DNS

Contents

[HTTP Discussion Continued… 2](#_Toc185539559)

[HTTP Response 2](#_Toc185539560)

[HTTP Response Headers 4](#_Toc185539561)

[HTTP Cookies 5](#_Toc185539562)

[How the Internet Works: Accessing a Website 7](#_Toc185539563)

[Understanding DNS (Domain Name System) 9](#_Toc185539564)

[Summary of DNS Workflow 14](#_Toc185539565)

[Key Concepts Discussed 17](#_Toc185539566)

## 

**Agenda**

The following topics to be covered:

1. Remaining HTTP concepts
2. Cookies
3. DNS
4. TCP and UDP (differences)

## HTTP Discussion Continued…

**General Overview**

* HTTP is an **application-layer protocol**.
* It follows the **client-server architecture** where the **client requests** data and the **server respond** with data.
* Initially invented for **hypertext**, HTTP now handles various types of objects (bold text, italics, images, videos, etc.).

**Request and Response**

1. HTTP **Request**: Already covered in the last class, including request headers, body, and methods.
2. HTTP **Response**: Discussed in detail in this session.

## HTTP Response

An **HTTP response** is when the server sends data back to the client.

* The response can contain **HTML, JavaScript, JSON,** or any other content.
* Along with the response content, the server sends a **status code** that describes how the server acted on the request.

**HTTP Status Code Categories**

The speaker mentions **5 categories** of status codes:

1. **1xx (Informational)**
   * Indicates that the server received the request and is processing it.
   * Rarely used in practice.
2. **2xx (Success)**
   * Means the server successfully processed the request.
   * Most common example:
     + **200 OK**: The request was successful.
   * Other examples:
     + **201 Created**
     + **202 Accepted**
3. **3xx (Redirection)**
   * Indicates that the requested resource has moved to another location.
   * Examples:
     + **301 Moved Permanently**: The resource has been permanently moved to a new URL.
     + **307 Temporary Redirect**: The resource has been temporarily moved.
   * Explanation:
     + The browser automatically follows the new location specified in the header.
     + Example shown: Redirection from http://scalar.com to https://scalar.com.
4. **4xx (Client Error)**
   * Indicates an error caused by the **client**.
   * Examples:
     + **404 Not Found**: The requested resource does not exist.
5. **5xx (Server Error)**: Indicates the server failed to fulfil the request. Examples include:
   * **500 Internal Server Error**: A generic server-side failure.
   * **503 Service Unavailable**: The server is temporarily unavailable.

**Additional Clarifications**

* **3xx Redirection**:
  + Explained how **search engines** treat 301 (permanent) vs. 307 (temporary) differently.
  + Redirects prevent users from encountering **404 errors** and leaving the site.

## HTTP Response Headers

HTTP response headers provide information about the response sent by the server. They are crucial for understanding how to process the response. Some common headers include:

1. **Content-Type**  
   Indicates the media type of the resource (e.g., text/html, application/json, image/png).
2. **Status Code**
   * **2XX**: Success (200 OK, 201 Created).
   * **3XX**: Redirection (301 Moved Permanently, 302 Found).
   * **4XX**: Client errors (404 Not Found, 403 Forbidden).
   * **5XX**: Server errors (500 Internal Server Error, 503 Service Unavailable).  
     These codes guide the browser or client on how to proceed.
3. **Cache-Control**  
   Dictates caching policies (no-cache, max-age=3600).
4. **Set-Cookie**  
   Used to send cookies from the server to the client for state management.
5. **Location**  
   Specifies the URL to redirect the client (commonly used with 3XX status codes).
6. **Content-Encoding**  
   Indicates the compression applied to the response (gzip, deflate).
7. **Access-Control-Allow-Origin**  
   Used for Cross-Origin Resource Sharing (CORS).

**Conventions and Contracts in HTTP**

* HTTP is based on agreed-upon conventions to ensure consistent communication across clients and servers.
* Misusing status codes (e.g., sending 4XX for successful responses) violates these conventions and can lead to unreliable client behaviour.

## HTTP Cookies

HTTP is a stateless protocol, but cookies allow the server to maintain state.

