Discussion 1:

TCP Sockets and Mininet VM

By Joey Buiteweg

(slide idea credits to Yiwen Zhang and Ben Reeves)

Greetings!

I took 489 in F18, this is my second year teaching the course

Graduated in Computer Engineering at Umich, CSE SUGS Masters

I played Tenor Sax in the Marching Band for 4 years

Industry experience in network security

^By no means an expert, but I know some things

DEI and Imposter Syndrome

I care very much about making you all feel included and welcomed here

Some of you may have prior knowledge of the concepts learned in this class

- Please use this to help, not intimidate your peers

None of you are imposters, you all belong here

- Much of networking and software in general isn’t intuitive
- We’re here to help
- You can do this!

Learning Objectives

By the end of this discussion we will:

- Know how to write TCP socket programs
- Be able to use the Mininet VM

TCP Socket Background

TCP Socket Background

What is a syscall? What’s an example?

TCP Socket Background

What is a syscall? What's an example?

- Ex: fopen(), fclose(), write() <- called by printf(), read()
- Allows us to ask the kernel/OS to do things (related to hardware)

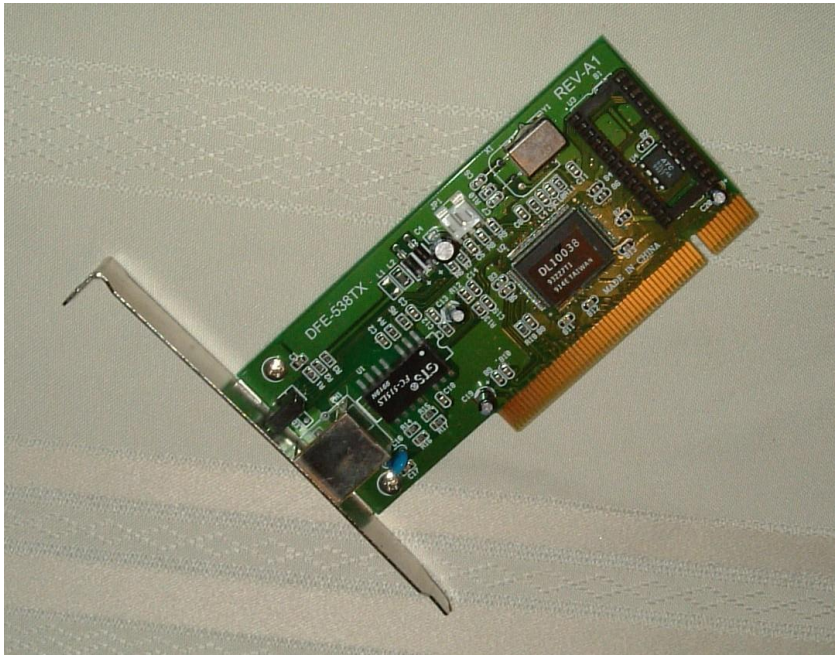
TCP Socket Background

What is a syscall? What's an example?

- Ex: fopen(), fclose(), write() <- called by printf(), read()
- Allows us to ask the kernel/OS to do things (related to hardware)

What hardware are we interacting with for networking/the internet?

