* Difference between HTTP1.1 vs HTTP2

A] Protocol structure:

**HTTP 1.1:** A typical HTTP 1.1 request is composed of text and is human readable.

**HTTP 2:** It is composed only of binary and is not human readable

B] Order of requests:

**HTTP 1.1:** It requires requests to be sent in the correct order, thereby creating an overhead and adding complications

**HTTP 2:** It handles parallel requests for the same connection and therefore solves the problem

C] Compressed headers:

**HTTP 1.1:** The HTTP 1.1 protocol had headers in text which in the case of multiple requests be repeated and create unnecessary duplication, thereby generating over head

**HTTP 2:** This protocol compressed headers thereby solving the problem of over head

* HTTP version history

By definition, HTTP is a protocol for exchange of HyperText documents over a network

A] HTTP 0.9:

HTTP 0.9 was the first official version of HTTP announced in 1991.

It is known as one-line protocol since a typical request message consisted of only one line,

Eg: GET /mypage.html

In, response the server would send just the required HTML file

It did, not contain any headers and was restricted to transferring HTML files ONLY. Error messages were also sent in the form of HTML files.

B] HTTP 1.0:

HTTP 1.0 request was no longer a one-line protocol, instead it also mentioned version information along with the GET method.

For the browser to understand if the connection request was successful or not, a status code line was also added at the beginning of the message in response to the request sent.

A major evolution that has occurred in HTTP 1.0 is that other types of documents apart from HTML, could also be sent e.g. images, videos etc. This was implemented by the use of HTTP headers for both request and responses. In the case of response from a server for a particular resource, it specified the resource type in the header with the **Content-Type** field in the header

C] HTTP 1.1:

In an attempt to fix the complex implementations of HTTP 1.0, HTTP 1.1 was introduced with certain improvements and fixes.

Multiple requests could be sent without having to wait for the response to be fully transmitted thereby reducing latency issues. This was implemented using Pipelining

Multiple domains can be accessed at a single IP Address using the **Host** field in the header.

Client and server can agree on what kind of content is to be exchanged, e.g. language, encoding etc. To specify that, **Accept, Accept-Language, Accept-Encoding** etc fields are specified

Network traffic is lowered by reusing the same connection for multiple requests, thereby saving the time for multiple TCP 3 way handshakes. This is achieved using the **Connection: Keep-Alive** field in the header.

Data is sent across in chunks using the **Transfer-Encoding** field.

D] HTTP 2:

To fix the overhead issues faced due to HTTP 1.1 pipelining mechanism that maintains order of requests, HTTP 2 was rolled out with an alternate way of exchanging data between client and server.

Following a multiplexed protocol, parallel requests could be processed simultaneously, thereby solving the overhead issue due to pipelining in HTTP 1.1.

Unlike HTTP 1.x, the request and response messages were in binary and therefore not human readable and also could not be manually created.

This protocol compressed headers thereby solving the problem of duplication and over of data transmitted.

Server push was introduced where a server maintains a client cache and populate data in it for responses in the future to the client

E] Post HTTP 2 evolution:

To ensure that secure cookies are not tampered with, security related prefixes are added in the **Cookie** field in the header.

The client/browser can specify more information about its requirements by providing data about its requirements, hardware constraints to the server for further optimization of request and response. This information is provided in the **Client-Hints** field in the header

To improve caching of resource availability and routing, the server provides alternate services to reach the same resource. This is implemented by the **Alt-Svc** field

* Abstract working of the JS Engine:

When a JS file is received form the server by the client browser i.e Google Chrome in this case, it goes through a series of processes in the V8 JS Engine through the different components of the V8, as follows:

1. The JS file is broken down by the **Parser** into **Tokens**
2. The Tokens are assembled in the form of an **Abstract Syntax Tree(AST)**. The tree has fields like **type, start, end, body, sourceType.** The **body** contains a systematic set of function and variable declarations, each having their own set of fields within them.
3. The AST is then fed into the **Interpreter** which converts it into Binary 0’s and 1’s called **Byte Code**. The Interpreter in this V8 engine is called ‘Ignition’
4. While the code is being interpreted line by line, a **Profiler** monitors the code and checks if the code can be optimized, if so, it takes note of it.
5. If there is scope that the code can be further optimized, the application sends the code to the **Compiler**, which then optimizes and compiles the optimized code which then replaces the Byte Code generated by the Interpreter. The compile in this V8 engine is named as the ‘Turbo Fan’.
6. Finally the Byte Code/Compiled Code, is further broken down into Machine code which is then executed on the local machine and displayed/rendered for the client user.

