The Internet

Client-Server Computing

The Client-Server model, where many browser instances can interact with a single web server, which in turn may be connected to some database:

```
C                     S
+---------+         +------------+
| browser | <-----> | web server |
+---------+         +-----+-----+
C                     ^     |
+---------+           |  +--+--+
| browser | <--------+  | DB  |
+---------+              +-----+
```

Alternatives to the Client-Server Model

0. **Single computer** (CS 31 assignments, where the logistics of networking, OS, etc. were abstracted away).

1. **Peer-to-peer (P2P)**: Decentralized approach. If a peer doesn't have a resource, the request is redirected to another resource. The main advantage of this is that it is more fault-tolerant: a single peer going down doesn't bring the system down. The downside is that it is more involved to maintain a consistent state across every peer. This is in contrast to the less fault-tolerant but more state-consistent client-server model.

2. **Primary secondary**: One primary machine that serves as the "overseer" - it keeps track of how the application is split up among numerous secondary servers. The secondary servers receive a small "subproblem" of the application from the primary server and return any results.

Performance and Correctness Issues

Traditional programming performance metrics:

- CPU time: how many CPU instructions executed (roughly proportional to the amount of energy consumed)
- Real time: amount of human time elapsed
- RAM
- I/O

Networking performance metrics:

- **Throughput**: Number of client requests per second that the system can handle (assuming individual requests and responses are reasonably small and approximately equally sized); bigger is better.
- **Latency**: Delay between a request to the server and the response back from the server; smaller is better.

To improve **throughput**:

- ➕ You can perform actions "out of order" (compared to "request order").
- ➕ You can perform actions *in parallel*.
- ➖ This can cause out-of-order execution, which can "mis-order" transactions.

To improve **latency**:

- ➕ You can use **caching** to speed up responses.
- ➖ This can cause **stale caches**, which requires **cache validation** to fix, which could be an expensive operation.

Task Networking Styles

Circuit Switching

- System is connected to the nearest central office, which can connect to other central offices. In the end, you get a path between the one computer to the other.
- However, you have temporary ownership of a wire during the transaction.

Packet Switching

- Connected to a little computer that breaks the signal into a bunch of small messages called **packets**. Each packet is sent to a local **router** that sends the packets along the network to the destination. Each packet travels independently and possible along different paths, and they do so very quickly, so it does not back up the network

Packet Switching

Idea originally proposed by Paul Baran (1960s, RAND).

Sold to Department of Defense as a reliable way to communicate in the event of nuclear war because of its ability to *reroute* in the event of an office goes down, which gives it an edge over traditional **circuit switching**.

Packet switching is a best effort transmission/no guarantees (circuit-switching is a *guarantee* between the two machines).

Problems:

- Packets can be lost - if some router along the way gets overloaded, some packets may be discarded
- Packets can be received out of order
- Packets can be duplicated via a mechanism called **bridges**
- All routing decisions are local to the individual routers, so packets may be directed into a loop

Internet Protocol Suite (TCP/IP)

A **protocol** is simply a set of rules. This particular set of rules forms the foundation of the Internet.

Basic idea: **layering**. Protocols are built on top of each other, with each layer abstracting and extending the one below it:

```
                          End user
                            |
+-----------------------+   |
|   Application Layer   | Applications
+-----------------------+   |
|    Transport Layer    | Data channels
+-----------------------+   |
|    Internet Layer     | Packets
+-----------------------+   |
|      Link Layer       | Bits and bytes
+-----------------------+   |
                            |
                          Hardware
```

Link Layer

Point-to-point.

The bits and bytes you send on a single link from one node to another (no routers in between). Very hardware oriented. Each technology used has its own link layer protocol.

Internet Layer

Packets. Software almost never operate directly at this level because it is too low level.

The IP (**internet protocol**) comes in different versions:

IPv4

Created in 1983, specified by Jon Postel (UCLA), etc. Involves packets, connection.

We are running out of IPv4 addresses because there are only about 4 billion, and the US has most of them.

Packet Anatomy

A packet is just a sequence of bytes. The **payload** is prefixed by a **header** that stores **metadata** like:

- Length.
- Protocol number (to support any protocol that ends up being built atop IP) tells you the *type* of packet that's being transported so algorithms can determine their priority. For example, a single video frame is much less important than part of a image file.
- Source IP address (a 32-bit number often expressed in Base 256).
- Destination IP address (ibid).
- A **checksum** (16-bit).
- **A time-to-live (TTL)** field (8-bit "hop count" that keeps track of how many routers it goes through; packets with abnormally high TTL values get dropped to prevent packet loops).

When routers receive packets, they look at the header, especially the destination address, to determine what to do with it.