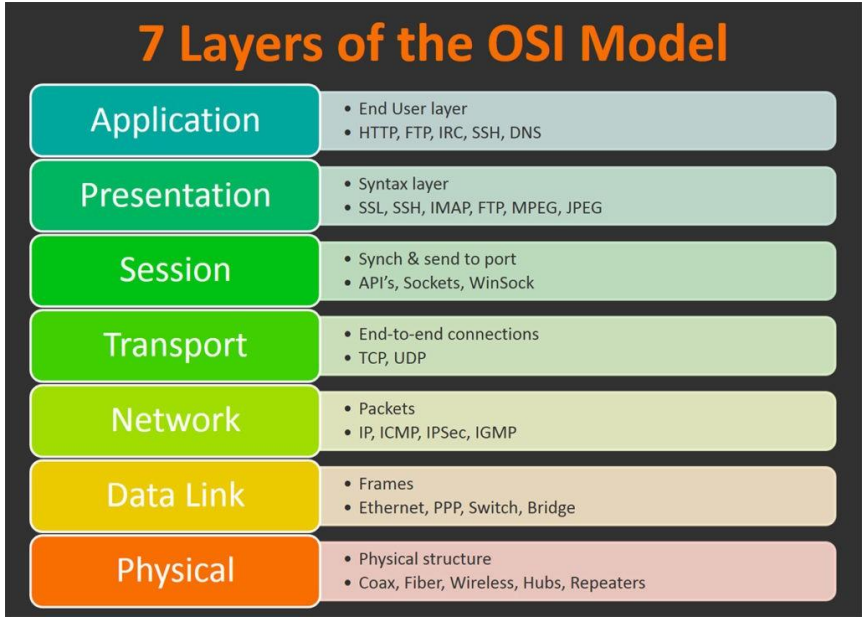
The NIC (Network Interface Controller)



https://www.tbray.org/ongoing/When/200x/2003/02/18/-big/NIC.jpg.html

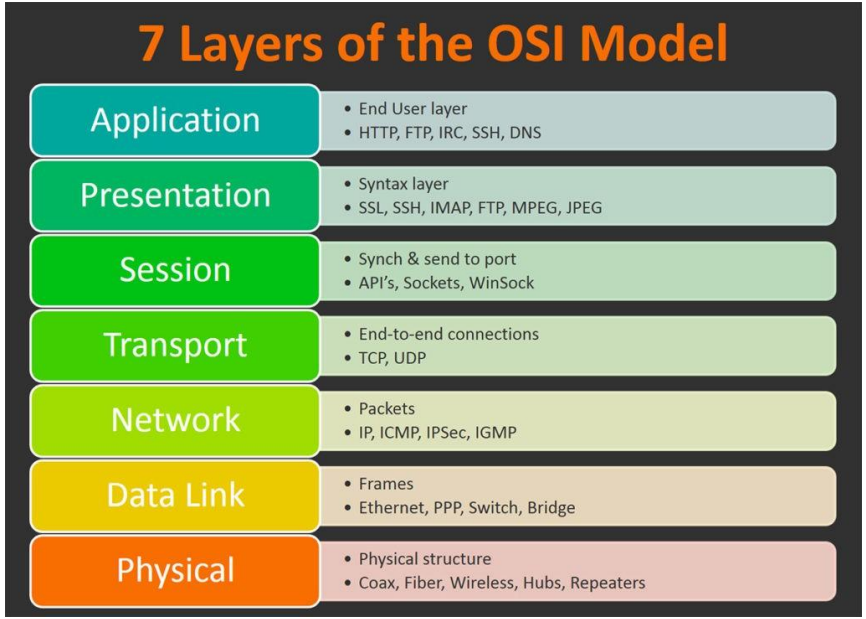
OSI (Open System Interconnect) Model

Abstracted by (for socket programming)