**Definition**

Cookies are key-value pairs stored in the browser and sent with every request to the corresponding domain.

**Usage**

1. **Authentication**
   * The server generates a token upon login and sends it to the client via a Set-Cookie header.
   * Subsequent requests include the token in the Cookie header, enabling the server to recognize the user.
2. **Session Management**
   * Cookies store session IDs, allowing the server to link requests to a specific user session.
3. **Tracking and Analytics**
   * Cookies like GA (Google Analytics) track user behavior.
4. **Preferences**
   * Store user-specific settings (e.g., dark mode, language preferences).

**Lifecycle of Cookies**

* **Session Cookies**

Deleted when the browser is closed.

* **Persistent Cookies**

Have an expiration date (e.g., expires=Wed, 27 Apr 2024 07:28:00 GMT).

**Cookie Scope**

* Associated with a specific domain and path.
* Sent only with requests matching that domain/path.

**Security Concerns**

1. **Stealing Cookies (Session Hijacking)**
   * Attackers can impersonate users if they obtain session cookies.
   * Use Secure (HTTPS-only) and HttpOnly (inaccessible to JavaScript) flags.
2. **Cross-Site Scripting (XSS)**
   * Malicious scripts can access cookies. Mitigation: HttpOnly and proper input validation.
3. **Cross-Site Request Forgery (CSRF)**
   * Cookies are automatically sent with requests, making it easier to forge actions.

**Cookies in Practice**

* **Set-Cookie Header**: Used by the server to set cookies.
* **Browser Behaviour**:
  + Stores cookies and sends them with each request to the same domain.
  + Users can view or clear cookies in browser settings.

**Examples**

1. **Authentication Cookie**

|  |
| --- |
| Set-Cookie: auth\_token=abc123; HttpOnly; Secure; Path=/; Expires=Wed, 27 Apr 2024 07:28:00 GMT |

1. **Tracking** **Cookie.**

|  |
| --- |
| Set-Cookie: GA=GA1.2.1234567890.1234567890; Path=/; Expires=Wed, 27 Apr 2024 07:28:00 GMT |

**Legal Considerations**

* **GDPR (Europe)** and similar laws require user consent before storing cookies, especially for tracking purposes.

## How the Internet Works: Accessing a Website

**Key Concepts**

1. **Internet Addresses**
   * Every device or website on the Internet has an **IP address** (e.g., 101.31.42.71), which is a unique numerical identifier.
   * Humans find it difficult to remember numbers (IP addresses) but can easily remember **names** (e.g., google.com, facebook.com).
2. **The Problem**
   * Internet works on numbers (IP addresses).
   * Humans work better with names (domain names).
   * There needs to be a system that maps **domain names** to their corresponding **IP addresses**.

**The Solution: Domain Name System (DNS)**

1. **What is DNS?**
   * **Domain Name System (DNS)** maps domain names (e.g., scalar.com) to IP addresses (e.g., 10.41.31.41).
2. **How Does DNS Work?**
   * When you type a domain name (e.g., scalar.com) in your browser:
     1. **Browser Request**: The browser sends a request to the **DNS server**.
     2. **DNS Server Response**: The DNS server translates the domain name into its IP address.
     3. **Access Website**: The browser uses the IP address to contact the server hosting the website.
3. **DNS Server Configuration**
   * DNS server settings are configured in your **router** or **operating system**.
   * Example: Google's public DNS server (8.8.8.8).
4. **Example Workflow**
   * Open scalar.com in the browser.
   * Browser contacts DNS server to get the IP address (e.g., 10.41.31.41).
   * Browser connects to 10.41.31.41 to load scalar.com.

**Role of the Operating System**

* Applications (like browsers) rely on the **operating system** to handle DNS requests.
* The OS uses the configured DNS server to resolve domain names into IP addresses.

