eece 350

# Chapter 2:

## • Application Layer

### • Client / server paradigm :

- **Server :** → always-on host
  → permanent IP address
  → often in data centers, for scaling

- **Clients :** → contact, communicate with server
  → may be intermittently connected
  → may have dynamic IP addresses
  → do not communicate directly with each other
  → examples: HTTP, IMAP, FTP

### • Peer-to-Peer architecture :

→ there's no always-on server
→ All peers (clients) communicate directly with each other.
→ Each peer both requests services and provides services to other peers.
→ self scalability : - New peers joining the network increase service capacity
- system can scale up without a central server.

# • Processes communicating:

- process ⟶ program running within a host.

→ A host is a computer or device (e.g phone, laptop...). When we talk about processes within the same host, we're referring to multiple programs or tasks running on that same computer at the same time. (e.g you could be running your web browser and music player on the same laptop).

→ IPC (inter-Process Communication) is the method that allows programs or processes running on the same device to talk to each other.

→ Common IPC methods:

① Shared Memory: 2 programs can share a section of memory. they write and read data from this shared space to communicate with each other.

② Message Passing: Programs can send messages to each other using a message queue.

③ Pipes: A program can send data directly to another program in real-time using Pipes, which connect the two.

## • In Summary :

→ within a host : Processes use IPC to communicate.

→ Between hosts : Processes exchange messages over the network

→ Client-Server : The client process requests, and the server responds.

→ P2P networks : Both clients and server processes are present on each peer.

---

• Sockets : → A socket is a communication endpoint through which a process (program) sends and receives messages over the network

→ It's like a door through which messages go in and out

ex : Imagine that each process has a door (socket). The sending process pushes the message out of its "door" (socket). The message is then transmitted through the network (through the transport layer like TCP/IP) to reach the other side.

• 2 sockets → sending side, receiving side } communicate with each other over the internet and are controlled by : ① Application Developer / ② operating system

## • Addressing Processes :

① To receive Messages, a Process must have an identifier : each process running on a device needs a unique identifier so that messages can be delivered correctly.

② Host Device : Every host device has a unique 32-bit IP address.

→ A process on a host requires both an IP address (identifies host) and a port number (identifies the specific process on the host).

→ the IP address alone is not enough because multiple processes can be running on the same device.

---

## • An application layer protocol defines :

① type of messages exchanged
→ e.g : requests, response

② message syntax
→ what fields in messages and how fields are delineated.

③ message semantics
→ meaning of information in fields

④ Rules for when and how processes send and respond to messages.

⑤ open protocols.
→ defined in RFCs, everyone has access to protocol definition, allows for interoperability (HTTP, SMTP).

⑥ Proprietary protocols
→ Skype, Zoom.

## • What transport service does an app need ?

### (1) Data integrity

→ Some apps (file transfer, web transactions) require 100% reliable data transfer

→ other apps (e.g audio) can tolerate some loss.

### (2) timing

→ Some apps (e.g Internet, telephony, interactive games) require low delay to be "effective".

### (3) throughput

→ Some apps (e.g, multimedia) require minimum amount of throughput to be "effective"

→ other apps ("elastic apps") make use of whatever throughput they get.

### (4) Security

→ Encryption, data integrity ---

## • Internet transport protocol services :

### ① TCP :

→ reliable transport between sending and receiving process

→ flow control : sender won't overwhelm receiver

→ congestion control

→ connection - oriented : set up required between client and server processes

→ does not provide : timing, minimum throughput guarantee, security

### ② UDP :

→ Unreliable data transfer between sending and receiving process.

→ doesn't provide : reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

→ **UDP is often used :**

for speed when it's more important than reliability (e.g streaming, online gaming), where loss of data is acceptable.

---

## • Securing TCP : → TLS (Transport Layer Security)

- TLS encrypts the data transmitted over the internet, ensuring that sensitive info cannot be read by outsiders

- TLS provides ① encrypted TCP connections, ② Data integrity, and ③ End - Point Authentication.

## • Web and HTTP :

→ A web page is composed of different objects. These objects can be stored on different web servers.

