Internet def: A network of interconnected networks.

Internet Protocol: Defines the format, order of msg (sent/received) among network entities, and actions taken on msg transmission and receipt.

Network edge consists of hosts (client & servers), **access networks** (home, mobile, enterprise using wired or wireless), and **physical media** (cable, DSL, ethernet, Fiber, etc.).

The **network core** is a mesh of interconnected routers & switches that uses packet-switching.

Packet transmission delay takes $\frac{L\ (bits)}{R\ (\frac{bits}{sec})}$to push over link.

Store and forward describes the way a packet is stored in a router's buffer (until fully received) before being sent out.

End-to-End delay is the total time it takes a packet to be received (from send time)

$Total_d = proc_d + queuing_d + trans_d + prop_d$

Circuit switching is where resources are dedicated to a user. It gives guaranteed performance, but allows circuits to be idle

FDM (Frequency Division Multiplexing) frequency divided into narrow bands. Each user allocated a narrow band. Bandwidth guaranteed on that band.

TDM (Time Division Multiplexing) frequency divided into time slots. Each user gets entire band for entire time slot. Full speed of band preserved.

The **internet structure** consists of millions of access networks connected to ISP's which communicate to each other via internet exchange points. Multiple ISP's tiers service different areas (regional or international). Content provider networks often bypass some networks to provide a more dedicated service.

Packet delay sources: Processing, queueing, transmission, and propagation delays.

Packet loss occurs when a router's buffer becomes full, making it necessary to drop some packets.

Throughput is the rate at which bits are being sent from the sender to receiver. It is the min(all link speeds), or the sum of all the min(link speeds) if paths are used in parallel.

Protocol Layer Stack: Application layer (networked apps), transport layer (TCP & UDP), network layer (IP, routing protocol), link layer (wi-fi & Ethernet), and physical layer (bits on wire).

ISO/OSI adds two layers after application layer: presentation layer (data interpretation) and session layer (data sync).

Encapsulation occurs when the layer above another layer handles its packet. Devices only need to deal with the layer above the layer they are designed to handle.

Packet Sniffing: Reading a packets contents over the wire.

IP Spoofing: Injecting a packet with a false source address.

DDOS: Overwhelming a resource with invalid requests.

Client-Server an application model that connects client to an always-on server with a permanent IP and communicates with the client through protocols such as HTTP, IMAP, FTP.

P2P is where clients act as both server and clients. Directly communicating (complex management, but scalable).

Sockets are a door that allow processes to send/receive msg's. A socket is associated with a port number. Sends a msg through the transport layer.

TCP (Transmission Control Protocol) is a transport layer protocol that gives reliable transport, flow/congestion control. It requires a handshake and does not provide timing, or min throughput guarantee's or security.

UDP (User Datagram Protocol) is a transport layer protocol that gives unreliable data transfer. No bells or whistles. No handshake required and is generally faster than TCP.

HTTP is a web application layer protocol. It uses the client-server paradigm, it uses TCP, and is stateless (no info maintained). A request consists of 3 parts: header line (get, post, URL, version). Header line (header name & values) and body.

Persistent HTTP can receive multiple objects over the same connection. **Non-Persistent HTTP** can only receive one object.

HTTP response requires 2 RTT's (round-trip-time).

HTTP response msg contains a status line (protocol version, status code/phrase) and a body with requested object.

Cookies are a file managed by the user's browser that is communicated to the server (sent over HTTP header line). Allows state to be maintained.

Web caches maintain copies of objects. If a request is cached, it has no need to query the server. If not, the object is queried, and cached for later user.

Conditional Get is user to verify if a cached copy is up to date. If object has been modified since, new object is cached, else not.

Thee major email components: user agents (reader, composer), mail server (mailbox, msg queue), and SMPTP (communication between mail servers to send msg's).

SMTP uses TCP to transfer mail from a user-agent to mail server. Requires 3 phases (handshake, transfer, closure). Similar to HTTP but encoded in 7bit ASCII.

IMAP provides for retrieval/deletion, folder management on mail servers.

HTTP provides a web-based interface on top of SMTP (send) and IMAP/POP3 (retrieve).

HTTP Vs SMTP: HTTP is pull, SMTP is push, both have a cmd/response interaction. HTTP encapsulates object in response, SMTP sends multiple objects in multipart msg. SMTP is persistent, encoded in 7bit ASCII, and uses CRLF.CRLF to end message.

DNS maps between IP and URL's and vice versa through a series of distributed db's.

DNS Services: Hostname -> IP, host aliasing, mail server aliasing.

DNS Tiers: Many tiers to distribute load (each tier has many redundant copies). Authoritative tier (organization mainted server). Top-level tier (manages domains such as .com, .org, etc.) Root DNS (all requests that failed to be handled by lower tiers). Local DNS (serves a region).

DNS Name Resolution can occur 1 of 2 ways: **Iterated query** (contact server replies with name of new server if it fails to resolve it, client is responsible for contacting next server). **Recursive query** (contacted server forwards request until resolved, no client involvement).

DNS Records: Database entries (RR's):

- **A:** Name is hostname, value is IP address
- **NS:** Name is domain, value is hostname of authoritative DNS server for that domain.
- **CNAME:** Nam is alias for some real name, value is canonical name.
- **MX:** name is mail server, value is mail server IP

File distribution: <u>Client-sever</u> (slows when more clients need file), <u>P2P</u> (new clients improve ability to distribute files).

