NOTES - UNIT IV

Instructor: Ankur Nahar

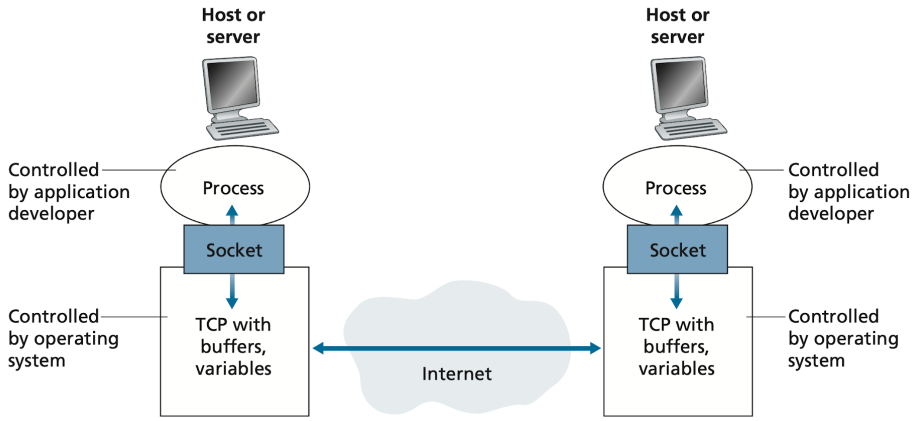
Course: CS2002: Computer Networks

Topic: Apllication Layer

---

1 Principle of Network Applications

- **Client-Server Architecture**: In a client server architecture, there is a dedicated server that services requests from multiple client hosts. Clients do not directly communicate with each other but interact with the server. The server has a fixed, well-known IP address. Examples include the Web, FTP, Telnet, and email.
- **Peer-to-Peer (P2P) Architecture**: In a P2P architecture, there is minimal reliance on dedicated servers in data centers. Peers, which are intermittently connected hosts, communicate directly with each other without a dedicated server intermediary.
- **P2P Scalability**: P2P architectures offer self scalability, with peers contributing service capacity by distributing files to other peers. They are cost effective and do not require significant server infrastructure.
- **Challenges of P2P Architectures**: P2P applications face challenges related to security, performance, and reliability due to their highly decentralized structure.
- **Communication Between Processes**: Processes running on different end systems communicate with each other by exchanging messages across the computer network.
- **Socket**: Messages sent between processes must pass through the network using a software interface called a 'socket'. A socket is analogous to a door through which a process sends and receives messages.
- **Socket Communication**: Sockets serve as the interface between the application layer and the transport layer within a host. They are the 'Application Programming Interface (API)' for building network applications.
- **Addressing Processes**: To send messages between processes, the receiving process's address needs to be specified. This includes the host's IP address and a destination port number. The IP address identifies the host, while the port number identifies the specific receiving process.
- **Port Numbers**: Popular applications are assigned specific port numbers (e.g., Web server on port 80, mail server on port 25)



2 Transport Services (TCP and UDP)

- **Choosing a Transport Layer Protocol**: When developing an application, you must choose a transport layer protocol that suits your application's needs. The choice is typically based on services such as reliable data transfer, throughput, timing, and security.
- **Reliable Data Transfer**: Reliable data transfer ensures that data sent by one end is delivered correctly and completely to the other end. Some applications require this service to prevent data loss, while others like multimedia apps can tolerate some loss.
- **Throughput**: Throughput is the rate at which bits are delivered from the sender to the receiver in a communication session. Bandwidth sensitive applications need guaranteed throughput.
- **Timing**: Timing guarantees are essential for real time applications like Internet telephony, teleconferencing, and multiplayer games. Low delay is crucial for their effectiveness.
- **Security**: Transport protocols can provide security services like encryption for confidentiality, data integrity, and end-point authentication.
- **UDP and TCP**: The Internet offers two transport layer protocols: UDP and TCP. UDP is lightweight and provides unreliable data transfer, while TCP offers reliable data transfer and connection oriented services.
- **Services Not Provided**: Neither UDP nor TCP provide throughput or timing guarantees, which means the Internet cannot guarantee specific timing or throughput for time sensitive applications.

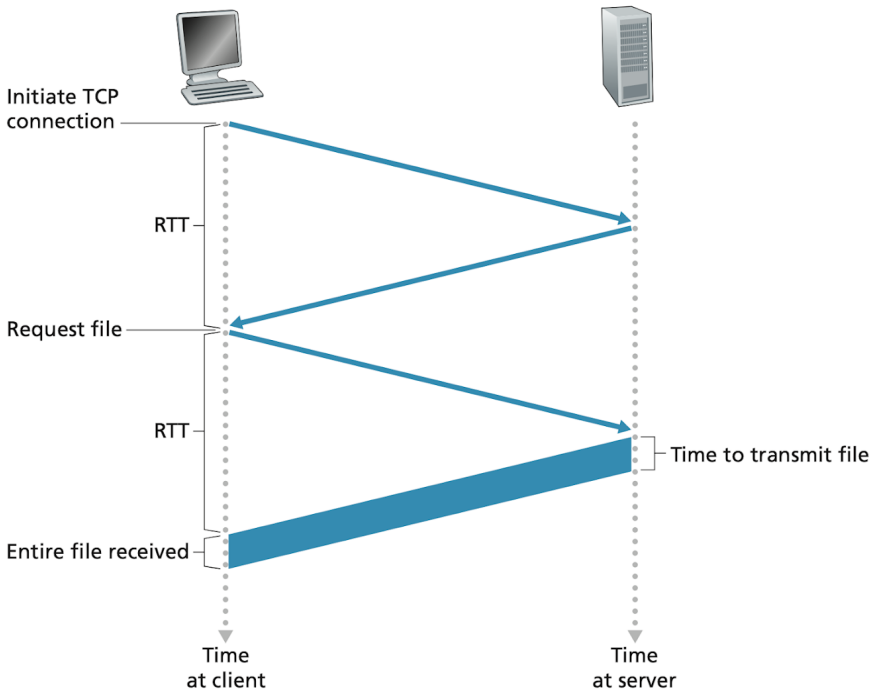
3 Application Layer Protocol: Web and HTTP

- **HTTP**: HTTP (HyperText Transfer Protocol) is the primary application layer protocol of the World Wide Web. It relies on client and server programs that communicate by exchanging HTTP messages. Web pages are composed of objects, which are individual files with unique URLs.

