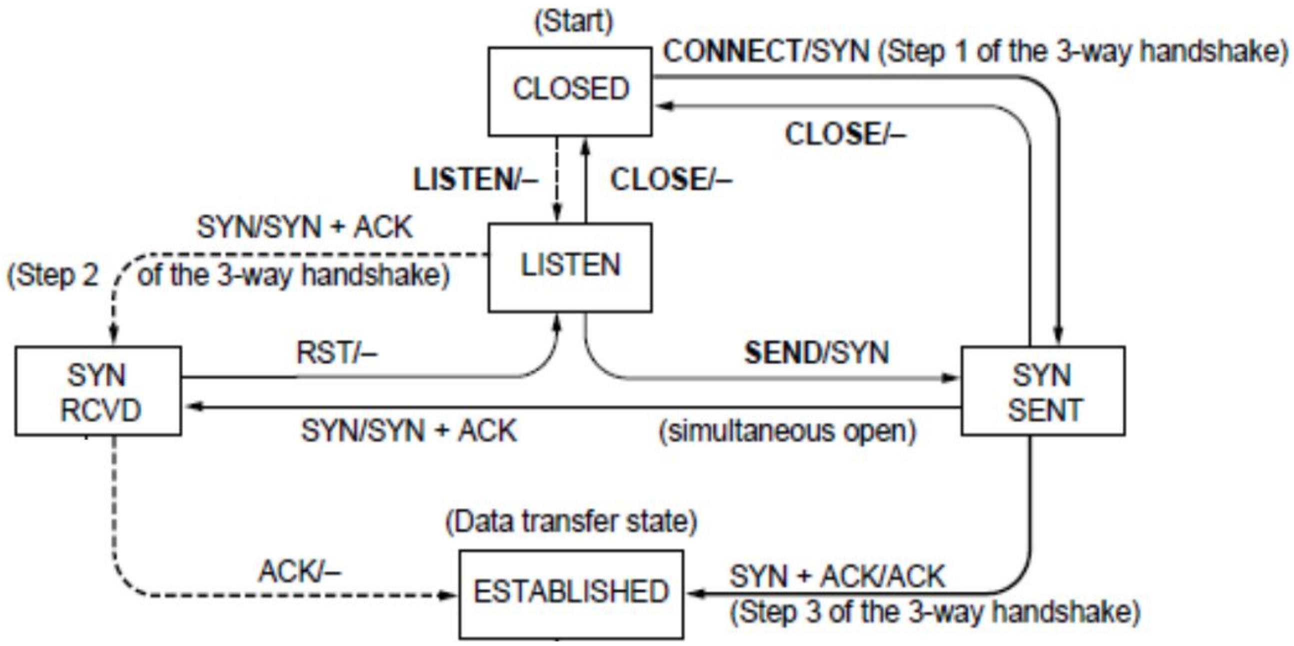


Figure 5.7.: TCP state-transition diagram.

How to use this diagram?

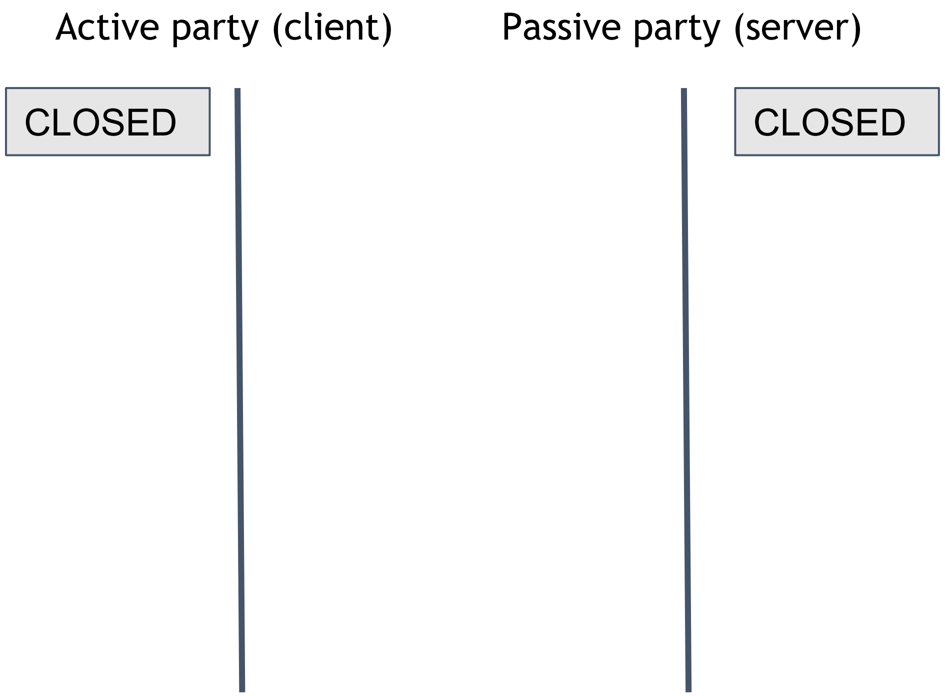
Given the sequence of events, you can derive the sequence of states of both sides.

