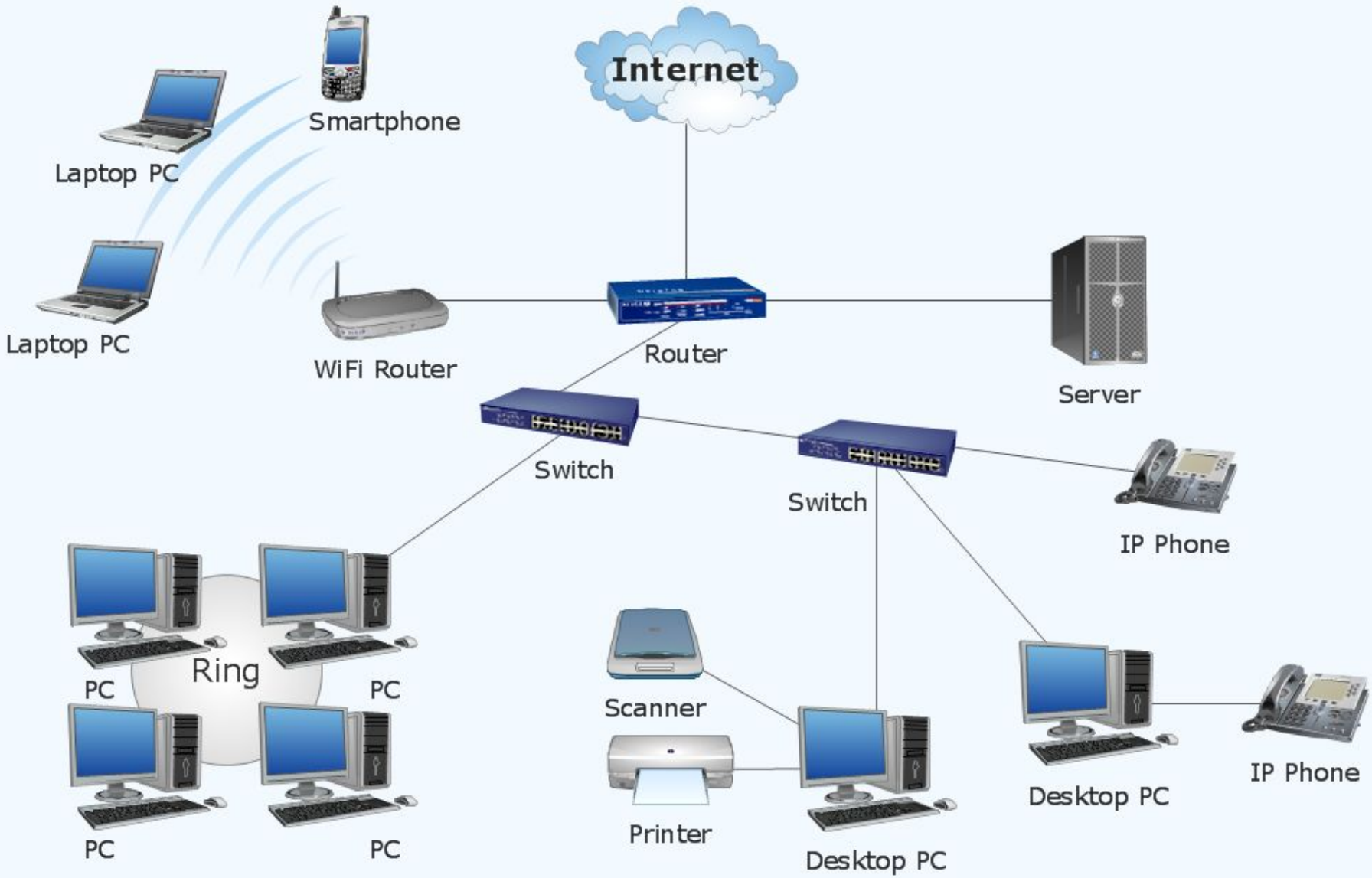
- Networking -

Given by Yours Truly’s: Sith and Mary


Internet
Smartphone
Laptop PC
Laptop PC
WiFi Router
Router
Server
Switch
Switch
IP Phone
Ring
PC
PC
PC
PC
Scanner
Printer
Desktop PC
Desktop PC
IP Phone

Components of the Network

- Hosts
  - Individual computers
- Links
  - Connect different components such as hosts and routers together
- Switches
  - Ties many hosts together using through links
  - This is “local area network”
- Routers
  - Connect different switches together

Identifying Things

- Computers are identified by their addresses
  - IPv4 Address
    - 123.45.67.89
    - Four numbers each from 0 to 255 => 256 ^ 4 possible addresses
  - Too hard to remember 50.18.174.64?
    - Pshhhhh...just type in “www.caltech.edu” instead
    - We call this “hostname”
- DNS servers maintain mappings from hostnames to IP addresses
- Computers can run more than one network application simultaneously
  - How does the host direct the correct type of traffic to the correct application?
  - Port numbers!
  - HTTP traffic runs on port 80
  - SSH traffic runs on port 22
  - HTTP clients don’t receive SSH traffic

Sockets

- A construct provided by the operating system.
- We can use it without any knowledge of the underlying hardware
- Can read/write at will
- Socket also controls the flow of data:
  - TCP & UDP
- We provide you with the CS2Net class to make your life easier

CS2Net Socket

We can create a socket object with:

CS2Net::Socket sock;

Suppose we are a client and we would like to connect to another host with hostname “host1.example.net”, port 9001.

```
std::string hostname("host1.example.net");
uint16_t port = 9001;
int con_ret = sock.Connect(&hostname, port);
```

CS2Net Socket

Connect() has a return value

- < 0, something bad happened
- 0, we're fine.

