CPSC 3600 Final Exam Study Guide

Brendan McGuire (bmmcgui@clemson.edu)

Disclaimer: While I strive for accuracy with these guides, I am a student just like you. If you see anything wrong or inaccurate, please let me know, and I will correct it!

Network Structure

Computer Networks are built on layers of abstraction, with each layer providing certain guarantees, and higher layers relying on the guarantees of the lower levels to provide more advanced or complex services.

Layer	**Responsibilities/Description**	**Protocol Examples**
Application	*Supports specific network applications: web, file delivery, git, streaming, etc..*	HTTP, FTP, git
Transport	*Main focus is process to process data transfer across the network, in a reliable and dependable abstraction*	TCP, UDP
Network	*Routing of datagrams from source computer to destination computer*	IP, Routing Protocols
Link	*Specifically moving data from one machine to an adjacent machine*	Etherent, 802.11 (Wifi), PPP
Physical	Direct connection from one network card to another	

Layering & Encapsulation

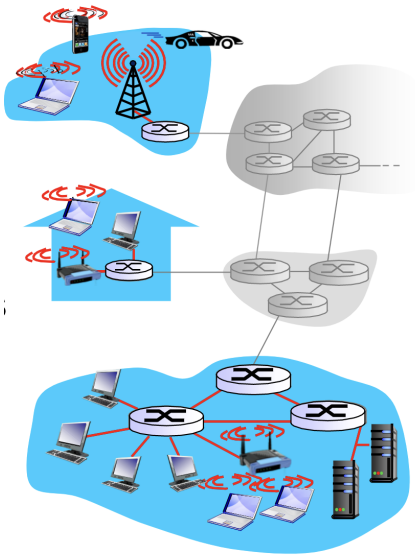
Each layer needs to add its own control information to the data in order to do its job. Typically, this is in packet headers, and prefixed on the payload before being passed down to a lower level.

			DATA		***APPLICATION***
		UDP HEADER	**DATA**		***TRANSPORT***
	IP HEADER	**UDP HEADER**	**DATA**		***NETWORK***
FRAME HEADER	**IP HEADER**	**UDP HEADER**	**DATA**	**FRAME FOOTER**	***LINK***

Access Networks

In order to be connected to the broader internet, you need to be connected to an access network, which connects your end system to an edge router. Examples of access networks include:

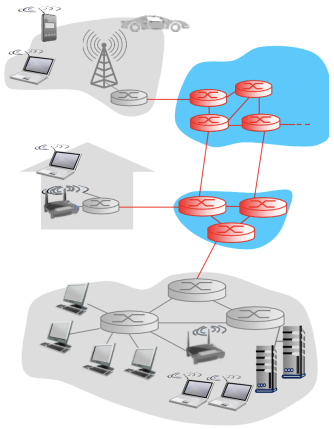
- ➔ Residential Access Networks
- ➔ Institutional Access Networks (schools, enterprise)
- ➔ Mobile Access Networks (your cell carrier)



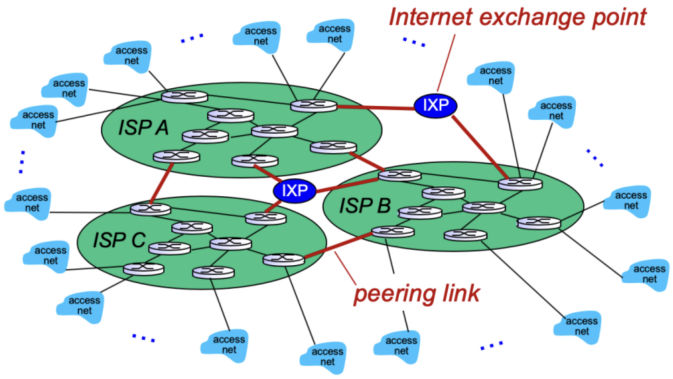
Core Networks

Seperate from Access Networks, which you usually interact with are Core Networks, a mesh of interconnected routers.

