# Week 1

*Prof. Nitin Chandrachoodan*
*Department of EE, IIT Madras*

*Notes by Adarsh (23f2003570)*

## Lecture 1: Introduction to the Web (00:27:11)

### Packet Switched Network

Packet-switched networks are the backbone of modern digital communication, including the internet.

#### Data Segmentation

**Packetization** When you send data over a packet-switched network, it is broken down into smaller chunks called packets. Each packet contains a portion of the data, plus metadata such as the destination address, source address, and sequence number

#### Routing

Each packet is routed independently through the network. The routing decision is made based on the packet's destination address. The path a packet takes can vary, which means packets from the same message *might arrive out of order*.

#### Network Nodes

**Switches and Routers:** The network is made up of various nodes like routers and switches. Routers are used in wide area networks (WANs) and handle the routing of packets between different networks. Switches are used in local area networks (LANs) and facilitate communication within a network segment. **Forwarding:** Each node examines the packet's destination address and decides the next hop for the packet. This process continues until the packet reaches its final destination.

#### Reassembly

*Reassembly:* At the destination, packets are reassembled into the original message based on their sequence numbers. If some packets are lost or corrupted, the system can request retransmission of the affected packets.

#### Error Handling and Quality of Service

**Error Checking:** Packets include checksums or other error-detection codes to verify their integrity. If a packet is corrupted, it can be retransmitted. **Quality of Service (QoS):**

Some packet-switched networks implement QoS mechanisms to prioritize certain types of traffic (e.g., voice or video) to ensure they meet performance requirements.

**Advantages of Packet Switching**

**Efficiency:** Network resources are used more efficiently because packets can be routed along the least congested paths and network links are shared among many users. **Robustness:** If a path becomes congested or fails, packets can be rerouted, making the network resilient to failures. Scalability: It scales well with the number of users and the amount of data being transmitted.

**Comparison with Circuit Switching**

Packet switching contrasts with circuit switching (used in traditional telephone networks), where a dedicated communication path is established between the sender and receiver for the duration of the call. Packet switching, on the other hand, dynamically allocates resources and shares them among multiple users, making it more adaptable to varying traffic loads.

In summary, packet-switched networks break data into packets, route them independently, and reassemble them at the destination, offering efficient, flexible, and robust communication.

## Network Protocols

A network protocol is a set of rules and conventions that governs how data is transmitted, received, and processed across a network. It defines the format, timing, sequencing, and error-checking mechanisms for data exchanges between devices. In essence, network protocols ensure that different devices and applications can communicate with each other in a consistent and reliable manner.

## Key Components of a Network Protocol

1. Syntax: Defines the structure or format of the data being transmitted. This includes the arrangement of data in packets or frames, as well as headers and footers that provide metadata.
2. Semantics: Specifies the meaning of each part of the communication. It describes what actions should be taken based on the data received, such as how to handle errors or interpret commands.
3. Timing: Involves rules for when data should be sent and received. This includes synchronization of data flow, handling delays, and managing retransmissions in case of lost or corrupted packets.
4. Error Handling: Includes mechanisms for detecting and correcting errors in data transmission. This ensures data integrity and reliability.
5. State Management: Defines how the state of the connection or session is managed, including how devices keep track of ongoing interactions and transitions between

different states.

Examples of Protocols TCP, IP, DECNET, HTTP, FTP, POP3, SMTP

**Importance of Network Protocols**

- **Interoperability:** Protocols enable devices from different manufacturers and with different designs to communicate effectively.
- **Reliability:** They ensure that data is transmitted accurately and that errors are handled appropriately.
- **Efficiency:** Protocols optimize the use of network resources and manage data flow to prevent congestion and bottlenecks.

## Network Layers

The **OSI model** (Open Systems Interconnection model) is a conceptual framework used to understand and standardize the functions of a telecommunication or computing system. It divides the process of network communication into seven distinct layers, each with specific responsibilities. This layered approach helps in designing and troubleshooting networks by isolating different aspects of communication.

Here's a detailed look at each layer of the OSI model, starting from the top:

**1. Application Layer (Layer 7)**

- **Function:** The top layer where network applications and services operate. It provides network services directly to end-users or applications. This layer is responsible for network-related functions such as email, file transfers, and web browsing.
- **Examples:** HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol).

**2. Presentation Layer (Layer 6)**

- **Function:** Handles data translation, encryption, and compression. It ensures that the data is in a readable format for the application layer and translates data from application-level formats to network formats and vice versa.
- **Examples:** Encryption protocols like SSL/TLS, data format conversions like JPEG, GIF, or ASCII.

**3. Session Layer (Layer 5)**

- **Function:** Manages sessions or connections between applications. It establishes, maintains, and terminates communication sessions between applications. This layer is responsible for session recovery and synchronization.
- **Examples:** APIs for session management, NetBIOS, RPC (Remote Procedure Call).

**4. Transport Layer (Layer 4)**

- **Function:** Provides reliable or best-effort delivery of data between systems and handles error detection, correction, and flow control. It ensures complete data transfer by managing data segmentation and reassembly.
- **Examples:** TCP (Transmission Control Protocol) for reliable connections, UDP (User Datagram Protocol) for connectionless communication.

**5. Network Layer (Layer 3)**

- **Function:** Handles routing and forwarding of data packets between devices across different networks. It is responsible for logical addressing and path selection.
- **Examples:** IP (Internet Protocol), ICMP (Internet Control Message Protocol), routers.

**6. Data Link Layer (Layer 2)**

- **Function:** Provides node-to-node data transfer and handles error detection and correction at the physical layer. It packages data into frames and manages physical addressing (MAC addresses).
- **Examples:** Ethernet, PPP (Point-to-Point Protocol), switches.

**7. Physical Layer (Layer 1)**

- **Function:** Deals with the physical transmission of data over a medium. It defines the hardware elements and electrical/optical signals used for data transmission.
- **Examples:** Cables, switches, and the physical aspects of network interfaces.

#### How the OSI Model Works

- **Layer Interaction:** Data communication starts at the application layer on the sender's side, moving down the layers, where each layer adds its own headers or trailers. On the receiver's side, data moves up the layers, where headers or trailers are stripped away, and the data is processed at each layer according to its function.
- **Encapsulation and Decapsulation:** At each layer, data is encapsulated into a format suitable for the layer below it. On the receiving side, each layer decapsulates the data by removing the headers or trailers added by the corresponding layer on the sending side.

#### Importance of the OSI Model

1. **Standardization:** Provides a common framework and terminology for understanding and designing network protocols and systems.
2. **Troubleshooting:** Helps in isolating problems by allowing a focus on specific layers of the network communication process.
3. **Interoperability:** Promotes compatibility and interoperability between products

and technologies from different vendors.

While the OSI model is a theoretical construct and the TCP/IP model is more commonly used in practical networking, the OSI model remains an essential reference for understanding network architecture and functions.

**Inter Level Network (Internet)**

The **Internet Protocol (IP)** is a fundamental protocol used for addressing and routing packets of data across networks, including the internet. It ensures that data is delivered from the source to the destination across potentially complex networks.

**Key Points of IP:**

1. **Addressing:** IP assigns unique addresses (IP addresses) to each device on a network, ensuring that data packets are sent to the correct destination.
2. **Routing:** IP determines the best path for packets to travel from the source to the destination, potentially passing through multiple intermediate devices (routers).
3. **Packetization:** IP breaks down data into packets, each of which contains a portion of the data, as well as addressing information.
4. **Version:** There are two versions of IP:
    - **IPv4 (Internet Protocol version 4):** Uses 32-bit addresses, allowing for about $2^{32}$ unique addresses.
        - examples 192.168.0.1, 172.9.0.1
    - **IPv6 (Internet Protocol version 6):** Uses 128-bit addresses, providing a vastly larger address space to accommodate the growing number of devices.
        - Examples
            - 2001:0db8:85a3:0000:0000:8a2e:0370:7334
            - Range 2001:0db8::5678

In summary, IP is crucial for the operation of the internet, handling addressing, routing, and packetization to ensure data reaches its intended destination across diverse and interconnected networks. No matter what kind of hardware or network you use, please follow IPV6 or IPv4 to be connected into the internet

**Transmission Control Protocol (TCP)** is a core protocol in the Internet protocol suite that ensures reliable, ordered, and error-checked delivery of data between applications over a network. It establishes a connection between sender and receiver, manages data segmentation and reassembly, and handles error correction and retransmission to guarantee that data is transmitted accurately and completely.