**DNS Server Performance and Challenges**

1. **Handling Billions of Requests**
   * DNS servers handle billions of requests per second globally.
   * A single server cannot manage such a high load.
   * DNS infrastructure is distributed, with many servers worldwide, to handle the load efficiently.
2. **What Happens if DNS Fails?**
   * Without DNS, users cannot resolve domain names to IP addresses.
   * Internet access becomes impossible unless users directly use IP addresses.
3. **Efficiency of DNS**
   * DNS is designed to handle vast numbers of requests quickly by employing:
     + **Caching**: Frequently accessed domain names are cached locally or on intermediary servers.
     + **Redundancy**: Multiple DNS servers ensure reliability and load balancing.

**DNS Registration**

* When a website is created, its domain name and corresponding IP address must be registered in a DNS server.

**Key Takeaways**

* **IP addresses** are how the Internet identifies devices, but humans rely on **domain names**.
* **DNS** bridges the gap, enabling easy navigation of the Internet.
* DNS is crucial for Internet functionality, handling billions of requests with distributed, efficient systems.

## Understanding DNS (Domain Name System)

**Purpose of DNS**

* DNS resolves human-readable domain names (e.g., google.com) into machine-readable IP addresses.
* Without DNS, users would need to memorize numerical IP addresses for websites.

**Challenges of a Single DNS Server**

1. **Scalability Issues**: A single DNS server cannot handle the vast number of requests generated globally.
2. **Performance Issues**: If every browser query first requires contacting the DNS server before accessing a website, it would significantly slow down the Internet.

**How DNS Works**

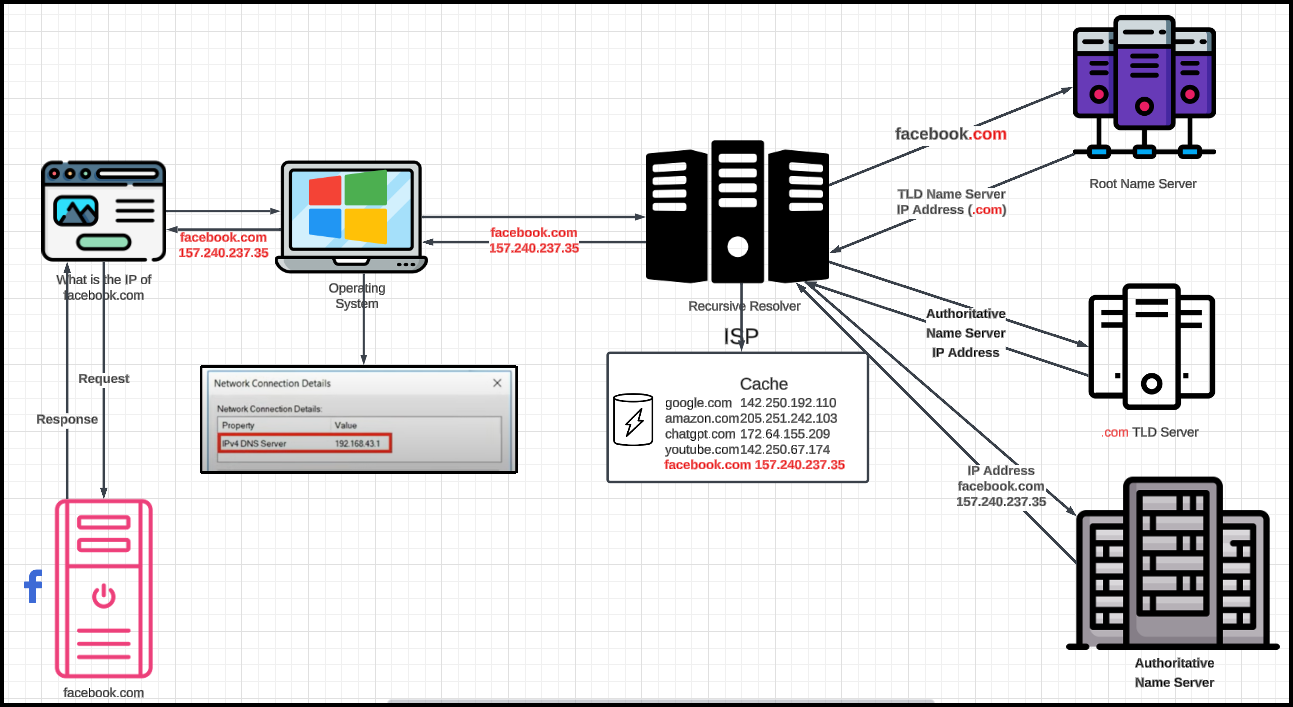
* DNS operates using **recursion** and a hierarchical structure.