- **Web Page Structure**: A web page typically includes a base HTML file and referenced objects like images, stylesheets, and videos. Objects are identified by URLs, which consist of a hostname and a path name.
- **HTTP and TCP**: HTTP uses TCP (Transmission Control Protocol) as its underlying transport protocol. Clients initiate TCP connections with servers to exchange HTTP messages.
- The HTTP client first initiates a TCP connection with the server. Once the connection is established, the browser and the server processes access TCP through their socket interfaces.
- HTTP need not worry about lost data or the details of how TCP recovers from loss or reordering of data within the network.
- **Stateless Protocol**: HTTP is a stateless protocol, meaning servers *don't store client specific information*. If a client requests the same object multiple times, the server doesn't remember previous requests.
- **HTTP Versions**: HTTP/1.0 and HTTP/1.1 are common versions, with HTTP/1.1 supporting persistent connections. Newer versions like HTTP/2 are also emerging.

3.1 Non-Persistent and Persistent Connections

In many Internet applications, clients and servers exchange multiple request-response messages. These can occur back-to-back, at regular intervals, or sporadically. When using TCP as the transport protocol, a crucial design decision arises: whether to use a separate TCP connection for each request-response pair (non-persistent connections) or to reuse the same connection for multiple exchanges (persistent connections).



3.1.1 HTTP with Non-Persistent Connections

- Each TCP connection is used for exactly one request and one response.
- After the server sends the requested object, it closes the TCP connection.
- For a webpage with multiple objects (e.g., HTML file + images), multiple separate TCP connections are established.
- Example: If a page has 1 HTML file and 10 images, 11 TCP connections will be created sequentially or in parallel.
- Modern browsers allow parallel TCP connections (typically 5–10), reducing total load time.
- Each connection requires a 3-way TCP handshake (1 RTT) and a request-response round (another RTT), totaling approximately 2 RTTs per object.
- Overhead increases due to frequent connection setup/teardown and repeated allocation of TCP buffers and state variables.

3.1.2 HTTP with Persistent Connections

- Introduced in HTTP/1.1 and used by default.
- A single TCP connection is maintained for multiple HTTP requests and responses.
- All objects of a webpage can be transferred over a single connection.
- Reduces the total number of TCP handshakes and eliminates repeated connection setups.
- Lowers server load by minimizing memory and state management for connections.
- Supports pipelining: multiple requests can be sent without waiting for responses.
- The server sends responses back-to-back in the same order as requests.
- Connection remains open until it is unused for a configurable timeout period.

3.1.3 Performance Comparison and Advancements

- Non-persistent connections incur higher delay (2 RTTs per object) and more overhead.
- Persistent connections reduce RTT delays and are more resource-efficient.
- HTTP/2 further improves performance by enabling multiplexing: interleaving multiple request-response pairs over a single connection.
- HTTP/2 also allows prioritization of messages within a connection.

In conclusion, persistent connections are preferred for modern web applications due to their efficiency, scalability, and reduced latency. Non-persistent connections, while simpler, are less suited for today's complex and media-rich websites.

3.2 HTTP Message Format:

- HTTP messages have two types: **request messages** and **response messages**. Request messages include a method (e.g., GET), URL, and HTTP version, followed by header lines. Response messages include a **protocol version, status code** (e.g., 200 OK), and **header lines**, followed by the **entity body**.
- **Header Lines**: Header lines provide additional information in HTTP messages. Examples of request headers include *Host, Connection, User-agent, and Accept-language*. Examples of response headers include *Connection, Date, Server, Last-Modified, Content-Length, and Content-Type*.
- **Status Codes**: Status codes (e.g., 404 Not Found) indicate the result of an HTTP request.
- **HTTP Methods**: HTTP includes methods like GET, POST, PUT, and DELETE for different types of requests and actions.
- **Entity Body**: The entity body in HTTP messages contains data related to the request method.

3.2.1 Status Codes

Status codes are categorized into five groups based on the first digit:

- **1xx:** Informational
- **2xx:** Successful
- **3xx:** Redirection
- **4xx:** Client errors
- **5xx:** Server errors