Brief overview of IP Addresses:

1. **Early Beginnings (1970s):**

    - **1973:** Vint Cerf and Bob Kahn introduced the concept of packet switching and the Transmission Control Protocol (TCP) as part of the ARPANET project, a precursor to the internet. They developed the idea of addressing packets of data to ensure they reached their destination.

2. **IPv4 Introduction (1981):**
    - **1981:** The Internet Engineering Task Force (IETF) standardized IPv4 in RFC 791. IPv4 uses a 32-bit address space, allowing for about 4.3 billion unique addresses. This version became the foundation for addressing on the internet.

3. **Expansion and Limitations (1990s-2000s):**
    - **1990s:** The rapid growth of the internet began to strain the IPv4 address space. Techniques like Network Address Translation (NAT) were developed to alleviate some of the pressure by allowing multiple devices to share a single public IP address.

4. **IPv6 Introduction (1998):**
    - **1998:** To address the limitations of IPv4, IPv6 was introduced. IPv6 uses a 128-bit address space, offering a virtually limitless number of unique addresses (about 340 undecillion). This upgrade aimed to accommodate the expanding number of internet-connected devices.

5. **Gradual Transition (2000s-Present):**
    - **2000s:** The transition to IPv6 has been gradual. Many networks and organizations began implementing IPv6 to future-proof their infrastructure and handle growing demand.
    - **2020s:** Adoption of IPv6 has been increasing, but IPv4 is still widely used due to its established presence and the complexity involved in switching entirely to IPv6.

**Octet** is a 8 bit byte. Infact Octet is a noun - a group of eight people or things.

The octet is used in representations of Internet Protocol computer network addresses.An IPv4 address consists of four octets, usually displayed individually as a series of decimal values ranging from $[0, 255]$ or [0x00, 0xFF], each separated by a full stop (dot). Using octets with all eight bits set, the representation of the highest-numbered IPv4 address is 255.255.255.255.

- IPv4 - 4 Octets
- IPv6 - 16 Octets like `2001:0db8:0000:0000:0123:4567:89ab:cdef`
- A variable-length sequence of octets, as in Abstract Syntax Notation One (ASN.1), is referred to as an octet string.

**Domain Names (1985)**

The Domain Name System (DNS) has a crucial role in the functionality of the internet, converting human-readable domain names into IP addresses. Here's a brief history of its development:

1. **Early Networking (1960s-1970s):**
    - **1960s:** Early networks like ARPANET used hosts.txt, a simple text file listing hostnames and their corresponding IP addresses. This file was manually updated and distributed to all ARPANET nodes.
2. **DNS Invention (1980s):**
    - **1983:** The Domain Name System was introduced by Paul Mockapetris and his colleagues. The system was detailed in RFC 882 and RFC 883. DNS replaced the hosts.txt file with a hierarchical and decentralized system, which made managing the growing number of hosts more scalable and efficient.
3. **DNS Structure and Delegation:**
    - The DNS hierarchy consists of several levels: root, top-level domains (TLDs), second-level domains, and subdomains. This hierarchical system allows for efficient management and resolution of domain names.
    - The root servers, which are crucial for the DNS infrastructure, were initially operated by various organizations. Over time, the number and distribution of these servers were optimized to improve reliability and speed.
4. **Domain Registration and Management (1990s):**
    - **1991:** The Internet Assigned Numbers Authority (IANA) began managing domain name registrations. The introduction of the Internet Corporation for Assigned Names and Numbers (ICANN) in 1998 formalized and expanded this role, overseeing domain name policies and operations.
5. **DNS Security (2000s-Present):**
    - **2000s:** The DNS Security Extensions (DNSSEC) were developed to protect against certain types of attacks, such as cache poisoning. DNSSEC adds cryptographic signatures to DNS data to ensure its authenticity.
    - **2010s:** The introduction of DNS over HTTPS (DoH) and DNS over TLS (DoT) aimed to enhance privacy and security by encrypting DNS queries and responses, protecting them from eavesdropping and tampering.
6. **Ongoing Developments:**
    - **2020s:** The DNS continues to evolve with advancements in security, privacy, and efficiency. Efforts to address issues like domain abuse, phishing, and the need for better global DNS performance are ongoing.

**Hypertext (early foundation of WWW)**

1. **Early Concepts (1960s):**
    - **1965:** Ted Nelson coined the term “hypertext” and introduced the idea in his work on “Xanadu,” a conceptual hypertext system that aimed to create a universal, interconnected information network. Though Xanadu was never fully realized, Nelson’s ideas influenced later developments.
2. **Theoretical Foundations (1960s-1970s):**
    - **1963:** Vannevar Bush's essay "As We May Think" proposed the concept of a “memex,” a hypothetical device that would allow users to access and link information in a non-linear way. This vision laid the groundwork for hypertext concepts.
    - **1968:** Douglas Engelbart demonstrated his NLS (oN-Line System) at the “Mother of All Demos,” showcasing concepts like hypertext linking, which were integral to future hypertext systems.
3. **Early Implementations (1980s):**
    - **1987:** The term "hypertext" gained more prominence with the publication of “Hypertext and Hypermedia” by George P. Landow. This period saw the development of early hypertext systems and applications.
4. **World Wide Web and HTML (1990s):**
    - **1991:** Tim Berners-Lee, a computer scientist at CERN, proposed the World Wide Web. His idea was to create a system for sharing and accessing information using hypertext. He developed HTML (HyperText Markup Language), HTTP (HyperText Transfer Protocol), and the first web browser, Mosaic, which made the web accessible to the public.
    - **1993:** The release of the Mosaic web browser brought hypertext to a broader audience, enabling users to navigate the web with clickable links and multimedia content.
5. **Modern Hypertext (2000s-Present):**
    - **2000s:** The development of web technologies continued with the introduction of XHTML, XML, and advanced JavaScript frameworks, which enhanced the capabilities of hypertext and interactivity on the web.
    - **2010s:** The rise of web standards like HTML5 and CSS3 further improved the functionality and design of web content. Hypertext has become more sophisticated with dynamic web applications, integrating multimedia, and interactive elements.

**CGI - Common Gateway Interface.** Basically a console program (executed by web server) that writes to STDIO and the web server retrieves the HTML output from STDIO and sends it back to the browser in the open HTTP/s socket connection.

**Web 2.0**

Web 2.0 represents a significant shift in the use and functionality of the internet, characterized by increased user interaction, collaboration, and the growth of social media.

1. **Conceptual Beginnings (1990s):**
    - **1999:** The term “Web 2.0” was coined by Darcy DiNucci in her article “Fragmented Future,” where she discussed the evolving nature of the web. However, the term gained wider recognition and definition in the early 2000s.
2. **Defining Web 2.0 (2000s):**
    - **2001:** Tim O'Reilly and Dale Dougherty of O'Reilly Media popularized the term “Web 2.0” during the first Web 2.0 Conference. They defined Web 2.0 as a new era of web development characterized by user-generated content, social networking, and the shift from static to dynamic web applications.
    - **2004:** The Web 2.0 Summit, organized by O'Reilly Media and MediaLive International, further solidified the concept. The summit highlighted the changes in how people interacted with the web, focusing on collaborative and interactive technologies.
3. **Key Technologies and Trends (2000s):**
    - **Social Media:** The rise of platforms like Facebook (2004), Twitter (2006), and YouTube (2005) transformed how users interacted online, emphasizing user-generated content and social networking.
    - **Ajax and Dynamic Content:** The development of Ajax (Asynchronous JavaScript and XML) allowed web pages to update content dynamically without requiring a full page reload, enhancing the interactivity and responsiveness of web applications.
    - **Rich User Interfaces:** Technologies like Flash and later HTML5, CSS3, and JavaScript frameworks (e.g., jQuery) enabled the creation of more engaging and interactive user experiences.
4. **Expansion and Evolution (2010s-Present):**
    - **Mobile and Responsive Design:** The proliferation of smartphones and tablets led to the adoption of responsive web design techniques to ensure websites function well across various devices.
    - **Cloud Computing and APIs:** Web 2.0 saw the rise of cloud-based services and the use of APIs (Application Programming Interfaces) to enable seamless integration and data exchange between applications.
    - **User-Centric Models:** Platforms and services increasingly focus on user-generated content, personalization, and community-driven features. Examples include platforms like Instagram, TikTok, and various collaborative tools like Slack and Trello.
5. **Beyond Web 2.0 (Current Trends):**

- **Web 3.0 and the Decentralized Web:** While Web 2.0 marked a shift toward interactive and collaborative internet usage, discussions around Web 3.0 focus on decentralization, blockchain technology, and a more semantic web that aims to enhance data interoperability and user control.

