IP addresses

- Every computer on a network has an “IP address” uniquely identifying it on the network
  - An IPv4 address is 4 bytes. Usually written as 4 numbers, 0-255, separated by periods (e.g. 192.168.1.230)
- If you want to talk to a computer, you need to know its IP address
- How do you find the IP address? (Too hard to remember!)
  - Your computer is configured with the address of a DNS server (can be hardcoded)
  - When you want to reach “www.google.com,” ask the DNS server for the IP address
  - IP address of www.google.com:
    - 🍌 dig +noall +answer www.google.com
    - www.google.com. 204 IN A 216.58.194.16
DNS resolution

Hi 8.8.8.8, what's the IP address for www.google.com?

www.google.com is at 216.58.194.16!

Hi 216.58.194.16, can you give me the www.google.com home page?

Here you go!
Understanding port numbers
“Host” (computer) = apartment complex
“Host” (computer) = apartment complex
“Host” (computer) = apartment complex
“IP address” = apartment complex address
“Host” (computer) = apartment complex
“IP address” = apartment complex address
“Host” (computer) = apartment complex
“IP address” = apartment complex address
“Port number” = apartment number
“Host” (computer) = apartment complex
“IP address” = apartment complex address
“Port number” = apartment number

Want to go to http://web.stanford.edu?
Use DNS to find web.stanford.edu's IP address: 171.67.215.200
Go to that apartment complex
Knock on the apartment that runs the HTTP service (port 80)
“Host” (computer) = apartment complex
“IP address” = apartment complex address
“Port number” = apartment number

Want to SSH into myth.stanford.edu?
Use DNS to find myth.stanford.edu’s IP address: 171.64.15.29
Go to that apartment complex
Knock on the apartment that runs the SSH service (port 22)
Starting a server
Apartment complex = host

171.67.215.200

... 22 ... 80 ... 443 ...
Apartment complex = host
Each host will have some processes running on it

171.67.215.200

... 22 ... 80 ... 443 ...
Each host will have some processes running on it
“Binding” to a port:

pid 1234

FD table

OF table

Vnode table

171.67.215.200

... 22 ... 80 ... 443 ...

terminal R/W...
“Binding” to a port:
Process “sets up shop” in an apartment. (Only one process per apartment)
“Binding” to a port:
Process “sets up shop” in an apartment. (Only one process per apartment)
“Binding” to a port:
Process “sets up shop” in an apartment. (Only one process per apartment)
Process installs a “waiting list” outside the apartment

pid 1234

FD table

OF table

Vnode table

terminal

171.67.215.200
... 22 ... 80 ... 443 ...
“Binding” to a port:
Process “sets up shop” in an apartment. (Only one process per apartment)
Process installs a “waiting list” outside the apartment

 pid 1234
 FD table

 OF table
 R/W

 Vnode table
 terminal

pid 1234

171.67.215.200

… 22  … 80  … 443  …
"Binding" to a port:
Process “sets up shop” in an apartment. (Only one process per apartment)
Process installs a “waiting list” outside the apartment
Waiting list is attached to a file descriptor, so the process can see when someone arrives

<table>
<thead>
<tr>
<th>FD table</th>
<th>OF table</th>
<th>Vnode table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>terminal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
171.67.215.200
```

```
... 22 ...
```

```
... 80 ...
```

```
... 443 ...
```
“Binding” to a port:
Process “sets up shop” in an apartment. (Only one process per apartment)
Process installs a “waiting list” outside the apartment
Waiting list is attached to a file descriptor, so the process can see when someone arrives

![Diagram]

- FD table
- OF table
- Vnode table

- pid 1234
- 171.67.215.200

- 22
- 80
- 443
“Binding” to a port:
Other processes can bind to other ports
(no two processes can bind to the same port — one application per apartment!)

pid 1234

FD table

OF table
R/W  R/W

Vnode table
terminal  socket

171.67.215.200

... 22  ... 80  ... 443  ...
“Binding” to a port:
Other processes can bind to other ports
(no two processes can bind to the same port — one application per apartment!)
“Binding” to a port:
A process can bind to multiple ports, if it desires
“Binding” to a port:
A process can bind to multiple ports, if it desires

171.67.215.200
Connecting a client
Say we have a server bound on 171.67.215.200:80
On some other computer, we want to talk to that server
The “client” walks out to try to find 171.67.215.200:80
If successful, it adds itself to the waiting list

<table>
<thead>
<tr>
<th>FD table</th>
<th>OF table</th>
<th>Vnode table</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R/W</td>
<td>terminal</td>
</tr>
<tr>
<td></td>
<td>R/W</td>
<td>socket</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>171.67.215.200</th>
<th>10.0.4.110</th>
</tr>
</thead>
<tbody>
<tr>
<td>... 22 ... 80 ... 443 ...</td>
<td>...  ...  ...  ...</td>
</tr>
</tbody>
</table>
The server sees the client through its waiting list file descriptor.

171.67.215.200

10.0.4.110

Garage/
outgoing ports
It takes the client off the waiting list and creates a new bidirectional “socket” that it can use to talk directly with the client.
It takes the client off the waiting list and creates a new bidirectional "socket" that it can use to talk directly with the client
Successful in making a connection, the client also creates a new file descriptor it can use to talk to the server.
If the client writes to its fd 3, it will be readable on the server’s fd 4.