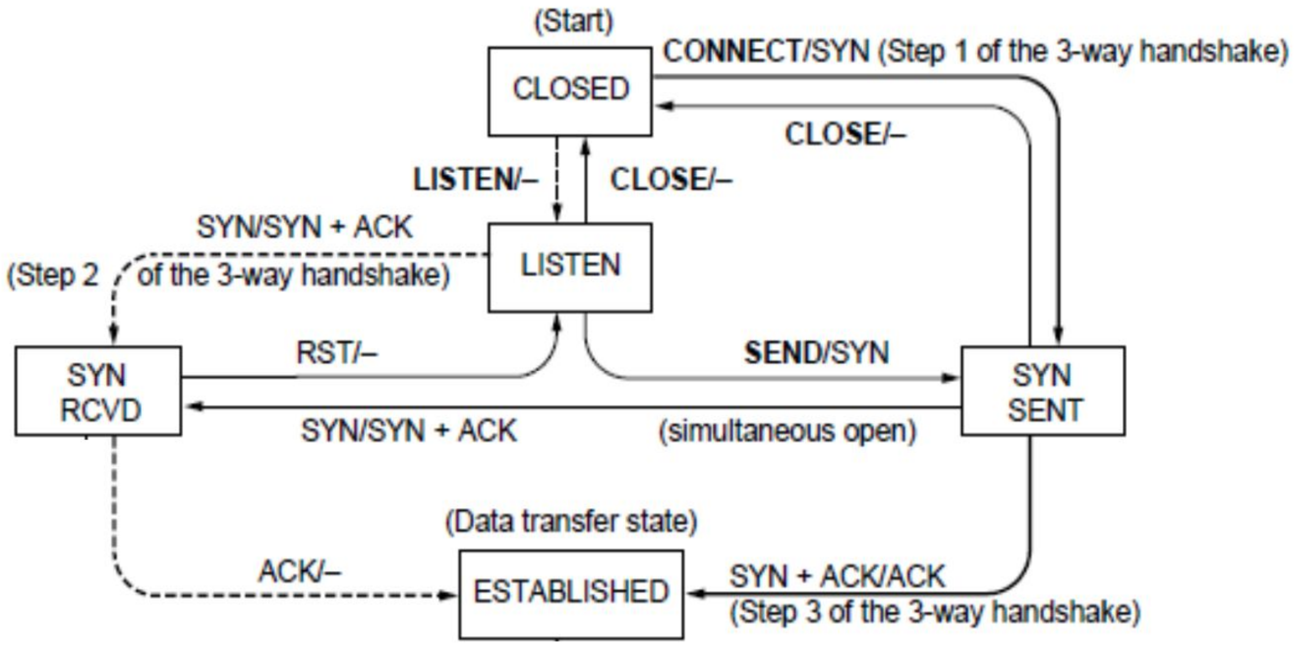




Exercise: 3-way handshake

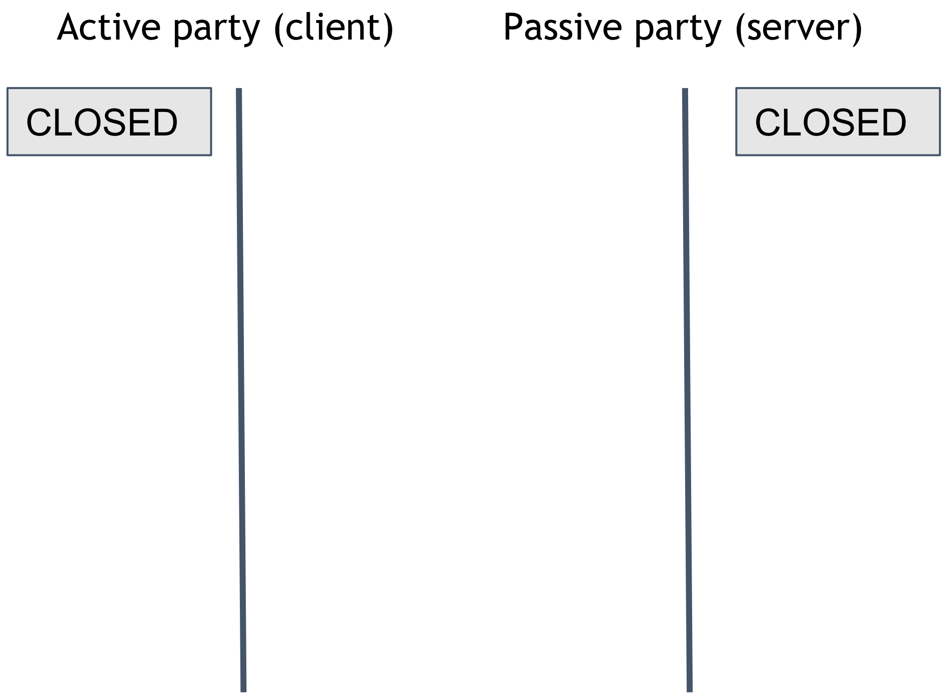
1. Identify two parties, draw timelines for both.