- ➔ Packet Switching: hosts break data into manageable packets when they go across the core network



Structure of the Network



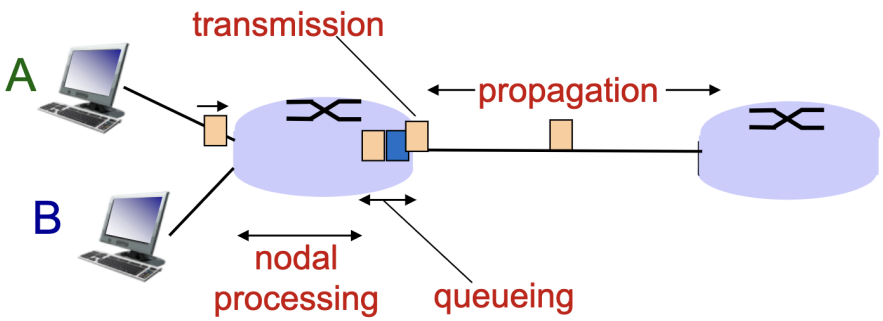
Many access networks, connected to an ISP

Many ISPs, with core networks

Core Networks connected by Internet Exchange Points

Packet Loss and Delay

In the network, there are many possible sources of packet delay. Let's examine the basic formula for packet delay.



DELAY = NODAL PROCESSING + QUEUING DELAY + TRANSMISSION DELAY + PROPAGATION DELAY

Nodal Processing – Check for bit errors, determine output link, typically very small (> 1ms)

Queuing Delay – Time waiting at output link for transmission, higher during high levels of network congestion

Transmission Delay – Time it takes to actually transmit the bits (packet length / link bandwidth)

Propagation Delay – Time it takes to actually cross the physical link (length of physical link / propagation speed)

Application Layer

The focus of the application layer is *actually doing things* with the network services we have. There are two kinds of Applications: client-server and peer-to-peer

Client-Server Applications

In client-server applications, request agents can take the form of either a client on a server. Clients are usually small, and usually connect to a single server at once. Servers are large and can have many simultaneous clients.

Example: HTTP (HyperText Transfer Protocol)

The protocol which allows the transmission of resources on the web. *Websites are made up of many different resources (HTML documents, CSS stylesheets, JavaScript, Images, Video, etc.) that the browser needs to download from the server in order to render the page.*

Every resource is addressable by a URL, a uniform resource locator, which takes on the following form:

> https://send.bren.app/app.d5396eb7.js
> **protocol://domain/path**

HTTP operates on the client-server model, where a client requests resources, and the server sends them. Some important details about HTTP:

- ➔ HTTP operates over TCP, and uses port 80 (or 443 for HTTPS)
- ➔ HTTP is “stateless” – server does not keep any information about clients in between requests
- ➔ Persistent vs. Nonpersistent
  - ◆ In persistent connections, the client and server keep open the TCP connection for multiple resources
  - ◆ In non-persistent connections, a new TCP connection is established for each resource

The general format for an HTTP request or response is a number of *Headers*, followed by the *Request/Response* Body. The headers and body are separated by 2 newlines.

```
GET /app.d5396eb7.js HTTP/1.1
DNT: 1
Referer: https://www.google.com/
sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="100", "Google Chrome";v="100"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/100.0.4896.127 Safari/537.36
```

An example of an HTTP request. The GET is the HTTP verb, and can be one of (GET, POST, PUT, PATCH, DELETE). There is no request body in a GET request.