**DNS Architecture**

1. **Recursive Resolver**:
   * The starting point for a DNS query.
   * Sends requests to other DNS servers (root, TLD, authoritative) to resolve domain names.
2. **Root Name Servers**:
   * The top-most level in the DNS hierarchy.
   * A diagram of a computer network

     Description automatically generatedKnows the IP addresses of all **TLD Name Servers**.
   * There are **13 root name servers** globally, operated by ICANN and volunteers.
3. **TLD Name Servers** (Top-Level Domain):
   * Maintain records for domains under a specific TLD (e.g., .com, .org, .dev).
   * Examples of TLD owners:
     + .dev: Google
     + .apple: Apple
     + .sbi: State Bank of India
   * TLDs are auctioned or assigned by **ICANN** (Internet Corporation for Assigned Names and Numbers).
4. **Authoritative Name Servers**:
   * Maintained by the domain owners (e.g., Google for google.com).
   * Store the final IP address of the domain or subdomains (e.g., blog.google.com).

**Step-by-Step Query Resolution**

1. **Client Request**:
   * User sends a request to the recursive resolver to resolve a domain (e.g., blog.google.com).
2. **Recursive Resolver → Root Name Server**:
   * The recursive resolver queries the root name server for the IP address of the TLD (.com).
   * The root name server responds with the IP address of the .com TLD Name Server.
3. **Recursive Resolver → TLD Name Server**:
   * The resolver queries the TLD name server for the IP address of the domain (e.g., google.com).
   * The TLD name server responds with the IP address of the **Authoritative Name Server** for google.com.
4. **Recursive Resolver → Authoritative Name Server**:
   * The resolver queries the authoritative name server for the specific subdomain (blog.google.com).
   * The authoritative name server responds with the final IP address.
5. **Response to Client**:
   * The recursive resolver returns the resolved IP address to the client.
6. **Client talks to the server:**
   * Client sends request to the server and the server sends response back.

**Key Characteristics of DNS**

* **Caching**:
  + DNS servers cache responses to improve performance and reduce repetitive queries.
  + Browsers also maintain a local DNS cache.
* **Hierarchical Structure**:
  + The DNS system is modelled as a tree structure:
    - Root → TLD → Domain → Subdomains.

**Ownership and Management**

1. **Root Name Servers**:
   * Managed by ICANN and volunteer organizations.
   * Serve as the backbone of the Internet.
2. **TLD Name Servers**:
   * Owned by organizations or companies that win the rights to specific TLDs.
   * Example: .dev is owned by Google.
3. **Authoritative Name Servers**:
   * Maintained by domain owners.
   * Ensure accurate mapping of their domains to IP addresses.

A computer network diagram with different colored objects

Description automatically generated with medium confidence

**ICANN and TLDs**

* ICANN manages the assignment of TLDs.
* Organizations or companies bid for rights to specific TLDs (e.g., .dev, .app).
* TLD owners maintain the TLD name servers and issue domains under their TLD.

**Key Takeaways**

1. **DNS Resolves Queries Efficiently**:
   * By distributing responsibility across multiple servers (root, TLD, authoritative), DNS avoids bottlenecks.
2. **Source of Truth**:
   * The authoritative name server is the final authority for domain-to-IP mapping.
3. **ICANN's Role**:
   * ICANN oversees DNS operations, including assigning TLDs and managing root servers.
4. **Recursive Resolution**:
   * DNS uses a recursive process to systematically narrow down the search for the correct IP address.