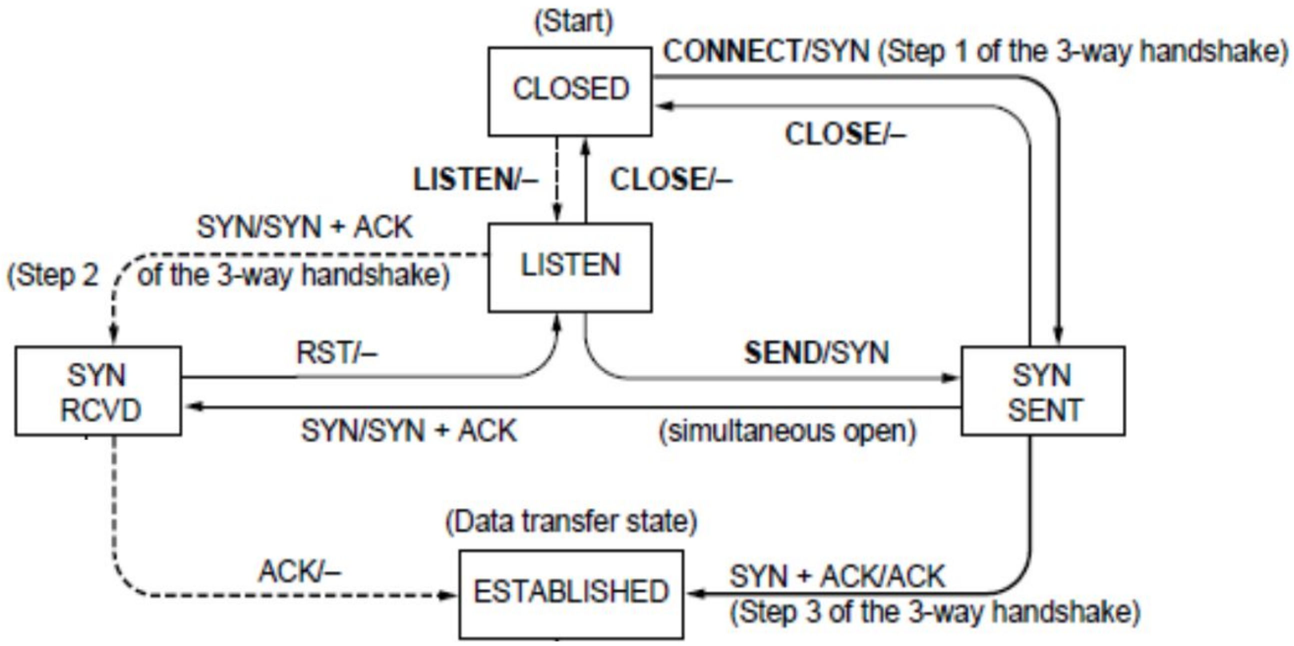




Exercise: 3-way handshake

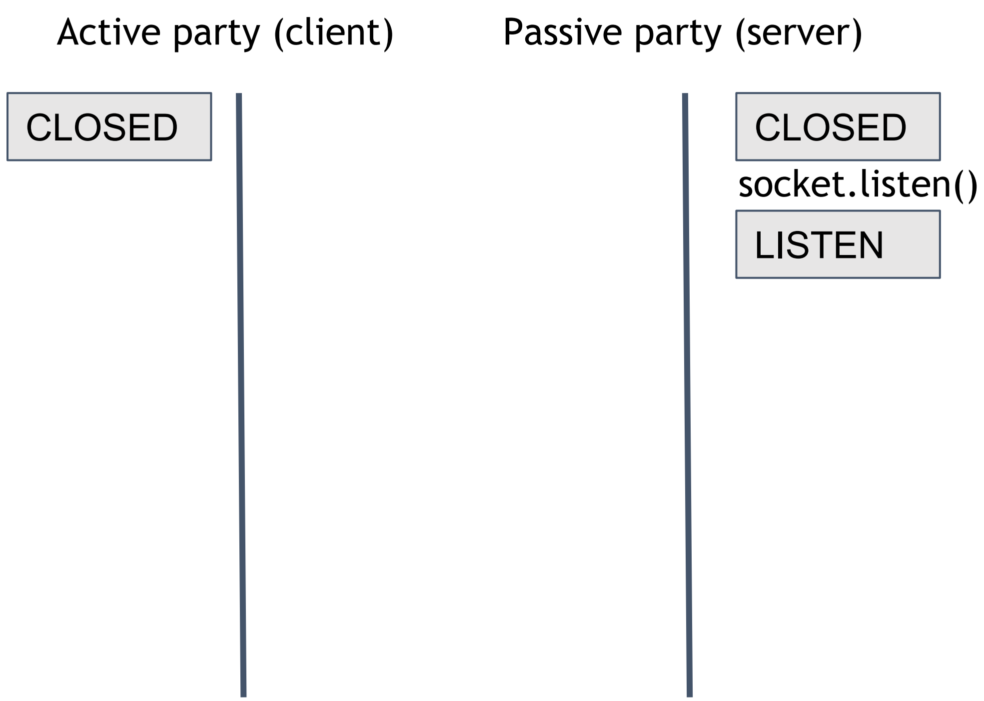
2. Initial states of both sides.