HTTP Status Codes

The status code is a 3 digit number that the server responds with to indicate the status of the request. Some common status codes include

1xx - Informational Responses	200 OK
2xx - Successful	302 Moved Permanently
3xx - Redirection	401 Unauthorized
4xx - Client Error	404 Not Found
5xx - Server Error	500 Internal Server Error

An example of an HTTP response might look like (the response to the request above)

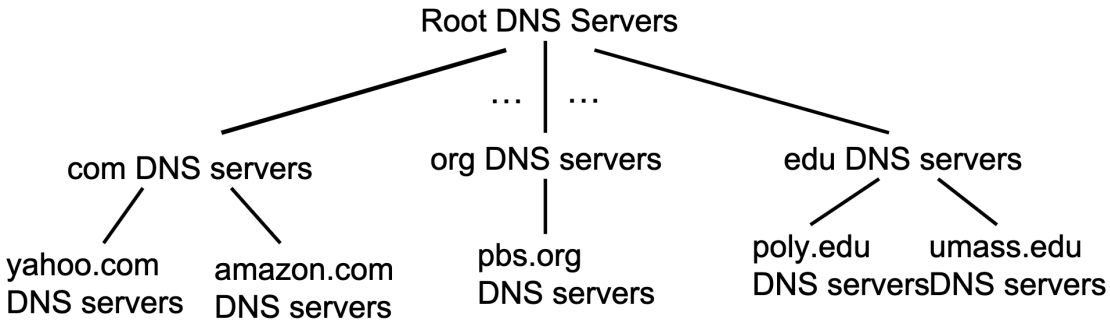
```
HTTP/1.1 200 OK
alt-svc: h3=":443"; ma=86400, h3-29=":443"; ma=86400
cache-control: public, max-age=31536000, immutable
cf-cache-status: MISS
cf-ray: 6db7992f28d61018-ATL
content-encoding: br
content-type: application/javascript; charset=UTF-8
date: Thu, 10 Feb 2022 19:03:35 GMT
etag: W/"5b98f-17aa4b74f68"
expect-ct: max-age=604800,
report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
last-modified: Wed, 14 Jul 2021 11:11:54 GMT
nel: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
report-to:
{"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v3?s=FbwnbJ2QIr9A1aOT%2BL
Vt8Q4jUSMF%2BWBdJREmvHpylq0JlJn42mbpa%2FL8%2BF8s6dQbGvppg93eZcJtTqO8h1S5OWi6eaiI46CDKvi
G4iSt9OGpL3DQ%2Fx9dQNZpe869kofE"}],"group":"cf-nel","max_age":604800}
server: cloudflare
strict-transport-security: max-age=0; includeSubDomains; preload
vary: Accept-Encoding
x-content-type-options: nosniff
x-dns-prefetch-control: off
x-download-options: noopen
x-frame-options: SAMEORIGIN, DENY
x-xss-protection: 1; mode=block, 1; mode=block

!function(t){function e(e){for(var
n,i,o=e[0],s=e[1],a=0,c=[];a<o.length;a++)i=o[a],r[i]&&c.push(r[i][0]),r[i]=0;for(n in
s)Object.prototype.hasOwnProperty.call(s,n)&&(t[n]=s[n]);for(u&&u(e);c.length;)c.shift(
)()}var n={},r={1:0};function i(e){if(n[e])return n[e].exports;
```

Cookies

Earlier, we said that HTTP is stateless. This is true at the protocol level, but obviously modern web applications must keep *some* sort of state about the client connections (to preserve authentication or preferences). This is accomplished by **Cookies**, a special type of header. The client sends the cookie, which is essentially a random bearer token, with every request, so the server can distinguish an individual client from the rest.

Example: DNS (Domain Name Service)

Resolves domain names into navigable IP addresses that clients can use to connect to foreign servers. **It is a distributed, federated system**, with many different nameservers for different purposes.