The NIC/Motherboard/CPU/Hardware

OSI (Open System Interconnect) Model

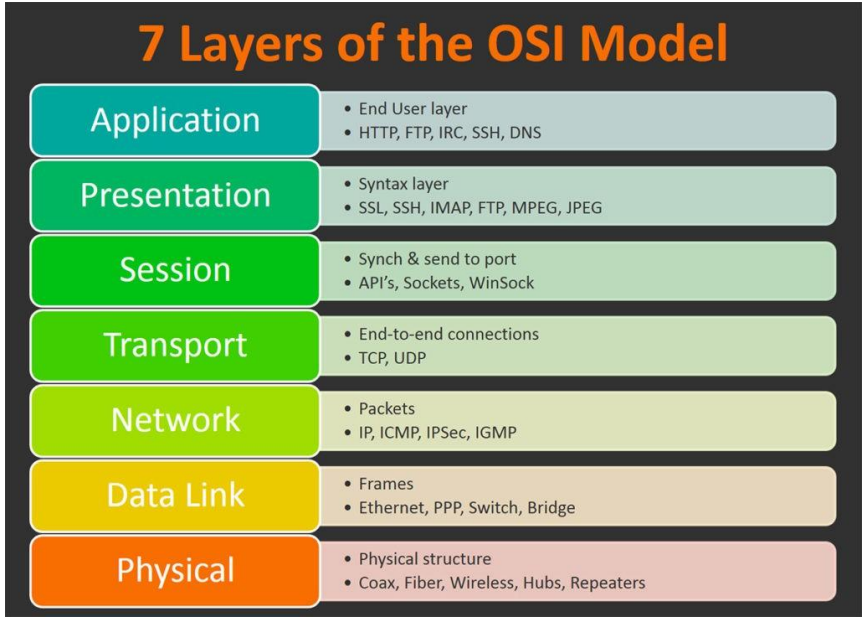


Abstracted by (for socket programming)

Kernel

The NIC/Motherboard/CPU/Hardware

OSI (Open System Interconnect) Model



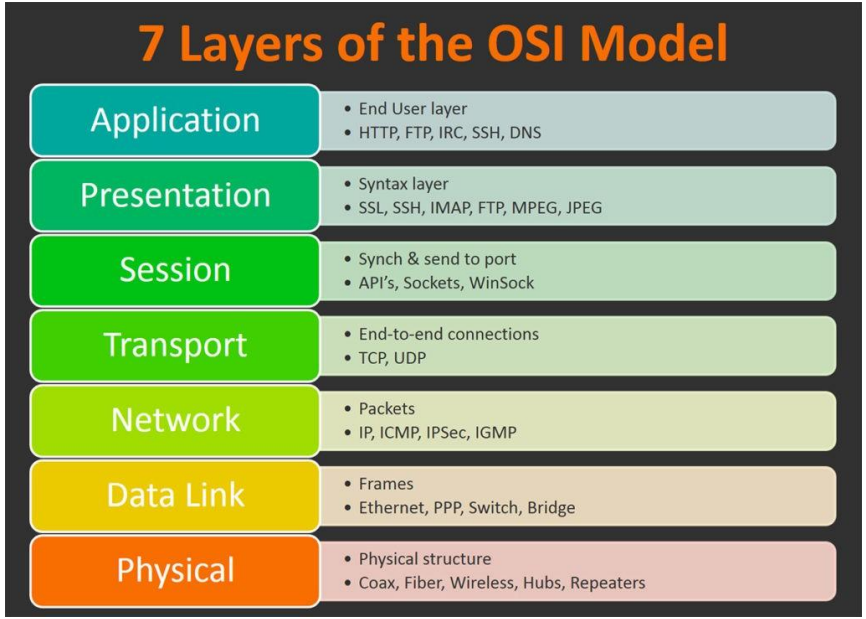
Abstracted by (for socket programming)

Kernel or use of raw sockets

Kernel

The NIC/Motherboard/CPU/Hardware

OSI (Open System Interconnect) Model



Abstracted by (for socket programming)

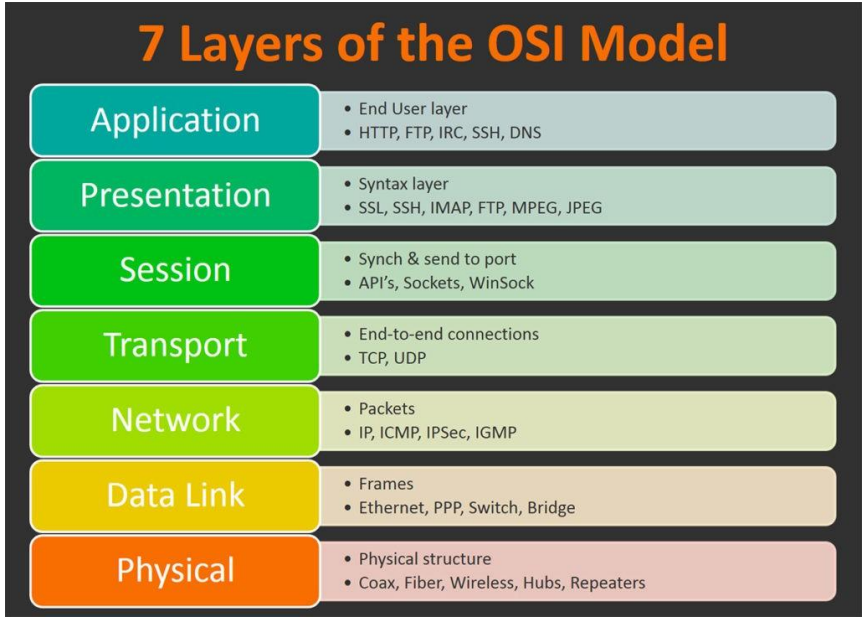
TCP/UDP Sockets C style API (to the kernel)

