The course "Internet Systems" (5CCS2INS) covers a wide range of topics across several layers of internet and web technologies.

The Internet and Web: A Layered Approach

The Internet's complexity in sending messages, such as from computer A to computer B, is managed through **layers of abstraction**. This layered model includes the Network Interface, Network, Transport, Web, and Application layers.

1. The Network Layer

The **Network layer** is responsible for ensuring that a **datagram** (also known as an **IP Message** or **packet**) is routed to a remote destination identified by its address. Its key aims include explaining the architecture of the Internet network layer and the IP protocol.

1.1 Naming and Topology

- **Topology**: A computer network is a network of devices. A topology is a graph that illustrates the connections of devices. **Hosts** are devices connected to networks by network interface controllers, while **routers** are devices connected to multiple networks using multiple network interface controllers. Networks can be **Local Area Networks (LANs)**, typically less than 1km (e.g., a building), or **Wide Area Networks (WANs)**, covering wider areas and potentially involving leased lines. Topologies can be incomplete graphs (not all nodes connected) and directed graphs (e.g., your desktop can access a server, but not vice-versa).
- **MAC Address**: The **Media Access Control (MAC) address** is a unique identifier printed in the hardware for a network interface controller. It consists of six groups of two hexadecimal digits, with manufacturers assigned specific ranges (e.g., Microsoft's range is 00:15:5D:00:00:00 - 00:15:5D:FF:FF:FF). MAC addresses are used for communication only at the physical layer.
- **IPv4 Addresses**: An **IPv4 address** is a 32-bit long identifier that identifies the network interface of a device over the Internet. A device may have multiple addresses. They are typically represented in **dotted notation**, with four bytes separated by dots, where each byte's value is between 0 and 255 (e.g., 137.73.113.232). A 32-bit address space provides a maximum of 4,294,967,296 addresses. Despite over 30 billion devices connected globally in 2023, the IPv4 lifespan was extended through technologies like **Network**

Address Translation (NAT) and private addresses, in addition to the advent of IPv6.

- **Structure**: An IPv4 address is composed of a **network prefix** (n bits) followed by a **host suffix** (32-n bits). All hosts within a network share the same network prefix. For instance, in the notation 137.73.113.232/24, the network address is 137.73.113.0 (suffix set to 0s) and the broadcast address is 137.73.113.255 (suffix set to 1s). These two addresses are reserved, leaving 2^(32-n)-2 addresses for hosts.
- **Special IPv4 Addresses**:
  - **Loopback address (127.0.0.1)** is a special IP number designated as the software loopback interface of a machine, with no associated hardware or physical network connection.
  - **Private addresses** are reserved ranges, such as 10.0.0.0/8 and 192.168.0.0/16, which can communicate with remote hosts over the Internet via NAT.
  - The **broadcast address** is 255.255.255.255/32.

- **IPv6 Addresses**: An **IPv6 address** is 128 bits long, offering a vast number of possible addresses (2^128, a significantly larger number than IPv4). It is noted using **colon hexadecimal notation**, consisting of eight sections of two bytes each, with each pair of bytes represented by four hexadecimal digits (e.g., fe80::227e:fd2e:3eea:b64). The IPv6 netmask typically uses the 64 leftmost bits (48 network address, 16 subnet address), and the device address is given by the 64 rightmost bits.

1.2 IP Message Structure and Routing

- **IP Message (Datagram/Packet)**: An IP message consists of a **header** and **data contents**. The data is typically provided by the transport layer (e.g., TCP) or other protocols like ICMP. The entire IP Message (Header + Data) becomes the **payload** for the layer below.
- **IPv4 Header**: The mandatory part of the IPv4 header is 20 bytes long, potentially followed by an optional section of the same length. The header identifies the source and destination of the message. The **source address** is identified so that error or acknowledgment messages can be returned. The **protocol field** specifies the protocol used in the data portion of the IP datagram. The **Total Length field** (16 bits) defines the entire packet size in bytes, including both header and data, with a minimum size of 20 bytes (header only) and a maximum of 65,535 bytes.

- **IP Routing Components**: IP routing involves three key components: **Routing tables**, a **Forwarding algorithm**, and **Routing protocols**.
    - **Routing Table**: A routing table contains entries with fields such as **Destination**, **Gateway** (the next-hop intermediary for routing packets), **Genmask** (network mask for the destination), and **Iface** (network controller interface).
    - **Forwarding Algorithm**: This algorithm uses the routing table to determine the next hop for a packet.
        - If the **Time To Live (TTL)** value of a packet is less than or equal to 1, an **ICMP Time Exceeded message** is sent to the source, and the packet is discarded.
        - Otherwise, the algorithm searches the routing table for an entry that matches the destination host address.
        - If no host-specific match is found, it looks for an entry matching a Genmask (a subnet match).
        - If still no match, it falls back to the default (0.0.0.0) entry.
        - If a next-hop (H) and interface (I) are found, the packet is forwarded to address H via interface I with the TTL value decremented by 1 (T-1).
        - If no route is available, an **ICMP Destination Unreachable message** is sent to the source, and the packet is discarded.
    - **Time To Live (TTL)**: **TTL** is an integer field (0-255) in the IP header that serves as a mechanism to prevent messages from being forwarded infinitely in a loop. Its value is decreased each time the message is forwarded.
    - **Routing Protocols**: These protocols are used to maintain dynamic routing tables.
        - **Between administrative domains**: Protocols like **BGP (Border Gateway Protocol)** are used to address political and commercial constraints, such as preferring specific providers or avoiding certain country networks.
        - **Within an administrative domain**: These protocols build on distance vector routing (e.g., Bellman-Ford algorithm). Early examples include **RIP (Routing Information Protocol)**, which did not scale well. Modern variants like **OSPF (Open Shortest Path First)** and **IS-IS (Intermediate-System to Intermediate-System)** are now used.

1.3 Fragmentation and Diagnostics

- **Fragmentation**: At the physical layer, protocols specify a **Maximum Transmission Unit (MTU)**, which is the maximum size for the data field within a frame. For Ethernet, the MTU is 1500 bytes. Since the maximum length of an IP datagram is 65,535 bytes, a datagram may need to be divided into smaller packets (fragments), each with a size smaller than the MTU, when traversing a network with a smaller MTU. Each fragment has its own header, and a fragment can be further fragmented if it passes through a network with an even smaller MTU.
- **Reassembly**: The reassembly of fragmented datagrams is **performed only by the final destination**. This is because each fragment may be routed via a different path. IP itself does not have the capability to request missing or corrupted fragments. The destination uses the **fragmentation offset** (a 13-bit field) to determine the correct order for reassembly. The **Identification field** (16 bits) uniquely identifies fragments belonging to the same original datagram. The **MF Flag (More Fragments)**, a 1-bit field, is set to 1 for all fragments except the last one, which has it set to 0. Every fragment, except the last, must be a multiple of 8 bytes.
- **Internet Control Message Protocol (ICMP)**: The IP protocol itself lacks a mechanism to query the network. ICMP assists in determining if a host or route is operational. It offers two types of messages:
    - **Query messages**:
        - **Echo-request and echo-reply** are used to check communication between two systems.
        - **Timestamp-request and timestamp-reply** determine the round-trip time for an IP datagram.
    - **Error reporting messages**: These messages report error situations back to the original source.
        - **Destination unreachable** is generated by a router when the destination address cannot be reached.
        - **Time exceeded** is generated by a router when the TTL field in the IP header is 1 or less.
- **Utilities using ICMP**:
    - **PING**: This utility sends ICMP echo requests to a target host/router and listens for echo responses to estimate the round-trip time.
    - **Traceroute**: This utility sends UDP messages with progressively increasing TTL values. When a router receives a message with TTL=1, it discards it and sends back an ICMP Time exceeded message. When

the destination receives the UDP message at an unexpected port, it responds with an ICMP Host Unreachable message.