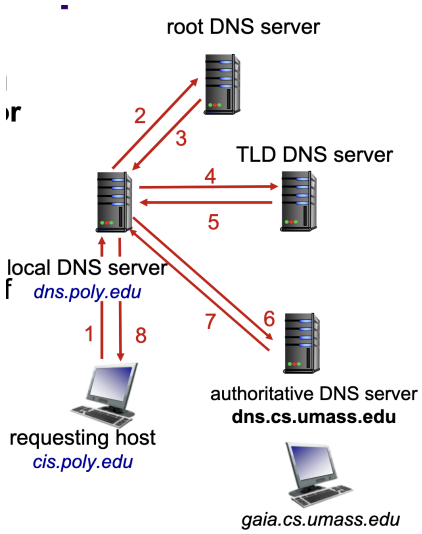
The process for DNS name resolution is as follows:

1. **Consult Local DNS Server**. This server is a part of your local network, and provided by your institution or internet service provider. It can include local DNS records, and caches for your local network
2. **Consult Root DNS Server.** One of around a dozen root DNS servers across the world. It can either give you a cached answer, or redirects to the TLD nameserver
3. **Consult TLD Server**. Every Top Level Domain (.com, .org. .edu) has its own name server, which records who owns a specific domain. This server will direct you to the *Authoritative Name Server*
4. **Authoritative Name Server.** This server is owned/operated by the *entity that owns the domain itself*. It will contain the definitive (authoritative) record of the address, if it does not exist above in cache.

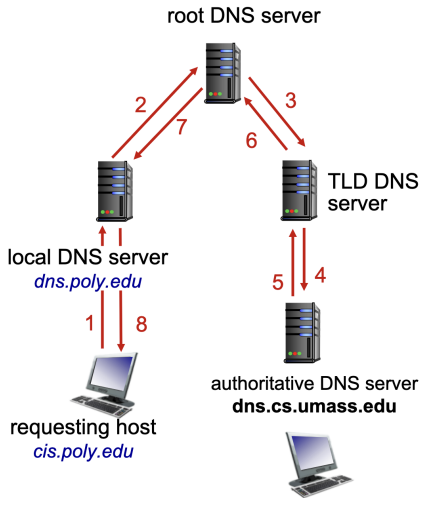
Iterated vs Recursive Query

The default scheme is an *iterated query*. If the server you are asking does not have the domain response in its cache, it will direct you to another server to contact. You are responsible for contacting the next server.

In a *Recursive Query*, the burden of contacting the next server is performed by the root server. This is done to improve speed, but puts additional load on the root servers, and can lead to denial of service amplification attacks.



An iterative query. The local server receives answers from each level on who to contact next.



A recursive query. The local server is responsible for making all of the additional calls to reach the authoritative DNS server.

Cache and Time to Live

Once any server learns about a record, it **caches** that result for the length of the Time To Live attached to the record. The cache is vital to reduce the resolve time for a record. Once a record has been cached, for the duration of the time to live, the server can respond with that record instead of directing to another server.

Types of Records

A DNS response is usually a series of records, which come in the form `(name, value, type, ttl)`

A records / AAAA records
Stores an IPv4 (A) or IPv6 (AAAA) record in *value* for a specified name

CNAME records
Stores a "canonical" address for the given address. Example: ibm.com is actually servereast.backup2.ibm.com

NS records
Name is the domain (*bren.app*)
Value is the nameserver for that domain (*brad.ns.cloudflare.com*)

MX records
Used to indicate mail servers. Value is the address of the nameserver associated with name.

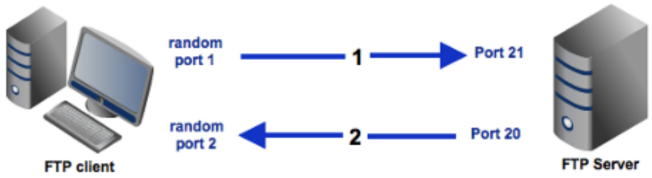
Example: FTP (File Transfer Protocol)

FTP allows the transmission of files across the network, to/from the client to/from the remote host. Similar to HTTP, it has a client/server model: the *client* initiates the connection, the *server* accepts the connection. Files can flow in either direction (upload or download). FTP uses two TCP ports: 20 (*used for data transfer*), and 21 (*used for control signals*).