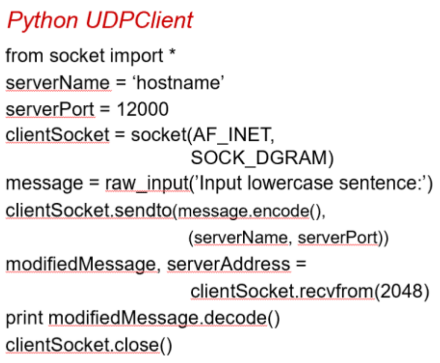
BitTorrent divides files in 256kb chunks, peers in torrent receive and send chunks. **Requested chunks**: different peers have different chunks, peers informed what chunks each have periodically, chunks requested from rarest first. **Sending chunks** sent to 4 peers sending at highest rate to itself, revaluates every 10 sec, every 30, random peer is chosen to receive chunks, if it does not become a top 4, it is dropped.

CDN files are scattered across the internet (many copies) at various nodes. Subscriber requests content and retrieves content from nearby node (depending on congestion, bandwidth, etc.).

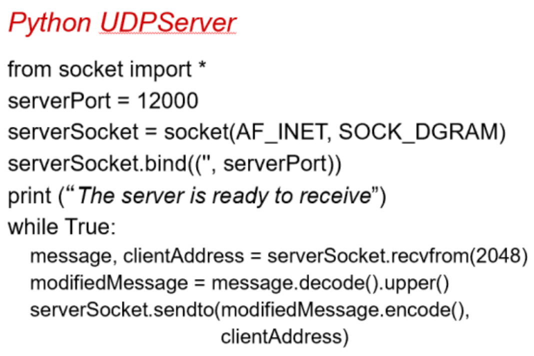
DASH (Dynamic Adaptive Streaming over HTTP)**:** Client can request content at a encoding rate (defined by server) that is sustainable over current bandwidth. Also, can choose best server. Files replicated across servers. A manifest file allows the client to determine where to get its chunks.

Streaming Video Challenges: Continuous playout constraint, network jitter, bandwidth varies, packet loss. A client-side buffer helps smooth out video. Client-side interactivity (play, pause, etc.).

Socket Programming (UDP)

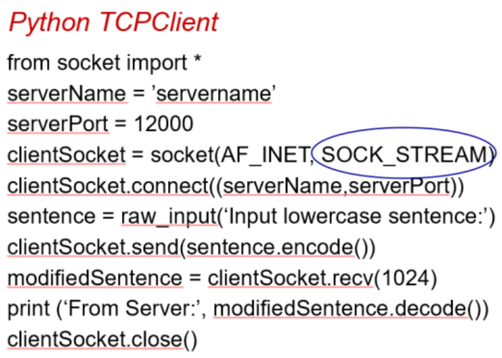

Python UDPClient

```
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()
```

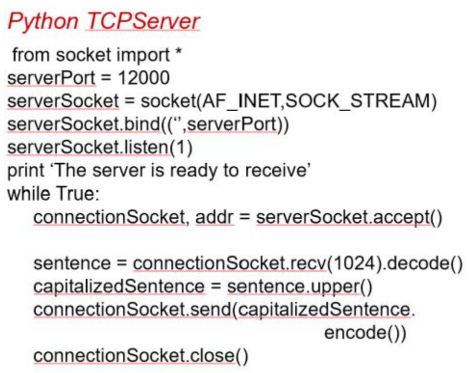

Python UDPServer

```
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)
```

Socket Programming (TCP)


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()
```

Probability a specific user is transmitting & others are not (N = #users)

$$P(transmitting) * P(!\,transmitting)^{N-1}$$

Probability 1 user is transmitting & others are not (N = #users)

$$N * P(transmitting) * P(!\,transmitting)^{N-1}$$

Probability k users are transmitting, and others are not. (N = #users, k = #transmitting)

$$\frac{N!}{k!\,(N-k)!} * P(trans)^{k} * P(!\,trans)^{N-k}$$

Probability k or more users are transmitting (>=) k++ if (>): (N = #users, k = #transmititng)

$$\sum_{k}^{N} P(trans)^{k} * P(!\,trans)^{N-k}$$

Transmission Delay = Packet Size (L) / Transmission Rate (R)

Propagation Delay = Distance (D) / Speed (S)

Traffic Intensity (I) = (Packet Length (L) * Packet Rate (a)) / Trans Rate (R)

Queuing Delay = I * (L / R) * (1 – I)

Packets in buffer after certain time period: (N = #packets, t = time period)

N – floor(t / queuing delay)

Client-Server distribution time (N = #clients, d_{min} = min client download rate, F = file size, U_s = server upload speed)

Max{ (N * F) / U_s , F / d_{min} }

P2P distribution time (N = #clients, d_{min} = min client download rate, F = file size, U_s = server upload speed, U_i = sum of client download speeds)

Max{ F / U_s , F / d_{min} , (N * F) / U_s + U_i }

RTT Delays (3 DNS Servers and Local DNS cache):

- **Time from click to object retrieval:**
  RTT_{cache} + RTT_1 + RTT_2 + RTT_3 + 2(RTT_{HTTP})
- **Referencing 3 objects (no parallel, non-persistent):**
  RTT_{cache} + RTT_1 + RTT_2 + RTT_3 + 2(RTT_{HTTP}) + 2(RTT_{HTTP}) + 2(RTT_{HTTP}) + 2(RTT_{HTTP})
- **Same as above but 5 parallel connections:**
  RTT_{cache} + RTT_1 + RTT_2 + RTT_3 + 2(RTT_{HTTP}) + 2(RTT_{HTTP})
- **Same as above but persistent**
  RTT_{cache} + RTT_1 + RTT_2 + RTT_3 + 2(RTT_{HTTP}) + RTT_{HTTP}

  1000bits = 1kb = 0.001mb = 1 * 10^{-6} gb