• Examples of objects in a web page :
  → HTML files (structure of web page)
  → JPEG images
  → Java applets (interactive content)
  → audio files

→ Each object on the webpage is addressed by a URL (Uniform Resource Locator). The URL specifies both the host (where object is located) and the path (location of the object on that host).

ex : www.someschool.edu/someDept/pic.gif.
host name — path name

→ HTTP : (hypertext transfer Protocol) is the protocol used to transfer data on the web. (It operates in the application layer of the internet).

e.g :- Client sends HTTP request to the server.
- the sever processes the request and sends an HTTP response back with the requested content.

- HTTP uses TCP to establish a connection.
  → the client initiates the TCP connection (creates a socket) to the server on port 80.

→ HTTP is "stateless" : the server does not store any information about past client requests.
Each HTTP request is independent, the server doesn't "remember" previous interactions.

- **HTTP connections :**

**2 types :**

**① Non-persistent**

**TCP connection process :**

→ Open TCP connection : the client open a TCP connection to the server.
→ One object is sent over the TCP connection
→ Connection is closed after sending the object.

- If a webpage requires multiple objects, each object requires its own TCP connection. this results in multiple connections being established for a single webpage.

**② Persistent**

**TCP connection Process :**

→ Open TCP connection : the client open a single TCP connection to server.
→ Multiple objects can be sent over this single connection. The client can continue to request and receive multiple objects without opening new connections each time.
→ Once all objects are transferred, connection is closed.

- Persistent is more efficient bcz it reduces the overhead of repeatedly opening and closing connection for each object.

- **Non-Persistent : Response time**

→ RTT : (round-trip-time) refers to the time it takes for a small packet to travel from the client to the server and back.

→ requires 2 RTTs per object.

Response time = 2RTT + file transmission time

1 for RTT to initiate TCP connection ; 1 for HTTP request

- **Persistent :**

→ only one RTT is needed for all referenced objects which can cut the response time in half for web pages with multiples objects.

- **Web Caches and Proxy Servers :**

→ web caches are used to satisfy client requests without involving the origin server, which improves response time and reduces server load.

- **How they work :** ① Browser configuration :
  → users configure their browsers to point to a local web cache.
  → the browser sends all HTTP requests to the cache.

② Cache Behavior:

→ If the object is in the cache: the cache returns the object directly to client

→ If the object is not in cache: the cache sends an HTTP request to the origin server, caches the object once recieved, and returns it to the client.

## • Why web caches is useful:

① reduces Response Time

② Reduces traffic on access link

③ Helps with Internet efficiency.

## • Performance Improvement Strategies:

### Option 1: Buy a Faster Access Link

#### • Performance:

→ the end-to-end delay is calculated by:

> End-to-End = Internet delay + Access link Delay + LAN Delay

- **Option 2: Install a Web Cache:**

→ the web cache reduces traffic going to origin server by serving cached content for frequently requested objects.

→ Installing a web cache is much cheaper compared to upgrading the access link.

---

- **Email Overview:**

- **Three major Components:**

① User Agent (UA):

→ Also known as mail readers

→ Used for composing, editing, and reading email messages

→ Ex: Outlook, Iphone mail client, ...

→ outgoing and incoming messages are stored in the server.

② Mail Servers:

→ Mailboxes store incoming messages for users

→ Message queues hold outgoing (to be sent) messages.

③ Simple Mail Transfer Protocol (SMTP):

→ the protocol used to transfer email messages between mail servers.

## • Comparison (SMTP vs HTTP):

### ① Client push vs Pull:

→ HTTP is client pull protocol, where the client requests resources from the server.

→ SMTP is client push protocol, where the client (sending mail server) pushes messages to the receiving mail server.

### ② Command/Response Interaction:

→ both use ASCII.

### ③ Object handling:

→ In HTTP, each object is encapsulated in its own response message.

→ In SMTP, multiple objects (such as mail components) can be sent in a multipart message.

### ④ Persistent connections:

→ SMTP uses persistent connections

→ HTTP uses a new connection for each object retrieval.

## • Retrieving Email: Mail access Protocols

### ① SMTP

→ used for delivering and storing email messages on the receiver's email server.

→ handles the sending of email from the sender's server to receiver's server.

### ② Mail Access Protocol.

→ once the email is delivered and stored, mail access protocols are used to retrieve the email from the receiver's email server.