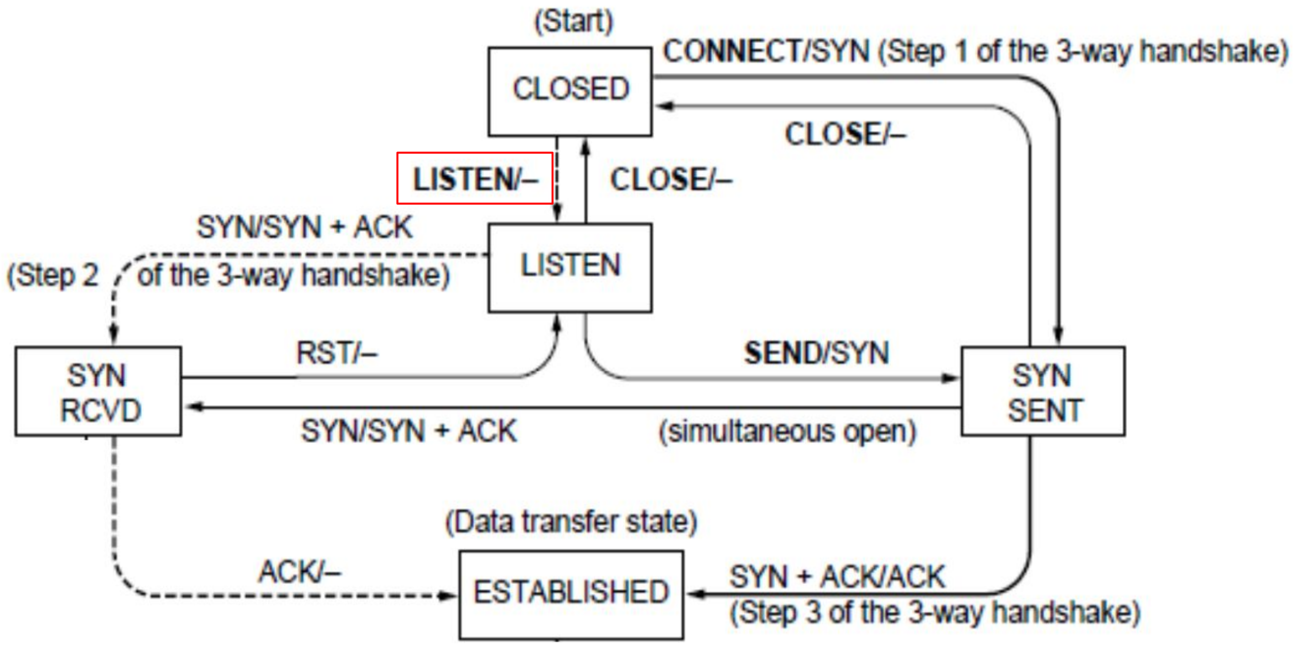




Exercise: 3-way handshake

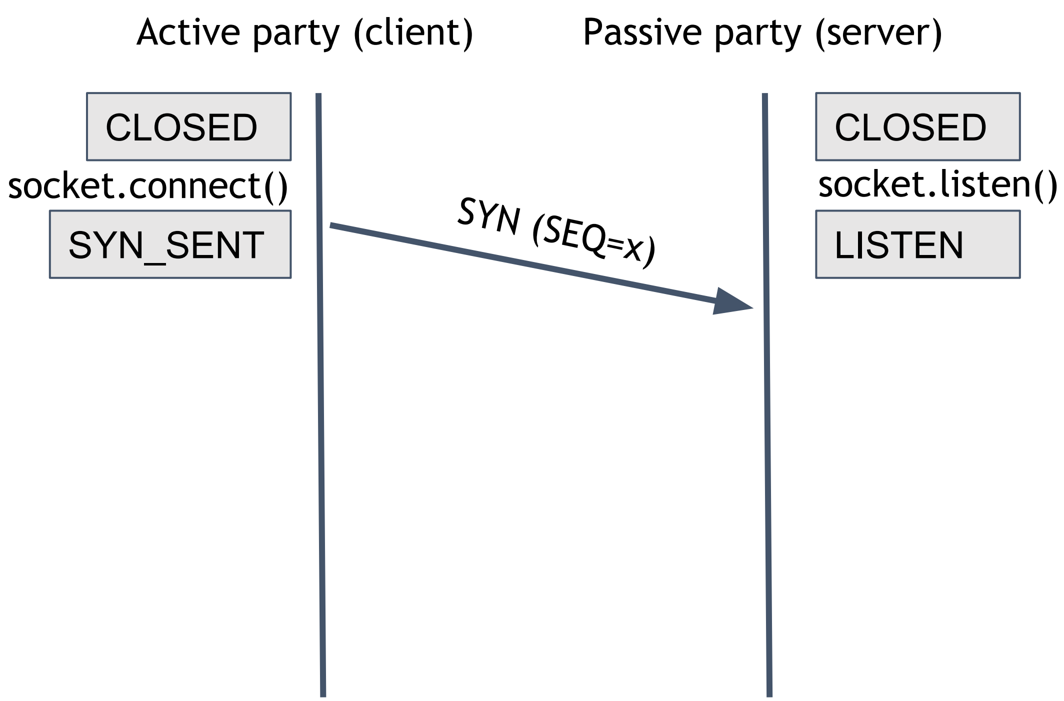
3. The server starts listening.