1.4 Cross-Cutting Concerns of the Network Layer

- **Naming**: Network interface controllers are identified by IP addresses (32-bit for IPv4). The Internet's hierarchical structure supports decentralized and localized name management. IP addresses can be allocated statically or dynamically. IPv4 addresses are fully allocated, prompting the use of IPv6 addresses (128-bit). Other naming systems like host/domain names (managed by DNS) and MAC addresses are also relevant.
- **Session/State**: The IP layer is inherently **connection-less** (packet switched). Datagrams are routed independently and do not require a connection setup. IP provides a "best effort, unreliable message-delivery service". Datagrams are not persistently stored at the IP layer beyond the first delivery attempt by a router, meaning there are no re-retry mechanisms or management of received/unreceived packets.
- **Security**: The original IP protocol has **no built-in security**; packet headers and contents are visible. **IPSec** was introduced to provide security and is a mandatory feature for IPv6. IPSec is frequently used for **Virtual Private Networks (VPNs)**. Common attacks include IP Address spoofing, Denial of Service (DoS) attacks, and routing attacks.
- **Reliability**: The IP layer is **unreliable**, offering no guarantee of message delivery. Messages may arrive corrupted, out of order, duplicated, or be lost entirely, without notification of delivery failure. Network redundancy can provide potential alternate routes, contributing to network resilience if links fail.
- **Scalability**: The network layer generally exhibits **good scalability**. While the IPv4 address space has been depleted, technologies like NAT extended its lifetime. IPv6 offers a significantly larger address range. Overall bandwidth continues to increase as more physical links are added and protocols are deployed to leverage higher transmission rates.
- **Management**: The hierarchical network structure enables **decentralized management**. Network administrators are responsible for allocating IP addresses within their assigned blocks. Dynamic IP address allocation is handled by protocols like **DHCP (Dynamic Host Configuration Protocol)**. The network allows for the dynamic addition and removal of links, and routing protocols facilitate dynamic updates to router configurations.
- **Governance**: The **Internet Assigned Numbers Authority (IANA)** manages the IP Address space and standardizes protocol operations over IP. IANA

delegates IP address blocks to **Regional Internet Registries (RIRs)**, such as the RIPE NCC for Europe. RIRs, in turn, divide these pools into smaller blocks for delegation to Internet service providers and other organizations. This hierarchy supports decentralized management.
- **Description**: The IP layer is **not a self-described layer**. It possesses only limited querying capabilities through ICMP.

2. The Transport Layer

The **Transport layer** builds upon the Network layer to deliver packets specifically **between processes**, identified by a host's IP address and a **port number**. This enables **multiplexing**, where two hosts can have multiple simultaneous conversations.

2.1 Naming: Ports and Sockets

- **Ports**: TCP conceptually divides host communications into ports. Ports are 16 bits long, allowing values between 0 and 65535.
- **Sockets**: A **socket** is a combination of an IP address and a port.
- **Port Governance**: While the notion of a socket is part of the TCP specification, the assignment of sockets to services is outside it. IANA maintains a list of assigned ports, with services assigned on a first-come, first-served basis as per RFC6335.
    - **Well-known ports**: 0-1023 (e.g., HTTP: 80, SMTP: 25, HTTPS: 443).
    - **Registered ports**: 1024-49151, assigned by IANA for specific services.
    - **Dynamic, private, or ephemeral ports**: 49152-65535, allocated dynamically by a host and cannot be registered with IANA.

2.2 TCP Message Structure and Session Management

- **TCP Message (Segment)**: A TCP message, or segment, consists of a **header** and **data contents**. The data is provided by the Web or Application layer, and the full TCP Message becomes the payload for the underlying IP layer.
- **TCP Header**: The TCP header has a mandatory 20-byte section, followed by an optional section. It identifies the source and destination ports.
    - **Sequence number**: A number assigned to the first data byte of a segment, ensuring ordering of bytes within the current session.

    - **Acknowledgement Number (ACK)**: This number is 1 plus the number of the byte acknowledged so far, ensuring reliability of data delivery.
    - **Window size**: This field indicates the number of bytes a receiver is ready to receive, enabling flow control.
- **TCP Session (Connection)**: Unlike IP which is packet-switched and routes each datagram independently, TCP is **session-based (connection-oriented)**. Messages can only be sent once a session has been established between sender and receiver. A TCP connection is not a pre-planned route for messages; messages are still routed independently by the IP layer. The session, however, aids in achieving reliability and flow control.
    - **Lifecycle**: A TCP session has three phases: **session set up**, **data transmission**, and **session tear down**. The TCP protocol defines a **finite state machine** that specifies this lifecycle.
    - **3-Way Handshake (Session Setup)**: This process involves three messages:
        1. **Client synchronizes (SYN)**: The client initiates the connection by sending a SYN message.
        2. **Server acknowledges and synchronizes (SYN-ACK)**: The server responds with a SYN-ACK message.
        3. **Client acknowledges (ACK)**: The client completes the handshake by sending an ACK message.
        - Once these steps are complete, the connection is established.
    - **Session Tear-down Protocol**:
        0. **Client sends finalize (FIN)**: One side (e.g., client) initiates termination.
        1. **Server acknowledges and sends finalize (FIN + ACK)**: The other side (server) acknowledges and sends its own FIN.
        2. **Client acknowledges (ACK)**: The initiating side acknowledges the final FIN.
        - The connection is then terminated.

2.3 TCP Functionalities: Reliability and Flow Control

- **Segmentation**: Application or Web data is segmented into multiple TCP segments, each carrying its own TCP header. The **Maximum Segment Size (MSS)** is the largest IP datagram that the host can handle, minus the sizes of the IP and TCP headers. For example, with an Ethernet MTU of 1500 bytes, the MSS is typically 1460 bytes (1500 - 20 IP header - 20 TCP header). The MSS value is negotiated during session setup by being sent as a TCP header

option *only* in SYN segments. Each side reports its MSS, and the sender must limit the size of data in a single TCP segment to a value less than or equal to the MSS reported by the receiver.

- **Reliability**: TCP is considered 'reliable' because it ensures that all transmitted data is delivered at the receiving end. It provides mechanisms to recover from segments that are lost, damaged, duplicated, or received out of order. TCP mandates that an acknowledgment message (ACK) be returned after data transmission.
    - **Duplicate Acknowledgements**: If a sender receives multiple duplicate ACKs for the same segment (indicating that subsequent segments have been received out of order, or a segment is missing), it will retransmit the unacknowledged segment.
    - **Time-out**: The sender starts a timer after transmitting data. If an unacknowledged segment is not acknowledged before the timer expires, the sender retransmits it.
- **Flow Control (Sliding Window)**: TCP uses a **sliding window mechanism** to prevent the sender from transmitting data too quickly for the receiver to process reliably. In each TCP segment, the receiver specifies the amount of additional data (in bytes) it is willing to buffer for the connection in the **receive window field**.
    - The receive window shifts as the receiver obtains and acknowledges new data segments.
    - If the receiver advertises a window size of 0, the sender must stop transmitting data.
    - To make the protocol robust against lost window size update messages, the sender initiates a **persist timer**. If the timer expires without a new update, the sender sends a small packet to prompt the receiver to respond with an acknowledgment containing the updated window size.

2.4 Cross-Cutting Concerns of the Transport Layer

- **Naming**: Managed through well-known, registered, and dynamic/private/ephemeral ports.
- **Session/State**: Parties maintain state within a connection. The sender tracks the receiver's MSS, the next byte to send, and the first byte sent but not yet acknowledged, as well as the receiver's window size. The receiver tracks received segments, the successor of the last acknowledged byte, and its sliding window status.