## Summary of DNS Workflow

1. **Client Request:**
   * The client (typically an operating system) sends a request to the **recursive resolver** to obtain the IP address of a domain.
2. **Recursive Resolver:**
   * Examples: Google's public DNS (8.8.8.8), ISP-provided resolvers, etc.
   * If the resolver does not have the IP address cached, it initiates a query to find it.
3. **Query Process:**
   * **Step 1:** Recursive resolver asks the **Root Name Server** (RNS): “Do you have the IP address of this domain?”
     + RNS either:
       - Provides the IP address (rarely), or
       - Provides the address of the **TLD (Top-Level Domain) Name Server**.
   * **Step 2:** Recursive resolver contacts the **TLD Name Server** (e.g., for .com, .dev, etc.).
     + TLD Name Server either:
       - Provides the IP address of the website, or
       - Provides the address of the **Authoritative Name Server (ANS)** for the domain.
   * **Step 3:** Recursive resolver contacts the **Authoritative Name Server**:
     + ANS provides the definitive IP address for the domain.
4. **Caching:**
   * To improve performance, caching occurs at multiple levels:
     + **Recursive Resolver:** Stores recently resolved IP addresses.
     + **Root and TLD Name Servers:** Cache frequently requested IP mappings.
     + **Operating System and Browser:** Store DNS results to avoid repeated queries.
5. **Final Response:**
   * The recursive resolver returns the IP address to the client.

**Key Components of DNS**

1. **Root Name Server (RNS):**
   * Owned and maintained by ICANN volunteers (organizations funding ICANN).
   * Provides IP addresses of TLD Name Servers.
2. **TLD Name Server:**
   * Maintained by entities owning specific TLD rights (e.g., .com, .dev).
   * Provides IP addresses of Authoritative Name Servers for specific domains.
3. **Authoritative Name Server (ANS):**
   * Maintained by the domain owner or hosting providers (e.g., AWS Route 53, Cloudflare).
   * Contains definitive mappings of domain names to IP addresses.
   * Smaller websites often use name servers provided by domain registrars or hosting services.
4. **Recursive Resolver:**
   * Performs the iterative query process.
   * Examples include public DNS servers like Google’s 8.8.8.8 or ISP-provided resolvers.

**Hosting and Name Servers**

1. **Static vs Dynamic IP Addresses:**
   * Websites use **static IP addresses** to ensure stability and accessibility.
   * Dynamic IP addresses are not suitable for web hosting.
2. **Load Balancers:**
   * IP addresses often map to load balancers instead of specific servers.
   * Load balancers distribute traffic across multiple backend servers.
3. **Domain Registrars and Hosting Services:**
   * Examples: GoDaddy, AWS Route 53, Cloudflare, etc.
   * Provide tools for DNS management and authoritative name servers.

**Caching and Performance**

* **Why caching is important:**
  + Reduces latency by avoiding repetitive queries.
  + Minimizes load on upstream servers (e.g., root and TLD servers).
  + Enhances user experience by providing faster responses.
* **Levels of caching:**
  1. **Browser Cache:**
     + Stores DNS results locally for recently visited websites.
  2. **Operating System Cache:**
     + Maintains DNS results for all applications on the device.
  3. **Recursive Resolver Cache:**
     + Retains DNS results for domains queried by multiple clients.

**Key Terms and Concepts**

1. **DNS Proxying:**
   * Some services (e.g., Cloudflare) act as proxies, masking the real IP address of a website for security purposes.
2. **Subdomains:**
   * Example: blog.example.com (subdomain of example.com).
   * Managed under the same authoritative name server as the parent domain.
3. **Dynamic Hosting on Personal Computers:**
   * Requires a **static public IP address** to make a personal computer accessible as a server.
   * Typically involves purchasing a static IP from an Internet Service Provider (ISP).

**Examples of DNS Configurations**

1. **Cloudflare DNS:**
   * Provides both authoritative name servers and proxying services.
   * Example: IP address of a proxied website differs from the real hosting server.
2. **AWS Route 53:**
   * An authoritative DNS service offered by AWS.
   * Commonly used for scalable and reliable DNS management.
3. **Load Balancing with DNS:**
   * Websites like google.com use a load balancer’s IP address as the resolved address.
   * Load balancer forwards requests to multiple backend servers.