→ some common mail access protocoles:

- IMAP: allows the client to retrieve, delete, and organize emails stored on the server. It supports folders of stored messages and allows actions like retrieval and deletion on the serve-side
- HTTP: Many email services like Gmail, Hotmail, and Yahoo Mail use a web-based interface that allows clients to access their email using HTTP.
  → these services rely on IMAP to retrieve email msgs, while SMTP is still used to send emails.

---

## • DNS: Domain Name System

→ Distributed database implemented in hierarchy of many name servers.

→ DNS is an application-layer protocol where hosts and DNS servers communicate to resolve names (mapping domain names to IP addresses and vice-versa.

→ The DNS is like a phone book that helps you find someone's phone number by their name, it helps you find a website IP address by using its domain name.

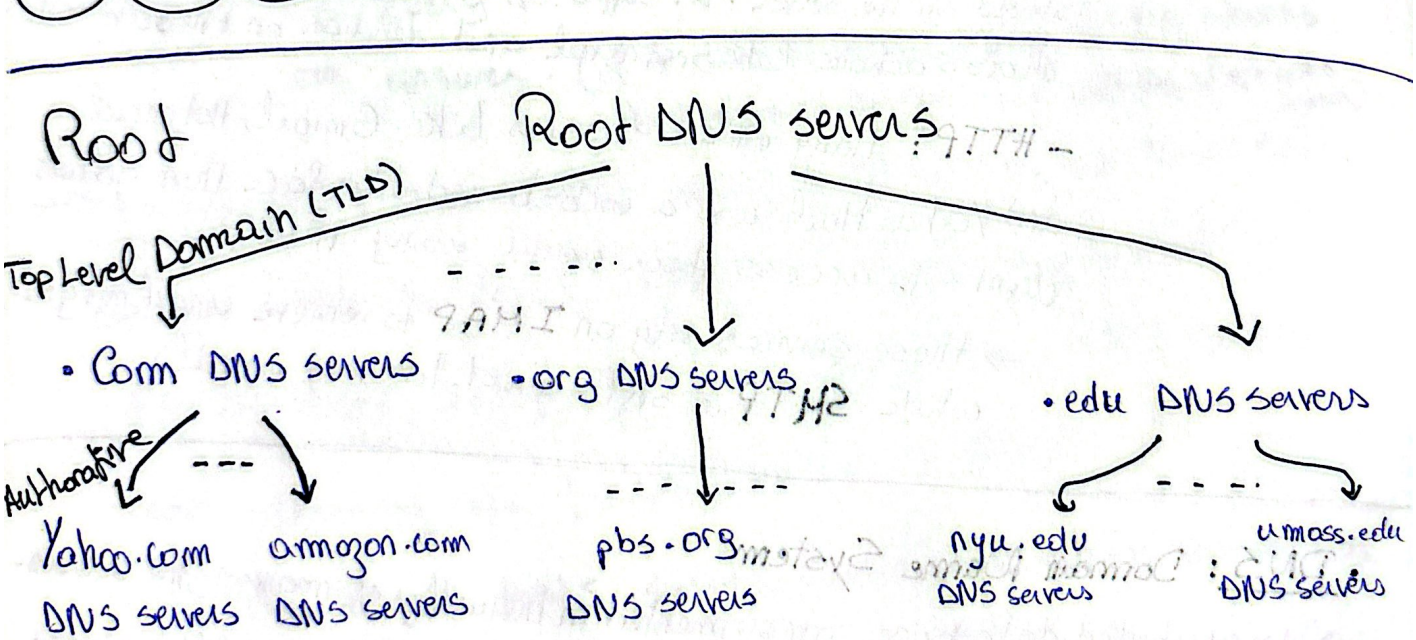
- how it works in general:

When you type a website's name, like google.com, into your browser, the computer doesn't understand names. The DNS translate those names into IP-addresses that the computer can understand:

① You type "www.google.com" into your browser.

② Your browser asks the DNS server: "Where is google.com?"

③ the DNS server looks up the IP for google.com and sends it back to your browser.




→ Client wants IP for www.amazon.com, 1st approx.

1- Client queries root server to find .com DNS server.