In summary, Web 2.0 represents a transformative phase in internet history, characterized by dynamic content, social connectivity, and user participation. It laid the groundwork for the interactive, user-driven web experiences that have become central to modern online activity.

# Lecture 2: How does the Web Work (00:09:30)

## Web Server

A web server is a software application or hardware device that delivers web pages and other content over the internet to users' web browsers. It processes requests from clients (typically web browsers), retrieves the requested content, and sends it back to the client. Here's a more detailed breakdown:

**Key Functions of a Web Server:**

1. **Handling HTTP Requests:**
    - The web server listens for incoming requests on port 80 (for HTTP) or port 443 (for HTTPS) from clients. When a request is received, the server processes it based on the HTTP protocol. Of-course you can use other ports too
2. **Serving Web Content:**
    - The server fetches the requested web content, which can be static files (like HTML, CSS, and JavaScript files) or dynamic content generated by server-side scripts (such as PHP, Python, or Node.js).
3. **Response Handling:**
    - After processing the request, the web server sends back a response to the client. This response includes a status code (indicating success or failure) and the requested content.
4. **Static vs. Dynamic Content:**
    - **Static Content:** Directly served files (e.g., HTML pages, images) stored on the server.
    - **Dynamic Content:** Generated on-the-fly based on server-side processing, such as executing scripts or querying databases.

**Examples of Web Servers:**

- **Apache HTTP Server:** One of the most widely used web servers, known for its flexibility and extensive features.
- **Nginx:** Known for its high performance and efficiency, often used as a reverse proxy server as well as a web server.
- **Microsoft Internet Information Services (IIS):** A web server for Windows Server environments, integrated with other Microsoft technologies.
- **LiteSpeed:** A commercial web server known for its speed and security features.

**Key Components of a Web Server:**

1. **HTTP Server Software:** The core application that handles requests and responses. Examples include Apache, Nginx, and IIS.
2. **Server Hardware or Virtual Instance:** The physical or virtual machine where the web server software runs. This can range from a dedicated server to a cloud-based virtual machine.
3. **Configuration Files:** Files that dictate how the web server should operate, including settings for security, performance, and content handling.

**Additional Features:**

- **Security:** Web servers often include features to enhance security, such as SSL/TLS for encrypted connections, access control mechanisms, and firewall settings.
- **Logging:** Web servers typically log details about requests and errors, which is useful for monitoring, debugging, and analyzing traffic.

In essence, a web server plays a central role in the functioning of the web by delivering content to users and managing interactions between clients and servers.

**Ports**

In networking, **ports** are logical endpoints used to identify specific processes or services on a device. They work with IP addresses to direct data to the correct application or service. Here's a concise explanation:

- **Port Numbers:** Each port is identified by a number, ranging from 0 to 65,535. Ports are categorized as:
    - **Well-Known Ports (0-1023):** Reserved for common protocols (e.g., HTTP on port 80, HTTPS on port 443).
        - In most operating systems the application has to be launched with administrator rights to listen on these ports
        - Only one application can listen on a port on a specific IP.
        - Port is a 16 bit unsigned integer. So you have $2^{16}$ or $65536$ ports
        - But, if you look at the IANA Service Name and Transport Protocol Port Number Registry, PORT 0 is reserved. So there are 65535 ports that you can use.

    - **Registered Ports (1024-49151):** Used by applications and services not covered by well-known ports.
    - **Dynamic/Private Ports (49152-65535):** Used for ephemeral or temporary connections.

- **Purpose:** Ports allow multiple services to run simultaneously on a single IP address by differentiating traffic intended for different applications.
- **Port Uniqueness:** Each port on a given IP address is associated with a single process or application. The port number helps the operating system direct incoming network traffic to the correct application. If two applications tried to listen on the same port, the operating system would be unable to distinguish which application should handle the incoming traffic.
- **Binding Conflicts:** When an application binds to a port, it essentially locks that port for its exclusive use. If another application attempts to bind to the same port, it will typically encounter an error indicating that the port is already in use.

**However**, multiple applications can use the same port number if they are listening on different IP addresses or if they are using different transport protocols (e.g., TCP and UDP). For example:

- **Different IP Addresses:** An application can listen on port 80 for IP address 192.168.1.1, while another application can listen on port 80 for IP address 192.168.1.2.
- **Different Protocols:** An application can listen on TCP port 80, and another application can listen on UDP port 80, as TCP and UDP are different transport protocols.

In essence, ports are essential for directing network traffic to the appropriate service on a device.

### HTTP (HyperText Transfer Protocol)

**Hypertext Transfer Protocol (HTTP)** is a fundamental protocol used for transferring data over the web. It forms the basis of data communication on the World Wide Web. Here's a concise note on HTTP:

#### Overview

- **Purpose:** HTTP is designed for transferring hypertext (web pages) and other data from a web server to a client (usually a web browser). It enables users to interact with websites and web applications by requesting resources and receiving responses.

### HTTP is stateless

Stateless means that each HTTP request from a client to a server is independent and

contains all the information necessary for the server to fulfill the request. The server does not retain any information about previous requests from the same client. Each request is processed in isolation, with no memory of previous interactions. But, of-course you can make it stateful by adding cookies, Headers, but that's not the point

#### How HTTP Works

1. **Request-Response Model:** HTTP operates on a client-server model where:
    - **Client:** The web browser or other client initiates a request for resources.
    - **Server:** The web server processes the request and returns the requested resource (e.g., HTML pages, images).
2. **Request Structure:**
    - **Method:** Defines the action to be performed (e.g., GET, POST, PUT, DELETE).
    - **URL:** Specifies the resource location.
    - **Headers:** Provide additional information (e.g., content type, authentication).
    - **Body:** Contains data sent to the server (e.g., form submissions).
3. **Response Structure:**
    - **Status Code:** Indicates the result of the request (e.g., 200 OK, 404 Not Found).
    - **Headers:** Provide metadata about the response (e.g., content type, length).
    - **Body:** Contains the requested resource or data (e.g., HTML content).

#### Key Versions

- **HTTP/1.0:** The original version, introduced in 1996, which established the basic framework for HTTP.
- **HTTP/1.1:** Introduced in 1999, it added features like persistent connections, chunked transfer encoding, and better caching mechanisms.
- **HTTP/2:** Published in 2015, it improves performance with features like multiplexing, header compression, and server push.
- **HTTP/3:** The latest version, built on QUIC (a transport layer protocol), aims to reduce latency and improve connection security.

#### Security

- **HTTPS:** HTTP Secure (HTTPS) is an extension of HTTP that incorporates SSL/TLS encryption to secure the data transmitted between the client and server, protecting it from eavesdropping and tampering.

#### Importance

HTTP is crucial for the functioning of the web, enabling the seamless transfer of web

pages and resources. It supports the browsing experience by defining how data is formatted and transmitted, and how web servers and clients should respond to various requests.

In summary, HTTP is the backbone protocol of the web, facilitating communication between clients and servers, and enabling the dynamic and interactive experience of modern web applications.

### Loop back

Loopback addresses are special IP addresses used to test and manage network software and services on the local machine. They allow a device to communicate with itself, which is useful for troubleshooting and development. Here's a breakdown:

#### IPv4 Loopback Address

- **Address**: The most common loopback address in IPv4 is **127.0.0.1**.
- **Range**: The entire range of loopback addresses in IPv4 is **127.0.0.0 to 127.255.255.255**. Any address in this range routes back to the local device.

#### IPv6 Loopback Address

- **Address**: In IPv6, the loopback address is **::1**.
- **Range**: The IPv6 loopback address is specifically just **::1**, indicating that it is designated for loopback communication.

#### Uses of Loopback Addresses

1. **Testing**: Developers can test network applications without sending data over an actual network.
2. **Configuration**: Useful in network configurations to ensure that network interfaces are working correctly.
3. **Troubleshooting**: Helps in diagnosing network issues; if a service responds to loopback requests but not to external requests, the problem may lie with the network configuration.

