Application Layer

Madiba Hudson-Quansah

Contents

Chapter 1

Principles of Network Applications

Network applications are programs that run on end systems and rely on the network to provide services to users. Examples include web browsers, email clients, and file sharing applications. When writing network applications developers do not need to write software that runs on network-core devices, such as router or link-layer switches.

1.1 Network Application Architectures

Definition 1.1.1: Application Architecture

Designed by the application developer and dictates how the application is structured and how it communicates over the network.

There are two commonly used application architectures:

- Client-Server Architecture
- Peer-to-Peer Architecture

1.1.1 Client-Server Architecture

The server:

- Always-on host
- Permanent IP address, i.e. well-known address
- Often in data centres for scaling
- Serves requests from other hosts (clients)

The client:

- Communicates with server
- May be intermittently connected
- May have dynamic IP address
- Do not communicate directly with other clients

1.1.2 Peer-to-Peer Architecture

There is close to no reliance on dedicated servers in data centres, with clients directly communicating with each other as **peers**.

- Peers are not owned by the service provider

- Peers are intermittently connected with dynamic IP addresses
- Peers directly communicate with each other
- Self-scalable as more peers join the network and bring new service capacity as well as new services demands.
- Relatively cost effective as less server infrastructure is required.

1.2 Process Communicating

Definition 1.2.1: Process

A program running within a host. Processes on the same host can communicate with each other with interprocess communication (IPC). Processes on two different hosts communicate with each other by exchanging messages across the network.

Definition 1.2.2: Message

The data exchanged between processes.

Processes on different hosts communicate by exchanging messages across the computer network.

1.2.1 Client and Server Processes

Definition 1.2.3: Client Process

A process that initiates communication.

Definition 1.2.4: Server Process

A process that waits to be contacted.

Note:-

A process can be both a client and a server. For example, in P2P

Definition 1.2.5: Socket

An endpoint of a communication channel. Analogous to a door in the sense that a socket is where messages enter and leave the host. Sockets can be bound to a specific port number and an IP address, which is used to determine which process a message should be delivered to. Sockets are the interface between the application process and the transport layer protocol, such as TCP or UDP.

Definition 1.2.6: Port Number

A 16-bit number used to identify a socket. Port numbers are used to determine which process a message should be delivered to.

A network applications is made up of pairs of processes that send messages to each other over a network. Each process in a pair can be labelled as client or a server process, sometimes a process can be both a client and a server.

A process sends messages into and receives message from its **socket**. Each socket is identified by a **port number** that is used to determine which process a message should be delivered to.

In the internet hosts are identified by their IP addresses, which are 32-bit unique identifiers for hosts on the internet. Additionally to access a specific process on the host a **port number** is used. Thus, to access a process on a host the client needs to know the IP address of the host and the port number of the process. The combination of the IP address and port number is called a **socket address**.

1.3 Transport Services Available to Applications

A socket is the interface between the application process and the transport layer protocol, such as TCP or UDP. The services that a transport layer protocol can over to applications can be broadly classified along four dimensions:

- Reliable data transfer
- Throughput
- Timing
- Security

1.3.1 Reliable Data Transfer

Packets can get lost within a computer network, thus a transport layer protocol can provide reliable data transfer by implementing mechanisms to detect and recover from lost packets. This is know as guaranteed data delivery. When a transport protocol provides this service the sending process can just pass its data into the socket and known with certainty that the data will arrive without errors at the receiving process.

When a transport protocol does not provide reliable data transfer there is no guarantee that all the data sent by the client will arrive at the server. This may be acceptable for **loss-tolerant applications** like audio and video streaming, where the occasional loss of a packet may not be noticeable to the user. However, for applications that require reliable data transfer, such as file transfer or email, it is important to use a transport protocol that provides this service.

1.3.2 Throughput

Throughput is the amount of data that can be transferred from one process to another in a given amount of time. A transport protocol can provide a guaranteed available throughput at some specified rate. This is important for application that require a certain level of performance such as video streaming or online gaming. Such applications are called **bandwidth sensitive applications**. Applications that do not require a guaranteed level of performance are called **elastic applications**.

1.3.3 Timing

Timing guarantees refer to the ability of a transport protocol t provide guarantees on the maximum amount of time it takes for a message to be delivered from the sender to the receiver. This is important for applications that require real-time communication, such as video conferencing or online gaming. Such applications are called **delay-sensitive applications**

1.3.4 Security

A transport protocol can provide security services such as encryption and authentication to protect the confidentiality and integrity of the data being transmitted. This is important for applications that handle sensitive information, such as online banking or email. Such applications are called **security-sensitive applications**.