2- Client queries .com DNS server to get amazon.com DNS server

3- Client queries amazon.com DNS server to get IP for www.amazon.com

## • Iterative vs Recursive query:

→ Iterative : the client sends a query to the local DNS server, which then contacts other DNS servers to resolve the the domain, passing the query along until it reaches the authoritative server. Each server provides the next server's address until the client gets the final IP address

→ the client sends the query to the local DNS server, if it doesn't know the answer, it passes the query to other servers until the client receives an answer. (if it doesn't know the local DNS returns the address of the next server and the client asks again)

→ Recursive : the client sends a query to the local DNS server, and the local server takes full responsibility for resolving the query by querying other DNS servers and returning the final answer to client.

→ the local DNS server handles the entire resolution process and sends the final result directly to the client.

## • DNS Caching:

→ ~~resolver~~ improves performance by storing previously resolved name-to-IP mappings. Once a name server resolves an address, it stores (caches) the mapping so that subsequent queries can be answered faster, as they don't need to go through the full process again.

→ Caching improves response time.

→ Cache entries only stay for a limited time and will expire after a specific time (called Time To Live).

## • DNS Records:

RR format: (name, value, type, ttl)

① type A
→ name: the hostname (e.g www.example.com)
→ value: IP address

② type NS (name Server Record)
→ name: domain (e.g example.com)
→ value: hostname of the authoritative name server for that domain (where the domain's information is stored)
e.g. ns1.example.com.

③ type CNAME (canonical name Record)
→ name: alias for another domain (the "real" name)
→ value: the actual canonical domain name it refers to.
(e.g server.example.com)

④ type MX (Mail Exchange Record)
→ value: name of SMTP mail server associated with domain
(e.g. mail.example.com)

## · File distribution: Client-server vs P2P:

→ $u_s$ = server upload capacity
→ $d_i$ = peer i download capacity
→ $u_i$ = peer i upload capacity
→ F = size

### · Client-Server:

→ Server transmission:

• the server needs to sequentially upload N copies of the file:

→ the time to send one file copy is: $\frac{F}{u_s}$

→ total time to send all N copies is: $N \times \left(\frac{F}{u_s}\right)$

→ client transmission:

• Each client must download the file from the server.

→ the download time for each client is given by the minimum download rate $d_{min}$ and download time is: $\frac{F}{d_{min}}$

→ So the total time to distribute the File F to N clients using client-server approach is:

$$D_{cs} \geq \max\left(\frac{NF}{u_s}, \frac{F}{d_{min}}\right)$$

- **P2P:**
  - server transmission: must upload at least one copy:
    → time to send one copy = $\frac{F}{u_s}$
  - client: each client must download file copy
    → min client download time = $\frac{F}{d_{min}}$
  - clients: as aggregate must download NF bits
    → max upload rate (limiting max download rate) is: $u_s + \sum u_i$

So:

→ time to distribute F to N clients using P2P:

$$D_{P2P} \geq \max\left\{\frac{F}{u_s}, \frac{F}{d_{min}}, \frac{NF}{(u_s + \Sigma u_i)}\right\}$$

## • Content Distribution Networks (CDN):

→ Streaming content such as videos to hundreds of thousands of users at the same time can be very challenging because of the resources required to handle such a large number of requests.

• Option 1: Single Large "Mega Server".

→ In this mode, there's only one large server handling the distribution of content. If the server goes down, everyone will lose access.

→ There will be network congestion

→ long and congested Path to distant clients

· Option 2 : Distribute content via Multiple servers (CDN)

→ Instead of relying on a single mega-server, CDNs use multiple geographically distributed servers to store and serve content to users. The idea is to bring the content physically closer to the end users to reduce latency and distribute the load efficiently.

· ex of how Netflix work :

→ Netflix stores copies of its content on its Open Connect CDN nodes located globally. These nodes are part of the CDN, which allows Netflix to deliver content from various locations close to users.

→ When a user wants to watch content, their device sends a request for the content to the service provider (in this case Netflix).

→ In response, Netflix's service provider returns a manifest file : A manifest file is essentially a list that includes details about the content, such as different versions, bitrates, qualities that can be streamed. This file helps user's device choose the best version of the content based on the user's internet connection speed and other.

→ After recieving manifest file, the client device retrieves the content at the highest supported rate.

→ If the network path between the user and CDN servers is congested (slow/unreliable), the client may switch to a lower bitrate or request a different copy of the content from another nearby server. This flexibility ensures that the user can still stream the content, even under less-than-ideal network conditions.