The assignment will tell you how to handle each case

There are many reasons why a connection would fail:

- Network interface is down.
- Address is incorrect
- Remote host is down
- Remote host is firewalled

In our case, it could be that the server we provide is down. Check with a TA if you are unsure. Please don't try to crash the server yourself.

Transmitting and Receiving Data on the Client Side

```
/* Connect the socket, as shown in a previous slide */
CS2Net::Socket sock;
std::string hostname("foo.example.net");
uint16_t port = 42000;
int ret = sock.Connect(&hostname, port);

/* Let's send some data to the host we're connected to. */
std::string to_send("some stuff to send");
int ret = sock.Send(&to_send);

/* Whooooooo...error checking stuff. */
if(ret == -1)  {
    ERROR("send error: %s", strerror(errno));
}
else {
    // do stuff;
}

/* Check for data coming in. Obviously, this only works if you
 * know data is coming in. */
std::string * incoming = sock.Recv(1024, false);

/* Whoooooo...error checking stuff. */
if(incoming == NULL) {
    ERROR("recv error: %s", strerror(errno));
}
else {
    // do stuff;
}
```

Blocking functions

The sockets created by the CS2Net class are blocking.

- This means, send/receive function calls hang until completed

Completion is when any amount of data is successfully sent or received. However, our **Recv()** function could also wait for all requested data to arrive.

- Recv(1024, true) blocks until all 1024 bytes come in or an error occurs.

Of course, we shouldn't wait all day for Recv(). Other things need to happen. For this assignment, we will be Polling.

- Ask socket periodically if there is data.

Polling Sockets

```
/* Connect the socket, as shown in a previous slide */
CS2Net::Socket sock;
std::string hostname("foo.example.net");
uint16_t port = 42000;
int ret = sock.Connect(&hostname, port);

/* Create an vector in which we will add the sockets we want to poll. */
std::vector<CS2Net::PollFD> poll_vec(1);
poll_vec[0].sock = sock;
poll_vec[0].SetRead(true);

// now do the poll (10 ms timeout)
int poll_err = CS2Net::Poll(&poll_vec, 10);
REQUIRE(poll_err >= 0, "error on poll!?");

// is there a hangup or error?
if(poll_vec[0].HasHangup() || poll_vec[0].HasError()) {
    // handle error;
}

// did we get anything to read?
if(poll_vec[0].CanRead()) {
    // do stuff;
}
```

Cleaning up CS2Net::Sockets

They should autodisconnect when destroyed. However, we can disconnect explicitly with:

```
sock.Disconnect();
```

For this assignment, you'll be making a chat

You will need handle different types of incoming message. The details are in **CS2ChatProtocol.h.** The message you send and receive will need to follow the protocol below.

Type (1)	Payload Length (2)	Data (n)

Every message requires a type and a payload.

20	0D 00	48 65 6C 6C 6F 2C 20 77 6F 72 6C 64 21

This chat message says "Hello, world!" in Hexadecimal.

Endianness

Little-endian: least significant bytes first

- 1000 = E8 03

Big-endian: most significant bytes first

- 1000 = 03 E8

"Network byte order": standard byte order chosen for platform interoperability the standard is big-endian.

For this assignment, payload size is littleendian

- slightly less work on your part

Type Casting is your friend

```
std::string foo = ...;

unsigned short int bar = 1000;

const char * __bar_bits;

__bar_bits = (const char *) (&bar);

foo.append(__bar_bits, sizeof(unsigned short int));
```

Getting the length of message

```
std::string * fbyte = sock->Recv(2, true);
unsigned short length = *(unsigned short *) (fbyte->data());
```

Server-Side

```
/* Applications are trying to connect to us. Handle them. */
CS2Net::Socket listener;
CS2Net::Socket * incoming_conn = NULL;

/* Bind to some arbitrarily chosen port with a backlog of 3 connections allowed */
int err = listener.Bind(31337, 3);
REQUIRE(err == 0, "Failed to bind!");

/* The listener socket is now listening for incoming connections. We can now
 * accept connections as they come in. */
incoming_conn = listener.Accept();
REQUIRE(incoming_conn != NULL, "Failed to accept!");

/* Now incoming_conn can be used to send and receive from the host that's
 * connecting to it. Don't forget to deallocate! (Disconnection happens
 * automatically. */
```

Good luck!

Ask a TA if you're stuck!

Useful Functions:

- std::string::find_first_of(char A, size_t start)
  - returns the index of the first occurrence of A
- std::string::substr(size_t start, size_t len)
  - returns a len-length substring starting from start