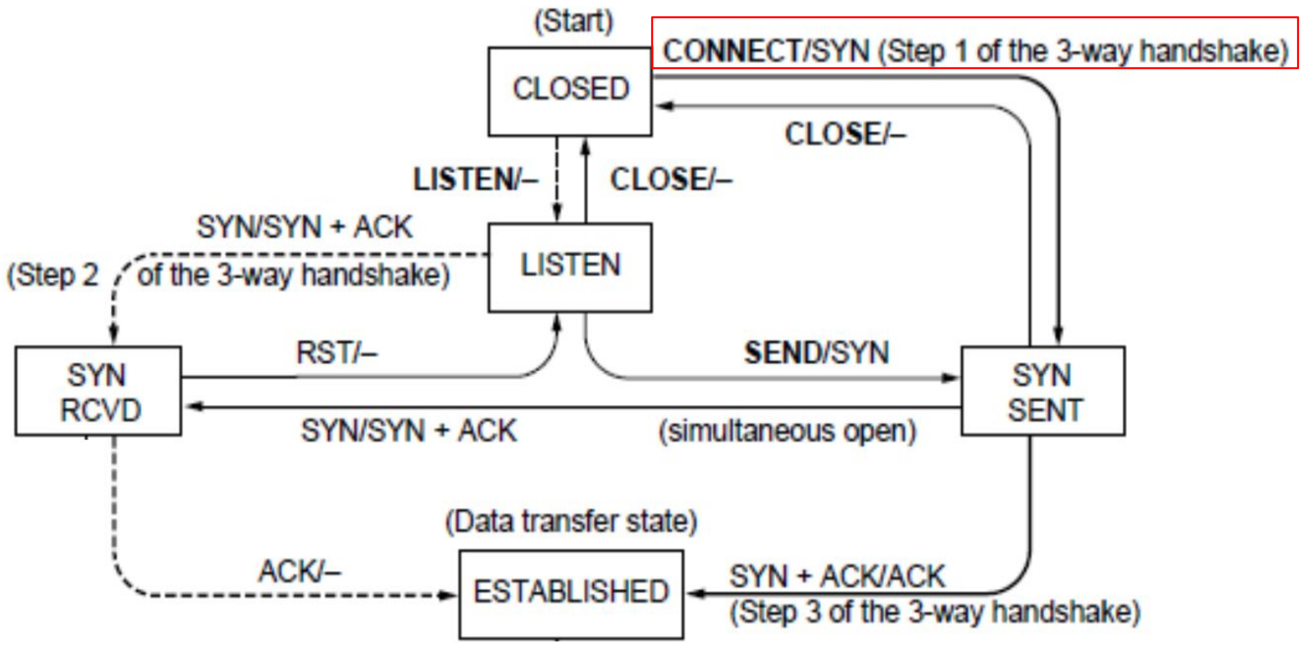




Exercise: 3-way handshake

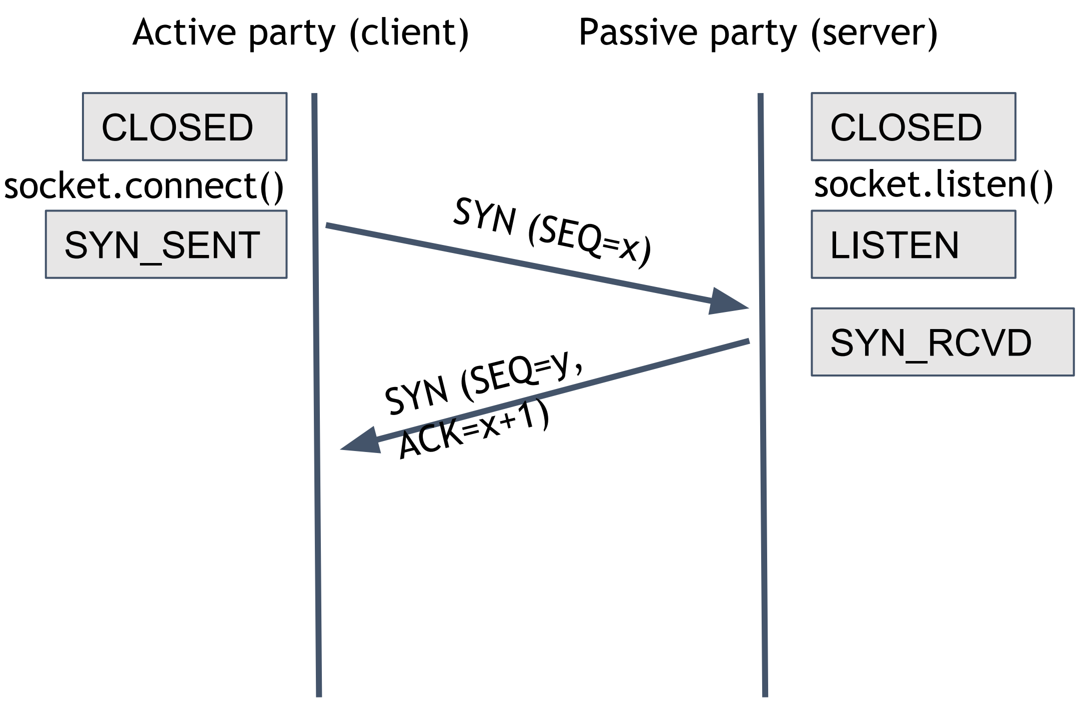
4. The client initiates the connection with a SYN segment