Kernel or use of raw sockets

Kernel

The NIC/Motherboard/CPU/Hardware

OSI (Open System Interconnect) Model



Abstracted by (for socket programming)

US!

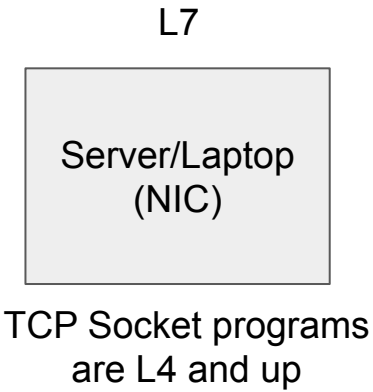
TCP/UDP Sockets C style API (to the kernel)

Kernel or use of raw sockets

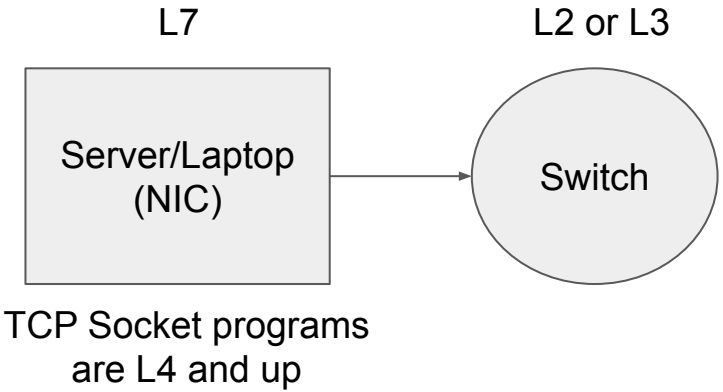
Kernel

The NIC/Motherboard/CPU/Hardware