### IPV6 Notes

IPv6, or Internet Protocol version 6, is the most recent version of the Internet Protocol, designed to replace IPv4. It addresses several limitations of IPv4, primarily the exhaustion of available IP addresses. Here's a detailed overview of IPv6, along with examples:

#### Key Features of IPv6

1. **Address Length**:
    - IPv6 addresses are 128 bits long, allowing for a vastly larger number of

unique addresses compared to the 32-bit IPv4.
    - This means IPv6 can support approximately (3.4 \times 10^{38}) addresses.
2. **Address Notation**:
    - IPv6 addresses are typically written in hexadecimal and separated by colons. For example:
        - `2001:0db8:85a3:0000:0000:8a2e:0370:7334`
3. **Address Simplification**:
    - Consecutive sections of zeros can be abbreviated. For instance:
        - The address above can be simplified to:
            - `2001:db8:85a3::8a2e:370:7334`
        - Here, `::` represents one or more groups of 16 bits of zeros.
4. **Types of Addresses**:
    - **Unicast**: A single sender and a single receiver. For example, `2001:0db8:85a3:0000:0000:8a2e:0370:7334` is a unicast address.
    - **Multicast**: A single sender and multiple receivers. For example, `FF02::1` is a multicast address that targets all nodes on the local network.
    - **Anycast**: A single sender and the nearest receiver from a group of potential receivers.
5. **No Broadcast**:
    - IPv6 does not use broadcast addresses like IPv4; instead, it relies on multicast for similar functionality.

#### Examples of IPv6 Addresses

1. **Global Unicast Address**:
    - An example of a global unicast address is `2001:0db8:1234:5678:90ab:cdef:1234:5678`. This type of address is routable on the internet.
2. **Link-Local Address**:
    - Link-local addresses are used for communication within a single network segment. An example is `fe80::1a2b:3c4d:5e6f:7g8h`. These addresses are automatically configured and have a scope limited to the local network.
3. **Loopback Address**:
    - Similar to the IPv4 loopback address (127.0.0.1), the IPv6 loopback address is `::1`. This address is used to test network applications locally.
    - `::` or `::0` is equal to `0.0.0.0`
4. **Multicast Address**:

- An example of a multicast address is `FF02::1`, which represents all nodes on the local network.

#### Benefits of IPv6

- **Larger Address Space**: Eliminates the shortage of IP addresses, accommodating the growing number of devices connected to the internet.
- **Improved Routing**: Simplifies routing tables and improves efficiency.
- **Built-in Security**: IPv6 was designed with security in mind, incorporating IPsec for encryption and authentication

### CGI Notes

The **Common Gateway Interface (CGI)** is a standard protocol that allows web servers to interact with external programs, usually scripts or executables. It enables dynamic content generation and allows web applications to process user inputs. Here's a detailed overview of CGI:

#### Key Features of CGI

1. **Interaction with Web Servers**:
    - CGI allows web servers to execute scripts (often written in languages like Perl, Python, or PHP) in response to client requests (e.g., from web browsers).
2. **Request and Response Model**:
    - When a client (browser) makes a request for a CGI resource, the web server invokes the CGI script and sends the request data to it. The script processes this data and generates a response, which the server sends back to the client.
3. **Environment Variables**:
    - CGI scripts can access environment variables set by the web server, providing information about the request (e.g., request method, query string, client IP address).
4. **Standard Input and Output**:
    - CGI scripts read input from standard input (stdin) and write output to standard output (stdout). The output typically consists of HTTP headers followed by the content.

#### How CGI Works

1. **Client Request**:
    - A user makes a request (e.g., submitting a form on a webpage).

2. **Web Server Handling**:
    - The web server recognizes the request as a CGI call (e.g., based on the URL pointing to a script).
3. **Script Execution**:
    - The server executes the specified CGI script, passing any input data.
4. **Generating Output**:
    - The script processes the input and generates an appropriate response, which includes HTTP headers (like `Content-Type`) and the body content.
5. **Response Sent Back**:
    - The server sends the generated response back to the client's browser.

#### Example of a CGI Script

Here's a simple example of a CGI script written in Perl:

```perl
#!/usr/bin/perl
print "Content-type: text/html\n\n";
print "<html><body>";
print "<h1>Hello, World!</h1>";
print "</body></html>";
```

#### Advantages of CGI

- **Language Agnostic**: CGI can be implemented using various programming languages.
- **Separation of Concerns**: It allows for a clear separation between web server tasks and application logic.

#### Disadvantages of CGI

- **Performance**: Each request results in a new process, which can be resource-intensive and slow compared to other methods like FastCGI or server-side scripting.
- **State Management**: CGI does not maintain state between requests, which can complicate user sessions.

#### Alternatives to CGI

With advancements in web technology, several alternatives to CGI have emerged, including:

- **FastCGI**: An improved version of CGI that keeps processes alive for multiple requests, improving performance.

- **Server-Side Includes (SSI)**: Allows for simpler dynamic content generation without the overhead of full CGI scripts.
- **Web Frameworks**: Modern web frameworks (like Flask, Django, or Express) offer more efficient and powerful ways to build dynamic web applications.

## MIME

**MIME** stands for **Multipurpose Internet Mail Extensions**. It is a standard that extends the format of email messages to support text in character sets other than ASCII, as well as attachments of multimedia files like images, audio, and video.

#### Key Features of MIME

1. **Content Types**:
    - MIME defines various content types to describe the nature of the data being transmitted. For example:
        - `text/plain`: Plain text.
        - `text/html`: HTML content.
        - `image/jpeg`: JPEG images.
        - `application/pdf`: PDF documents.
2. **Headers**:
    - MIME adds specific headers to email messages to indicate the type of content. The most common headers include:
        - `Content-Type`: Specifies the type of data (e.g., `Content-Type: text/html`).
        - `Content-Disposition`: Provides information about how the content should be displayed (inline or as an attachment).
3. **Encoding**:
    - MIME allows for encoding of binary data into a text format, which is necessary for sending non-text files via email. Common encoding methods include:
        - **Base64**: Encodes binary data into ASCII, making it safe for email transmission.
        - **Quoted-printable**: Encodes data so that it remains mostly human-readable while still allowing for special characters.
4. **Multipart Messages**:
    - MIME supports multipart messages, allowing multiple types of content to be included in a single message. For example, an email might contain both text and an image:
        - `Content-Type: multipart/mixed`
        - In this case, the email can have multiple parts, each with its own

content type.

**Example of a MIME Message**

Here's a simple example of a MIME email message:

```
From: sender@example.com
To: recipient@example.com
Subject: Example Email
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="boundary123"

--boundary123
Content-Type: text/plain

Hello, this is a plain text message.

--boundary123
Content-Type: image/jpeg
Content-Disposition: attachment; filename="example.jpg"

[Base64-encoded image data goes here]

--boundary123--
```

**Importance of MIME**

- **Email Functionality**: MIME is essential for the modern functionality of email, enabling the transmission of rich content beyond simple text.
- **Web Content**: MIME types are also used in HTTP to indicate the type of content being served by web servers, helping browsers understand how to handle different types of files.

## Lecture 3: Simple Web Server (00:20:34)

```
while true; do
   echo -e "HTTP/1.1 200 OK\n\n $(date)" | nc -N -l localhost 1500;
done
# Some bug here.. nc is not closing the socket after serving date..
```

Use your browser to point to http://localhost.ai:1500/

### Typical HTTP request

```
GET /favicon.ico HTTP/1.1
Host: localhost:1500
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/5
DNT: 1
Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;
Referer: http://localhost:1500/
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en;q=0.9,en-US;q=0.8,en-IN;q=0.7
Cookie: _vwo_uuid=...
```

1. Host is required, as multiple port-ip bindings can happen in the same system
2. The Accept: header defines the content type the client accepts, or expects to be returned by the server. Depending on the situation this can be text/css, text/html, image/png, .. etc. - just some mime type.
3. Blank line after headers is VERY IMPORTANT!