- **Security**: The original TCP lacks security features; packet headers and contents are visible. **TLS (Transport Layer Security)** aims to provide end-to-end secure communication. Potential attacks include TCP reset attacks, TCP connection hijacking, and TCP SYN floods.
- **Reliability**: TCP is designed for reliability, ensuring delivery and providing recovery mechanisms for lost, damaged, duplicated, or out-of-order segments by requiring acknowledgments.
- **Scalability**: TCP benefits from the underlying IP layer's scalability. Its performance, in terms of throughput and latency, has been extensively studied. The TCP connection setup time is a non-negligible factor affecting performance. QUIC is an alternative protocol for HTTP/3 designed to minimize connection establishment time and overall transport latency.
- **Management**: Beyond IP management, the primary management concern is the dynamic allocation of dynamic, private, or ephemeral ports by a host.
- **Governance**: IANA standardizes TCP ports.
- **Description**: TCP is generally **not a self-described layer** and is not queryable.

3. The Web Layer

The **Web layer** operates at a higher level of abstraction than networking layers, with its universe of discourse being **resources**. The Web is viewed as a collection of static documents and a network of resources.

3.1 Resources and Representation

- **Resource**: A **resource** is anything that has identity. This can include documents, images, services (e.g., weather forecasts), collections of resources, or even abstract concepts like human beings, corporations, books, friendships, or mathematical operations. Not all resources are network-retrievable (e.g., human beings).
    - **Lifecycle**: Resources have a lifecycle where they can be created, their representations obtained, modified, and deleted. This is analogous to **CRUD (Create, Read, Update, Delete)** operations in data storage.
- **Representation**: All interactions with resources are mediated by the **uniform interface provided by HTTP**. Since direct interaction with a resource is not possible, an **abstraction called a "representation"** is used to reflect the past, current, or desired state of that resource in communications. A representation includes metadata and a potentially unbounded stream of data.

3.2 Naming Resources (URIs)

- **URI (Uniform Resource Identifier)**: A URI is a compact string of characters used for identifying a resource, acting as its name. URIs follow a well-defined syntax and are intended to reference resources.
    - **Opacity of URIs**: Clients are generally not meant to derive information by looking at the contents of the URI string itself; the URI is merely a name. Good practices are encouraged for designing and managing URIs.
    - **Variants**: **URL (Uniform Resource Locator)**, **URN (Uniform Resource Name)**, and **IRI (Internationalized Resource Identifier)** are variants of URIs.

3.3 HTTP Messages and Methods

- **HTTP Message Structure**: An HTTP message consists of a **Start line**, **Headers**, and a **Body**.
    - **Request Start Line**: Includes an HTTP method (e.g., GET, POST), an absolute URL path, and the HTTP version.
    - **Response Start Line**: Includes the HTTP version, a status code, and status text.
- **HTTP Methods and CRUD**: HTTP provides a uniform interface to Web resources. Clients interact by sending request messages to the target URI. The HTTP methods map to the lifecycle (CRUD) of a resource:
    - **Create**: **POST**.
    - **Read (Retrieve)**: **GET**.
    - **Update (Modify)**: **PUT**, **POST**, or **PATCH**.
    - **Delete (Destroy)**: **DELETE**.
- **HTTP Method Properties**:
    - **Safe**: A method is **safe** if it is read-only and does not modify the resource. GET, HEAD, OPTIONS, and TRACE methods are safe.
    - **Idempotent**: A method is **idempotent** if repeating the request multiple times has the same effect as sending it once. GET, HEAD, PUT, DELETE, OPTIONS, and TRACE are idempotent.
    - **Cacheable**: A method's response can be stored and reused. GET and HEAD are cacheable, while POST is rarely so.
- **Media Type**: The Content-Type header field specifies the nature of the data in the body of an HTTP message. It includes a media type, subtype identifier, and optional auxiliary information. Media types can be discrete (e.g.,

text/plain, image/jpeg, application/json) or composite (e.g., multipart/mixed). Character sets (e.g., us-ascii, iso-8859-1, utf-8) are specified via the charset parameter.
- **Multipart Messages**: HTTP supports composite messages, such as those used in file uploads, where different parts of the message body can have different content types.
- **Base64 Encoding**: Designed to encode arbitrary binary data into us-ascii form for transmission over channels that only reliably support text. It takes 24 bits of data, splits them into four 6-bit chunks, and converts each chunk into a letter. Padding with = is used for missing chunks.

3.4 Content Negotiation

- **Concept**: A resource can have representations in different formats, languages, or encodings. HTTP provides mechanisms for **content negotiation** between clients and servers to cater to varied user preferences, capabilities, or characteristics.
- **Server-Driven Negotiation (Proactive)**: Clients express preferences using specific HTTP headers like Accept (media type), Accept-Charset, Accept-Encoding, and Accept-Language. The server then selects the most suitable representation. However, servers may not always honor preferences (e.g., if they don't support negotiation or decide to send a non-conforming response instead of a 406 Not Acceptable error).
    - **Disadvantages**: It's difficult for the server to determine the "best" representation accurately, it can be inefficient to send client capabilities with every request, it complicates server implementation, and it limits the reusability of responses for shared caching. More precise headers can also increase HTTP fingerprinting and raise privacy concerns.
- **Agent-Driven Negotiation (Reactive)**: When faced with an ambiguous request, the server returns a page containing links to all available alternative resources, allowing the client to choose.
    - **Disadvantages**: The HTTP standard does not specify the format of the choice page, hindering automation. Additionally, an extra request is needed to fetch the actual resource, which can slow down availability.

3.5 Web Architecture and REST

- **Components**: The Web architecture operates on a **Client-Server model**. Clients can use **caches** to reuse previously retrieved representations. Servers

offer a uniform API. Clients and servers are not limited to browsers and websites; they can include various devices and applications.
    - **Intermediaries**: HTTP traffic can be intercepted by intermediaries that present the same uniform HTTP API to clients and interact with servers or other intermediaries, potentially using caching.
        - **Proxy**: An intermediary selected by the client (e.g., via local configuration) that receives requests for URIs and satisfies them through translation via the HTTP interface. Proxies are used for security, annotation, or shared caching (e.g., Squid).
        - **Reverse-Proxy/Gateway**: An intermediary that acts as a server for outbound connections but translates received requests and forwards them inbound to other servers. Gateways are used to encapsulate legacy or untrusted services, improve server performance through "accelerator" caching, or enable partitioning and load balancing (e.g., Nginx).
- **REST (Representational State Transfer)**: REST is an **architectural style** developed by Roy Fielding during his contributions to HTTP and URI standardization. It promotes a **resource-centric view**, which is strongly encouraged by HTTP specifications. REST defines four interface constraints:
    1. **Identification of resources**.
    2. **Manipulation of resources through representations**.
    3. **Self-descriptive messages**.
    4. **Hypermedia as the engine of application state (HATEOAS)**.
    - REST components communicate by transferring resource representations in standard data formats, dynamically selected based on recipient capabilities. The actual representation format is hidden behind the interface.
    - **Linking & State Transfer**: HTTP responses are enriched with URIs of related resources and their relationships. These links can be embedded in the application payload (e.g., <a> anchors, <link> in HTML header, JSON-LD, HAL) or inserted into HTTP response headers. Clients use these links to navigate and explore further resources.
    - **State Transfers**: In REST, the server does not "hold on" to the client's state. Instead, at any step, the client is sent a complete "picture" of where it can go next, including its current state and possible transitions. This is why it's called "Representational State Transfer".

    - It's important to note that not all uses of HTTP are compatible with HTTP specifications or REST principles. REST is an architectural style without strict criteria and is subject to interpretation.