**Conclusion**

* DNS is a hierarchical system designed to efficiently map domain names to IP addresses.
* Recursive queries involve multiple layers of servers (Root, TLD, Authoritative).
* Caching at various levels optimizes performance and reduces query times.
* Tools like AWS Route 53 and Cloudflare facilitate DNS management for domain owners.

## Key Concepts Discussed

**DNS and IP Address Caching:**

1. **DNS caching and invalidation**:
   * The importance of DNS caching for faster access to frequently visited websites.
   * What happens when a website’s IP address changes and how it can lead to downtime (e.g., the Facebook/WhatsApp outage example).
   * DNS records’ Time-To-Live (TTL) affects how long a cached address is valid.
2. **Private DNS and Ad Blocking**:
   * Custom DNS configurations on devices (e.g., using dns.adguard.com) block ads by resolving ad-serving domains to invalid or null addresses.
   * Explains how DNS servers remove entries of ad-related domains to prevent them from resolving.
3. **Recursive Resolvers**:
   * Recursive DNS servers query multiple levels of the DNS hierarchy (Root, TLD, and authoritative name servers) to resolve a domain name.
   * Google Public DNS (e.g., 8.8.8.8) and its global presence for resolving domains.
4. **GeoDNS**:
   * How GeoDNS assigns different IPs to a single domain based on geographical location, ensuring faster access and load distribution.

**Networking Protocols and Tools:**

1. **Socket Programming**:
   * The lifecycle of a socket: creation, binding to a port, sending requests, and receiving responses.
   * Explained in the context of browsers and tools like Postman.
2. **HTTP GET vs POST**:
   * GET has character limits (about 2083 characters in modern browsers), which can pose limitations in older systems.
   * POST is often used for large payloads (e.g., Flipkart's search APIs).

**Hosting and Infrastructure:**

1. **Hosting on Personal Machines**:
   * It is possible to host a website on a personal computer if a static public IP address is assigned by the ISP.
   * Challenges include cost, availability, and security concerns.
2. **Domain Name Hosting**:
   * Differences between domain registrars (e.g., GoDaddy) and authoritative name servers (e.g., Cloudflare).
   * Example setup: Registering a domain with GoDaddy and using Cloudflare for DNS management.

**Outages and Failures:**

1. **Facebook/WhatsApp Outage**:
   * Cited as an example of cascading issues in DNS and BGP protocols.
   * Explained how misconfigurations can propagate due to cached data.
2. **Fault Tolerance in DNS**:
   * High availability achieved through geographically distributed DNS servers and replicas.

**Common Questions and Answers Addressed**

1. **What happens if a website’s IP changes?**
   * Cached IPs become invalid until the DNS TTL expires and a fresh query resolves the updated IP.
2. **Can a single domain have multiple IPs?**
   * Yes, through techniques like GeoDNS.
3. **How does private DNS work?**
   * Filters domains by not resolving their IPs for blocked sites.
4. **How are sockets and ports assigned?**
   * Sockets are created first and then bound to ports.
5. **Can a personal computer act as a server?**
   * Yes, with a public static IP and appropriate configuration.

**Teaching and Engagement Style**

* The conversation uses **real-world analogies and examples** (e.g., the Facebook/WhatsApp outage, hosting services like GoDaddy/Cloudflare) to explain technical details.
* Questions from participants are addressed interactively and thoroughly.
* Concepts like **GeoDNS**, **BGP**, and **caching** are connected to practical issues like ad-blocking, website outages, and global DNS resolution.

**Opportunities for Improvement**

1. **Clarity and Structure**:
   * Some explanations are verbose and could be streamlined for clarity.
   * Avoid repeating the same concept multiple times (e.g., caching).
2. **Visual Aids**:
   * For a technical session, diagrams of DNS resolution processes, caching flows, and socket lifecycle could help clarify the concepts.
3. **Advanced Networking Concepts**:
   * Participants asked about distributed systems and fault tolerance. Including more in-depth explanations on DNS replication and failover mechanisms would be beneficial.
4. **Practical Demonstrations**:
   * Demonstrate tools (e.g., nslookup, dig, or Postman) to show how DNS resolution or HTTP requests work.