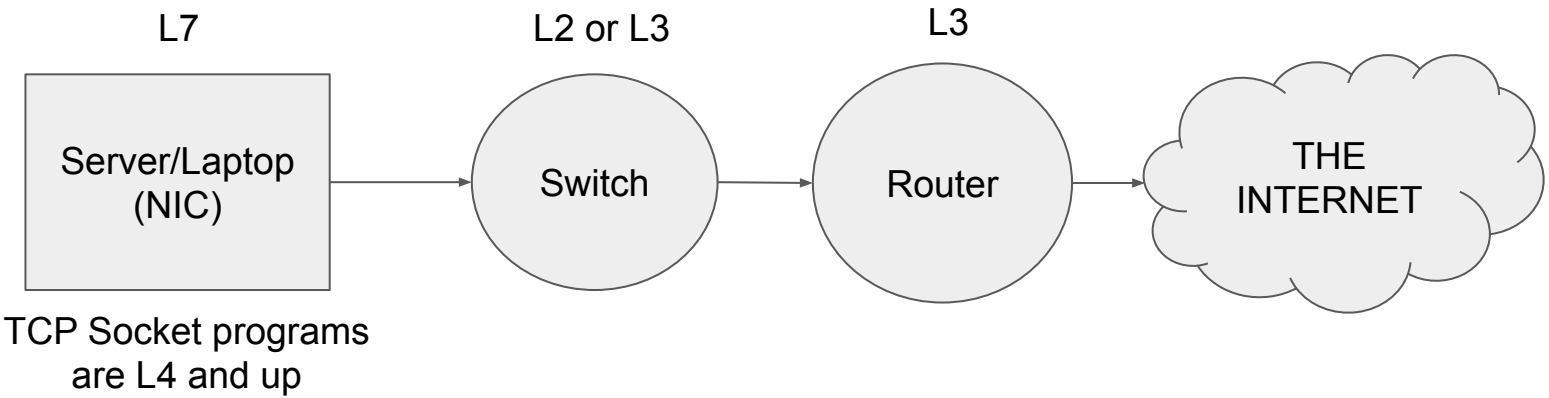
Greater Scope



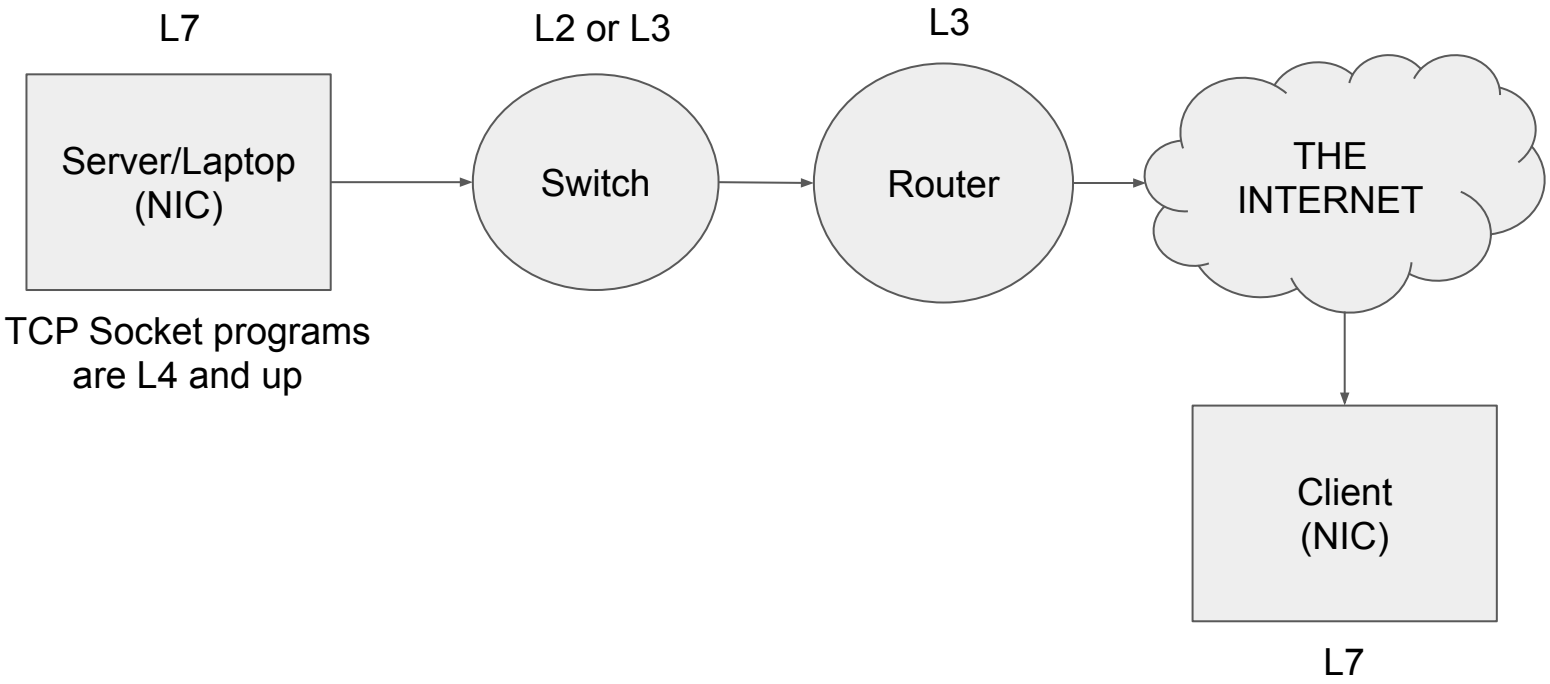
Greater Scope



Greater Scope



Greater Scope



TCP Socket API

TCP Socket API

C style

- OLD and hard to read at first
- Operations done on a variable, not method calls
- Guarantees in order byte stream of data (full duplex, so two byte streams)