Common HTTP Status Codes

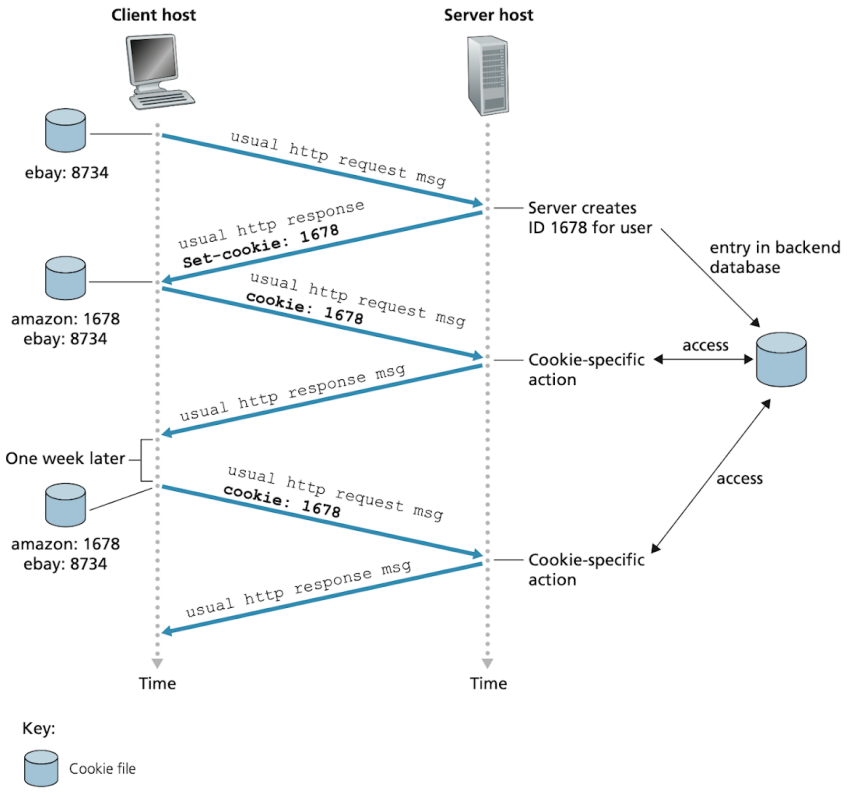
Status Code	Description
100 Continue	Request headers received, continue sending the body.
200 OK	Request successful, response contains the result.
201 Created	Successful response to a PUT request.
301 Moved Permanently	Permanent redirection to a different URL.
302 Found	Temporary redirection to a different URL.
304 Not Modified	Use cached copy of the resource if not modified.
400 Bad Request	Invalid HTTP request submitted.
401 Unauthorized	HTTP authentication required.
403 Forbidden	No access allowed to the requested resource.
404 Not Found	Requested resource does not exist.

405 Method Not Allowed	Unsupported method for the specified URL.
413 Request Entity Too Large	Request body exceeds server's handling capacity.
414 Request URI Too Long	URL used in the request is too large.
500 Internal Server Error	Server encountered an error fulfilling the request.
503 Service Unavailable	Application accessed via the server is not responding.

3.2.2 HTTP Methods

- When attacking web applications, the commonly used methods are **GET** and **POST**.
- **GET** is used to retrieve resources and can send parameters in the URL query string.
- GET URLs are visible, logged, and transmitted in the `Referer` header, so sensitive information should not be included in the query string.
- **POST** is used to perform actions and allows parameters in both the URL and message body.
- POST parameters in the message body are not included in bookmarks, logs, or the `Referer` header.
- The browser warns users when navigating back to a page accessed with POST to prevent unintended actions.
- POST should be used for performing actions to prevent unintentional repetitions.
- **HEAD**: Similar to GET but without a message body, used to check resource presence before a GET request.
- **TRACE**: Returns the exact contents of the received request, helpful for diagnosing proxy manipulation.
- **OPTIONS**: Requests available HTTP methods for a resource; server responds with an `Allow` header listing the methods.
- **PUT**: Attempts to upload a resource to the server using the request body, potentially exploitable for attacks like arbitrary script execution.
- Other HTTP methods exist but are not directly relevant to attacking web applications.

However, the presence of certain dangerous methods can make a web server vulnerable to attack.



3.3 Cookies

- There are situations where web sites need to identify users, for security or personalization purposes. HTTP uses cookies to achieve user identification and tracking.
- Cookies have four components: a *cookie header* in HTTP *response* and *request* messages, a *cookie file* on the user's end system, and a *back end database* on the web site.
- Cookies work by sending a unique identification number in a *Set-cookie* header from the server to the user's browser.
- The browser stores the identification number and sends it back to the server in a Cookie header with subsequent requests.
- Cookies are used to track user activity and offer personalized services, like shopping carts or recommendations.
- Users can be identified over multiple sessions by maintaining the same identification number in cookies.
- Cookies are controversial due to potential privacy concerns, as websites can gather and potentially sell user information.

3.4 Web Caching

- Web caches, or proxy servers, handle HTTP requests on behalf of origin servers.

- Caches store copies of requested objects, reducing the need to fetch them from the origin server.
- Users can configure browsers to direct requests through caches for faster responses.
- Caches serve as both servers (to clients) and clients (to servers) in the caching process.
- Installed by ISPs and institutions, caches enhance performance and lower bandwidth costs.
- Benefits include faster responses, reduced bandwidth upgrades, and decreased overall Internet traffic.
- Content Distribution Networks (CDNs) use distributed caching to localize content delivery.
- An HTTP request message is a so called conditional GET message if
  1. the request message uses the GET method and
  2. the request message includes an If-Modified-Since: header line.
- *Conditional GET* checks for freshness by comparing the *If-Modified-Since* header with object modification date.
- If an object hasn't changed, a *304 Not Modified response* allows the cache to serve the locally cached object.

3.5 HTTP/2

The primary goals for HTTP/2 are to reduce perceived latency by enabling request and response multiplexing over a single TCP connection, provide request prioritization and server push, and provide efficient compression of HTTP header fields.

- **HTTP/2 Motivation:** HTTP/1.1's persistent TCP connections caused HOL blocking. Browsers used multiple parallel TCP connections to work around this issue.

  Note: Head of Line (HOL) blocking: occurs when a web page has a large video clip and numerous small objects. With a slow bottleneck link, the video clip causes delays for small objects queued behind it.

- **HTTP/2 Solution (Framing):** Reduces the need for parallel TCP connections by breaking messages into frames and interleaving them, significantly reducing user perceived delay. Includes binary frame encoding for efficiency.

  Note: The ability to break down an HTTP message into independent frames, inter leave them, and then reassemble them on the other end is the single most important enhancement of HTTP/2.