* What happens when you type a URL in the address bar in the browser?

A] DNS

When a user types a URL in the address bar, a series of processes take place in the background. When we type a URL and hit enter, we are basically invoking the server that hosts the website that we wish to visit. However, machines do not understand the language that the URL is typed in and require the IP address of the website. The following is the procedure:

1. The browser forwards the request to the OS for the IP address, the OS further forwards the request to the DNS resolver which is handled by the ISP. The DNS resolver checks (forward lookup) in its cache if it has the IP address.
2. If the cache hit is failed, the DNS resolver asks the Root Name Server for the IP address.
3. Based on the extension of the domain name (.org, .net, .com), the RNS, forwards the IP address of the appropriate Top Level Domain Name server to the DNS resolver.
4. The TLD, when requested forwards the IP address of the Authoritative Name Server which stores the IP address of the requested URL, to the DNS resolver.
5. DNS resolver then contacts the ANS which provides the IP address to the DNS resolver.
6. Finally, the ISP then stores the IP address in its cache and forwards it to the OS which forwards it to the browser.
7. Finally, the browser initiates a TCP connection with the server that hosts the website.

B] TCP

The client browser will establish a TCP connection by means of the TCP/IP 3-way handshake procedure.

1. The browser sends a SYN packet to the server asking if it is open to make a connection
2. The server will send a SYN/ACK packet to acknowledge and affirm the connection
3. Finally, the client will send an ACK packet to acknowledge the server’s respons

C] HTTP

The browser then sends an HTTP request to the server.

It includes all the necessary HTTP method(s) like GET, POST etc. Also it specifies all fields in the HTTP headers and also information from cookies.

The server responds with an appropriate HTTP response and in a particular format (HTML, JSON,XML) etc.

Within the response, the requested website is contained along with other necessary information in fields like Encoding etc.

At the top of this response will be a status code indicating the failure/success of the request. The codes are of 5 types:

1. 1xx indicates an informational message only
2. 2xx indicates success of some kind
3. 3xx redirects the client to another URL
4. 4xx indicates an error on the client’s part
5. 5xx indicates an error on the server’s part

Finally, if the status code is 2xx, the browser receives the HTML/CSS/JS files from the server.

D] WEBSITE RENDERING AND DISPLAY

The HTML files are then parsed by the HTML parser and broken down into tokens by the tokenizer which are then converted into a Document Object Model(DOM) tree.

Similarly, the CSS sheets are parsed by the CSS parser and broken down into tokens by the tokenizer which are then converted into a CSS Object Model(CSSOM) tree.

The JS file is processed in the JS engine of the browser and the Machine Code is mapped with the DOM and CSSOM trees. Then a Render Tree is made where the entire layout of the website is assembled.

Finally the layout undergoes the Painting process and the website is displayed as the Document Object.

* List 5 difference between Browser JS(console) vs NodeJS

1] FUNCTION:

**Browser JS:** Browser JS executes JS on the client side

**NodeJS:** NodeJS executed JS on the server side

2] WINDOW OBJECT:

**Browser JS:** It has a predefined Window Object which has attributes and functions used for drawing on the window

**NodeJS:** It doesn’t have a Window Object as it doesn’t have to draw anything on the server side

3] LOCATION OBJECT:

**Browser JS:** Location Object is a predefined object which stores information about the URL that has been loaded

**NodeJS:** NodeJS doesn’t have to store page specific information about a particular URL, so it doesn’t have a Location Object

4] DOCUMENT OBJECT:

**Browser JS:** Document is a predefined global variable that has the HTML that has been rendered for display

**NodeJS:** Since it doesn’t have to display anything, it doesn’t have a Document object

5] MODULE:

**Browser JS:** In client side JS, moduling is not necessary

**NodeJS:** In NodeJS, your code is kept in a module