TCP Socket API

C style

- OLD and hard to read at first
- Operations done on a variable, not method calls
- Guarantees in order byte stream of data (full duplex, so two byte streams)

```
// Don't get to do this
Socket s = Socket();
s.setsockopt(...);
s.bind(...);
```

TCP Socket API

C style

- OLD and hard to read at first
- Operations done on a variable, not method calls
- Guarantees in order byte stream of data (full duplex, so two byte streams)

```
// Don't get to do this
Socket s = Socket();
s.setsockopt(...);
s.bind(...);
```

```
// Instead have to do this
int sockfd = socket();
setsockopt(sockfd, ...);
bind(sockfd, ...);
```

TCP Socket API

C style

- OLD and hard to read at first
- Operations done on a variable, not method calls
- Guarantees in order byte stream of data (full **duplex**, so two byte streams)

```
// Don't get to do this
Socket s = Socket();
s.setsockopt(...);
s.bind(...);
```

```
// Instead have to do this
int sockfd = socket();
setsockopt(sockfd, ...);
bind(sockfd, ...);
```

Duplex

Client -> Server

0xDE	0xAD	0xBE	0xEF	0x00	0x00

Server -> Client

0xBE	0xEF	0xDE	0xAD	0xFF	0xFF

TWO SEPARATE BYTE STREAMS

TCP Socket API

```
// Server functions (in order)
socket()

setsockopt()

bind()

listen()

accept()

recv() or send()

close()
```

TCP Socket API

```
// Server functions (in order)
socket()

setsockopt()

bind()

listen()

accept()

recv() or send()

close()
```

```
// Client functions (in order)
socket()

connect() // does TCP handshake

recv() or send()

close()
```

TCP Socket Telephone Analogy

Can view these function calls with analogy to a telephone conversation

Server Functions: socket()

Creates a socket. Like buying a telephone.

```
int socket(int domain, int type, int protocol);

// Example

sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

Server Functions: setsockopt()

Allows port number to be reused. Similar to configuring the telephone

Prevents “port is already in use” error. PLEASE USE THIS.

```
setsockopt(int sockfd, int level, ...)

// Example

int yes = 1;
setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &yes,
           sizeof(yes));
```

Server Functions: bind()

Binds a socket to an address. Like registering a phone number for the telephone.

```
int bind(int sockfd, const struct sockaddr * addr,
         socklent_t addrlen);

// Example
struct sockaddr_in addr;
memset(&addr, 0, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY; //INADDRY_ANY == 0.0.0.0
addr.sin_port = htons(port);
bind(sockfd, (struct sockaddr *) & addr, sizeof(addr));
```

AF_INET

```
addr.sin_family = AF_INET; // Use IPv4

addr.sin_family = AF_INET6; // Use IPv6
```

For this class we will only be using IPv4

INADDR_ANY vs 127.0.0.1

```
addr.sin_addr.s_addr = INADDR_ANY; //INADDRY_ANY == 0.0.0.0

addr.sin_addr.s_addr = inet_addr("127.0.0.1");
```

`INADDR_ANY` binds to all interfaces

127.0.0.1 binds to connections only from localhost, meaning no traffic from other hosts

htons / htonl

```
addr.sin_port = htons(port);
```

htons stands for “host-to-network short”

htonl stands for “host-to-network long”

network might operate in different endianness than the host

- i.e little-endian vs big-endian

Server Functions: listen()

Listens for a connection on a socket.

Like plugging a phone into the wall, but being able to listen to “backlog” number of calls.