1.3.5 Transport Services Provided by the Internet

Application	Data Loss	Throughput	Time-Sensitive
File Transfer	No loss	Elastic	No
Email	No loss s	Elastic	No
Web Documents	No loss	Elastic	No
Video conferencing	Loss-tolerant	Bandwidth-sensitive	Yes
Streaming stored video	Loss-tolerant	Bandwidth-sensitive	No
Interactive games	Loss-tolerant	Bandwidth-sensitive	Yes
Online messaging	No loss	Elastic	Yes

1.3.6 TCP Services

The TCP service model is a connection-oriented service model, offering reliable data transfer.

Connection Oriented - TCP requires the client and server to exchange transport layer control information with each other before the exchange of messages. This is called a **handshake** an alerts the client and server allowing them to prepare for the data transfer. After the handshaking phase a TCP connection is established between the sockets of the two processes. The connection is a full-duplex connection, meaning that data can be sent in both directions at the same time. After the data transfer is complete the client and server exchange control information to terminate the connection.

Reliable Data Transfer - The communicating processes are guaranteed that all data sent by the client will arrive at the server without errors and in the correct order. TCP achieves this by implementing mechanisms to detect and recover from lost packets

Flow Control - TCP implements flow control to ensure that the sender does not overwhelm the receiver with too much data at once.

Congestion Control - TCP also implements congestion control to prevent the sender from overwhelming the network with too much data at once, which can lead to network congestion and packet loss.

Does not provide - Timing, guaranteed minimum throughput, security.

1.3.7 UDP Services

A lightweight transport protocol providing minimal services. UDP is connectionless, and unreliable meaning it does not guarantee the delivery of messages.

Connectionless - UDP does not require the client and server to exchange transport layer control information with each other before the exchange of messages. The client can just send a message to the server without establishing a connection first. This makes it faster than TCP for applications that do not require reliable data transfer.

Unreliable Data Transfer - UDP does not guarantee that all data sent by the client will arrive at the server without errors and in the correct order. This may be acceptable for loss-tolerant applications like audio and video streaming, where the occasional loss of a packet may not be noticeable to the user. However, for applications that require reliable data transfer, such as file transfer or email, it is important to use a transport protocol that provides this service.

Does not provide - Reliability, flow control, congestion control, timing, guaranteed minimum throughput, security, connection setup

1.3.8 Securing TCP

Default TCP and UDP sockets provide no encryption, meaning that data sent over a TCP connection is sent in clear text. This means that if an attacker is able to intercept the data being sent over the TCP connection, they can read the data, including any sensitive information such as passwords or credit card numbers. To address this issue, Transport Layer Security (TLS) can be used to provide encryption for TCP connections. TLS is a cryptographic protocol that provides data integrity, end-point authentication, and encryption for TCP connections. When TLS is used, the data sent over the TCP connection is encrypted, making it much more difficult for an attacker to intercept and read the data.

1.4 Application Layer Protocol

An application layer protocol defines how an application's processes, running on different end systems, pass messages to each other. It defines:

- The types of messages exchanged, i.e. request and response messages
- The syntax of the various message types, i.e. the fields in each message and how the fields are delineated
- The semantics of the fields, i.e. the meaning of the information in the fields and how the receiving process should interpret the information

- The rules for determining when and how a process sends and responds to messages, i.e. the order of message exchange and the actions taken when a message is received

The main difference between network applications and application-layer protocols is that an application-layer protocol is only a section of a network application. For example, a web browser is a network application that implements the HTTP application-layer protocol to communicate with web servers. The web browser also implements other protocols such as DNS to resolve domain names to IP addresses, and TLS to secure the communication with the web server.

There are two types of protocols

Open

Proprietary

Chapter 2

Web and HTTP

2.1 HyperText Markup Protocol (HTTP)

Definition 2.1.1: Object

A file, such as an HTML file, a JPEG image, a Java applet, or a video clip, that can be sent over the internet.

HTTP is implemented in two programs, a client program and a server program usually executing on different hosts. HTTP defines the structure of messages exchanged between the client and server and how the client and server should respond to various messages. HTTP is the foundation of data communication for the World Wide Web, and it is used to transfer web pages and other web resources between web servers and web browsers.

