CSC 138 – Midterm Study Guide

The internet is a network of interconnected networks.

Protocols define the format order of msg's sent and received among network entities, and action taken.

The network edge is where hosts (client & servers) sit. They are connected to each other through **access networks & physical media** (cable, DSL, Fiber, Ethernet, Wi-Fi, LTE).

The host sending function consists of taking an application msg, breaking it into small chunks of L length (packets) and transmitting it through an access network at a rate of R. Packet Transmission Delay = $\frac{L\ (bits)}{R\ (\frac{bits}{sec})}$

The network core is a mesh of interconnected routers and switches that uses packet switching to forwards packets to their destination.

Store and forward: Every router has a queue where packets enter, and eventually leave. This allows the router to temporarily receive more packets than it can send for short periods of time.

End-to-End Delay is the total time it takes a packet from send time, to be received.

Circuit Switching is where resources are dedicated to a user. Dedicated resources give guaranteed performance but allows circuits to be idle if there is no activity.

Frequency Division Multiplexing (FDM) is where a frequency is divided into narrow frequency bands to be used. Each "call" is allocated a band and gets the max speed of that band. In **time division multiplexing (TDM)** entire band is divided into slots. Each "call" gets the entire band for its time slot.

The current **internet structure** has access networks connected to ISP's which communicate to each other via internet exchange points. There are multiple tiers of ISP's with tier 1 being commercial, tier 2 being regional. Content provider networks often bypass such networks to provide more dedicated service.

Packet delay is affected by nodal processing time, queuing delay, transmission delay, and propagation delay.

Packet loss occurs when a buffer's memory is exceeded.

Throughput is the rate at which bits are being sent from the sender to receiver. Throughput is the min() of all link speeds taken to get to the receiver. Or the sum of all the min() if multiple paths are used.

Protocol Layer is where an internet protocol exists. It performs its job and communicates with other layers in the stack.

The **internet protocol stack** consists of 5 layers. Application layer (networked apps), transport layer (UDP, TCP), Network (IP, routing protocols), Link (Ethernet, Wi-fi), and physical (bits on wire).

ISO/OSI model has two extra layers. It goes from application, presentation (data interpretation), session (data sync), transport, network, link, and physical.

The layer above another layer completely **encapsulates** it. Therefore, devices only need to deal with the layer above the layer they are designed to handle.

The internet was not originally designed with **network security** in mind. Much work has gone into securing our networks:

- Packet Sniffing: Reading the content of a packet as its sent over the wire.

- IP Spoofing: Injecting a packet with a false source address
- DDos: Overwhelming resource with hundreds if not thousands of requests.

The **client-server paradigm** is a application model that connects a client (intermittent connection) to a server that is always-on. The server has a permanent IP and communicates data to the client through internet protocols such as HTTP, IMAP, and FTP.

The **peer-to-peer architecture** does not require an always on server. Instead end-to-end system directly communicate each acting both as clients and servers (complex to manager, but scalable).

Sockets are a "door" that allow processes to send/receive messages. A socket on a computer is associated with a port number and sends/receives msg's from the transport layer.

TCP (transmission control protocol) is an internet transport layer protocol that guarantees reliable transport between sender and receiver. It requires a handshake (single RTT) to setup and includes features such as flow control (won't overwhelm receiver), congestion control (can throttle sender). It does not provide a min guarantee on throughput or security.

UDP (User Datagram Protocol) is an unreliable transport layer protocol that is simple and does not provide many features. It does not guarantee reliability, nor congestion control or throughput/timing. Requires no setup and is generally faster than TCP.

HTTP (Hypertext transfer protocol) is a web app protocol that follows the client-server paradigm. It uses TCP and exchanges HTTP messages. These msg lack state (no info is mainted). An HTTP request consists of 3 parts, a request line (get, post, url, version). A header line (header names and values) and the body (contains info).

Non-persistent HTTP requires 2 RTT's per object received. A TCP connection is opened (handshake = 1 RTT) and then request/response occurs. Once the response is received the connection is close. Multiple objects require multiple connections.

Persistent HTTP allows multiple objects to be sent over the same connection. Once all of the objects have been sent, the connection is then close.

RTT (round trip time) is the time it takes a packet to travel from client to server & back. An **HTTP response** requires 2 RTT's.

A **HTTP Response Msg** consists of a status line (protocol version, status code/phrase) and body with the requested object.