## Acronyms

1. MIME = Multipurpose Internet Mail Extension
    1. MIMES are like `application/text`, `image/jpeg`

## HTTP/2.0

**HTTP/2** is a major revision of the HTTP protocol designed to improve web performance and efficiency. Key features include:

- **Multiplexing:** Allows multiple requests and responses to be sent simultaneously over a single connection, reducing latency and improving page load times.
- **Header Compression:** Uses HPACK to compress HTTP headers, reducing the amount of data sent and improving efficiency.
- **Binary Protocol:** Uses a binary format rather than text, which is more efficient for parsing and processing.
- **Stream Prioritization:** Enables prioritization of streams to ensure more important resources are delivered first.
- **Server Push:** Allows servers to send resources proactively to the client before they are explicitly requested, speeding up page rendering.

HTTP/2 aims to enhance web performance, reduce latency, and improve overall efficiency compared to its predecessor, HTTP/1.1.

#### HTTP/3.0

**HTTP/3** is the latest version of the HTTP protocol, designed to further enhance web performance and security. Key features include:

- **QUIC Protocol:** Built on the QUIC transport protocol, which improves connection speed and reduces latency by incorporating features like multiplexing, encryption, and congestion control at the transport layer.
- **Improved Connection Establishment:** Reduces connection setup time with 0-RTT (zero round-trip time) and faster handshakes.
- **Better Handling of Packet Loss:** QUIC's built-in mechanisms handle packet loss more efficiently, improving overall reliability and performance.

HTTP/3 aims to provide a faster, more secure browsing experience by addressing the limitations of previous versions and leveraging modern transport technologies.

## Notes on Domains

In the context of networking and the internet, a **domain** refers to a distinct and identifiable name that is used to locate and access resources on the web. Here's a brief overview of the concept:

1. **Definition:**
    - A domain is a human-readable address used to identify a location on the internet. It maps to an IP address, which is used by computers to find and connect to each other.
2. **Domain Structure:**
    - **Domain Name:** The part of the URL that specifies the address of a website. It typically consists of two main parts:
        - **Second-Level Domain (SLD):** This is the main part of the domain name that usually represents the name of the organization or the purpose of the website (e.g., `example` in `example.com`).
        - **Top-Level Domain (TLD):** This follows the SLD and indicates the domain's category or origin (e.g., `.com`, `.org`, `.net`, `.edu`).
3. **Hierarchy:**
    - Domains are structured hierarchically, with different levels separated by dots. For example, in `www.example.com`:
        - `com` is the TLD.
        - `example` is the SLD.
        - `www` is a subdomain that may specify a particular server or service.
4. **Subdomains:**
    - Subdomains are divisions of a domain and can be used to organize different sections of a website or to point to specific servers (e.g., `blog.example.com`, `shop.example.com`).
5. **Domain Registration:**
    - To use a domain name, it must be registered with a domain registrar, which

involves reserving the domain name for a specified period and maintaining it in a global database known as the Domain Name System (DNS).

6. **DNS and Resolution:**
    - The Domain Name System (DNS) is responsible for translating domain names into IP addresses. When a user types a domain name into a browser, DNS servers resolve the domain to its corresponding IP address to connect the user to the correct website.

#### Example

**URL:** `https://www.example.com/page`

- **Domain Name:** `example.com`
    - **Top-Level Domain (TLD):** `.com`
    - **Second-Level Domain (SLD):** `example`
    - **Subdomain:** `www`

## Lecture 3: What is a Protocol (00:12:31)

**Protocol** is how computers are supposed to speak to each other

### Notes on URL

A URL (Uniform Resource Locator) is a reference or address used to access resources on the internet.

#### Basic Format of a URL

```
scheme://domain:port/path?query#fragment
```

#### Components of a URL

1. **Scheme (or Protocol):**
    - **Format:** `scheme://`
    - **Description:** Indicates the protocol used to access the resource. Common schemes include `http`, `https`, `ftp`, `mailto`, etc.
    - **Example:** `https://` or `ftp://`
2. **Domain (or Host):**
    - **Format:** `domain`
    - **Description:** Specifies the domain name or IP address of the server hosting the resource. It may include subdomains.

- **Example:** `www.example.com`, `sub.example.com`, `192.168.1.1`

3. **Port:**
    - **Format:** `:port`
    - **Description:** (Optional) Specifies the port number on the server to connect to. If omitted, the default port for the scheme is used (e.g., port 80 for `http`, port 443 for `https`).
    - **Example:** `:8080`
4. **Path:**
    - **Format:** `/path`
    - **Description:** Specifies the specific resource or file path on the server. It often corresponds to directories and filenames.
    - **Example:** `/index.html`, `/articles/article1`
5. **Query String:**
    - **Format:** `?query`
    - **Description:** (Optional) Contains parameters and values used for querying or filtering resources. Parameters are separated by &.
    - **Example:** `?search=keyword&page=2`
    - Starts with a ?
6. **Fragment:**
    - **Format:** `#fragment`
    - **Description:** (Optional) Points to a specific section or anchor within the resource. It is used to navigate to a particular part of a page.
    - **Example:** `#section1`, `#top`

#### Example URL Breakdown

**URL:** `https://www.example.com:8080/path/to/resource?query=example#section1`

- **Scheme:** `https://` - Specifies that the Hypertext Transfer Protocol Secure (HTTPS) is used.
- **Domain:** `www.example.com` - Indicates the domain name of the server.
- **Port:** `:8080` - Specifies the port number 8080 on the server (optional, and if omitted, the default port 443 for HTTPS would be used).
- **Path:** `/path/to/resource` - The path to the specific resource on the server.
- **Query String:** `?query=example` - Contains parameters for querying or filtering the resource.
- **Fragment:** `#section1` - Points to a specific section within the resource.

#### Additional Details

- **URL Encoding:** Special characters in URLs (like spaces or symbols) must be percent-encoded. For instance, a space is encoded as `%20`.
- **Case Sensitivity:** Domain names are case-insensitive, but paths and queries might be case-sensitive depending on the server configuration.

Understanding the structure of a URL helps in navigating the web and constructing valid URLs for accessing various resources and services.

## HTTP Protocol

HTTP (Hypertext Transfer Protocol) has a structured format that includes several key parts in both requests and responses. Here's a breakdown of these parts:

#### HTTP Request

1. **Request Line:**
    - **Method:** Specifies the action to be performed (e.g., GET, POST, PUT, DELETE).
    - **URL:** Indicates the resource being requested.
    - **HTTP Version:** Shows the version of HTTP being used (e.g., HTTP/1.1).

    **Example:** `GET /index.html HTTP/1.1`
2. **Headers:**
    - **General Headers:** Apply to both request and response messages (e.g., `Cache-Control`, `Connection`).
    - **Request Headers:** Provide additional information about the request (e.g., `Accept`, `User-Agent`, `Authorization`).
    - **Example Header:** `Accept: text/html`
3. **Body (Optional):**
    - Contains data sent to the server (used in methods like POST or PUT). It's not present in GET requests.

    **Example Body (in a POST request):**

```
{
  "username": "user1",
  "password": "pass123"
}
```

#### HTTP Response

1. **Status Line:**
    - **HTTP Version:** Indicates the HTTP version used (e.g., HTTP/1.1).

- **Status Code:** A three-digit code indicating the result of the request (e.g., `200` for OK, `404` for Not Found).
- **Reason Phrase:** A textual description of the status code (e.g., `OK`, `Not Found`).

**Example:** `HTTP/1.1 200 OK`

2. **Headers:**
   - **General Headers:** Apply to both request and response messages (e.g., `Date`, `Server`).
   - **Response Headers:** Provide information specific to the response (e.g., `Content-Type`, `Content-Length`).
   - **Example Header:** `Content-Type: text/html`
3. **Body (Optional):**
   - Contains the data sent back to the client, such as the content of a web page, an image, or other resources.

   **Example Body (HTML content):**

```
<html>
  <head><title>Example</title></head>
  <body><h1>Hello, world!</h1></body>
</html>
```

- **Requests** consist of a request line, headers, and optionally a body.
- **Responses** consist of a status line, headers, and optionally a body.

```
python -m http.server

Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...

# Will list the current directory
```

## Lecture 4: Performance of a website (00:16:11)

**Latency** to the time it takes for data to travel from its source to its destination across a network. It is typically measured in milliseconds (ms) and represents the delay experienced during this data transfer. Latency encompasses various factors, including:

1. **Propagation Delay:** The time required for a signal to travel through the physical medium (e.g., cables, fiber optics) from the sender to the receiver.
2. **Transmission Delay:** The time it takes to push all the packet's bits onto the wire or link.