A web page consists of objects, which are files that can be sent over the internet. Examples of objects include HTML files, JPEG images, Java applets, and video clips. When a user requests a web page, the web browser sends an HTTP request to the web server, which then responds with the requested objects. The web browser then renders the web page using the received objects.

HTTP is based on TCP, i.e.:

- Client initiates TCP, creates socket, to a server at port 80
- Server accepts TCP connections from client
- HTTP messages are exchanged between browser and web server
- TCP connection is closed

TCP provides a reliable data transfer service to HTTP, which means that each request sent by a client eventually arrives intact at the server, and each response sent by the server eventually arrives intact at the client.

HTTP is stateless as the server maintains no information about past client requests. Each HTTP request is treated independently of any previous requests. This means that the server does not keep track of any information about the client or the client's previous requests.

2.1.1 Non-Persistent vs Persistent HTTP

In many applications the client and server communicate for an extended period of time, with the client making a series of requests and the server responding to each of the requests. Depending on the application the requests may be made back to back periodically at regular intervals or intermittently. This introduces the question of whether to use a single TCP connection for all the requests and responses, or to use a separate TCP connection for each request and response. These types of connections are known as persistent and non-persistent connections respectively. HTTP can be implemented using either persistent or non-persistent connections.

2.1.1.1 HTTP with Non-Persistent connections

In the case of a non-persistent connection a separate TCP connection is used for each request and response. This means that for each request the client establishes a new TCP connection to the server, sends the request, receives the response, and then closes the TCP connection.

HTTP/1.0 uses non persistent connections, supporting exactly one request and response per TCP connection.

Definition 2.1.2: Round Trip Time (RTT)

The time it takes for a small packet to travel from the client to the server and back again.

The RTT includes packet-propagation, packet-queuing, and packet-processing delays. Since TCP uses a connection-oriented service model, the client and server initiate a **three-way handshake** to establish a TCP connection before the client can send its request. The first handshake message (SYN) is sent from the client to the server, then the second handshake message (SYN-ACK) is sent from the server to the client, and finally the third handshake message (ACK) is sent from the client to the server with the request. This means at minimum two RTTs are needed before the client receives the response from the server.

The first RTT is for the three-way handshake to establish the TCP connection, and the second RTT is for the client to send the request and receive the response from the server. In the case of non-persistent HTTP, the RTT is incurred for each request and response, which can lead to significant delays if the client needs to make multiple requests to the server.

Non-Persistent HTTP is inefficient as it requires multiple TCP connections to be established and closed for each request and response, which can lead to significant delays due to the overhead of establishing and closing TCP connections, as well as the RTT for each request and response.

Non-Persistent HTTP connections also require TCP buffers and variables to be allocated for each connection, which can lead to increased memory usage on the server.

2.1.1.2 HTTP with Persistent connections

With HTTP/1.1, persistent connections are the default, meaning the server leaves the TCP connection open after sending a response. Subsequent requests and responses are sent over the same connection, avoiding the need for a new three-way handshake for every object. While the first object still requires 2 RTTs (handshake plus request), subsequent objects only require 1 RTT (or less with pipelining). Pipelining further improves performance by allowing the client to send multiple requests back-to-back without waiting for intervening responses.

2.1.2 Message Format

There are two types of HTTP messages: request messages and response

2.1.2.1 HTTP Request Message

Below is a typical HTTP request message:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/5.0
Accept-language: fr
```

The components of a HTTP are:

Request Line - The first line of an HTTP request message. Contains three fields

- **Method** - The method field specifies the type of request being made. Common methods include GET, POST, and HEAD.
- **URL** - The URL field specifies the resource being requested. It includes the protocol (e.g. http), the hostname (e.g. www.someschool.edu), and the path to the resource (e.g. /somedir/page.html).
- **HTTP Version** - The HTTP version field specifies the version of HTTP being used by the client.

Header Fields - The subsequent lines after the request line. Each header field consists of a name and a value, separated by a colon. These fields provide additional information about the request, such as the client's capabilities and preferences.

Message Body - The message body is an optional part of the HTTP request message. It is used to send data from the client to the server, such as form data or file uploads. The message body is typically used with the POST method, but it can also be used with other methods as well.

2.1.2.2 HTTP Response Message

Below is a typical HTTP response message:

```
HTTP/1.1 200 OK
Connection: close
Date: Tue, 18 Aug 2015 15:44:04 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT
Content-Length: 6821
Content-Type: text/html

<html>
<body>
<h1>Welcome to my homepage</h1>
<p>My name is Madiba</p>
</body>
</html>
```

The components of a HTTP response message are:

Status Line - The first line of an HTTP response message. Contains three fields

HTTP Version - The HTTP version field specifies the version of HTTP being used by the server.

Status Code - The status code field is a three-digit code that indicates the status of the response. Common status codes include 200 (OK), 404 (Not Found), and 500 (Internal Server Error).

Reason Phrase - The reason phrase field is a brief description of the status code. It provides additional information about the status of the response.

Header Fields - The subsequent lines after the status line. Each header field consists of a name and a value, separated by a colon. These fields provide additional information about the response, such as the server's capabilities and preferences.

Message Body - The message body is an optional part of the HTTP response message. It is used to send data from the server to the client, such as the requested resource or an error message. The message body is typically used with successful responses (e.g. 200 OK), but it can also be used with error responses as well.

2.1.3 Cookies and State

Definition 2.1.3: Cookie

A small piece of data stored on the client's computer by the web browser. Cookies are used to maintain state and store information about the client, such as login credentials or user preferences.

HTTP is stateless, meaning that the server does not maintain any information across requests from the same client. Sometimes it is desirable for the server to maintain state and this is achieved using cookies.

Cookies are comprised of four components:

Cookie Header in Response - The Set-Cookie, header is a field in the HTTP response message that is sent from the server to the client. It contains the name and value of the cookie, as well as any additional attributes such as the expiration date or the domain for which the cookie is valid.

Cookie Header in Request - The Cookie, header is a field in the HTTP request message that is sent from the client to the server. It contains the name and value of the cookie that was previously set by the server. This allows the server to identify the client and maintain state across multiple requests from the same client.

Cookie File - The cookie file is a file stored on the client's computer by the web browser. It contains the name and value of the cookies that have been set by the server, as well as any additional attributes such as the expiration date or the domain for which the cookie is valid. The web browser uses the cookie file to manage cookies and send them back to the server in subsequent requests.

Backend Database - The backend database is a database maintained by the server that stores information about the clients, such as login credentials or user preferences. When a client sends a request to the server with a cookie header, the server can use the value of the cookie to look up the corresponding information in the backend database and maintain state across multiple requests from the same client.

2.1.4 Web Caching

Definition 2.1.4: Web Cache / Proxy Server

A network entity that satisfies HTTP requests on behalf of an origin web server.

When a browser is configured to use a web cache it goes through the following steps:

1. The browser establishes a TCP connection to the web cache and sends a HTTP request for the object to the web cache.
2. The web cache checks to so if it has a copy of the object stored locally. If it does the we cache returns the object within an HTTP response message to the client browser.
3. If the web cache does not have the object the web cache opens a TCP connection to the origin server and requests the object. After receiving the request the origin server services the request and sends the HTTP response to the web cache.
4. When the web cache receives the response it stores a copy in its local storage and sends a copy, within a HTTP response message, to the client browser.

The cache is both a server and client at the same time. When it receives requests from and sends responses to a browser its a server and when it sends requests and receives responses from an origin server its a client.

A web cache can be used to reduce the response time for a client request, particularly if the bottleneck bandwidth between the client and the origin server is much less than the bottleneck bandwidth between the client and the web cache. A web cache can also be used to reduce the traffic on an institution's access link, particularly if many clients within the institution request the same object.

Web caches can also reduce the traffic on an access link to the internet. By reducing the traffic on the access link, bandwidth is not needed as much, reducing the cost of internet access for the institution. Additionally, by reducing the traffic on the access link, the performance of other applications that use the access link can be improved.

A method of calculating the traffic intensity on the LAN is to use the formula:

$$\text{Traffic Intensity} = \frac{\text{Average Request Rate} \times \text{Average Object Size}}{\text{Bottleneck Bandwidth}}$$

2.1.4.1 Conditional GET

Caching introduces the problem of stale objects, where the objects stored in a web cache may have been modified since last cache. The Conditional GET method allows a cache to verify that its objects are up to data.

An HTTP request message is a conditional GET if:

- The request verb is GET
- It includes the If-Modified-Since header line