```
int listen(int sockfd, int backlog);

// Example
listen(sockfd, 10);
```

Server Functions: accept()

Accepts a connection on a socket. Returns another socket representing client.

Like picking up the telephone/answering a call.

```
int accept(int sockfd, struct sockaddr * addr,
           socklen_t * addrlen);

// Example
socklen_t addr_len = sizeof(addr);
int client_sock = accept(sockfd, (struct sockaddr *) & addr,
                         &addr_len);
```

Client Functions: socket()

Creates a socket, same as server. Like buying a telephone

```
int socket(int domain, int type, int protocol);

// Example

sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

Client Functions: connect()

Connects to another machine/the server's binded socket.

Similar to calling someone on a telephone.

```
int connect(int sockfd, const struct sockaddr* addr,
            socklen_t addrlen);
// Example
int sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
struct sockaddr_in server;
server.sin_family = AF_INET;
server.sin_port = htons((u_short) portNum);
struct hostent* sp = gethostbyname(hostname.c_str());
memcpy(&server.sin_addr, sp->h_addr, sp->h_length);
connect(sockfd, (sockaddr*) &server, sizeof(server));
```

Communication Functions: send()

Send a message.

Similar to saying something during the phone conversation.

```
ssize_t send(int sockfd, const void * buf, size_t len,
             int flags);
// Example
ssize_t bytes_sent = send(sockfd, buffer, MSG_SIZE,
                          MSG_NOSIGNAL);
// bytes_sent is not always == to MSG_SIZE, but for our
// purposes it is (very rarely isn't)
```

Communication Functions: recv()

Receive a message.

Similar to hearing something during the phone conversation.

```
ssize_t recv(int sockfd, const void * buf, size_t len,
             int flags);
// Example
ssize_t bytes_recvd = recv(sockfd, buffer, MSG_SIZE, 0);
// here bytes_recvd is not always == to MSG_SIZE

// But we also can do
ssize_t bytes_recvd = recv(sockfd, buffer, MSG_SIZE
                           MSG_WAITALL);
// here bytes_recvd is == MSG_SIZE, but we block until then
```

Communication Functions: close()

Close a socket connection.

Similar to hanging up the phone (or even throwing it away haha)

```
int close(inf sockfd);

// Example
int ret = close(sockfd);
```

Putting it all together

```
// Server functions (in order)
socket() // Buy phone

setsockopt() // Set up phone

bind() // Register phone num

listen() // Plug in phone

accept() // Pick up phone

recv()/send() // Talk w/Client

close() // Hang up phone
```