3.6 Web Sessions and Cookies

- **Web Session**: An application-level concept representing a series of contiguous actions by a visitor on a website within a given timeframe. During an anonymous session, elements like shopping carts, browsing history, and personalized pages are maintained. In a non-anonymous session, shopping history is retrieved, and user preferences are made explicit.
- **HTTP Cookie**: A small piece of data sent by a server to a user's web browser, which the browser stores and sends back with subsequent requests to the same server. Cookies are primarily used to remember stateful information for the stateless HTTP protocol, such as keeping a user logged in.
    - **Applications**: Session management (logins, shopping carts), personalization (user preferences), and tracking (user behavior analysis).
    - **Modern Alternatives**: While historically used for general client-side storage, modern storage APIs are now recommended over cookies for this purpose.
    - **Cookie Law**: EU privacy legislation requires websites to obtain consent from visitors before storing or retrieving information on their devices.
    - **REST Implications**: Cookies contain user-specific configuration choices or tokens, but they are typically attached to all future requests for an entire site. This can lead to a mismatch between the browser's application state and the stored state in the cookie, particularly when using browser history (e.g., "Back" button), causing confusion. According to Fielding, HTTP cookies represent an "inappropriate extension" to the protocol as they introduce site-wide state information that contradicts the desired properties of a generic interface.

3.7 Cross-Cutting Concerns of the Web Layer

- **Naming**: URIs act as opaque names for resources. Good practices are essential for URI design and management.
- **Session/State**: While REST resources are stateful, HTTP itself is stateless. Clients and servers exchange resource representations. Web sessions are an

application concept, and state can be managed using mechanisms like cookies or client-side web storage. The distribution of web application state across clients and servers involves various, sometimes conflicting, requirements related to privacy, auditability, persistence, and adherence to REST principles.

- **Security**: HTTP itself has **no security**. **HTTPS** provides end-to-end security based on **TLS**. Other security aspects include Simple Access Authentication, Authorization and Identity frameworks, and Cross-Origin Resource Sharing. Security considerations involve attacks exploiting URI interpretation, browser disclosure of personal information, server disclosure of sensitive URI information, and browser fingerprinting.
- **Reliability**: HTTP's reliability is tied to its underlying TCP layer and the properties of its methods (safe and idempotent methods contribute to predictability).
- **Scalability**: The Web architecture supports scalability through features like replication, load balancing, caching, and proxies.
- **Management**: Management at the Web layer is decentralized; services can be deployed locally without registration, and URIs are managed within their respective domain names.
- **Governance**: Key governing bodies and standards include W3C, IETF, Unicode, IANA, EU privacy laws, and Certificate Authorities (CAs) with browser developers.
- **Description**: The Web layer is **self-describable**. This is evident through explicit message headers, metadata about payload (Content-Type, Content-Language, charset), the use of OPTIONS and HEAD methods to query services, and the inclusion of links (HATEOAS). This self-descriptive nature is a step towards the **Semantic Web**, where descriptions are included in the payload using formats like HTML, RDF (Resource Description Framework), RDFa (RDF embedded in HTML), and JSON-LD (RDF overlaid over JSON).

4. Internet and Web Security

Security mechanisms are essential to support end-to-end encryption at the Transport and Web layers, and authentication and authorization at the Web layer.

4.1 End-to-End Encryption

- **Purpose**: End-to-end encryption ensures that only the communicating parties can read the information exchanged, preventing eavesdropping and man-in-the-middle attacks.
- **SSL/TLS**: **SSL (Secure Socket Layer)**, developed by Netscape in 1995, was an early encryption-based Internet security protocol superseded by **TLS (Transport Layer Security)**.
    - **TLS Aims**:
        - **Authentication**: The server side is always authenticated, while the client side is optionally authenticated.
        - **Confidentiality**: Data sent over the channel after establishment is only visible to the endpoints.
        - **Integrity**: Data sent over the channel cannot be modified by attackers.
    - **Evolution**: TLS has evolved through versions (1.0, 1.1, 1.2, 1.3), with older SSL versions (2.0, 3.0) and early TLS versions (1.0, 1.1) now deprecated. TLS 1.3, the latest version as of 2018, focuses on performance (fewer round trips) and improved cryptography.
    - **Handshake**: The TLS handshake involves **key exchange**, **server parameters**, and **authentication**. It occurs after the TCP 3-way handshake is completed and enables bidirectional secure communication.
    - **Applications**:
        - **HTTPS**: TLS secures HTTP traffic, forming the **HTTPS protocol**, which typically uses port 443. URIs can use both HTTP and HTTPS schemes. Certificates are associated with the URI hostname for secure connections.
        - **Secure Mail**: Protocols like SMTP can be protected by TLS.
        - **VPNs**: TLS can be used for tunnelling an entire network stack to create VPNs (e.g., OpenVPN, OpenConnect).
        - **DNS Security**: **DNS over HTTPS (DoH)** on port 443 and **DNS over TLS (DoT)** on port 853 enhance DNS privacy and security.
    - **Performance Cost**: While necessary, the TLS handshake adds a setup cost to TCP connections.

4.2 HTTP Simple Authentication

- **Authentication vs. Authorization**: **Authentication** is the process of identifying an individual (e.g., via username/password or fingerprints), ensuring they are who they claim to be. It does not, however, grant access

rights. **Authorization**, on the other hand, is the process of granting or denying access to a network resource, occurring after successful authentication.

- **Basic Authentication (RFC7617)**: This scheme uses the Authorization HTTP header field. It relies on a **Base64 encoding** of the username and password string (e.g., user:hello becomes dXNlcjpoZWxsbw==).
    - **Vulnerability**: **Base64 encoding is reversible**, meaning credentials are transmitted in cleartext over the network. This is its most serious flaw, and it **SHOULD NOT be used without enhancements like HTTPS** for sensitive information. The danger is compounded by users reusing passwords, risking unauthorized access to other systems. Servers respond with a 401 Unauthorized status and a WWW-Authenticate header to challenge the client.
- **Digest Access Authentication (RFC7616)**: Instead of a plain password string, the client submits H1, a hash (e.g., SHA-256) of username:realm:password. The server only needs to store a copy of this hash for authorized users.
    - **Vulnerability**: While an improvement, it is still vulnerable to **dictionary attacks**. It **SHOULD be used only with passwords having reasonable entropy** and, ideally, **over a secure channel like HTTPS**. It does not provide strong authentication.

4.3 Web Authorization and Identity Frameworks

- **Limitations of HTTP Simple Access Authentication**: It is insufficient for modern web applications due to client-side issues (plain credentials, no user verification, no time limits, no specific purpose) and server-side challenges (authorization decisions, managing sensitive info, compliance, repeated challenges for each service).
- **Motivation for Frameworks**: Modern scenarios require users to grant third-party services (e.g., a printing service) access to protected resources (e.g., photos on a photo-sharing service) without sharing their primary username and password. Instead, the user authenticates directly with a trusted server (e.g., by the photo-sharing service), which then issues **delegation-specific credentials (access tokens)** to the third-party service.
- **OAuth 2.0**: This is a **standard protocol for authorization**. It provides specific authorization flows for various application types (web, desktop, mobile) and is developed by the IETF OAuth Working Group.
- **OpenID Connect 1.0**: This is an **identity layer built on top of OAuth 2.0**. It allows clients to verify the identity of the end-user (based on authentication by

an Authorization Server) and to obtain basic profile information in an interoperable, REST-like manner.