In the case of a conditional GET if the object has not been modified since the date specified in the If-Modified-Since header line, the origin server responds with a 304 Not Modified status code and does not include the object in the response message. This allows the web cache to use the stale object without having to download it again from the origin server, thus reducing the response time for the client request and reducing the traffic on the access link to the internet.

2.1.5 HTTP/2

HTTP/2 introduces several improvements over HTTP/1.1, including:

Latency Reduction - Achieved by enabling request and response multiplexing over a single TCP connection

Request Prioritization - HTTP/2 allows clients to assign a priority level to each request, allowing the server to prioritize the processing of requests based on their importance.

Server Push - HTTP/2 allows servers to proactively send resources to clients before the client requests them, which can improve performance by reducing the number of round trips needed to fetch resources.

Header Compression - HTTP/2 uses a more efficient header compression algorithm, which reduces the size of HTTP headers and improves performance.

Definition 2.1.5: Head of Line (HOL) Blocking

A performance issue that occurs when the processing of one request is blocked by the processing of another request. This can occur when multiple requests are sent over a single TCP connection, and the processing of one request is delayed due to network congestion or other issues.

HTTP/1.1 allows for persistent connections but this comes with the caveat that sending all the objects in a web page over a single TCP connection could lead to Head of Line (HOL) blocking, where the processing of one request is blocked by the processing of another request.

HTTP/2 addresses this issue by allowing for request and response multiplexing over multiple TCP connections, which allows for multiple requests and responses to be sent simultaneously without being blocked by each other. This allows for improved performance and reduced latency when fetching resources for a web page.

2.1.5.1 HTTP/2 Framing

HTTP/2 breaks down each HTTP message into smaller units called frames, and then interleaves the request and response messages on the same TCP connections. This allows for multiple requests and responses to be sent simultaneously without being blocked by each other, thus improving performance and reducing latency when fetching resources for a web page.

This framing mechanism allows for more efficient use of the TCP connection, as it allows for multiple requests and responses to be sent at the same time without being blocked by each other, significantly decreasing user-perceived delay.

Framing is done by the framing sub-layer of the HTTP/2 protocol. When a server wants to send an HTTP response, the response is processed by the framing sub-layer, where it is broken-down into frames, with the header field becoming one frame, and the body being broken down into multiple frames if necessary. The response frames are then interleaved by the framing sub-layer in the server with the frames of other responses and sent over the same TCP connection to the client.

When the client receives the frames, they are first reassembled into the original response messages at the framing sub-layer and then processed by the browser. Similarly a client's HTTP requests are broken into frames and interleaved.

The framing sub layer also binary encodes the frames as binary protocols are more efficient to parse, lead to smaller frames, and are less error prone.

2.1.5.2 Response Message Prioritization and Server Pushing

Message prioritization allows developers to set the relative priority of requests to better optimize application performance. When the framing sub-layer breaks messages down into frames and organizes them into parallel streams to be sent to the same destination, it can use the priority information to determine how to interleave the frames of different messages. This allows for more important messages to be sent before less important messages, improving the overall performance of the application.

Server pushing allows servers to proactively send resources to clients before the client requests them. This can improve performance by reducing the number of round trips needed to fetch resources. For example, when a client requests an HTML page, the server can also push the associated CSS and JavaScript files to the client before the client requests them, thus reducing the latency of fetching these resources and improving the overall performance of the application.

2.1.6 HTTP/3

HTTP/3 introduces several improvements over HTTP/2, including:

Quick UDP Internet Connections (QUIC) - HTTP/3 uses QUIC, a new transport protocol that is designed to provide improved performance and security compared to TCP. QUIC is built on top of UDP and provides features such as multiplexing, connection migration, and improved congestion control. QUIC supports message multiplexing, per-stream flow control, and low-latency connection establishment.

Improved Performance - HTTP/3 is designed to provide improved performance compared to HTTP/2, particularly in terms of latency and throughput. This is achieved through the use of QUIC, which provides features such as multiplexing, connection migration, and improved congestion control.

Improved Security - HTTP/3 is designed to provide improved security compared to HTTP/2, particularly in terms of encryption and authentication. This is achieved through the use of QUIC, which provides features such as built-in encryption and improved authentication mechanisms.

Chapter 3

Electronic Mail in the Internet

The internet mail system has three main components:

User Agent (UA) - Allows users to read, reply to, forward, save, and compose messages.

Mail Server - A server that stores and forwards email messages. It typically uses the Simple Mail Transfer Protocol (SMTP) to send and receive email messages. Each user has a mailbox on the mail server, where their email messages are stored until they are retrieved by the user agent. Messages are sent from the user agent to the mail server, and then from the mail server to the recipient's mail server, and finally from the recipient's mail server to the recipient's user agent. If the recipient's mail server is not available, the sender's mail the sender server will store the message in a **message queue** and attempt to resend the message at a later time.

Simple Mail Transfer Protocol (SMTP) - A protocol used to send and receive email messages. It is a text-based protocol that uses a client-server model, where the client sends email messages to the server, and the server forwards the messages to the appropriate recipients. It uses TCP as its transport protocol, i.e. reliable data transfer, and it typically uses port 25 for communication.

3.1 Simple Mail Transfer Protocol (SMTP)

SMTP transfers messages from senders' mail servers to the recipients' mail servers. It is a text-based protocol that uses a client-server model, where the client sends email messages to the server, and the server forwards the messages to the appropriate recipients. SMTP uses TCP as its transport protocol, i.e. reliable data transfer, and it typically uses port 25 for communication.

The order of operations in SMTP is as follows:

1. The sender's user agent is instructed by the user to send an email to a recipient at a particular email address.
2. The sender's user agent sends the message to the sender's mail server where it is placed in a message queue.
3. The client side of SMTP running on the sender's mail server initiates a TCP connection a SMTP server running on the recipient's mail server.
4. After the TCP connection is established the sender's mail server sends the email message to the recipient's mail server.
5. At the recipient's mail server the server side of SMTP receives the message and places it in the recipient's mailbox.
6. The recipient invokes their user agent to read the email message, which retrieves the message from the recipient's mail server and displays it to the recipient.

It is important to note that SMTP does not usually use intermediate mail servers for sending mail, even when the two mail servers are not directly connected.

From the operation order it is clear that SMTP is a push protocol, meaning that it only facilitates the sending of email messages from the sender's mail server to the recipient's mail server. It does not provide any mechanism for retrieving

email messages from the recipient's mail server to the recipient's user agent.

Pull protocols, such as Post Office Protocol (POP) and Internet Message Access Protocol (IMAP), are used to retrieve email messages from the recipient's mail server to the recipient's user agent. These protocols allow the recipient's user agent to connect to the recipient's mail server and retrieve email messages from the recipient's mailbox. POP and IMAP are typically used in conjunction with SMTP, where SMTP is used to send email messages from the sender's mail server to the recipient's mail server, and POP or IMAP is used to retrieve email messages from the recipient's mail server to the recipient's user agent.

3.2 Mail Message Formats

SMTP does not specify the format of email messages, but it does specify that email messages must be in ASCII text format. All kinds of peripheral information can be included in the message header, at the top of the message, and the message body, at the bottom of the message. The message header contains information about the sender, recipient, subject, and other metadata about the email message. The message body contains the actual content of the email message.

Similar to HTTP, email messages can also include header fields that provide additional information about the email message, such as the sender's email address, the recipient's email address, the subject of the email message, and the date and time the email message was sent. These header fields are used by email clients to organize and display email messages to the user.

An example of an email message is as follows:

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

3.3 Mail Access Protocols

A message sent from a sender's user agent is first sent to the sender's mail server, before it is sent to the recipient's mail server. This two-step procedure allows the sender's mail server to store the message in a message queue in the case of an unreachable destination mail server.

A recipient user agent can retrieve messages from the recipient's mail server using a mail access protocol. There are two common ways for a recipient user agent to retrieve messages from the recipient's mail server:

HTTP - If the user agent is web based, the recipient can retrieve messages from the recipient's mail server using HTTP. In this case, the recipient's mail server would need to implement an HTTP server that allows the recipient user agent to access the email messages stored in the recipient's mailbox.

Internet Mail Access Protocol (IMAP) - Typically used by mail clients, such as Microsoft Outlook or Apple Mail, to retrieve email messages from the recipient's mail server. IMAP allows the recipient user agent to connect to the recipient's mail server and retrieve email messages from the recipient's mailbox. IMAP also allows the recipient user agent to manage email messages on the recipient's mail server, such as deleting or moving email messages between folders. IMAP is typically used in conjunction with SMTP, where SMTP is used to send email messages from the sender's mail server to the recipient's mail server, and IMAP is used to retrieve email messages from the recipient's mail server to the recipient's user agent.