A **cookie** is a method to maintain state between transactions. The file is kept on the user's host & managed by the browser. Cookie info is sent in the header line of an HTTP message so the server can identify the user.

Web caches maintain copies of objects (websites) with the goal of not involving the origin server with all requests. If the object requested is cached, it simple returns that object. Else, the web cache asks for the object, stores it for future use and reports back to the client. This reduces response time, traffic.

The **Conditional GET** is used verify if a cached copy is update. An HTTP request is sent to their server with an "if-modified—since:" clause. If the object has been modified since this date, the object is returned. Else, the server response with "HTTP/1.1 304 NOT MODIFIED."

The **three major components** of email is user agents (mail reader, composer, reader…), mail servers (contains mailbox, msg queue for outgoing mail), and SMTP (simple mail transfer protocol) which communicates between mail servers to send msg's.

SMTP uses TCP to reliably transfer mail from a client to a server. It has 3 phases of transfer (handshake, msg transfer, and closure). It uses a command/response interaction and messages must be encoded in 7 bit ASCII.

IMAP is a protocol that allows for retrieval, deletion, and folders of emails stored on web server (client -> server)

HTTP provides a web-based interface on top of STMP, IMAP to retrieve email messages.

SMTP Vs HTTP: HTTP each object encapsulated in own response msg, SMTP multiple objects sent in multipart msg. SMTP uses persistent connections, requires encoding in 7 bit ASCII uses CRLF.CRFL to end message.

The **DNS (Domain Name System)** maps between IP address & URL name & vice versa through a series of distributed databases.

DNS Services include hostname to IP translation, host aliasing, mail server aliasing.

DNS has three main **tiers of service,** each tier is responsible for servicing a subset of requests (there are many copies between each tier). The authoritative tier is an organizations own server maintained by the org. A top level tier manages all services such as .com or .org or .edu. The root DNS handles all requests that failed to be handled by other tiers.

DNS name resolution can occur one of two ways

- **Iterated query**: Contact server replies with name of server if it fails to resolve it
- **Recursive query**: Contacted server contacts another server if it fails to resolve. Client is not involved.

DNS records contain RR's (resource records) which hold four parameters. Name, value, type, and TTL (timeout).

- Type **A**: Name is hostname, value is IP address
- Type **NS**: Name is domain, value is hostname of authoritative name server for the domain
- Type **CNAME**: Name is alias for some real name, value is conical name
- Type **MX**: Value is name of SMTP mail server associated with name.

In the Client-server paradigm a file being delivered to a client slows down as more clients request the file. In P2P time to deliver increases by (LogN?) as more client request the file. This is because as each client joins, they act as a server delivering pieces they have to other clients.

BitTorrent divides files into 256kb chunks, peers in torrent receive and send chunks.

- **Requesting chunks:** Different peers have different subsets of chunks, peers are informed what chunks each other have periodically. Chunks are requested rarest first.
- **Sending chunks:** Peers send chunks to peers sending at the highest rate to itself, other peers choked by a peer do not receive chunks. It sends chunks to top 4, revaluating every 10 sec. Every 30sec a random peer is selected, if it does not become top 4, it is dropped.

DASH (Dynamic adaptive streaming over HTTP) allows for a client to request content at an encoding rate sustainable on its current bandwidth. It can choose different rates at different times from different servers.

Servers over videos in chunks encoded at different rates, files are replicated across various servers. A manifest file allows the client to determine where and how to get its chunks.

A **CDN (content distribution network)** is a method by which files are delivered to clients. Using a manifest, clients retrieve content from CDN nodes scattered around the internet. Many possible locations with copies of each files help eliminate network traffic.

Python UDPClient

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message.encode(),
                    (serverName, serverPort))
modifiedMessage, serverAddress =
                    clientSocket.recvfrom(2048)
print modifiedMessage.decode()
clientSocket.close()
```

Python UDPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print ("The server is ready to receive")
while True:
   message, clientAddress = serverSocket.recvfrom(2048)
   modifiedMessage = message.decode().upper()
   serverSocket.sendto(modifiedMessage.encode(),
                       clientAddress)
```

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print ('From Server:', modifiedSentence.decode())
clientSocket.close()
```

Python TCPServer

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while True:
    connectionSocket, addr = serverSocket.accept()

    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                                        encode())
    connectionSocket.close()
```