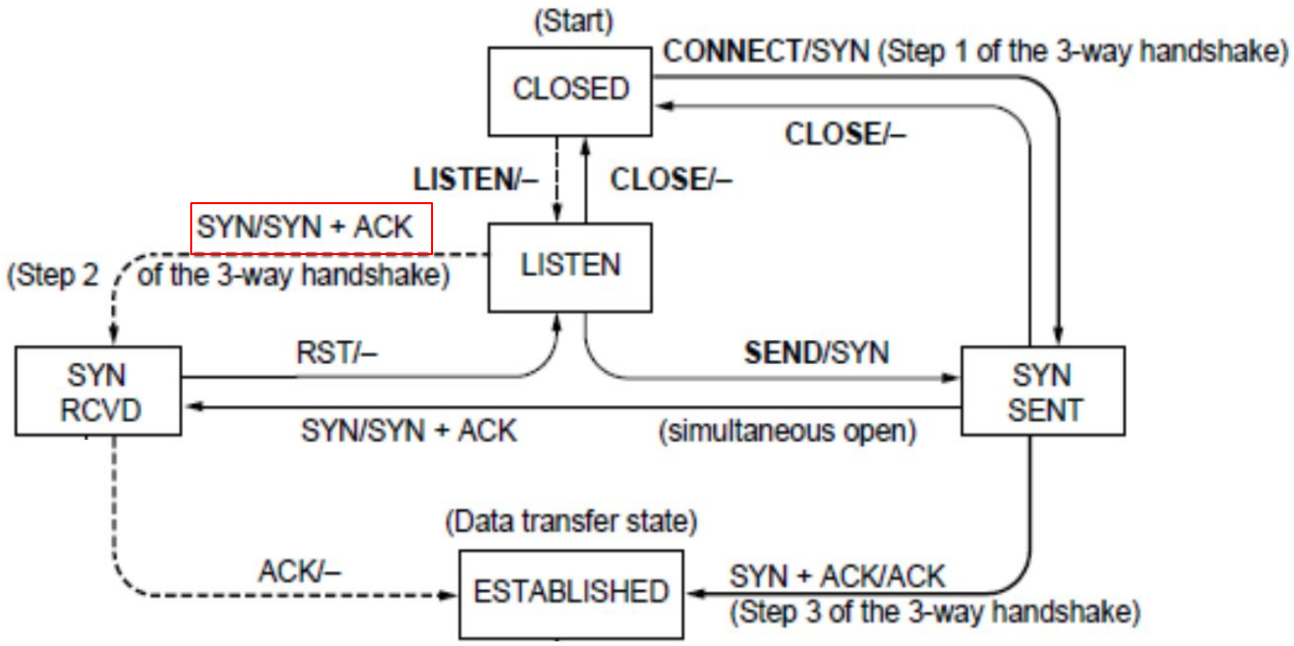




Exercise: 3-way handshake

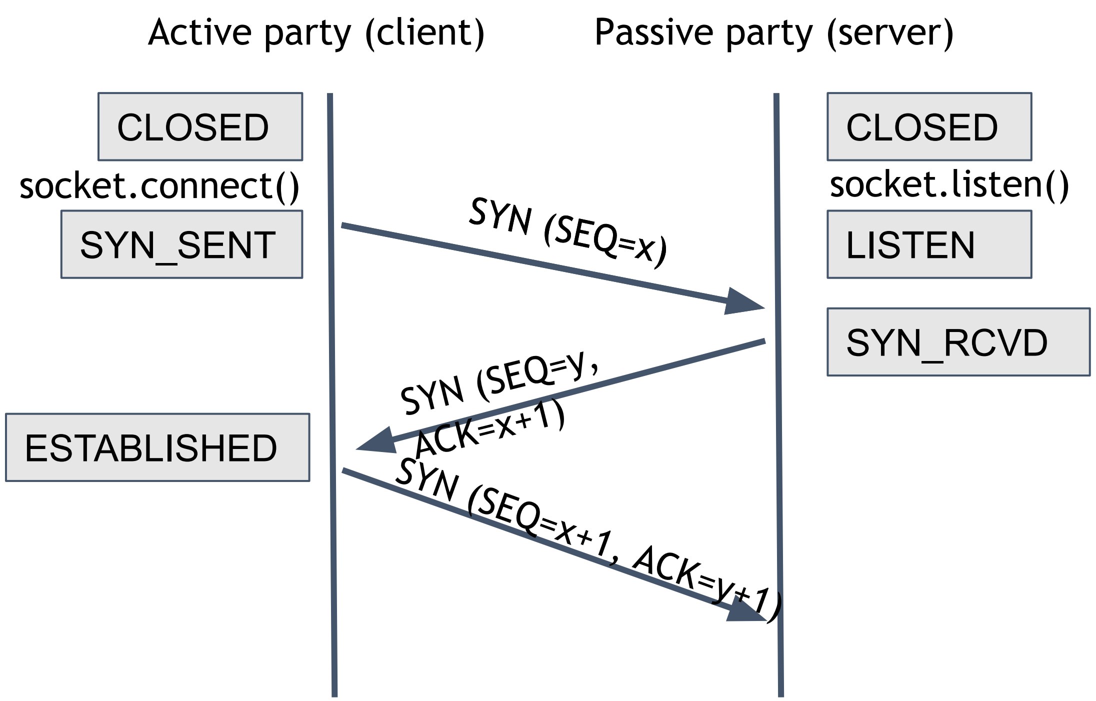
5. The server received the SYN segment, responses with a SYN/ACK

