**B22 WE - Task 1**

**Q: HTTP Version History**

**Ans:**

HTTP (Hypertext Transfer Protocol) is an application protocol that has been the de facto standard for communication on the World Wide Web since its invention in 1989. It functions as a request–response protocol in the client–server computing model and is the in-charge of delegating all the internet's media files between clients and servers which includes html, images, text files, movies, and everything in between.

HTTP standards are developed by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C), culminating in the publication of a series of Requests for Comments (RFCs).

Evolution of HTTP has concentrated mainly on increasing the performances. It has four versions — HTTP/0.9, HTTP/1.0, HTTP/1.1, and HTTP/2.0. Today the version in common use is HTTP/1.1 and HTTP/2.0 is gaining popularity and is being supported by many browsers today. Brief summary of these are given below:

1. First HTTP Implementation – 1990

* Tim Berners Lee at CERN

1. HTTP/0.9 – The One-line Protocol - 1991

* a simple client-server, request-response, telnet-friendly protocol
* Request nature: single line (method + path for requested document)
* Methods supported: GET only
* Response type: hypertext only
* Connection nature: terminated immediately after the response
* No HTTP headers (cannot transfer other content type files), No status/error codes, No URLs, No versioning

1. HTTP/1.0 Building extensibility – 1992

* Client Server Information, simple caching
* Browser-friendly protocol
* Provided header fields including rich metadata about both request and response (HTTP version number, status code, content type)
* Response: not limited to hypertext (Content-Type header provided ability to transmit files other than plain HTML files — e.g. scripts, stylesheets, media)
* Methods supported: GET , HEAD , POST
* Connection nature: terminated immediately after the response

1. HTTP/1.1 The standardized protocol – 1996

* This is the HTTP version currently in common use.
* Introduced critical performance optimizations and feature enhancements — persistent and pipelined connections, chunked transfers, compression / decompression, content negotiations, virtual hosting (a server with a single IP Address hosting multiple domains), faster response and great bandwidth savings by adding cache support.
* Methods supported: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS
* Connection nature: long-lived

1. HTTPS – HTTP over TLS - 2000

* Netscape created HTTPS in 1994, was formally specified by RFC2818 in 2000
* An HTTPS connection can protect the data transfer from the man-in-the-middle attacks and common security threats by providing bidirectional encryption for communications between a client and server.

1. HTTP/2.0 - Improving Transport Performance – 2015

* Based on SPDY protocol, developed primarily at Google
* enables a more efficient use of network resources and a reduced perception of latency by introducing header field compression and allowing multiple concurrent exchanges on the same connection.
* introduced binary framing and supports bidirectional flow of bytes within established connection. Message are in binary format and is broken down into frames which is the basic protocol unit
* allows request and response multiplexing, provides flow control mechanism which can be implemented also at application layer, and allows prioritization of requests
* adds a new interaction mode, whereby a server can push responses to a client
* Methods supported: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS
* Connection nature: persistent, One Connection Per Origin

1. Post-HTTP/2 evolution – HTTP/3

* The next major version of HTTP, HTTP/3, will use QUIC instead TCP/TLS for the transport layer portion

**Q: Difference between HTTP/1.1 and HTTP/2**

**Ans:**

Following table summarizes the major differences between HTTP/1.1 and HTTP/2:

|  |  |  |
| --- | --- | --- |
| Differentiator | HTTP/1.1 | HTTP/2 |
| Protocol Type | Text based protocol, request and response messages are in text format, human readable format | Binary protocol, messages are encoded into binary format, hence cannot be created or read without a tool |
| Purpose | Introduced critical performance optimizations and feature enhancements | HTTP/2 began as the SPDY protocol, developed primarily at Google with the intention of reducing web page load latency by using techniques such as compression, multiplexing, and prioritization. |
| Optimization Features/strategy | persistent connections and pipelining, concurrent TCP connections | Binary framing layer, Stream and Multiplexing, single persistent connection per origin, Stream Prioritization, Server Push, Header Compression |
| Flow Control | It relies on the transport layer to avoid buffer overflow; each new TCP connection requires a separate flow control mechanism. When this connection initiates, both client and server establish their buffer sizes using their system default settings and controls through receive window size subsequently | It allows the client and server to implement their own flow controls, rather than relying on the transport layer. The application layer communicates the available buffer space, allowing the client and server to set the receive window on the level of the multiplexed streams. |
| Predicting Resource Requests | Resource Inlining technique is used. E.g. if a client needs a specific CSS file to render a page, inlining that CSS file will provide the client with the needed resource before it asks for it, reducing the total number of requests that the client must send. | Server Push - a server can send a resource to a client along with the requested HTML page, providing the resource before the client asks for it. the client can decide to cache or decline the pushed resource separate from the main HTML document |
| Compression | Programs like gzip have long been used to compress the data sent in HTTP messages, especially to decrease the size of CSS and JavaScript files. The header component of a message, however, is always sent as plain text. | HTTP/2 can split headers from their data, resulting in a header frame and a data frame. The HTTP/2-specific compression program HPACK can then compress this header frame. This algorithm can encode the header metadata using Huffman coding, thereby greatly decreasing its size. |