- **ID Token (JSON Web Token - JWT)**: The ID token acts like an identity card. It is in a **standard JWT format** and is digitally signed by the OpenID Provider (OP).
    - **Features**: It asserts the user's identity (sub), specifies the issuing authority (iss), is generated for a particular client/audience (aud), may include a nonce, has issue (iat) and expiration (exp) times, and can contain additional requested user details (e.g., name, email).
- **Protocol Flow (Key Aspects)**: The OAuth 2.0/OpenID Connect protocol involves the User (Resource Owner), Client Application (e.g., printing app), Resource Server (e.g., photo-sharing service), OpenID Provider, and Authorization Server.
    - The user connects with the client app, which redirects the user to the OP's login page (potentially via a third-party identity provider).
    - The OP performs authentication and returns a single-usage token to the client app.
    - The client app converts this token into an identity token (JWT) and an access token.
    - The client app then uses the access token to request resources from the Resource Server, which validates the access token.
    - This approach exploits HTTP interaction patterns (redirects, callbacks). The browser only sees a single-usage token, while the ID token is visible to the client application, not the browser. The ID token is signed by the OP, and there's a back-channel for communication between the Resource Server and OP. This framework also supports token revocation and fine-grained permissions (read vs. write).

4.4 Cross-Origin Resource Sharing (CORS)

- **History and Same Origin Policy (SOP)**: Innovations like Netscape's "magic cookie" (1994) and JavaScript (1995), coupled with the Document Object Model (DOM) API (1995-97), enabled rich client-side interactivity. This richness necessitated a way to securely interact with various entities within the browser. Netscape introduced the **Same-Origin Policy (SOP)** in 1995, initially to protect DOM access, and later broadened its protection scope.
    - **Notion of Origin**: An origin is defined by three common characteristics of a web resource URI: **scheme (protocol)**, **hostname**

(domain/subdomain), and **port**. Any resources sharing these three elements have the same origin.
    - **SOP Function**: The SOP is a browser-based defense mechanism that permits scripts in one web page to access data in a second web page *only if* both pages have the same origin. Without SOP, a malicious site could, for example, embed a legitimate bank's login page in an iframe, and JavaScript on the malicious site could then access sensitive DOM elements (like account balances) from the legitimate site without user consent or knowledge.
- **Relaxing SOP: Why CORS?**: While SOP is crucial for security (preventing JavaScript from accessing resources on other origins by default), there are legitimate scenarios where cross-origin access is desirable, especially when an API needs to be shared across multiple domains.
- **CORS (Introduced 2009)**: **Cross-Origin Resource Sharing (CORS)** defines a mechanism for a browser and server to interact and determine if a cross-origin request is safe to allow. It provides greater flexibility than purely same-origin requests.
    - **Mechanism**: CORS leverages **HTTP Extension Headers** and a **Preflight Protocol**.
        - **HTTP Headers**:
            - The client sends an Origin request header indicating where the request originates (scheme, server name, port).
            - If the server accepts the origin, it responds with an Access-Control-Allow-Origin header, which can also use a wildcard (*) to allow any origin.
        - **CORS Success**: Occurs if the remote site supports CORS and accepts the origin, leading to a successful call in the browser.
        - **CORS Failure**: Can happen if the remote site supports CORS but *does not* accept the origin (server returns an error), or if the remote site ignores CORS headers but the browser's CORS policy is not met (browser flags an error despite a 200 OK from server).
        - **Preflight Requests**: For "complex requests" (e.g., XMLHttpRequest, JavaScript/HTTP methods that modify data like PUT, or POST with certain MIME types), browsers first send a **"preflight" request** using the **HTTP OPTIONS method**. This request solicits information from the server about supported

methods and headers. If the server "approves" (e.g., with a 204 No Content response and Access-Control-Allow-Methods header), the browser then sends the actual request. Resources can also notify if "security credentials" should be sent with requests.

- **Key Aspects**: The decision of whether JavaScript is allowed to access foreign domains via XMLHttpRequest is made in the browser. A resource cannot protect itself from arbitrary clients, and a modified browser might not implement CORS.

4.5 Cross-Cutting Concerns of Internet and Web Security

- **Security**: Involves end-to-end encryption (HTTPS/TLS), simple access authentication, authorization/identity frameworks, and CORS. The browser itself, being a sophisticated software system, is subject to many potential attacks.
- **Scalability**: Security protocols have become more involved, adding overhead (e.g., HTTPS setup costs, OAuth interactions, CORS preflight requests).
- **Governance**: Governed by organizations such as W3C, IETF, OpenID, WHATWG, EU privacy laws, and Certificate Authorities working with browser developers.
- **Description**: The Web layer is self-describable, using WWW-Authenticate headers and OPTIONS/HEAD methods to query services for security and CORS information.

5. Web Development Fundamentals

Web development involves HTML, CSS, and JavaScript. Front-end and back-end aspects are often distinguished, with HTML, CSS, and client-side JavaScript applicable to both, and server-side JavaScript, Java, Python, etc., for back-end.

5.1 HTML (Hypertext Markup Language)

- **Hypertext**: A term coined by Ted Nelson in the 1960s, referring to non-linear text that includes links to other text, allowing multiple reading paths. Clicking a link triggers an HTTP request to download the linked page.
- **HTML as a Language**: HTML is the language used to write web pages (resources). It uses predefined elements and attributes to describe page structure and presentation. When a browser visits a page, HTTP (over

TCP/IP) downloads the HTML document. HTML is an XML-like language for encoding hypertext documents.

- **HTML5**: The current standard, HTML5, can be written in forms similar to HTML 4.01 or XHTML 1.0 (a fully XML version). HTML5 provides additional elements and attributes for features like video embedding, typesetting, and new form input types. The WHATWG maintains a Living Standard for HTML.
- **HTML Document Structure**: Every HTML document has <html> as its root element, split into a <head> (containing document information like <title>) and a <body> (containing the webpage's content).
    - **Body Content**: Text within <body> appears in the browser. By default, browsers ignore most whitespace. The <pre> element allows for pre-formatted content where line breaks and spaces are preserved.
- **Markup**:
    - **Structural Markup**: Elements like <h1> (for headings) and <p> (for paragraphs) provide structural detail. HTML5 added elements like <SECTION>.
    - **Presentational Markup**: Elements like
 (line break), <b> or (bold text), and <i> or (italic text) specify presentation. Note that <CENTER> is deprecated in HTML5, with CSS text-align being the preferred method for centering.
    - **Lists**: HTML supports ordered lists (<ol>) and unordered lists (<ul>), with list items defined by <li> elements.
    - **Tables**: Complex structures are created using <table>, <tr> (rows), and <td> (cells).
    - **Images**: The <img> element includes images, with the SRC attribute providing the URL (often relative) for the image, which the browser downloads via an HTTP GET request.
    - **Links**: The <a> element creates links, with the HREF attribute specifying the linked URL.
    - **Fragment Anchors**: These allow linking to a specific position within a document. An anchor is named using <a> with an ID attribute (e.g., <a id="cats"/>), and linked to by appending #anchor to the URL (e.g., http://animals.com/all.html#cats).

5.2 HTML Forms

- **Structure**: A form is an HTML page component defined by a <form> element, which specifies the HTTP method and the server/destination for the request. Forms contain one or more input elements (<input>, <textarea>, <button>,

<select>, etc.), each typically having name and type attributes. A submit button is included using <input type="submit">.

- **Submission**: Upon clicking the submit button, form data is sent via HTTP to the URL specified in the form's action attribute. By default, data is sent in application/x-www-form-urlencoded MIME type. If a file input is used, enctype should be multipart/form-data and the method must be POST.
    - **Encoded Form Data**: For application/x-www-form-urlencoded, data is formatted as InputName=InputValue strings, with spaces converted to + and other URL-unfriendly characters converted to ASCII hexadecimal (e.g., %27 for an apostrophe), all concatenated with &.
- **HTTP GET Form Submission**: If the form's METHOD attribute is GET, the encoded form data is appended to the action URL with a ? (e.g., /personalData?fullname=Samhar+Mahmoud&eyecolour=Brown). The server then accesses this data as parameters. GET is preferred for safe and idempotent operations and can be faster due to less data exchange.
- **HTTP POST Form Submission**: If the METHOD is POST, the form data becomes the HTTP message's entity body, sent to the URL in the ACTION attribute without a ? extension. POST is generally better for large data volumes and security.