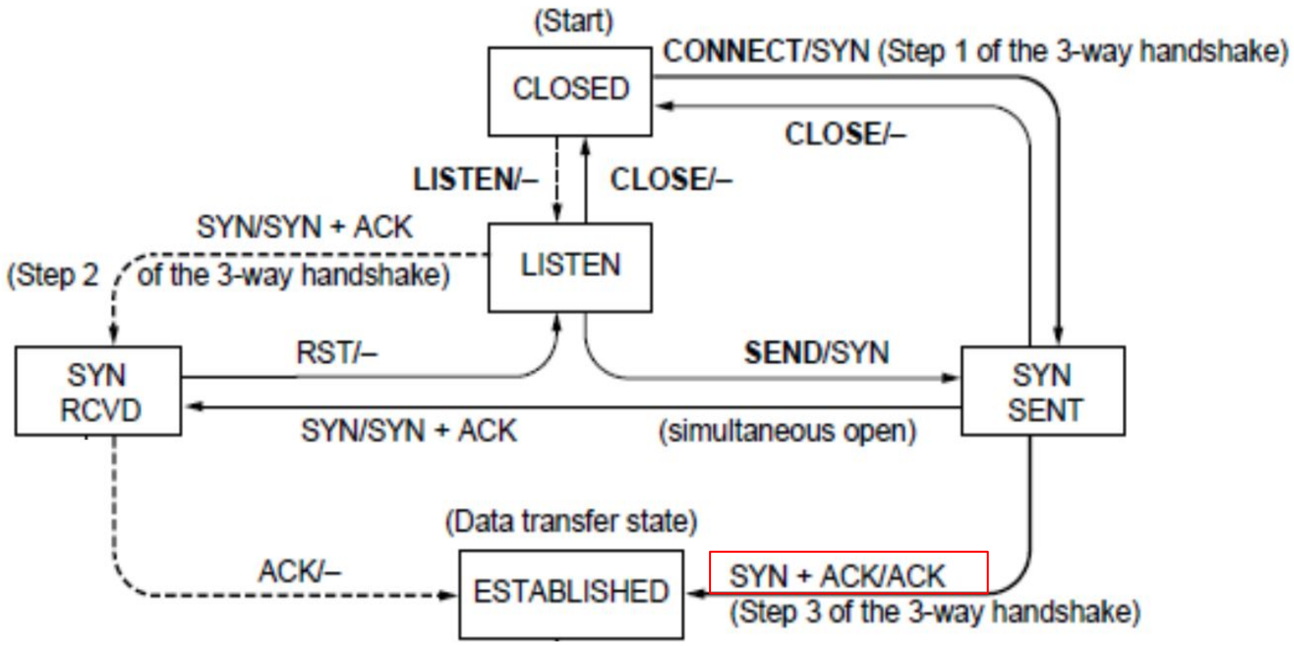




Exercise: 3-way handshake

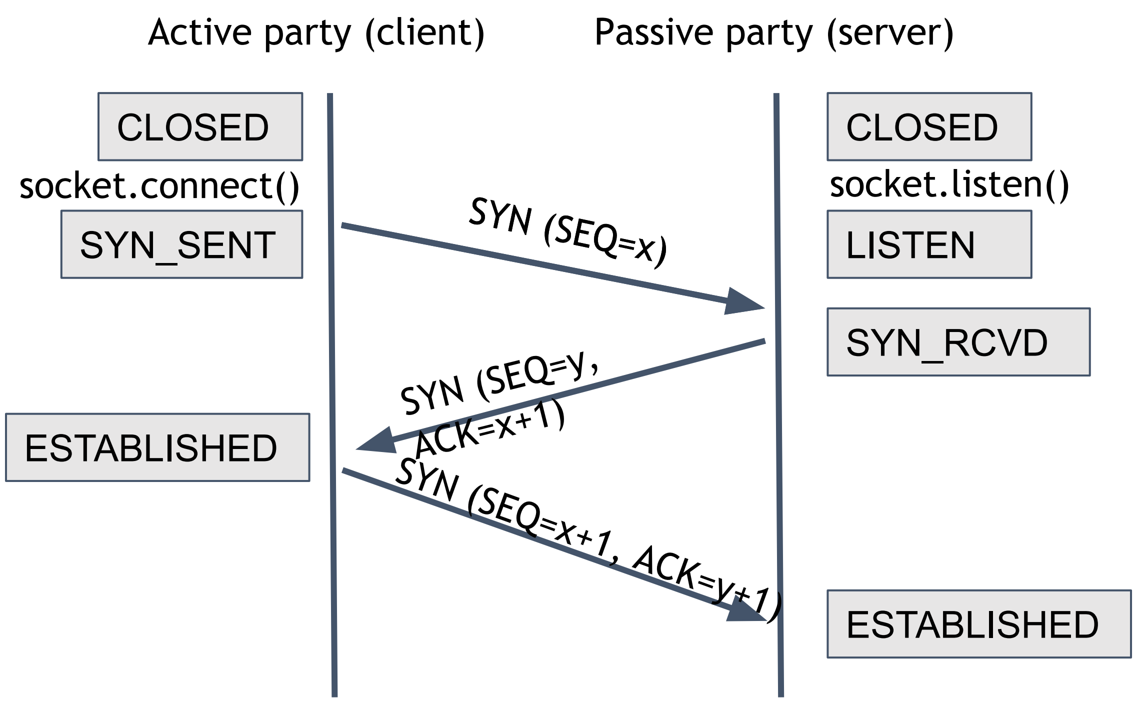
6. The client received the SYN/ACK segment, responses with an ACK