Chapter 4

Domain Name System (DNS)

Definition 4.0.1: Hostname

Human readable name of a host, such as www.someschool.edu. Host names are used to identify hosts on the internet and are typically easier for humans to remember than IP addresses.

The Domain Name System (DNS) is a distributed database that maps hostnames to IP addresses. It is an application layer protocol that allows hosts to query the distributed database.

4.1 DNS Services

DNS is commonly used by other application layer protocols like HTTP and SMTP to resolve hostnames to IP addresses. DNS provides other important services in addition to hostname resolution, such as:

Definition 4.1.1: Canonical Name (CNAME)

The canonical name that a hostname is an alias for. This allows multiple hostnames to be associated with a single IP address, which can be useful for load balancing and for providing multiple services on the same host.

Host Aliasing - DNS allows multiple hostnames to be associated with a single IP address. This is useful for load balancing and for providing multiple services on the same host.

Mail Server Aliasing - DNS allows multiple mail servers to be associated with a single hostname. This is useful for load balancing and for providing redundancy for email services.

Load Distrubuton - DNS allows for the distribution of load across multiple servers by allowing multiple hostnames to be associated with a single IP address. This allows for better performance and reliability for applications that rely on DNS for hostname resolution.

4.2 DNS Architecture

A naive approach to DNS would have one centralized DNS server that contains all the mappings of hostnames to IP addresses. This approach has several issues:

Single Point of Failure - If the centralized DNS server goes down, then hostname resolution would fail for all clients that rely on that server.

Traffic Volume - A single DNS server would have to handle all DNS queries from all clients, which could lead to performance issues and increased latency for hostname resolution.

Distant Centralized Database - A single DNS server cannot be located close to all clients, which can lead to increased latency for hostname resolution due to the distance between the client and the DNS server.

Maintenance - A single DNS server would have to keep records for all Internet hosts. This would be a huge database that would require a lot of maintenance and updates, which could lead to errors and inconsistencies in the DNS records.

4.2.1 Distributed Hierarchical Database

To address the issues of a centralized DNS server, the DNS uses a large number of servers organized in a hierarchical fashion and distributed across the world. There are four general types of DNS servers

Root DNS Servers - Copies of 13 different root servers managed by 12 different organizations. They are responsible for providing the IP addresses of the top level domain (TLD) DNS servers.

Top Level Domain (TLD) DNS Servers - For each top level domain (e.g. .com, .org, .edu) there is a TLD DNS server that is responsible for providing the IP addresses of the authoritative DNS servers for that top level domain.

Authoritative DNS Servers - Every organization with publicly accessible hosts on the Internet must provide publicly accessible DNS records that map the names of those hosts to their IP addresses. The Authoritative DNS server holds these DNS records.

Local DNS Servers - Local DNS servers are typically provided by Internet Service Providers (ISPs) and are responsible for providing DNS resolution services to clients within the ISP's network. Local DNS servers are typically configured to forward DNS queries to the appropriate root, TLD, or authoritative DNS servers as needed to resolve hostnames to IP addresses. Local DNS servers can also cache DNS records to improve performance for frequently accessed hostnames.

Definition 4.2.1: Iterative Query

A DNS query where the local DNS server is responsible for making the necessary requests to the root, TLD, and authoritative DNS servers to resolve a hostname to an IP address. It is iterative in the sense that all the responses from the other DNS servers are returned to the local DNS server, which then makes the next request based on the response received.

Definition 4.2.2: Recursive Query

A DNS query where the local DNS server is responsible for resolving a hostname to an IP address and returning the result to the client. It is recursive in the sense that the local DNS server is responsible for making the necessary requests to the root, TLD, and authoritative DNS servers to resolve the hostname to an IP address and returning the final result to the client.

The general process of DNS resolution is as follows (without caching):

1. The client sends a DNS query to the local DNS server to resolve a hostname to an IP address.
2. The local DNS server then makes a request to a root DNS server to get the IP address of the TLD DNS server for the top level domain of the hostname. This response is returned to the local DNS server.
3. The local DNS server then makes a request to the TLD DNS server to get the IP address of the authoritative DNS server for the hostname. This response is returned to the local DNS server.
4. The local DNS server then makes a request to the authoritative DNS server to get the IP address of the hostname. This response is returned to the local DNS server.
5. Finally, the local DNS server returns the IP address of the hostname to the client.