Active vs. Passive Mode

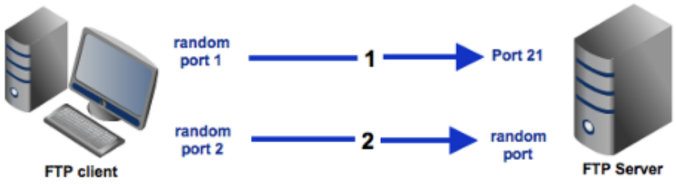
Active Mode is the default

1. Client connects to server's control port
2. Server will connect from port 20 to the client's specific port for data transfer
3. Data transfer initiated



When servers cannot connect directly to the clients (because they are behind a firewall, or Network Address Translation, or otherwise don't have a public IP address), they can use *Passive Mode*

1. Client connects to server, and indicates passive mode
2. Server gives client a port to connect to for the data transfer connection
3. Client connects to this second address.
4. Data transfer initiated



Peer-To-Peer Applications

In peer-to-peer, there is no always-on server; arbitrary end clients communicate *directly*. Peers are intermittently connected, and can change IP addresses. May require a server to broadcast or facilitate communication between clients.

Example: BitTorrent

Files divided into 256kb chunks. There is a tracker server which maintains a list of peers with the desired file. Peers connect to various torrents to download chunks.

Transport Layer

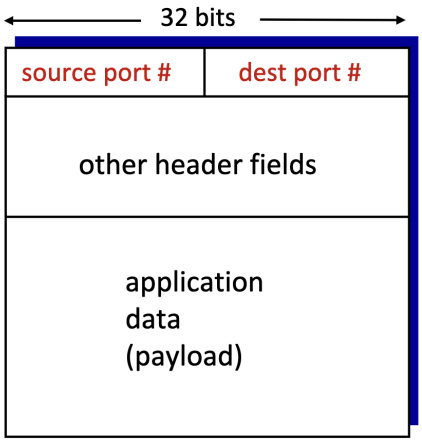
The transport layer primarily concerns itself with delivering messages from one process to another on the network. This is distinct from the network layer, which is primarily concerned with machine to machine communication. The transport layer introduces the idea of *ports*, which allow machines to maintain multiple simultaneous connections without getting packets mismatched.

TCP – Stateful connections, reliable, in-order delivery
UDP – Stateless. Unreliable, unordered delivery. Adds very little on top of IP. Good for low latency applications (streaming video)

Multiplexing / Demultiplexing

In order to distinguish multiple simultaneous connections, transport layer protocols use the idea of a *socket*, which represents a single connection. When sending information across the network, the transport layer adds port and IP information to distinguish between sockets before passing to the network layer.

The TCP segment is made of a header with source port and destination port, along with other fields. The IP address is handled in the network layer



TCP/UDP segment format

UDP Multiplexing/Demultiplexing

Since UDP has no connections, there's no handshake between nodes before sending. Instead, the client directly attaches the destination port and IP to the server and sends. When demultiplexing the node will read the destination port, it parses the segment header and redirects to the appropriate socket.

User Datagram Protocol

UDP is the barebones, no-frills extension of IP. UDP segments may be lost or delivered out of order. There is no concept of a connection, a client just immediately sends to a server.

- ➔ Good for multimedia streaming, DNS, SNMP
- ➔ No reliable delivery, but low latency and fast

UDP Checksum

First, divide segment contents (including header fields) into a sequence of 16 bit integer sequences. Checksum is the one's complement of the sum of the integer sequences. This value is placed in the checksum field in the UDP segment header. Handle wraparound by adding to the lowest bit.

```
  1  1  1  0  0  1  1  0  0  1  1  0  0  1  1  0
+ 1  1  0  1  0  1  0  1  0  1  0  1  0  1  0  1
  1  1  0  1  1  1  0  1  1  1  0  1  1  1  0  1   SUM
  0  1  0  0  0  1  0  0  0  1  0  0  0  0  1  1   ONES COMPLEMENT
```