5.3 Markup Languages (General Concept) and XML

- **Markup Language**: A format for documents where text is annotated ("marked-up") with computer-parseable information. These annotations indicate how text should be interpreted, presented, or how pieces of text relate. Historically, markup helped "mark-up men" in presentation. Non-overlapping markup inherently describes a hierarchy of annotations.
- **XML (eXtensible Markup Language)**: A general-purpose markup language primarily used for organizing, storing, and transferring arbitrary data, not just readable text documents. XML tags describe the structure of content.
- **XML vs. HTML**: XML and HTML were designed with different goals. XML focuses on *what* data is carried, while HTML focuses on *how* data looks when displayed. Unlike HTML, XML tags are not predefined.
- **XML Essentials**:
    - **Tags and Elements**: Data is marked up with opening (<NAME>) and closing (</NAME>) tags, which are case-sensitive. A section of a document enclosed by tags is an element. Elements can be nested, forming a hierarchy where meaning can be contextualized by surrounding tags.

    - **Attributes**: Elements can be parameterized with attributes, which consist of a name-value pair (in quotes) and contain data related to the specific element. Attributes are placed inside an element's opening tag. xmlns is a special attribute for namespace declarations.
    - **Document Structure**: An XML document contains a single root element enclosing all other elements. It can be preceded by a **prolog** containing an XML Declaration and an optional Document Type Declaration (DTD).
    - **XML Declaration**: The <?xml...? > tag specifies the XML version, encoding (character set), and whether the document is standalone.
    - **Entities**: Since XML uses specific characters for delimiting elements and attributes, **entities** (special strings like & for & or < for <) are used to represent these characters within the marked-up text. Parsers replace entities with their corresponding characters.
    - **Comments**: XML supports comments (<!-- Comment -->) that are ignored during parsing and can appear anywhere after the XML declaration to aid human readers.
- **XML Namespaces**: Crucial for interoperability when XML is exchanged between applications, as they help distinguish tags with different meanings but identical names (e.g., <title> for a book chapter vs. a person's honorific). A namespace is identified by a URI, and every element and attribute name has a namespace.
    - **Declaration Methods**:
        - **Default Namespaces**: An xmlns="NS-URI" attribute in an element sets a default namespace for that element and all elements within its hierarchy, unless overridden.
        - **Namespace Prefixes**: An xmlns:PREFIX="NS-URI" attribute defines a prefix, which precedes element/attribute names to indicate their namespace (e.g., <book:chapter xmlns:book="...">).
- **XML Validation**: XML documents can be validated against an **XML Schema**, which is a language for expressing constraints about XML documents. An XML document with correct syntax is "Well Formed". If it also validates against a schema, it is considered "Valid". Common schema languages include DTDs, Relax-NG, Schematron, and **W3C XSD (XML Schema Definitions)**. XSD documents are themselves written in XML and describe the structure of an XML document. They use the namespace

http://www.w3.org/2001/XMLSchema. Simple type elements are asserted using <element name="ELEM-NAME" type="TYPE"/>.

5.4 JSON (JavaScript Object Notation)

- **Definition**: JSON is a **text-based, lightweight, human-readable, and language-independent data exchange format** that uses key-value pairs.
- **JSON Object**: A JSON object maps keys to values, where values can be other JSON objects or various data types.
- **Datatypes**: JSON supports strings, numbers, booleans, arrays, objects, and null values. Unlike JavaScript objects, JSON does not permit comments.
- **JSON vs. JavaScript Objects**: JSON format is almost identical to native JavaScript objects, but with key differences: JSON keys *must* be strings, while JavaScript object keys can be strings, numbers, or identifier names. JavaScript values can also include functions, dates, or undefined, which JSON does not natively support.
- **Parsing and Stringifying**: To convert JSON text received from a server into a manipulable JavaScript object, JSON.parse() is used. When sending or returning data to a server, JavaScript objects are converted into a JSON string using JSON.stringify(). Special handling might be needed for dates, arrays, and functions.

5.5 JavaScript

- **Purpose**: JavaScript is used to create **client-side dynamic functionality** in web applications. It enables customization of interactions through events, form validation (though server-side validation is still critical for security), and connections to servers.
- **Characteristics**: JavaScript "derives" from ECMAScript and can be used on both client and server sides. It's built into browsers (though users can disable it). Each browser tab operates in its own execution environment for security, though advanced techniques exist for data/code exchange. It is an interpreted language (though modern browsers use just-in-time compiling for efficiency). Running order is important, and it is case-sensitive.
- **Basics**: JavaScript supports **dynamic typing** (no explicit type declaration for variables) and various **datatypes** including strings, booleans, integers, objects, and arrays. **Functions** are blocks of code for specific tasks, promoting reuse and modularity.

- **Adding Scripts to HTML**: Scripts are included using the HTML <script> element, typically with type="text/javascript". They can be written **in-line** (bad practice), **internally** within the <head> section (still not ideal for reuse), or most commonly, **externally** by linking to a separate file in the <head>.
- **Execution**: Scripts execute when a <script> tag is encountered in the document or when an event occurs (e.g., page load, click, keypress, form submission).
- **jQuery**: A JavaScript library that simplifies common tasks and encapsulates concepts like AJAX. It uses a basic syntax like $(selector).effect(handler), where $ is the jQuery object, selector targets elements, effect specifies an action, and handler is a callback function. Many other frameworks and libraries exist (e.g., Vue.js, React, Angular).

5.6 DOM (Document Object Model)

- **Definition**: The DOM is a **programming interface for HTML and XML documents**. It represents the structure of a web page, its elements, and attributes as objects that can be manipulated programmatically.
- **Standardization**: It is defined by a living standard from WHATWG. While its design is language-independent, its implementation can vary among browsers.
- **DOM Events**: These are actions that occur due to user interaction with an element/node (e.g., click, keypress), network activity, or changes in the DOM tree's state. Examples include mouse events (click, dblclick), keyboard events (keypress, keydown), form events (submit, change), and document/window events (load, resize). DOM events enable programmatic interaction and dynamic functionality in web pages.

5.7 AJAX (Asynchronous JavaScript and XML)

- **Concept**: AJAX is a group of technologies that allow JavaScript to make requests to a web server and process responses **without reloading the entire web document**. This enables the creation of more dynamic and responsive web applications.
- **History**: AJAX gained prominence in the mid-1990s with applications like Gmail and Google Maps.
- **Components**: Key components include HTML, CSS, JavaScript, the DOM, and the **XMLHttpRequest Object**. While "XML" is in its name, AJAX can

transmit data in various formats, including JSON, HTML, or any other data format.

- **How it Works**:
    1. An event occurs on a web page.
    2. A **XMLHttpRequest object** is created by JavaScript.
    3. This object sends a request to a web server.
    4. The server processes the request and sends a response.
    5. JavaScript reads the response.
    6. JavaScript performs an action, such as updating a part of the web page, without a full reload.
- **XMLHttpRequest Object**: This API provides scripted clients with the functionality to transfer data between the client and server. All modern browsers support it, and it's defined by a WHATWG living standard.
    - **Sending Requests**: Requests are opened using open(method, url, async) (specifying HTTP method, URL, and whether it's asynchronous) and sent using send() or send(string) for POST requests.
    - **Receiving Responses**: The onreadystatechange property defines a function to be executed when the request's status changes. The readyState property tracks the request's state (0: unsent, 1: opened, 2: headers received, 3: loading, 4: done), while status and statusText convey the HTTP response status (e.g., 200 OK, 404 Not found).
- **Security Issues**: AJAX applications can be susceptible to security threats, such as using a user's session data on a malicious server or **Cross-site scripting (XSS)**.
- **Libraries and APIs**: Libraries like **jQuery** simplify AJAX implementations by encapsulating browser-specific syntax. The **Fetch API** is another modern API used for making network requests.