- **Message Prioritization:** Developers assign weights (1-256) to messages, and the server prioritizes higher weight responses. Clients can specify message dependencies.

- **Server Push:** Enables sending additional objects to the client without explicit requests, reducing latency.
- **HTTP/3 and QUIC:** QUIC, a new transport protocol over UDP and supports features like message multiplexing, is used for HTTP/3. This streamlined design incorporates HTTP/2 features and leverages QUIC's advantages.

4 Application Layer Protocol: SMTP

A typical message starts its journey in the sender's user agent, then travels to the sender's mail server, and then travels to the recipient's mail server, where it is deposited in the recipient's mailbox. Reattempts are often done every 30 minutes

4.1 Email Components

- **User Agents:** Tools like Microsoft Outlook, Apple Mail, and Gmail, allowing users to manage emails.
- **Mail Servers:** The central infrastructure, hosting mailboxes for recipients like Bob.
- **SMTP (Simple Mail Transfer Protocol):** The principal protocol to send emails between servers.

4.2 SMTP Basics

- SMTP transfers messages between sender and recipient mail servers at 'Port 25'.
- The client (sender's server) initiates a connection to the recipient's server via TCP.
- It introduces the sender and recipient, transmits the message, and uses a persistent connection for multiple messages.

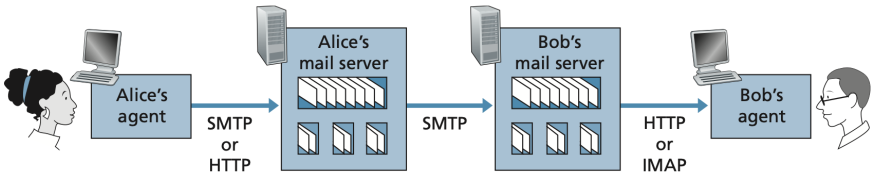
```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr ...  Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ...  Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

4.3 Mail Message Structure

- Email messages consist of a 'header' and a 'body'.
- The header includes sender and recipient information, such as '"From," "To," and "Subject."'. The header lines and the body of the message are separated by a blank line (that is, by CRLF).
- These headers are distinct from SMTP commands used for server handshake communication.