3. **Processing Delay:** The time taken by network devices (like routers or switches) to process the data, including routing and switching decisions.
4. **Queueing Delay:** The time data spends waiting in queues at intermediate devices due to congestion or high traffic volumes.

Low latency is crucial for applications that require real-time interactions, such as online gaming, video conferencing, and VoIP (Voice over IP). High latency can cause noticeable delays and affect the performance of these applications.

#### Latency problems

1. Calculate the propagation delay for data traveling over a fiber optic link of 1,000 kilometers. Assume the speed of light in fiber optic cables is approximately 200,000 kilometers per second.
    - $t^{1-way} = \frac{10^6}{2 \times 10^8} = 0.005$ or 5 ms
    - One way propagation delay is 5ms
2. 

```
PING www.google.com (172.217.12.36) 56(84) bytes of data.
64 bytes from lga34s12-in-f36.1e100.net (172.217.12.36): icmp_seq=1
64 bytes from lga34s12-in-f36.1e100.net (172.217.12.36): icmp_seq=2
```

- Network Latency is $\frac{14.5 + 14.2}{2}$ ms

3. Say you have a 100 Mbps (Mb = Megabits, MB = Megabytes) and you want to get a 1KB resource.
    1. $\frac{100,000,000}{8} B/s = 12,500,000 B/s = 12.5 MB/s$
    2. Total requests per second $= \frac{12500 KB}{1 KB}$ ~~ 12,5K requests per second.
    3. But you will not get this in real life.
    4. Mb, Gb, Tb are in terms of bits.. 1byte = 8 bits. always multiply bytes by 8
    5. MB, GB, TB are in terms of BYTES!

#### Google Analysis

1. Google website is 144KB
2. Headers is 100Bytes
3. We know from some publication that google serves 60K requests per second

- so thats $144.1 KB * 6 \times 10^5 == 8.646 \times 10^{10}$ bytes of traffic.
- or 86.46 GB of data per second for the home page
- So they need a bandwidth of at-least 87GB/second
- This is not possible. So it has to scale out

**Decimal Exponential Scales**

The exponential scale, commonly used in computing and data storage, represents large quantities in a compact form using prefixes like giga, tera, peta, etc. These prefixes are based on powers of 10 or 2, depending on the context. Here's a breakdown:

### Base-10 (Decimal) System

1. **Kilobyte (KB):** $10^3$ **bytes**
2. **Megabyte (MB):** $10^6$ **bytes**
3. **Gigabyte (GB):** $10^9$ **bytes**
4. **Terabyte (TB):** $10^{12}$ **bytes**
5. **Petabyte (PB):** $10^{15}$ **bytes**
6. **Exabyte (EB):** $10^{18}$ **bytes**
7. **Zettabyte (ZB):** $10^{21}$ **bytes**
8. **Yottabyte (EB):** $10^{24}$ **bytes**
9. ~~Ab Bas~~ Saaku

## Lecture 5 - What is an App (00:17:04)

An **app** (short for application) is a software program designed to perform specific tasks or functions for the user. Apps can be used on various devices, including computers, smartphones, tablets, and smartwatches.

**Key Aspects of an App:**

1. **Types:**
    - **Desktop Apps:** Installed on computers (e.g., Microsoft Word, Adobe Photoshop).
    - **Mobile Apps:** Designed for smartphones and tablets (e.g., Instagram, WhatsApp).
    - **Web Apps:** Accessed via web browsers and do not require installation (e.g., Google Docs, Slack).
    - **Hybrid Apps:** Combine elements of both web and native apps and can run on multiple platforms (e.g., Twitter).
2. **Platforms:**
    - **Operating Systems:** Apps are developed for specific operating systems like Windows, macOS, iOS, Android, or Linux.
    - **App Stores:** Mobile apps are often distributed through app stores like Apple App Store or Google Play Store.
3. **Functionality:** Apps can perform a wide range of functions, including:

- **Productivity:** Managing tasks, calendars, and documents.
- **Communication:** Sending messages, making calls, or video conferencing.
- **Entertainment:** Streaming media, gaming, or reading content.
- **Utilities:** Providing tools for specific tasks like photo editing, navigation, or health tracking.

## Computing Platforms

### 1. Android

- **Tools and IDEs:**
  - **Android Studio:** Official Integrated Development Environment (IDE) for Android development.
  - **Eclipse with ADT Plugin:** Older option, less commonly used now.
- **Languages:**
  - **Java**
  - **Kotlin** (recommended for new projects)
- **Frameworks:**
  - **Jetpack Compose:** Modern UI toolkit for building native UIs.
  - **Retrofit:** For network operations.
  - **Room:** For local database management.

### 2. iOS

- **Tools and IDEs:**
  - **Xcode:** Official IDE for iOS development.
- **Languages:**
  - **Swift** (recommended)
  - **Objective-C**
- **Frameworks:**
  - **UIKit:** Framework for building user interfaces.
  - **SwiftUI:** Modern framework for declarative UI.
  - **Core Data:** For data management.
  - **Combine:** For reactive programming.

### 3. Windows

- **Tools and IDEs:**
  - **Visual Studio:** The primary IDE for Windows app development.

- **Languages:**
  - **C#**
  - **C++**
  - **VB.NET**
- **Frameworks:**
  - **.NET Framework:** For traditional Windows applications.
  - **.NET Core / .NET 5+**: For cross-platform applications.
  - **UWP (Universal Windows Platform):** For modern Windows apps.
  - **WinUI:** For native user interface components.

4. **Web Applications**

- **Tools and IDEs:**
  - **Visual Studio Code:** Popular lightweight code editor.
  - **Sublime Text:** Versatile text editor.
  - **WebStorm:** IDE specifically for JavaScript development.
- **Languages:**
  - **HTML/CSS/JavaScript**
- **Frameworks and Libraries:**
  - **React:** JavaScript library for building user interfaces.
  - **Angular:** TypeScript-based framework for building web applications.
  - **Vue.js:** Progressive framework for building UIs.
  - **Node.js:** For server-side JavaScript.

5. **Cross-Platform Mobile Development**

- **Tools and IDEs:**
  - **Visual Studio:** For Xamarin development.
  - **Android Studio:** For Flutter and Dart.
- **Frameworks and Languages:**
  - **Flutter (Dart):** Framework by Google for natively compiled applications.
  - **Xamarin (C#):** Framework for building cross-platform apps using .NET.
  - **React Native (JavaScript):** Framework for building mobile apps using React.
  - **Ionic (TypeScript/JavaScript):** Framework for building cross-platform mobile apps using web technologies.

6. **Game Development**

- **Tools and IDEs:**

    - **Unity:** Popular game development engine.
    - **Unreal Engine:** Advanced game development engine.

- **Languages:**
    - **C# (Unity)**
    - **C++ (Unreal Engine)**

- **Frameworks and Libraries:**
    - **Unity:** Offers a wide range of tools and assets for game development.
    - **Unreal Engine:** Known for high-fidelity graphics and robust toolset.

**7. Embedded Systems**

- **Tools and IDEs:**
    - **Arduino IDE:** For programming Arduino boards.
    - **PlatformIO:** For multiple embedded systems and boards.
    - **Eclipse with CDT:** For C/C++ development in embedded systems.

- **Languages:**
    - **C**
    - **C++**

- **Frameworks:**
    - **FreeRTOS:** Real-time operating system for embedded devices.
    - **MBED OS:** For ARM-based embedded systems.

## MVC Architecture

The MVC (Model-View-Controller) architecture is a design pattern used in software development to separate an application into three interconnected components. This separation helps in organizing code, improving maintainability, and facilitating the management of complex applications. Here's a brief description of each component in the MVC architecture:

**1. Model**

- **Description:** The Model represents the data and the business logic of the application. It is responsible for retrieving, storing, and processing data, as well as implementing the rules and logic that govern how data is manipulated.
- **Responsibilities:**
    - Manage and update application data.
    - Perform business logic operations.
    - Notify the View of any changes in the data.
- **Example:** In a web application for managing user accounts, the Model would

handle database interactions, such as retrieving user profiles or updating user information.

### 2. View

- **Description:** The View is responsible for displaying the data to the user. It represents the user interface and is concerned with how the data is presented, rather than how it is processed.
- **Responsibilities:**
    - Render the data provided by the Model into a user interface.
    - Display data to the user in a format that is easy to understand.
    - Receive user input and send it to the Controller.
- **Example:** In the user account management application, the View would be the web pages or UI components that present user information and allow interaction, such as forms and buttons.