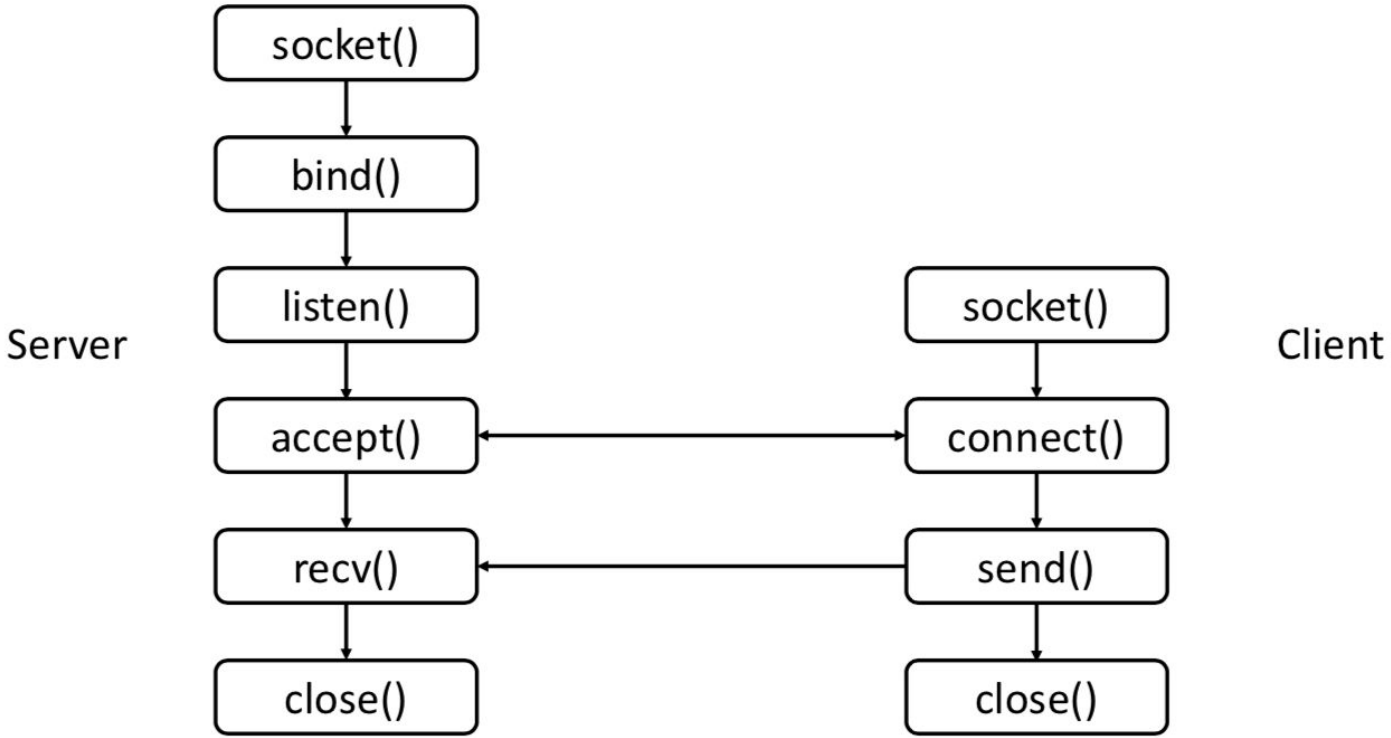
```
// Client functions (in order)
socket() // Buy phone

connect() // Call server

recv()/send()  // Talk to Server

close() // Hang up phone
```

TCP Socket API - Control Flow



Socket Demo

Credit to Ben Reeves (482 GSI/IA) for the code for the Demo
(link to the repo and other examples on next slide)

Socket example projects

Very good example here (demo code we looked at that I did not write):

https://github.com/eecs482/bgreeves-socket-example

Other examples can be found here (anything with socket in the name):

https://github.com/eecs482

Very good references here:

http://www.cs.rpi.edu/~moorthy/Courses/os98/Pgms/socket.html

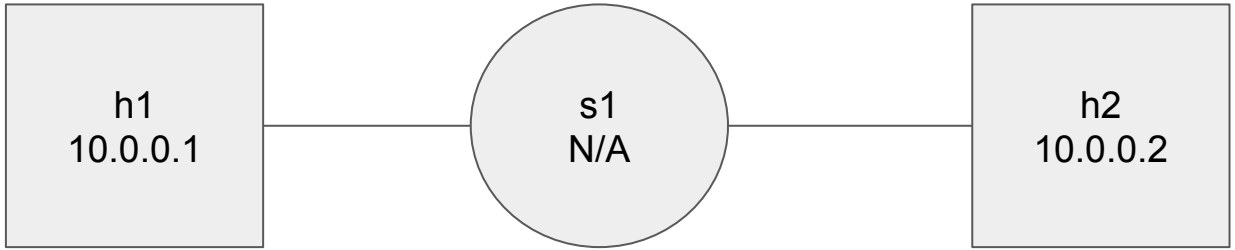
https://beej.us/guide/bgnet/html/

Also see the linux man pages (e.g “man recv”)

Mininet VM Info

Default Mininet Topology

```
$ sudo mn
```



Note: There is no DNS configured, no name resolution programs (e.g dig, nslookup, etc.) will work

Please ignore the switches, they don’t matter for this class

Mininet VM Tips

Use the one from the assignment spec! Already has a desktop env installed

Provision enough hardware resources

- About half of RAM, half of cores, as much VRAM as possible

Shared folders

Follow the mininet walkthrough mentioned in the spec