```
From:  alice@crepes.fr
To:  bob@hamburger.edu
Subject:  Searching for the meaning of life.
```

4.4 Mail Access Protocols



Bob's user agent can't use SMTP to obtain the messages because obtaining the messages is a pull operation, whereas SMTP is a push protocol.

- Users retrieve their email messages from a shared mail server using either HTTP or IMAP.
- HTTP is often used for web based email clients like Gmail, while IMAP is common with clients like Microsoft Outlook.
- Both the HTTP and IMAP approaches allow to manage folders, move messages to folders, delete messages, mark messages as important, and so on.

5 Application Layer Protocol: DNS

- DNS is an essential service that translates human friendly hostnames into IP addresses.
- It's a distributed database and an application layer protocol, implemented with DNS servers, often running BIND software and runs over UDP and uses port 53.
- DNS services include:
  - **hostname aliasing**: host with a complicated canonical hostname can have one or more alias names.
  - **Mail Server Aliasing**: DNS resolves alias hostnames to canonical forms for mail servers and retrieves corresponding IP addresses.

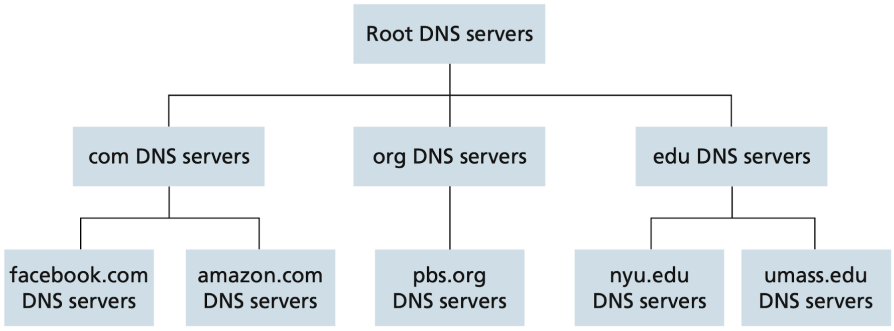
- **Load Distribution**: DNS balances traffic among replicated servers by rotating IP addresses within replies, ensuring even distribution. This technique is also applied to email servers with shared alias names.

5.1 How DNS Works: High Level Overview

gethostbyname() is the function call that an application calls in order to perform the translation.

- DNS operates through query and reply messages using UDP datagrams on port 53.
- DNS queries involve multiple servers globally distributed.
- A simple centralized design for DNS is not feasible due to scalability issues.
- Issues with centralized design: 'single point of failure,' 'high traffic volume', 'distant database', and 'maintenance'.
- DNS uses a hierarchical structure and a distributed database., to handle the vast number of hosts on the Internet.

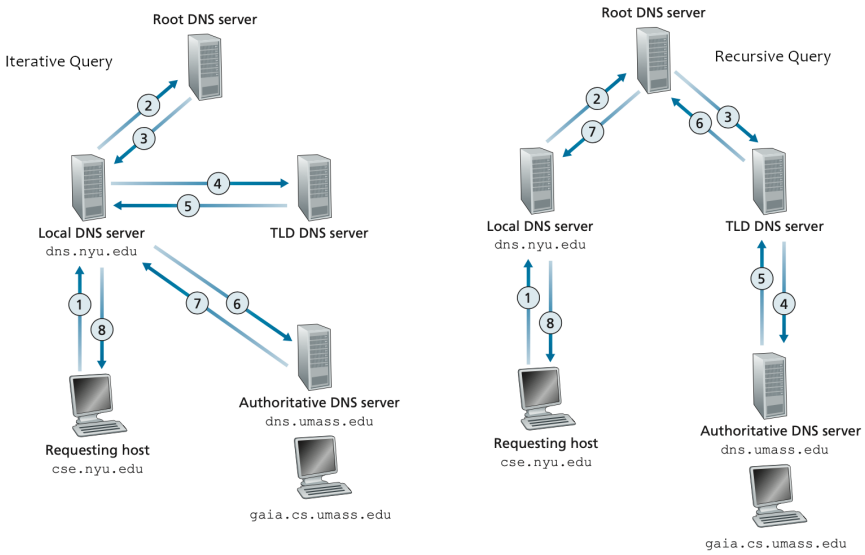
5.2 Distributed, Hierarchical Database



- DNS uses three classes of servers:
  - Root DNS servers,
  - top level domain (TLD) DNS servers, and
  - authoritative DNS servers.
- Root DNS servers provide IP addresses for TLD servers. TLD servers provide IP addresses for authoritative DNS servers Authoritative DNS servers store DNS records for specific organizations.
- A local DNS server, specific to an ISP, also plays a crucial role in DNS queries. It cache DNS information to reduce query traffic and improve performance.
- When a host makes a DNS query, the query is sent to the local DNS server, which acts a proxy, forwarding the query into the DNS server hierarchy.

- DNS extensively utilizes caching to enhance performance. These are stored temporarily and it allows DNS servers to quickly respond to subsequent queries for the same hostname.

5.3 Recursive vs Iterative DNS Queries



5.4 DNS Records and Messages

- DNS servers store **resource records (RRs)** in the distributed database.
- A resource record (RR) is a four-tuple: `(Name, Value, Type, TTL)`.
- **TTL (Time to Live)** determines when a resource should be removed from a cache.
- Types of resource records:
  - **Type = A**: Maps hostname to IP address.
  - **Type = NS**: Maps a domain to the hostname of an authoritative DNS server.
  - **Type = CNAME**: Provides the canonical name for an alias hostname.
  - **Type = MX**: Maps to the canonical name of a mail server with an alias hostname.
- To obtain the canonical name for a mail server, a DNS client queries for an **MX record**; to obtain the canonical name for another server, it queries for the **CNAME record**.

5.5 DNS Messages

- DNS messages have a **header section** with fields like query/reply flags, recursion flags, and more.

- DNS messages consist of the following sections:
  - **Question section**
  - **Answer section** (contains resource records)
  - **Authority section**
  - **Additional section**
- A **1-bit query/reply flag** indicates if the message is a query (0) or a reply (1).
- A **1-bit authoritative flag** is set in a reply when the DNS server is authoritative for the queried name.
- A **1-bit recursion desired flag** is set when a client requests the DNS server to perform recursion if it doesn't have the record.
- A **1-bit recursion available flag** is set in a reply if the DNS server supports recursion.

5.6 Inserting Records to DNS Database

A registrar is a commercial entity that verifies the uniqueness of the domain name, enters the domain name into the DNS database (as discussed below), and collects a small fee from you for its services.

- To register a domain name, you need to provide registrar with DNS server names and IP addresses. Registrar enters 'Type NS' and 'Type A' resource records for 'authoritative' DNS servers into 'TLD' servers.
- Additional resource records, like Type A and Type MX, must be added for Web and mail servers.

6 P2P