IPv6

- Created in 1998.
- 128 bit IP addresses.
- The headers are also longer.
- A superset of IPv4; the 32 bits of IPv4 can be mapped to 32 bits of an IPv6 address, so IPv4 users can communicate with IPv6 users. The converse is not as simple but made possible with complicated translation techniques.
- Less efficient because the extra length is overhead for the link layer

UDP (User Datagram Protocol)

- Created by David Reed at MIT.
- Designed as a thin layer over IP, but still at the Internet layer.
- You use UDP if your application sends single short messages over the Internet without much care if it is lost, duplicated, etc. Intended for apps that *want* to deal with packets.

Transport Layer

Data channels. Large data *streams* (TB of data) that cannot be individual packets. This layer oversees how a *stream of packets* is transmitted.

TCP (Transmission Control Protocol)

Vint Cerf from Stanford, Bob Kahn from Princeton.

Looks like a *stream of data* that:

- Is reliable (via **acknowledgements**).
- Have **sequence numbers** for packets.
- Is ordered (the recipient reassembles the packets that may be out of order in the lower layers).
- Is **end-to-end error-checked**.

The protocol has:

- Flow control; sends packets at the correct rate to not overload the network.
- Retransmission.
- Reassembly.

A single machine can listen to multiple TCP channels on different **ports**.

Application Layer

Application dependent: web, voice, etc.

RTP (Realtime Transmission Protocol)

Runs atop UDP because TCP is not suited for sending realtime data. TCP would cause **jitter**.

HTTP (HyperText Transfer Protocol)

Runs atop TCP because reliability is a must - a single misplaced bracket may break an HTML document.

Tim Berners-Lee at CERN in 1991. Honestly a real chad, he invented:

1. The Web, HTTP and its **request-response protocol**:
    - Create a TCP connection (default port 80)
    - Client sends the server a GET message
    - Server responds with the contents of the webpage
2. **HTML (HyperText Markup Language)**: a way to express the contents of a webpage in a machine-independent format.

> The Internet fundamentally is just HTML and HTTP combined together. HTML is like the content of the Internet, HTTP is how it gets around.

`telnet` Aside

You can use the `telnet` command to open a 2-way stream where you can send raw HTTP requests. Run `telnet` with the IP address and the port to connect to. THen enter the `GET` command with the resource you want to get and the protocol version to use. Then specify the `Host`:

```
$ host www.cs.ucla.edu
WWW.cs.ucla.edu has address 164.67.100.182
$ telnet 164.67.100.182 80
Trying 164.67.100.182...
Connected to 164.67.100.182.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.cs.ucla.edu

HTTP/1.1 301 Moved Permanently
Date: Wed, 19 Oct 2022 22:21:10 GMT
Server: Apache
X-Frame-Options: SAMEORIGIN
Location: https://www.cs.ucla.edu/
Content-Length: 232
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved <a href="https://www.cs.ucla.edu/">here</a>.</p>
</body></html>
```

The actual HTML data is prefixed with a response **header**, detailing metadata like the response code, content length, etc.

Variants of HTTP

HTTPS (HTTP Secure)

With plain HTTP, every router between the source and destination can see the raw data that is transmitted.

Uses shared private keys or something.

HTTP/2

Came out in 2015.

Added extra features to HTTP:

1. Header compression
2. Server push (lets the server send a response without a request)
3. Pipelining (allows client to send multiple requests so the server can respond in batches, allows more *parallel* communication instead of having the client wait for a response every single time)
4. Multiplexing (servers can respond to the easy requests before the harder ones - responses come in out of order)

HTTP/3

Came out in 2022. Motivation: TCP is bad for audio/video streams. These streams don't care if there is an occasional blip in the stream

1. Built atop **QUIC** (Jim Roskind, Google, 2012) instead of TCP:
   - Like TCPv2 + handles stream losses
2. Prevents freezing by avoiding TCP level **head-of-line** blocking delays; allows content after a missing packet to be delivered immediately
   - Discards out-of-order packets

Data Languages

HTML is an example of a data language. We also have:

SGML (Standard General Markup Language)

Markup language for documents (1980s).

A **declarative** (as opposed to an **imperative**) language. **Markup** specifies the structure and attributes of a document.

```
<QUOTE TYPE="example">
    OK <ITALICS>This text</ITALICS> is
    part of a block quote intended for example.
</QUOTE>
```

Extensible bracket types allow you to define new features without making breaking changes to the language. `<QUOTE>` is one type of bracket with its closing tag `</QUOTE>`, `<ITALIC>` is another, etc.

The structure of the document forms a tree structure: the content inside `<ITALICS>` is a child node to `<QUOTE>`.