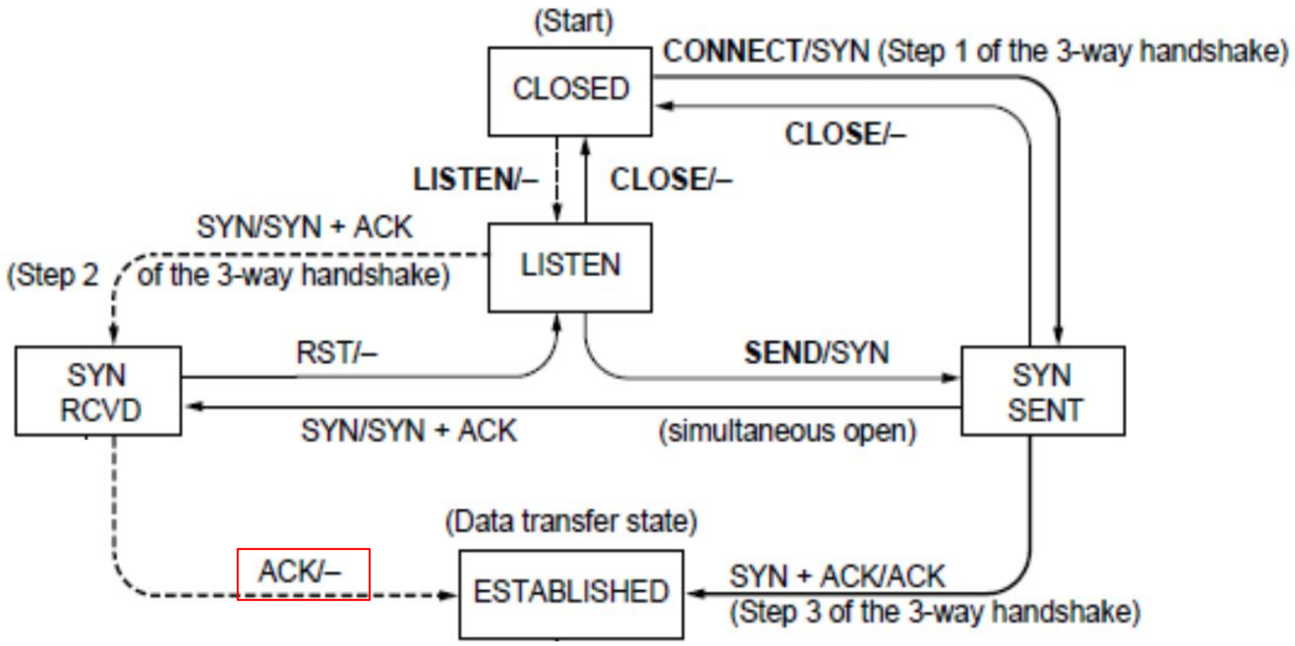




Exercise: 3-way handshake

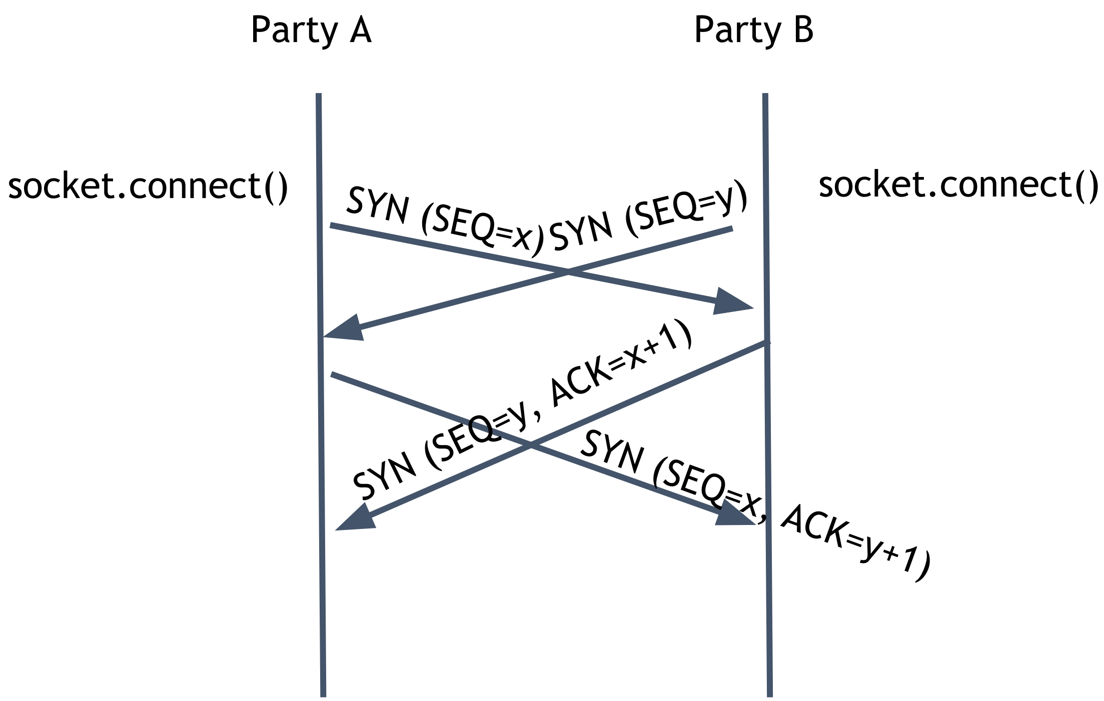
7. The server received the ACK, done!

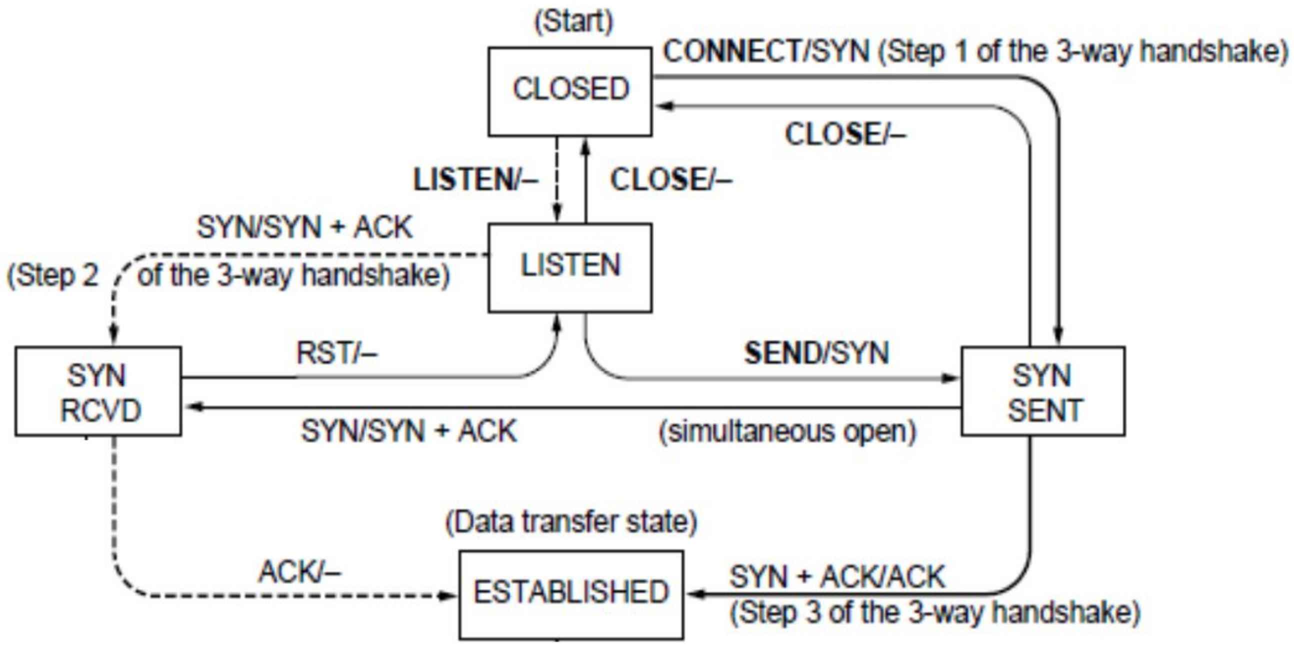




Bonus: simultaneous open

Sequence of events. Work out the sequence of states on your own :)





Thanks for coming!

Any questions?