6. Web Services

Web Services are software functionalities deployed using internet and web technology. They apply principles like decentralization and distributed administration.

6.1 Service-Oriented Computing

- **Services**: Services have a public **interface** (constant) and a private **implementation** (changeable by the owner). Messages exchanged between a client and a service provider must conform to this interface.

- **Interfaces**: Each service states the protocols it supports, which can be specific to its business function. The description of these protocols is its interface. Multiple services providing interchangeable functionality can share the same interface definition.
- **Advantages**: Web services offer **reusability** (for third parties or within a company), **language transparency** (can be written in different programming languages but communicate via common formats like XML/JSON), **usability** (secure data access for other systems), and **deployability** (easier to make available due to Internet standards).
- **Considerations**: Potential challenges include **latency** (time between request and response) and **partial failure** (due to network errors or server congestion).

6.2 SOAP (Simple Object Access Protocol)

- **Components**: A SOAP service involves a **Service Provider**, a **Service Requester**, and a **Service Registry** (e.g., UDDI - Universal Description Discovery and Integration) where providers publish and requesters find information about web services.
- **WSDL (Web Services Definition Language)**: WSDL is an **XML language used for describing services**. It specifies the form of messages a service understands or produces. Key elements of a WSDL definition include <types> (defining XML Schema data types), <message> (defining data elements for operations), <portType> (describing operations and messages), and <binding> (defining protocol and data format).
- **SOAP Messages**: SOAP is a **Web Service communication protocol**. SOAP messages have a common structure, expressed as an XML element hierarchy, with an outer **envelope** containing a **header** and a **body**. It can also include an optional Fault element for error information.
- **Advantages**: SOAP is **language, platform, and transport independent** (supporting HTTP, SMTP, UDP, etc.). It works well in distributed enterprise environments and is standardized, providing significant pre-built extensibility through WS* standards. It has **built-in error handling** and tighter security via **WS-Security** (in addition to SSL support). Designing complex operations (transactions, security) can be less complex with SOAP, requiring less application-layer coding.
- **Disadvantages**: SOAP generally has **poorer performance**, **more complexity**, and **less flexibility** compared to REST. It requires more bandwidth and computing power.

6.3 RESTful Services (Representational State Transfer)

- **Nature**: A RESTful service is a **stateless service**, meaning the Web Server does not store any information about the client session. Requests and responses typically use XML or JSON formats.
- **Advantages**: RESTful services are generally **faster** and use **less bandwidth**, requiring fewer resources and no expensive tools for interaction. They have a smaller learning curve and are closer in design philosophy to other web technologies. REST allows a **greater variety of data formats** (plain text, HTML, XML, JSON, YAML, etc.), unlike SOAP which is XML-only. They offer **superior performance**, especially through caching for non-dynamic information. REST is the protocol frequently used by major services like Yahoo, Ebay, Amazon, and Google. It's easier to integrate with existing websites without refactoring the infrastructure.
- **Disadvantages**: REST generally offers **less security** and is **not as suitable for distributed environments** as SOAP.

6.4 SOAP vs. REST Comparison (Key Differences)

- **Design**: SOAP is a standardized protocol with predefined rules, whereas REST is an architectural style with looser guidelines.
- **Approach**: SOAP is function-driven (e.g., "getUser"), while REST is data-driven (e.g., a "user" resource).
- **Statefulness**: SOAP is stateless by default but can be made stateful; REST is inherently stateless (no server-side sessions).
- **Caching**: SOAP API calls cannot be cached; REST API calls can be cached.
- **Security**: SOAP uses WS-Security with SSL support and built-in ACID compliance; REST supports HTTPS and SSL.
- **Message Format**: SOAP is limited to XML; REST supports plain text, HTML, XML, JSON, YAML, and others.
- **Transfer Protocols**: SOAP can use HTTP, SMTP, UDP, and others; REST typically uses only HTTP (as it is an architectural style that can be applied to other protocols).
- **Recommended Use**: SOAP is recommended for enterprise applications, high-security applications, distributed environments, financial services, and telecommunication services. REST is recommended for public APIs for web services, mobile services, and social networks.

7. Semantic Web and Linked Data

The **Semantic Web** aims to make the vast information on the web comprehensible to software, not just humans. This is achieved by including **computer-readable information** alongside the current human-readable content.

7.1 RDF (Resource Description Framework)

- **Core Structure**: RDF is a data structure for making computer-readable statements. An RDF document is a set of **statements (or triples)**, each asserting something about a resource or its relation to another resource.
- **Triples**: Every statement consists of three parts:
    - **Subject**: The resource the statement is about (e.g., https://transtechsocial.org/news).
    - **Object**: The resource or value the subject is related to (e.g., Angelica Ross).
    - **Predicate**: How the subject and object are related (e.g., contributor). Statements can be written in the form: Subject Predicate Object.
- **Resources**: Subjects and sometimes objects of RDF statements are **resources**, identifiable by **URIs** (Uniform Resource Identifiers). These can represent webpages, physical things (people, organizations), or abstract concepts (happiness, pi). URIs also uniquely identify predicates (e.g., http://purl.org/dc/elements/1.1/contributor).
- **Values**: Objects of RDF statements are not limited to URIs; they can also be literal data values like strings or integers (e.g., foaf:firstName "Robert").
- **Vocabularies**: A **vocabulary** is a set of terms defined together to allow descriptions within a specific domain, similar to namespaces. Each term is a URI, and all URIs within a vocabulary typically start with the same string (e.g., http://purl.org/dc/terms/ for library data terms like creator or publisher).
- **RDF Graphs**: A set of RDF statements is often called an **RDF graph** because the information forms a graph with resources and values as nodes, and predicates as edges.
- **Turtle Format**: RDF statements can be encoded in various formats, including XML. **Turtle (Terse RDF Triple Language)** is a human-readable format often used, which allows for the abbreviation of long URIs using **prefixes** declared at the start of the document (e.g., dc:creator where dc: maps to http://purl.org/dc/terms/).
- **RDF Storage**: RDF data can be stored in **TripleStores** or **RDF Stores**, which are often implemented using **Graph Databases** (e.g., Neo4j, Amazon Neptune) that store data in a graph structure of nodes and edges. These databases are typically queried using **SPARQL**.

7.2 Ontologies and OWL (Web Ontology Language)

- **Ontologies**: While RDF allows software to read statements, it doesn't enable "reasoning" about them. An **ontology** is data that encodes the **meaning of resources**, such as their type and what is known about resources of that general type. It is a specification of a domain described by its concepts and the relationships between them, going beyond simple vocabularies or taxonomies. Ontologies allow software to infer new knowledge (e.g., if "Every Estuary flows into a Sea" and "Amazon Estuary is an Estuary", a reasoner can infer that "Amazon Estuary flows into a Sea").
- **OWL (Web Ontology Language)**: OWL is a language specifically designed for **encoding ontologies in RDF**. It defines a vocabulary of terms and specifies how those terms relate to each other, providing extra meaning for software to reason over using components called **reasoners**. OWL itself is a vocabulary for defining the meaning of terms.
    - **Classes and Individuals**: OWL allows statements to say what **class** a resource belongs to (e.g., inf:simonm rdf:type ex:Person), using the rdf:type predicate (often abbreviated a). A **class** represents a kind of thing (e.g., ex:Person), while an **individual** is a specific instance of that class (e.g., inf:simonm).
    - **Class Hierarchies**: A resource can be an instance of multiple classes (e.g., a person can also be a reader and a man). Ontologies typically include **hierarchies of subclass relationships** (e.g., ex:Man rdfs:subclassOf ex:Person), allowing reasoners to automatically infer that an instance of a subclass is also an instance of its superclass.
    - **Properties**: In OWL, a predicate (e.g., ex:worksIn) is called a **property**. OWL allows defining a property's **domain** (the class of the subject) and **range** (the class of the object) to specify its meaning (e.g., ex:worksIn rdfs:domain ex:Person; rdfs:range ex:City).
    - **Datatypes**: For properties whose objects are literal values (strings, integers, etc., not resources), OWL uses **XML Schema datatypes** (e.g., xs:string, xs:integer, xs:dateTime) to define their range.
- **Social Methodology and Mapping**: Agreeing on ontologies can be difficult, especially with many participants. The semantic web's social approach involves small groups agreeing on small ontologies, with **mappings created between them**. OWL provides vocabulary for this, allowing statements that one class is **owl:equivalentClass** to another, or one individual is **owl:sameAs** another, implying their equivalence.