Peer-to-Peer (P2P) is a distributed network model where each node (peer) acts as both a client and a server. It allows users to share resources such as files, bandwidth, and processing power without relying on a centralized server. It is widely used for file sharing of multimedia files like videos, music, e-books, and games.

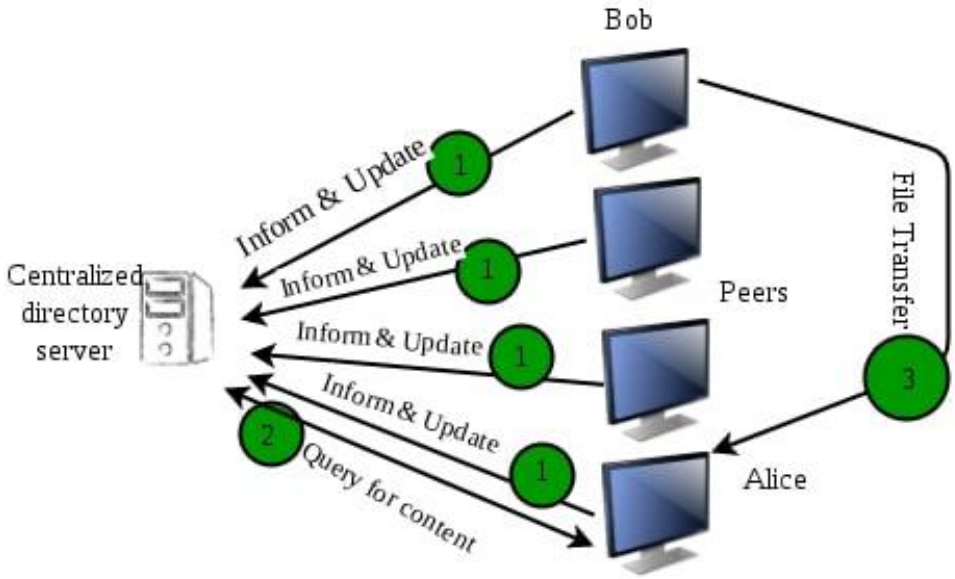
6.1 How Does P2P Work?

- Peers establish TCP or UDP connections to request files from one another.
- Unlike client-server models, there is no central server to handle requests.
- Peers interact directly for both requesting and delivering files.
- A requested file might be available on multiple peers; the IPs of such peers are discovered using the underlying P2P architecture.

6.2 P2P Architectures

6.2.1 Centralized Directory

The Centralized Directory approach to peer-to-peer networking resembles the client-server model in design. It is based on the idea of maintaining a large, centralized server that is responsible for handling the directory services of the entire network. In this model, every peer, upon joining the network, sends its IP address and the list of files it wants to share to this central server. The server periodically queries all the peers to verify their availability and maintain an up-to-date database containing the mapping of files to IP addresses of the peers that host them.



P2P paradigm with a centralised directory

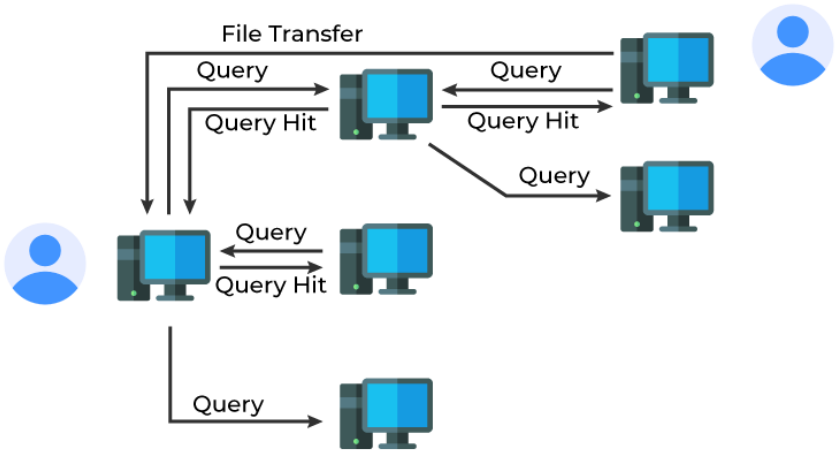
Working: When a new peer wants to request a file, it does not flood the network with queries. Instead, it simply contacts the centralized server with its request. Since the server already maintains the information about which files are stored on which peers, it quickly returns a list of peer IP addresses that contain the requested file. After receiving this list, the requesting peer initiates a direct file transfer with one of the peers from the list. Hence, the server only helps in file discovery, while the actual file transfer occurs between peers.

Limitations: The major drawback of this architecture is that it introduces a single point of failure. If the central server fails or is taken down, the entire network becomes non-functional. Additionally, this model requires a powerful and scalable server capable of handling large amounts of data and frequent updates, which can become a bottleneck in large-scale systems.

6.2.2 Query Flooding

The Query Flooding model uses a fully decentralized system, meaning there is no central server. Instead, all peers (also referred to as nodes) are interconnected in what

is known as an overlay network. In this structure, a path or connection between any two peers is considered an edge, and the entire network forms a dynamic graph-like topology. This model was first implemented in Gnutella, which was among the earliest decentralized peer-to-peer networks.



Working: In this system, when a peer wants to find a file, it sends a query to all of its neighboring nodes. If a neighbor node does not possess the requested file, it forwards the query to its own neighbors. This process continues recursively and is referred to as **query flooding**. When a peer with the requested file receives the query, it responds with a *query hit*, sending back the file details such as name and size to the original requesting peer via the reverse path. If there are multiple hits, the peer may choose any of the responding nodes to download the file directly.

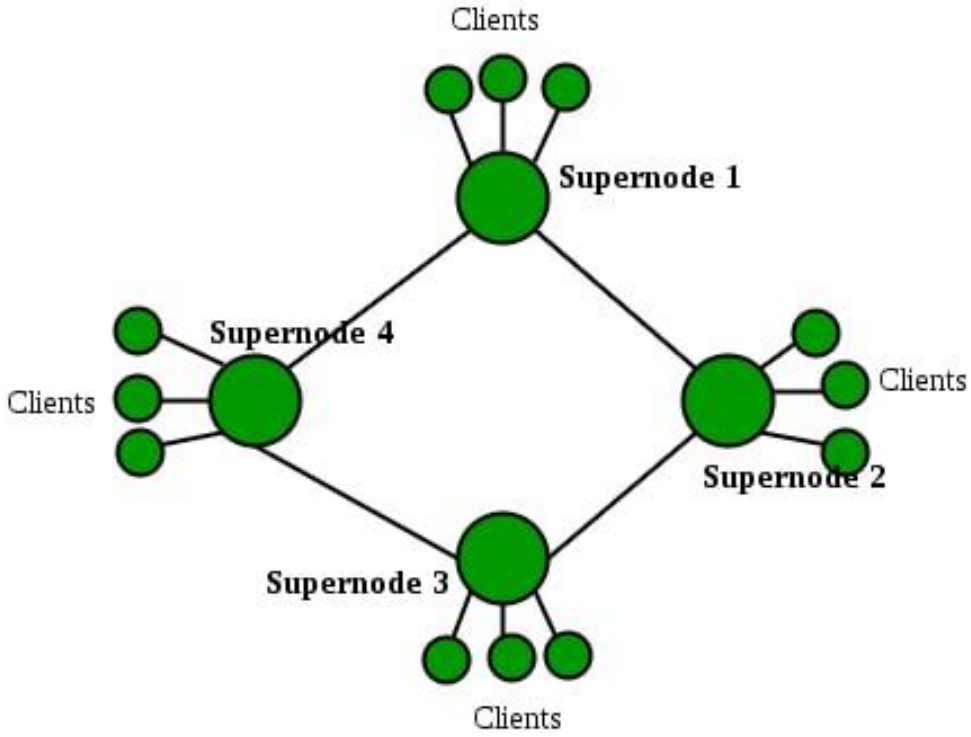
Limitations: The main disadvantage of this architecture is that flooding creates a large amount of network traffic, especially when the file is not widely available. Every node has to process and forward the query until a match is found or a time-to-live (TTL) value expires, which can lead to inefficient use of bandwidth.

6.2.3 Exploiting Heterogeneity

This model represents a hybrid peer-to-peer architecture that combines the benefits of both the centralized directory and query flooding approaches while minimizing their respective drawbacks. In this approach, peers are not treated equally. Instead, the system takes advantage of **heterogeneity** by assigning roles based on the capabilities of each peer.

Architecture: Peers with higher processing power, more bandwidth, and better connectivity are promoted to the role of **supernodes** or **group leaders**. These supernodes act as local directories for a set of connected peers, known as **child nodes**. The child nodes register themselves with their respective supernodes, providing details such as their IP address, connection status, and the list of available files. The supernodes maintain this metadata locally and perform query processing on behalf of their children.

These supernodes are then interconnected in a decentralized overlay network, similar to Gnutella. This allows them to communicate and cooperate in file search and resource discovery.



Working: There are two primary methods for query resolution in this architecture:

1. **Metadata Sharing:** Supernodes periodically exchange or merge their peer directories. This allows them to maintain a larger cache of file indices, improving the speed and success rate of query resolution.