UDP Programming

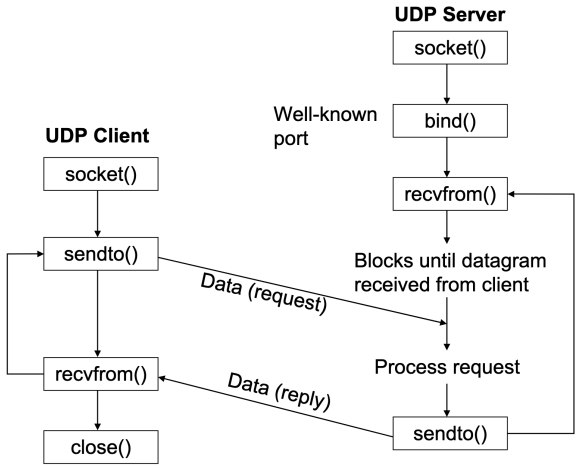
See the right for the general flow of system calls between the client and server.

socket() – Registers a new socket with the Operating System

bind() – On the server, binds the socket to the given port

sendto() – Send bytes to the other side of the connection

recvfrom() – Read bytes from the other side of the connection



Transport Control Protocol

TCP Programming

Secure Programming

Network Layer

Internet Control Messaging Protocol

Routing Algorithms

Internet Routing

Link Layer

The link layer undergirds the network layer and supports it by providing *reliable data transfer between directly adjacent nodes.* More specifically, it provides the following services

- ➔ **Framing** – Packets in the link layer are known as frames. The link layer will discretize larger packets into individual frames to be transmitted across the wire
- ➔ **MAC Address** – Different from an IP address, the MAC address directly refers to a single network card, and is the address for the Link Layer.
- ➔ **Reliable Delivery For Adjacent Nodes** – If two nodes are directly connected, then the Link Layer ensures that communication between them is reliable
  - ◆ **Error Detection & Correction**
  - ◆ **Half-Duplex & Full-Duplex** – whether or not both ends of the link can transmit at the same time. Important for flow control

Cyclic Redundancy Check

Adjacent nodes will ensure reliable transmission by the use of checksums and bit redundancy. For many protocols, this is done by the Cyclic Redundancy Check. The way this works is fairly simple:

1. The message you are checking is D bits long
2. Choose a generator *polynomial* G. For our purposes, this polynomial is just a binary number of length r + 1 (for example 101 means $x^2 + 1$).
3. Create r check bits such that:
   a. The sequence D followed by your check bits are exactly divisible by G
   b. The receiver knows G, and can perform the division to detect errors of up to length r + 1.

Multiple Access Protocols

Unlike other network layers, where the actual methods of transmission are mostly abstracted away (IP doesn't care if it is being transmitted over fiber or pigeon), the link layer concerns itself with the different means of connecting two nodes on the network. Common protocols include:

- ➔ PPP links directly between two nodes (two nodes directly connected by ethernet or dial up)
- ➔ *Broadcast links (shared wire or wireless medium).* These protocols need to regulate how to share this limited transmission link between all of the nodes.
  - ◆ Ethernet networks
  - ◆ 802.11 wireless (WIFI)

MAC Address and Address Resolution Protocol

As previously discussed, the MAC address is used to identify a specific network card on a machine. It is 48 bits (longer than an IPv4 address) and is used to get from one physical device to another. Generally takes the form of hexadecimal segments, like "1A-2F-BB-76-09-AD"

- ➔ *Why is the MAC Address Needed?* Unlike IP addresses, which can refer to an entire network of devices (through Network Address Translation), a MAC address is usually burned into the network card, and always represents a single device.
- ➔ However, when communicating on a network, we almost exclusive refer to devices by their IP address. To translate the IP address into the MAC address, we use the **Address Resolution Protocol**. One way to think of it is like DNS, but for MAC addresses.