### 3. Controller

- **Description:** The Controller acts as an intermediary between the Model and the View. It handles user input, processes it, and makes calls to the Model to retrieve or update data. It then updates the View to reflect the changes.
- **Responsibilities:**
    - Interpret user inputs and commands.
    - Update the Model based on user actions.
    - Select and return the appropriate View to display the updated data.
- **Example:** In the user account application, the Controller would handle actions such as user login, updating user details, and processing form submissions. It would then update the Model with new data and choose a View to present the updated information.

### Summary of MVC Architecture

- **Model:** Manages data and business logic. Notifies the View of any changes.
- **View:** Displays data and UI to the user. Receives user input.
- **Controller:** Handles user input, updates the Model, and selects the View to display.

### Benefits of MVC

- **Separation of Concerns:** By dividing the application into distinct components, MVC improves organization and maintainability. Each component has a specific role and can be developed or modified independently.
- **Scalability:** Makes it easier to scale and extend applications, as changes in one component (e.g., updating the View) do not affect other components.
- **Testability:** Enhances the ability to test individual components separately. For

example, business logic (Model) can be tested independently from the UI (View).

MVC is widely used in web development frameworks and applications, including frameworks like Ruby on Rails, Angular, and ASP.NET MVC, among others.

## Lecture 6 - Components of an App (00:09:44)

A modern application typically consists of several key components that work together to provide functionality and deliver a seamless user experience.

**1. User Interface (UI)**

- **Front-End:** The part of the application that users interact with. This includes:
  - **UI Design:** Visual elements like buttons, forms, and layouts.
  - **UX Design:** Ensures a good user experience through intuitive design and navigation.
  - **Responsive Design:** Ensures the app works well on various screen sizes and devices.

**2. Back-End**

- **Server:** The part of the application that processes requests, handles data, and performs business logic.
  - **Web Server:** Serves web pages or API endpoints (e.g., Apache, Nginx).
  - **Application Server:** Executes the app's business logic and processes requests (e.g., Node.js, Django, Ruby on Rails).
- **Database:** Stores and manages data (e.g., MySQL, PostgreSQL, MongoDB).
- **APIs (Application Programming Interfaces):** Interfaces for communication between different software components, including external services (e.g., RESTful APIs, GraphQL).

**3. Application Logic**

- **Business Logic:** Rules and processes that define how data is created, stored, and modified.
- **Services:** Modules or microservices that handle specific functions (e.g., payment processing, user authentication).

**4. Data Management**

- **Data Models:** Define the structure of data and relationships between different types of data.
- **Data Storage:** Methods for storing and retrieving data, including databases and file systems.

**5. Authentication and Authorization**

- **Authentication:** Verifies user identity (e.g., login systems, single sign-on).
- **Authorization:** Determines user permissions and access levels (e.g., role-based access control).

**6. Networking and Communication**

- **Networking:** Manages connections between the app and other systems or services.
- **Communication Protocols:** Includes HTTP/HTTPS, WebSockets, and other protocols used for data exchange.

**7. Security**

- **Data Encryption:** Protects data in transit and at rest.
- **Security Policies:** Defines how to handle user data, permissions, and other security aspects.
- **Threat Detection:** Monitors for and responds to potential security threats.

**8. Testing and Debugging**

- **Testing Frameworks:** Tools and frameworks for unit testing, integration testing, and end-to-end testing (e.g., Jest, Selenium).
- **Debugging Tools:** Tools for diagnosing and fixing issues in the application (e.g., Chrome DevTools, Log analysis).

**9. Deployment and Infrastructure**

- **Deployment Pipeline:** Automates the deployment of code changes to various environments (e.g., CI/CD pipelines).
- **Hosting and Infrastructure:** Services and environments where the application runs (e.g., cloud providers like AWS, Azure, or Google Cloud).

**10. Monitoring and Analytics**

- **Monitoring Tools:** Track application performance, errors, and system health (e.g., New Relic, Datadog).
- **Analytics:** Collect and analyze user data and application metrics to inform decisions (e.g., Google Analytics, Mixpanel).

**11. Integration with External Services**

- **Third-Party APIs:** Integrations with external services and APIs (e.g., payment gateways, social media).
- **Webhooks:** Allow real-time communication and notifications between services.

#### 12. Documentation

- **User Documentation:** Guides and help resources for end-users.
- **Developer Documentation:** Technical documentation for developers, including API docs and architecture diagrams.

These components work together to create a robust, scalable, and user-friendly modern application. Each component plays a crucial role in ensuring that the app meets user needs and operates efficiently.

## Lecture 7: Client Server and Peer To Peer Architecture (00:11:14)

### Peer-to-Peer (P2P) Applications and Architecture

**Peer-to-Peer (P2P)** is a decentralized network architecture where each participant, or **peer**, acts as both a client and a server. Unlike traditional client-server models where a central server handles all requests, in a P2P network, each peer can both request and provide resources or services. This design promotes decentralization, scalability, and resilience.

#### Key Concepts of P2P Applications

1. **Decentralization:**
    - **No Central Server:** There is no single central server managing the network. Instead, each peer in the network has equivalent status and can communicate directly with other peers.
    - **Distributed Resources:** Resources such as files, processing power, or data are distributed across all peers in the network.
2. **Scalability:**
    - **Dynamic Addition/Removal:** Peers can join or leave the network at any time, allowing the system to scale easily as more peers join or leave.
    - **Load Distribution:** Since each peer can handle requests and provide resources, the workload is distributed across the network.
3. **Redundancy and Fault Tolerance:**
    - **Redundancy:** Multiple copies of resources can exist on different peers, ensuring that the network remains functional even if some peers go offline.
    - **Resilience:** The network can adapt to the loss of individual peers, maintaining functionality as long as there are enough active peers.
4. **Resource Sharing:**
    - **Data Sharing:** Peers can share files or data directly with each other (e.g., file sharing applications).

- **Computational Resources:** Peers can share processing power or storage (e.g., distributed computing projects).

**Types of P2P Architectures**

1. **Pure P2P:**
    - **Definition:** In a pure P2P network, all peers have equal roles and capabilities, and there is no central server or authority.
    - **Examples:** Early file-sharing networks like Napster and Gnutella.
2. **Hybrid P2P:**
    - **Definition:** Hybrid networks combine P2P elements with central servers to enhance performance or functionality. Central servers may handle tasks such as indexing or directory services while the actual data exchange occurs between peers.
    - **Examples:** Modern file-sharing networks like BitTorrent and some social networks.
3. **Structured P2P:**
    - **Definition:** Structured P2P networks use specific algorithms and protocols to maintain a well-organized network. These networks often use Distributed Hash Tables (DHT) to efficiently locate resources.
    - **Examples:** Chord, CAN (Content Addressable Network).
4. **Unstructured P2P:**
    - **Definition:** In unstructured P2P networks, there is no predefined organization of the network. Peers connect to each other in a more ad-hoc manner, and resource location is typically less efficient.
    - **Examples:** Gnutella, early versions of file-sharing networks.

## Key Components of P2P Architecture

1. **Peers:**
    - **Nodes in the Network:** Each peer acts as both a consumer and provider of resources. They connect to other peers to share or request resources.
2. **Overlay Network:**
    - **Network Structure:** An abstract network that defines how peers are connected. It is built on top of the underlying physical network and can vary depending on the P2P protocol used.
3. **Discovery Mechanism:**
    - **Resource Location:** Methods for finding resources or other peers in the network. This can involve querying a central directory (in hybrid networks) or

using algorithms to search the network (in structured networks).

4. **Data Distribution:**
    - **Data Storage:** Mechanisms for distributing and replicating data across multiple peers to ensure availability and reliability.
5. **Communication Protocols:**
    - **Message Exchange:** Protocols for peer-to-peer communication, including request/response patterns, data transfer, and error handling.

**Advantages and Disadvantages**

**Advantages:**

- **Scalability:** Can handle large numbers of peers without requiring a central server.
- **Resilience:** More robust to peer failures due to redundancy and decentralization.
- **Cost Efficiency:** Reduces the need for centralized infrastructure and associated costs.

**Disadvantages:**

- **Security Risks:** Decentralized nature can make it harder to manage security and control access.
- **Resource Management:** Peers may have varying levels of resources and reliability, leading to inconsistent performance.
- **Complexity:** Managing and maintaining a decentralized network can be complex, particularly in terms of data consistency and coordination.