2. **Controlled Flooding:** When a peer sends a query to its supernode, and if the supernode doesn't have the required file in its local directory, it forwards the query to neighboring supernodes. These supernodes continue forwarding the query until a match is found, similar to Gnutella but within a smaller and more capable group. This significantly reduces network congestion.

Advantages: By exploiting peer heterogeneity, this architecture enhances scalability, improves resource utilization, and maintains better fault tolerance than purely centralized systems. It also reduces unnecessary traffic by limiting the scope of flooding to high-capacity nodes.

6.3 Applications of P2P

6.3.1 File Sharing

- Primary use-case for P2P networks.
- Allows multimedia file distribution without centralized control.
- Popular file-sharing tools include BitTorrent, Gnutella, and eDonkey.

6.3.2 Blockchain and Cryptocurrencies

- Blockchain relies on P2P for decentralized ledgers.
- Ensures transparency, fault-tolerance, and redundancy.
- **Example:** Bitcoin, Ethereum.

6.3.3 Direct Messaging and VoIP

- P2P supports real-time, encrypted communication.
- Examples include early versions of Skype for voice calling.

6.3.4 Content Distribution

- Useful for distributing updates or large files to multiple users efficiently.
- The more users involved, the faster the delivery becomes.

6.3.5 Collaborative Applications

- Small teams and research groups can collaborate by sharing resources.
- Examples: Collaborative design tools, P2P-based Git systems.

6.4 Security Concerns in P2P File Sharing

- Sensitive files may be unintentionally shared.
- Malware and viruses can spread through shared files.
- Unauthorized access to systems via open ports.
- Identity spoofing and man-in-the-middle attacks are possible.

6.4.1 Security Measures

- Delete or protect sensitive files before enabling sharing.
- Reduce or remove P2P programs on systems handling sensitive data.
- Monitor for unauthorized P2P activity.
- Block unapproved P2P software at the firewall.
- Use strong access controls and authentication.
- Encrypt data using SSL/TLS during transfer.
- Keep all software and P2P clients up-to-date.
- Educate users about P2P risks and safe practices.
- Implement intrusion detection and prevention systems.
- Segment the network to contain potential breaches.

- Use Data Loss Prevention (DLP) tools to stop leakage of sensitive files.

7 Video Streaming and Content Distribution Networks

Streaming prerecorded video now constitutes the majority of traffic in residential ISPs, with Netflix and YouTube alone accounting for over 50% of traffic in 2015. This section overviews the architecture and techniques behind modern video streaming services.

7.1 Internet Video

- **Definition:** Streaming of prerecorded video (e.g., movies, TV shows, user-generated content).
- **Examples:** Netflix, YouTube, Amazon Prime Video, Youku.
- **Video Structure:**
  - Sequence of images (24–30 fps).
  - Encoded with pixel data (luminance + color).
  - Highly compressible; compression trades video quality for bit rate.
- **Bit Rate Considerations:**
  - Compressed video: 100 kbps (low quality) to 3 Mbps+ (HD).
  - 4K: 10 Mbps+.
  - Example: 2 Mbps $\times$ 67 min $\Rightarrow$ 1 GB.
- **Performance Metric:** Average end-to-end throughput $\geq$ video bit rate.
- **Multiple Versions:** Videos can be stored in various qualities (e.g., 300 kbps, 1 Mbps, 3 Mbps).

7.2 HTTP Streaming and DASH

7.2.1 HTTP Streaming

- Videos stored as regular files on HTTP servers.
- Client sends HTTP GET request, receives file over TCP.
- Client buffers data and begins playback when threshold met.
- Limitation: All clients receive same version regardless of bandwidth.

7.2.2 DASH (Dynamic Adaptive Streaming over HTTP)

- Video encoded into multiple versions (different bit rates).
- Client downloads small video chunks (few seconds long) via HTTP GET.
- Enables **real-time adaptation** to network conditions and user capabilities.

- **Manifest file (MPD)**: Lists available versions and their URLs.
- Client uses HTTP GET with byte ranges to request chunks.
- Adaptive quality selection based on:
  - Available bandwidth.
  - Current buffer level.
- Client dynamically switches quality versions based on real-time conditions.

7.3 Content Distribution Networks (CDNs)

With the rise in Internet usage, delivering high-quality, fast-loading content to a global audience has become a challenge. For instance, YouTube has over 2 billion users, and Netflix over 160 million. Simply hosting content on a single large data center poses several problems:

7.3.1 What is a CDN?

A **Content Distribution Network (CDN)** is a geographically distributed network of servers that works together to deliver content faster and more reliably to users worldwide.

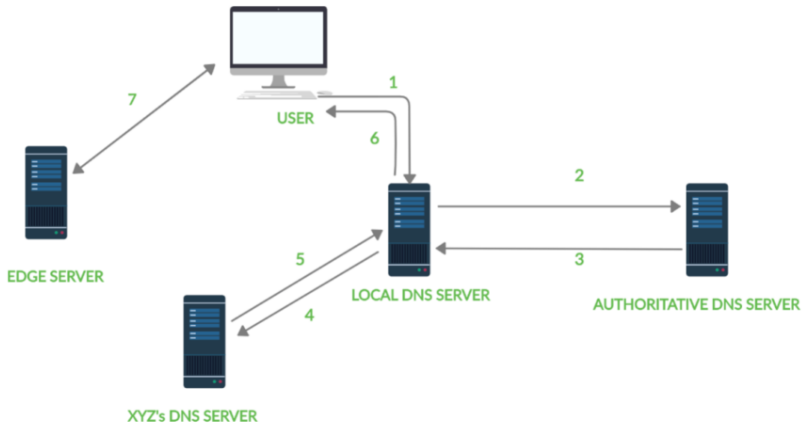
- Maintains globally distributed servers.
- Stores cached versions of web content.
- Redirects users to the nearest server (edge server) for faster delivery.