7.3 SPARQL (SPARQL Protocol and RDF Query Language)

- **Query Language**: SPARQL is the **standard query language for RDF data stores (triple stores)**, similar to SQL for relational databases. There is also a SPARQL Protocol for sending queries to online triple stores.
- **Queries**: A basic SPARQL query finds statements or combinations of statements matching a specific pattern and returns specified subjects and/or objects.
- **Variables**: **Variables**, denoted by ?var or $var, are used to represent parts of the data to be retrieved.
- **SELECT Statement**: The SELECT statement specifies which variables are desired as query results. For example, SELECT ?name ?mbox WHERE { ?x foaf:name ?name . ?x foaf:mbox ?mbox } retrieves names and email addresses.
- **Results**: Query results are presented as a table, with column headings as variable names and cells as values, where each row represents a set of bindings that match the query pattern in the triple store.

7.4 RDFa

- **Embedding RDF in HTML**: RDFa is a technology that allows **RDF data to be embedded directly within HTML pages** using specific markup attributes. This machine-readable RDF data is not directly displayed to the user but can be extracted by software for various applications (e.g., copying events to a calendar, adding contact info to an address book).
- **property attribute**: Adding a property attribute to an HTML element makes its value a predicate relating the webpage to the text content (e.g., <h1 property="http://purl.org/dc/terms/title">...</h1> embeds a title statement about the page).
- **resource attribute**: To embed RDF statements about a subject other than the webpage itself, the resource attribute is used within an HTML element. Statements embedded inside this element will then be about the specified resource URI (e.g., ... embeds statements about the TV series).

7.5 Linked Data

- **Concept**: Linked Data is about **making links between data** on the web, enabling people or machines to explore a "web of data". Unlike hypertext links

that relate anchors in HTML documents, Linked Data involves links between arbitrary things described by RDF, where URIs identify any kind of object or concept.

- **Four Rules (Principles)**: Tim Berners-Lee outlined four "rules" or expectations for Linked Data:
    1. **Use URIs as names for things**. This is fundamental to Semantic Web technology.
    2. **Use HTTP URIs so that people can look up those names**. This allows dereferencing URIs to retrieve information.
    3. *When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)**. This means serving data in formats like RDF/XML or Turtle, and potentially providing SPARQL query services.
    4. **Include links to other URIs so that they can discover more things**. This is crucial for connecting data into an unbounded web, much like the hypertext web. The value of information is enhanced by what it links to.
- **Value Added**: The unexpected re-use of information is a key value added by the web.
- **Methods of Linking Data**:
    - **Simple URI Referencing**: Using a URI in one file that points to another (e.g., http://example.org/smith#albert). **Dereferencing** a URI means truncating it before the hash and accessing the resulting document to obtain information.
    - **URIs without Slashes and HTTP 303**: For abstract concepts or when a single document describes multiple resources, an HTTP GET request to the resource's URI can return a 303 See Other status with a Location: header pointing to the document containing information about it.
    - **rdfs:seeAlso (FOAF convention)**: The Friend-Of-A-Friend (FOAF) convention initially linked people using foaf:mbox (email) and rdfs:seeAlso (a document describing them). While successful, this approach didn't provide direct URIs for people a **SPARQL query service** is a reasonable approach. To ensure data is effectively linked, there needs to be a way for users with a URI to discover the relevant SPARQL endpoint, often facilitated by HTTP 303 redirects to metadata documents.
- **5-Star Linked Open Data (LOD) Scheme**: This scheme encourages the progressive publication of open data with increasing levels of usability and

interconnectedness. **Linked Open Data (LOD)** is Linked Data released under an open license that allows free reuse (e.g., Creative Commons CC-BY, UK's Open Government Licence).
    - ★ **(1 Star)**: Data is available on the web in any format, but with an **open license**.
    - ★★ **(2 Stars)**: Data is available as **machine-readable structured data** (e.g., Excel instead of image scans).
    - ★★★ **(3 Stars)**: Data is available as (2) plus in a **non-proprietary format** (e.g., CSV instead of Excel).
    - ★★★★ **(4 Stars)**: All of the above, plus **uses open standards from W3C (RDF and SPARQL) to identify things**, allowing others to point to the data.
    - ★★★★★ **(5 Stars)**: All of the above, plus **links the data to other people's data** to provide context.
- **LOD Cloud**: The LOD Cloud diagram illustrates datasets published in the Linked Data format and their interconnections. As of December 2023, it contained 1314 datasets with 16308 links.
- **Linked Data Lifecycle**: An example lifecycle includes: **Generate** (map data to RDF), **Validate** (check spelling, integrity, semantics), **Publish** (via data dump, SPARQL endpoint, or LoD documents), **Query** (find, access, combine data), and **Enhance** (addressing missing/incorrect data, tracking provenance).
- **Publishing Methods**: Data can be published as a **SPARQL endpoint** (for live data, low bandwidth, low availability, low client cost, high server cost) or as a **Data dump** (for older data, high bandwidth, high availability, high client cost, low server cost).

7.6 JSON-LD (JavaScript Object Notation for Linking Data)

- **Format**: JSON-LD is a **lightweight Linked Data format** that is human-readable and writable. Its design allows existing JSON data to be interpreted as Linked Data with minimal changes, making it suitable for web-based programming, interoperable web services, and JSON-based storage engines.
- **Enhancements to JSON**: JSON-LD adds several features to standard JSON:
    - **@id**: A universal identifier mechanism for JSON objects using **IRIs** (Internationalized Resource Identifiers). This helps avoid ambiguity when different names are used for keys by linking them to vocabularies like schema.org.

    - **@context**: A mechanism to **disambiguate keys** shared across JSON documents by mapping terms to IRIs. A context can be embedded or referenced via a URL, making the document less verbose but still unambiguous.
    - **@type**: Used to **specify the class of a node**. A node can be assigned multiple types using an array, and the value of @type can be a term defined in the active context.
    - Ability to annotate strings with their language [for automating tests, ensuring high-quality client-submitted data, and combining different data sources.
- **Definitions**: A JSON Schema defines **data types**, **structure**, and **required data**. However, it does not define the meaning of the data.
- **Creation (Key elements)**:
    - The type keyword defines the expected data type (e.g., array of objects).
    - For arrays, items is used to define the schema for its elements.
    - For objects, properties define key-value pairs, where each property has its own JSON schema for validation.
    - required lists properties that must be present.
    - For numbers, ranges can be defined using minimum, exclusiveMinimum, maximum, and exclusiveMaximum.
    - **Combining Schemas**: Keywords like anyOf (valid against any subschema, e.g., a property being null or a string), allOf (valid against all subschemas), oneOf (valid against exactly one subschema), and not (must not be valid against the given schema) are used.
- **Capabilities**: JSON Schema offers extensive validation capabilities, including multiples for numbers, regular expressions for strings, list validation for array elements, and constraints on object properties like maxLength, minItems, required, and patternProperties (as supported by validators like Ajv).
- **Validity**: An XML document with correct syntax is "Well Formed". If it also validates against an XML Schema, it is both "Well Formed" and "Valid". This principle applies conceptually to JSON Schema validation as well.