```plaintext
171.67.215.200
... 22 ... 80 ... 443 ...

10.0.4.110
```
Similarly, if the server writes to fd 4, it will be readable on the client’s fd 3.
The server can talk to multiple clients at the same time, using separate file descriptors (often using a thread to facilitate each conversation over each fd).
Networking syscalls

💡 You don’t need to know these super well, but you should have some sense of what is happening behind the scenes.
int fd = socket(AF_INET, SOCK_STREAM, 0);
Allocates a socket that will use IPv4 and TCP (TCP provides a reliable pipe-like stream of communication — more next Wednesday).
The socket isn’t attached to anything yet.
int fd = socket(AF_INET, SOCK_STREAM, 0);
Allocates a socket that will use IPv4 and TCP (TCP provides a reliable pipe-like stream of communication — more next Wednesday).
The socket isn’t attached to anything yet.
struct sockaddr_in address;
memset(&address, 0, sizeof(address));
address.sin_family = AF_INET;
address.sin_addr.s_addr = htonl(INADDR_ANY);
address.sin_port = htons(port);

Initialize a `struct sockaddr_in` with the IP address and port that we wish to listen on
bind(fd, (struct sockaddr *)&address, sizeof(address))

“Move into the apartment”: Tell the OS that we would like to use the specified IP/port. If that port is already in use, bind will return -1.
bind(fd, (struct sockaddr *)&address, sizeof(address))

“Move into the apartment”: Tell the OS that we would like to use the specified IP/port. If that port is already in use, **bind** will return -1.
listen(fd, 128)
Install a waiting list with room for 128 waiting clients, and start listening for connections (when someone shows up, they will be added to the waiting list)
**listen(fd, 128)**

Install a waiting list with room for 128 waiting clients, and start listening for connections (when someone shows up, they will be added to the waiting list)
int fdConnectedToClient = accept(fd)
Watch the waiting list, waiting for someone to connect. (accept blocks until then.)
On some other computer, we want to talk to web.stanford.edu (the server)
First, we need to do a DNS lookup to figure out its IP address:

```c
struct hostent *he = gethostbyname("web.stanford.edu");
```
We allocate a socket to use for this connection:

```c
int fd = socket(AF_INET, SOCK_STREAM, 0);
```
We allocate a socket to use for this connection:

```c
int fd = socket(AF_INET, SOCK_STREAM, 0);
```
We construct a struct sockaddr_in to specify which host/port we wish to connect to:

```c
struct sockaddr_in address;
memset(&address, 0, sizeof(address));
address.sin_family = AF_INET;
address.sin_port = htons(80);
address.sin_addr = *((struct in_addr *)he->h_addr);
```
Finally, we tell the OS to use our socket to connect to the specified host/port:
\[
\text{connect}(fd, (\text{struct sockaddr }*) \&\text{address, sizeof(address)})
\]
Finally, we tell the OS to use our socket to connect to the specified host/port:

```c
connect(fd, (struct sockaddr *) &address, sizeof(address))
```

![Diagram of FD, OF, and Vnode tables with ip addresses and ports]
At this point, the server’s accept call returns:

```c
int fdConnectedToClient = accept(fd)
```
At this point, the server’s accept call returns:

```c
int fdConnectedToClient = accept(fd)
```
What can you do with this?
What can you do with this?

- Multiprocessing: you don’t need to implement *everything* within your program. You can use other executables on the machine.
- Networking: you don’t even need to have everything working on one machine. You can use other machines to help you out:
  - Google Images: search images of cats within a fraction of a second. It wouldn’t be possible to store all the images that Google Images has on a single machine.
  - Distributed computation: e.g. rendering an animated film using a large server farm.
What can you do with this?

- Look up words in a dictionary:
  ```bash
echo "define * networking" | nc dict.org 2628
  ```

- **Print to your networked printer** (!!!):
  ```bash
echo "Hello world" | nc 10.0.4.175 9100
  ```
Networking APIs

- API: structured way of asking for something and getting a response (more next Monday)
- [http://icanhazip.com](http://icanhazip.com): tells you your IP address
- [http://api.open-notify.org/astros.json](http://api.open-notify.org/astros.json): list astronauts currently in space
- [https://www.placecage.com/200/400](https://www.placecage.com/200/400): generate a placeholder image of the given dimensions featuring Nick Cage
- [https://placekitten.com/](https://placekitten.com/): same as above, but with kittens
- Other lists:
  - [https://apilist.fun/](https://apilist.fun/)
  - [https://www.reddit.com/r/webdev/comments/3wrswc/what_are_some_fun.Apis_to_play_with/](https://www.reddit.com/r/webdev/comments/3wrswc/what_are_some_funApis_to_play_with/)