**Examples of P2P Applications**

1. **File Sharing:** Applications like BitTorrent allow users to share files directly with each other.
2. **Messaging:** P2P messaging apps like Signal use end-to-end encryption and decentralized architectures for secure communication.
3. **Distributed Computing:** Projects like **SETI@home** use P2P to harness the computing power of many peers to process scientific data.

P2P architectures are fundamental to various modern technologies and applications, leveraging the collective resources of many participants to achieve decentralized, scalable, and efficient systems.

## Client Server Models

The client-server model describes how different parts of a system or application interact in terms of requesting and providing services. The model can be categorized into different tiers, with each tier having distinct roles and responsibilities. Here’s a concise explanation of the 1-tier, 2-tier, and 3-tier client-server models:

### 1-Tier Architecture

**Definition:** Also known as a **single-tier** or **monolithic** architecture, the 1-tier model combines the presentation layer, application logic, and data storage into a single entity.

**Characteristics:**

- **Single Layer:** All components (user interface, application logic, and database) are integrated within one system or application.
- **Local Processing:** Everything runs on the client's machine or a single server, so there is no separation between the user interface and the data layer.
- **Simplicity:** Suitable for simple applications or small-scale systems where complex scalability or distribution is not required.

**Example:** A desktop application where the application logic, user interface, and data storage (like a local database) are all part of the same software.

**Pros:**

- **Ease of Deployment:** Simple to deploy and manage since there is only one component.
- **Performance:** Fast access to data since everything is local.

**Cons:**

- **Scalability Issues:** Difficult to scale as the entire application is bound to a single system.
- **Maintenance:** Harder to maintain and update, especially as the application grows.

### 2-Tier Architecture

**Definition:** In the 2-tier architecture, also known as **client-server architecture**, the system is divided into two layers: the client and the server.

**Characteristics:**

- **Client Layer:** This is the front-end where users interact with the application. It handles the user interface and application logic.
- **Server Layer:** This is the back-end where data is stored and managed. It handles the database management and business logic.

**Interaction:**

- The client sends requests to the server for data or services.
- The server processes the requests, interacts with the database, and sends back the results to the client.

**Example:** A traditional client-server application where a desktop client communicates with a server to retrieve or update data from a central database.

**Pros:**

- **Separation of Concerns:** Clear separation between the user interface and data management.
- **Scalability:** Better scalability compared to 1-tier, as the server can be upgraded or scaled independently of the client.

**Cons:**

- **Limited Scalability:** While it improves scalability, it still has limitations compared to more complex architectures.
- **Network Dependency:** Performance can be affected by network latency and server load.

### 3-Tier Architecture

**Definition:** The 3-tier architecture adds an additional layer to the 2-tier model, creating a more modular and scalable system. It separates the application into three distinct layers: the presentation layer, the application logic layer, and the data layer.

**Characteristics:**

- **Presentation Layer:** This is the client-side layer where the user interface is handled. It interacts with the user and sends requests to the application layer.
- **Application Layer (Business Logic Layer):** This layer handles the application's business logic, processes requests from the presentation layer, and interacts with the data layer.
- **Data Layer:** This layer manages the database or data storage, handling data retrieval, updates, and storage.

**Interaction:**

- The presentation layer communicates with the application layer, which in turn interacts with the data layer.
- This separation allows for more organized and scalable development and deployment.

**Example:** A web application where the web browser (presentation layer) interacts with a web server (application layer) that communicates with a database server (data layer).

**Pros:**

- **Scalability:** Improved scalability and flexibility, as each layer can be scaled independently.
- **Maintainability:** Easier to maintain and update, as changes in one layer do not necessarily affect others.
- **Modularity:** Clear separation of concerns makes the system more modular and manageable.

**Cons:**

- **Complexity:** More complex to design and implement compared to 1-tier and 2-tier architectures.
- **Performance Overhead:** Additional network communication between layers may introduce latency.

#### Summary

- **1-Tier Architecture:** All components are combined into a single system. Simple and suitable for small applications.
- **2-Tier Architecture:** Separates the client and server, with the client handling the user interface and the server managing data. Improves separation of concerns and scalability.
- **3-Tier Architecture:** Further separates the application into presentation, business logic, and data layers. Provides better scalability, maintainability, and modularity but adds complexity.

Each architecture has its own use cases and trade-offs, and the choice between them depends on factors such as application size, scalability requirements, and complexity.

### n-tier Architecture

**N-Tier Architecture** (also known as **multi-tier architecture**) is a software architecture model where an application is divided into multiple layers or tiers, each with specific responsibilities. The "N" in "N-Tier" signifies that the architecture can have more than three tiers, depending on the complexity and requirements of the application.

#### Key Concepts of N-Tier Architecture

1. **Separation of Concerns:**
    - **Modularity:** Each tier is responsible for a distinct aspect of the application, promoting separation of concerns. This modularity allows for easier maintenance and scalability.
    - **Isolation:** Changes in one tier typically do not affect other tiers, improving flexibility and manageability.
2. **Layers/Tiers:**
    - **Presentation Tier:** Handles the user interface and user interaction. This is where the application’s visual elements are managed, such as web pages or mobile app screens.
    - **Application Tier (Business Logic Tier):** Contains the business logic and rules. It processes requests from the presentation tier, applies business rules, and coordinates interactions between other tiers.
    - **Data Tier:** Manages data storage and retrieval. This tier interacts with databases or other data storage systems to handle data operations.

3. **Communication:**
    - **Client-Server Communication:** Each tier communicates with adjacent tiers through well-defined interfaces. For example, the presentation tier communicates with the application tier, and the application tier communicates with the data tier.
    - **Inter-Tier Communication:** Data and requests flow through the tiers, often via APIs or service calls.

#### Typical Tiers in N-Tier Architecture

1. **Presentation Tier:**
    - **Role:** Provides the interface through which users interact with the application.
    - **Examples:** Web browsers, mobile apps, desktop applications.
2. **Business Logic Tier:**
    - **Role:** Contains the core application logic, including business rules, workflows, and algorithms.
    - **Examples:** Application servers, service-oriented architecture (SOA), microservices.
3. **Data Tier:**
    - **Role:** Manages data storage, retrieval, and persistence. It interacts with databases and data sources.
    - **Examples:** Relational databases (SQL), NoSQL databases, file storage.
4. **Additional Tiers (Depending on Complexity):**
    - **Integration Tier:** Handles interactions with external systems, services, or third-party APIs.
    - **Service Tier:** Manages various services such as authentication, authorization, and messaging.
    - **Caching Tier:** Stores frequently accessed data to improve performance and reduce load on the data tier.
    - **Logging and Monitoring Tier:** Collects and manages logs and performance metrics for monitoring and troubleshooting.

#### Advantages of N-Tier Architecture

- **Scalability:** Each tier can be scaled independently based on load and demand. For example, you can scale the presentation tier to handle more users or the data tier to handle larger volumes of data.
- **Maintainability:** Changes in one tier do not directly impact other tiers, making it easier to update or maintain individual components.
- **Flexibility:** Allows for the use of different technologies and platforms for different

tiers. For example, the presentation tier might be web-based, while the data tier uses a different database technology.
- **Reusability:** Business logic and data access layers can be reused across different presentation tiers or applications.

#### Disadvantages of N-Tier Architecture

- **Complexity:** The architecture can become complex due to the number of tiers and interactions between them. This complexity can make development and troubleshooting more challenging.
- **Performance Overhead:** Additional network communication between tiers may introduce latency and impact performance, especially if tiers are distributed across different servers or locations.
- **Deployment Challenges:** Managing and deploying multiple tiers requires careful planning and coordination to ensure that all components work together seamlessly.

#### Example

Consider an e-commerce application with the following tiers:

1. **Presentation Tier:** The web interface where users browse products, add items to their cart, and complete purchases.
2. **Business Logic Tier:** The application server that handles order processing, payment validation, and inventory management.
3. **Data Tier:** The database server that stores product details, user information, and order history.

In more complex systems, additional tiers might be added, such as a service tier for managing customer authentication or a caching tier for frequently accessed product data.

**N-Tier Architecture** is widely used in modern software development due to its ability to provide a scalable, maintainable, and flexible system design. It is particularly useful for complex applications where different layers can be optimized independently.