**Q: List 5 difference between Browser JS (console) and Node JS**

**Ans:**

node.js is a JavaScript runtime based on chrome’s JavaScript engine called V8 that allows JavaScript to run outside the browser. However, there are some runtime differences, few of them are listed below:

1. **CommonJS vs import:** Node.js has a module system that predates the modern import statement. While node.js has recently started supporting import, almost all code in the wild still uses common.js. In common.js one uses *require* to load a JavaScript module.
2. **Full user-level system access**: This is one of the most exciting things about node.js. Unlike the browser where JavaScript is sandboxed for safety, node.js has full access to the system like any other native application. This means one can read and write directly to/from the file system, have unrestricted access to the network, can execute software and more. This means writing full desktop software is possible with node.js even including a UI through modules like electron. This means that JavaScript ran through node.js needs to be treated with the same level of caution as running C++, Java, or any other language directly on the system.
3. **Global instead of Window:** A lot of the APIs of the browser are missing like anything related to DOM and CSS, Performance, Document, APIs related to window. And in the browser, we don’t have all the nice APIs that Node.js provides through its modules, like the filesystem access functionality.
4. **Concurrency / Asynchronous (non-blocking) IO:** Browsers do have multiple threads to support the execution of JavaScript but in node.js the thread pool is used for super-fast IO. Using Async IO makes it possible in nodejs to have high-performance JavaScript based servers even though JavaScript itself only runs on one thread.
5. **Control over environment**: Another big difference is that in Node.js, one knows which version of Node one will run the application on. Compared to the browser environment, where one doesn’t get the luxury to choose what browser users will use, this is very convenient. This also means that one can write all the modern ES6-7-8-9 JavaScript that the Node version supports.

**Q: What Happens when you type a URL in the address bar in the browser?**

**Ans:**

Here is a brief overview of what happens when you type in a URL in a address bar in a browser...

#### **1.** Type the URL in the browser and hit enter.

#### **2. Forward Lookup :** The browser looks for the **IP address**of the domain name in the **DNS(Domain Name Server)**. The DNS checks at the following places for the IP address.

* **Check Browser Cache:** The browser maintains a cache of the DNS records for some fixed amount of time. It is the first place to run a DNS query.
* **Check OS Cache:** If the browser doesn't contain the cache then it requests to the underlying Operating System as the OS also maintains a cache of the DNS records.
* **Router Cache:** If your computer doesn't have the cache, then it searches the routers as routers also have the cache of the DNS records.
* **ISP(Internet Service Provider) Cache:** If the IP address is not found at the above three places then it is searched at the cache that ISP maintains of the DNS records.
* If not found here also, then ISP’s DNS recursive search is done. In "DNS recursive search", a DNS server initiates a DNS query that communicates with several other DNS servers to find the IP address.

#### **3. The Browser initiates a TCP connection with the server.**

When the browser receives the IP address, it will build a connection between the browser and the server using the internet protocol. The most common protocol used is TCP protocol. The connection is established using a three-way handshake. It is a three-step process.

* **Step 1 (SYN):** As the client wants to establish a connection so it sends an SYN(Synchronize Sequence Number) to the server which informs the server that the client wants to start a communication.
* **Step 2 (SYN + ACK)**: If the server is ready to accept connections and has open ports then it acknowledges the packet sent by the server with the SYN-ACK packet.
* **Step 3 (ACK)**: In the last step, the client acknowledges the response of the server by sending an ACK packet. Hence, a reliable connection is established and data transmission can start now.

#### 4. The browser sends an HTTP request to the server.

The browser sends a GET request to the server asking for webpage. It will also send the cookies that the browser has for this domain.

#### 5. The server handles the incoming request and sends an HTTP response.

The server handles the HTTP request and sends a response. The first line is called the status line. A Status-Line consists of the protocol version(e.g HTTP/1.1) followed by numeric status code(e.g 200) and its associated textual phrase(e.g OK). The status code is important as it contains the status of the response.

* 1xx: Informational: It means the request was received and the process is continuing.
* 2xx: Success: It means the action was successful.
* 3xx: Redirection: It means further action must be taken in order to complete the request. It may redirect the client to some other URL.
* 4xx: Client Error: It means some sort of error in the client’s part.
* 5xx: Server Error: It means there is some error on the server-side.

It also contains response header fields like Server, Location, etc. These header fields give information about the server. A Content-Length header is a number denoting the exact byte length of the HTTP body. All these headers along with some additional information are added to form an HTTP response.

#### 6. The browser displays the HTML content.

Now the browser gets the response and the HTML web page is rendered in phases. First, it gets the HTML structure and HTMLParser creates the DOM tree. Browser then sends multiple GET requests to get the embedded links, images, CSS, javascript files. CSS parser creates CSSOM tree and js files are processed by JS engine. The web page will be rendered (rendering engine creates rendering/layout tree from DOM tree and CSSOM tree and painting process begins) and is then displayed.