7.3.2 How Does a CDN Work?

To reduce the physical distance between users and the content, a CDN caches content at multiple **Points of Presence (PoPs)**. Each PoP has multiple **edge servers** responsible for delivering cached content to nearby users.

7.3.3 Example:

- Assume your origin server is in Australia.
- A user in India clicks a video on your website.
- The request is routed via:
  - Local DNS $\rightarrow$ Authoritative DNS $\rightarrow$ CDN DNS.
  - CDN DNS identifies the closest edge server (in or near India).
  - Edge server delivers the content.
- Subsequent requests by users in the same region go directly to the identified edge server.



7.3.4 Need for CDNs or Challenges with Centralized Streaming

- Avoids:
  - **Latency:** A data center in the USA serving users in India results in slower content delivery.
  - **Single Point of Failure:** Centralized data centers are prone to outages.
  - **Bandwidth Wastage:** Frequently accessed content from distant regions causes repetitive traffic over long paths.
- Solution: Distribute content via geographically dispersed servers.

7.3.5 Benefits of CDN

- **Improved Security:** Includes DDoS protection, TLS/SSL certificates.
- **Increased Availability:** Handles large traffic spikes and hardware failures gracefully.
- **Faster Load Times:** Reduced latency due to local caching.
- **Reduced Bandwidth Costs:** Caching decreases load on origin servers, lowering data transfer expenses.

7.3.6 Types of CDNs

- **Private CDN:** Owned by content provider (e.g., Google for YouTube).
- **Third-party CDN:** External service providers (e.g., Akamai, Limelight).

7.3.7 CDN Deployment Strategies

1. **Enter Deep:**
   - Deploy clusters within access ISPs.
   - Minimizes user-to-server hops, reduces latency.

- Example: Akamai.

2. **Bring Home:**
   - Fewer, large clusters at Internet Exchange Points (IXPs).
   - Easier maintenance, but may result in higher delay.
   - Example: Limelight.

7.3.8 Content Replication and Caching

- Not all videos are pushed to every server.
- Popular content is replicated based on demand.
- Pull-based model: On-demand replication to local clusters.
- Uses eviction policies like Web caching to remove stale content.

7.3.9 CDN Operation with DNS Redirection

- DNS is used to redirect clients to appropriate CDN server clusters.
- Example (NetCinema using KingCDN):
  1. User clicks a video link (e.g., `video.netcinema.com/6Y7B23V`).
  2. Local DNS sends query to NetCinema's authoritative DNS.
  3. Recognizing "video", DNS redirects query to KingCDN (e.g., `a1105.kingcdn.com`).
  4. KingCDN resolves and returns IP address of a suitable CDN node.
  5. Client connects directly to that node and begins video streaming.

7.3.10 Limitations of DNS-based Redirection

- Mapping is often done based on LDNS location, not the actual user device.
- A mismatch may occur if the user and their LDNS server are in different locations.
- Leads to suboptimal server selection and degraded performance.
- Does not account for:
  - Temporary network congestion
  - Load imbalance across clusters

7.3.11 Cluster Selection Strategies

At the core of any CDN deployment is a **cluster selection strategy**—a mechanism for dynamically directing clients to a suitable server cluster or data center within the CDN.

- The CDN first learns the IP address of the client's **Local DNS (LDNS)** server via the client's DNS lookup.
- Based on the LDNS IP, the CDN selects an appropriate cluster.
- CDNs typically use **proprietary cluster selection algorithms**, but common approaches include:

1. **Geographic Proximity-Based Strategy**
    - Assigns client to the geographically closest cluster (shortest distance "as the bird flies").
    - Uses commercial geo-location databases like:
        - Quova (2016)
        - MaxMind (2016)
    - Maps LDNS IPs to geographic coordinates.
    - **Advantages:**
        - Simple and effective for many clients.
        - Minimal computational overhead.
    - **Disadvantages:**
        - **Geographic Network proximity:** Closest cluster geographically might not have the shortest or best-performing network path.
        - **Remote LDNS problem:** Many clients use LDNS servers that are geographically distant, skewing location estimation.
        - **Static mapping:** Does not adapt to real-time network conditions (e.g., congestion, outages).
2. **Performance-Based Strategy (Real-Time Measurements)**
    - Selects cluster based on current traffic and path performance (e.g., delay, loss).
    - CDN servers periodically send **probes** (e.g., `ping`, DNS queries) to LDNS servers worldwide.
    - Cluster assignment is dynamic and adapts to:
        - Varying Internet conditions
        - Server load and network congestion
    - **Advantages:**
        - More accurate and responsive to actual network performance.
        - Improves user QoE (Quality of Experience).
    - **Disadvantages:**
        - Some LDNS servers are configured to ignore probe traffic.

- Probing adds computational and communication overhead.