This is an iterative DNS resolution process, where the local DNS server is responsible for making the necessary requests to the root, TLD, and authoritative DNS servers to resolve the hostname to an IP address. However the first request sent from the host to the local DNS server is a recursive request, meaning that the local DNS server is responsible for resolving the hostname to an IP address and returning the result to the client.

Sometimes the TLD DNS server may not know the authoritative DNS server of the host name, and my only know of an intermediate DNS server which knows the authoritative DNS server. In this case, the local DNS server would make a

request to the intermediate DNS server to get the IP address of the authoritative DNS server for the hostname, and then make a request to the authoritative DNS server to get the IP address of the hostname.

4.2.2 DNS Caching

DNS caching is a technique used to improve the performance of DNS resolution by storing previously resolved DNS records in a cache. When a client makes a DNS query, the local DNS server first checks its cache to see if it has a valid record for the requested hostname. If it does, it returns the cached record to the client, which can significantly reduce the latency of DNS resolution.

A typical DNS process with caching is as follows:

1. The client sends a DNS query to the local DNS server to resolve a hostname to an IP address.
2. The local DNS server checks its cache to see if it has a valid record for the requested hostname. If it does, it returns the cached record to the client, and the process is complete.
3. If the local DNS server does not have a valid record in its cache, it proceeds with the iterative DNS resolution process as described in the previous section, making requests to the root, TLD, and authoritative DNS servers as needed to resolve the hostname to an IP address.
4. Once the local DNS server receives the IP address of the hostname from the authoritative DNS server, it stores the record in its cache for future use and returns the IP address to the client.

4.3 DNS Records and Messages

DNS servers store **Resource Records (RRs)** in their databases, which are used to map hostnames to IP addresses and provide other information about the host. A resource record is a four tuple that contains the following fields:

- Name
- Value
- Type
- Time to Live (TTL)

TTL is the time to live of the resource record, i.e. it determines when a resource should be removed from a cache.

The meaning of **Name** and **Value** fields depend on **Type**:

Type=A - The Name is a hostname and Value is the IP address for the host name. A Type A record provides the standard hostname to IP address mapping that is used for most DNS queries.

Type=NS - The Name is a domain and Value is the host-name of an authoritative DNS server that knowns how to obtain the IP addresses for hosts in the domain. This record is used to route DNS queries further along the query path to the authoritative DNS server for a particular domain.

Type=CNAME - Value is a canonical hostname for the alias hostname Name. This record is used to provide querying hosts the canonical name for a hostname.

Type=MX - The Value is the canonical name of a mail server that has an alias hostname Name. MX records allow the hostnames of mail servers to have simple aliases.

If a DNS server is authoritative for a particular hostname, then the DNS server will contain a Type A record for that hostname, or even if the server is not authoritative it may still contain cached Type A records for that hostname.

If a server is not authoritative then it will contain a Type NS record for the domain that includes the hostname, it will also contain Type A record that provides the IP address of the DNS server in the Value field of the NS record.

4.3.1 DNS Messages

There are two kinds of DNS messages: query messages and response messages. These messages have the same format and are as follows:

Header Section - 12 bytes, which contains identification and control information about the message.

- **Identification** - A 16-bit field that is used to match responses to queries. When a client sends a DNS query, it includes a unique identification number in the query message. When the DNS server responds to the query, it includes the same identification number in the response message, allowing the client to match the response to the original query.
- **Flags** - A 16-bit field that contains various flags and control bits that provide information about the message, such as whether it is a query or a response, whether the message is authoritative, and whether the message is truncated.
- **Counts** - Four 16-bit fields that specify the number of entries in the Question, Answer, Authority, and Additional sections of the message. These fields are used to indicate how many resource records are included in each section of the message.

Question Section Contains information about the query that is being made. It includes

- **Name** - Contains the name that is being queried, such as a hostname or a domain name.
- **Type** - Indicated the type of question being asked about the name, i.e. the type of resource record being queried for, such as A, NS, CNAME, or MX.

Answer Section - In a reply from a DNS server, contains the resource records for the name that was queried in the question section. A reply can return multiple RRs in the answer since a hostname can have multiple IP addresses

Authority Section - Contains records of other authoritative servers

Additional Section - Contains other helpful records, like the IP address of the authoritative server in the authority section

4.3.2 Inserting Records into the DNS Database

Chapter 5

Peer-to-Peer (P2P) File Sharing

5.1 Scalability

5.2 BitTorrent

Chapter 6

Video Streaming and Content Distribution Networks (CDNs)