**DAY-1 TASK**

**1.Difference between HTTP1.1 vs HTTP2**

HTTP stands for hypertext transfer protocol & it is used in client-server communication. By using HTTP user sends the request to the server & the server sends the response to the user. There are several stages of development of HTTP but we will focus mainly on HTTP/1.1 which was created in 1997 & the new one is HTTP/2 which was created in 2015.

**HTTP/1.1**

For better understanding, let’s assume the situation when you make a request to the server for the geeksforgeeks.html page & server responds to you as a resource geeksforgeeks.html page. before sending the request and the response there is a TCP connection established between client & server. again you make a request to the server for image img.jpg & the server gives a response as an image img.jpg. the connection was not lost here after the first request because we add a keep-alive header which is the part of the request so there is an open connection between the server & client. there is a persistent connection which means several requests & responses are merged in a single connection. These are the drawbacks that lead to the creation of HTTP/2: The first problem is HTTP/1.1 transfer all the requests & responses in the plain text message form. The second one is head of line blocking in which TCP connection is blocked all other requests until the response does not receive. all the information related to the header file is repeated in every request.

**HTTP/2**

HTTP/2 was developed over the SPDY protocol. HTTP/2 works on the binary framing layer instead of textual that converts all the messages in binary format. it works on fully multiplexed that is one TCP connection is used for multiple requests. HTTP/2 uses HPACK which is used to split data from header. it compresses the header. The server sends all the other files like CSS & JS without the request of the client using the PUSH frame.

**Difference:**

**The Background:**

For better contextualization of the certain alterations that HTTP/2 made to its precursor, we’ll take a quick look at their basic functionalities and development details first.

**HTTP/1.1**

HTTP protocol was developed in 1989 as the common language that enables client and server machines’ interaction. Process steps are as enlisted:

1. The client (browser) has to send a request to the server using the method (GET/POST).

2. Server responds with the demanded resource, for example – image, alongside the status of what it did to the client’s request.

**HTTP/2**

HTTP/2 was released at Google as the significant improvement of its predecessor. It was initially modeled after the SPDY protocol and went through significant changes to include features like multiplexing, header compression, and stream prioritization to minimize page load latency. After its release, Google announced that it would not provide support for SPDY in favor of HTTP/2.

**Delivery Models**

As discussed before, HTTP/1.1 sends messages as plain text, and HTTP/2 encodes them into binary data and arranged them carefully. This implies that HTTP/2 can have various delivery models. Most of the time, a client's initial response in return for an HTTP GET request is not the fully-loaded page. Fetching additional resources from the server requires that the client send repeated requests, break or form the TCP connection repeatedly for them.

**HTTP/1.1**

HTTP/1.1 addresses this problem by creating a persistent connection between server and client. Until explicitly closed, this connection will remain open. So, the client can use one TCP connection throughout the communication sans interrupting it again and again.

**HTTP/2**

Considering the bottleneck in the previous scenario, the HTTP/2 developers introduced a binary framing layer. This layer partitions requests and responses in tiny data packets and encodes them. Due to this, multiple requests and responses becomes able to run parallelly with HTTP/2 and chances of HOL blocking are bleak.

**Compression**

Every HTTP transfer contains headers that describe the sent resource and its properties. This metadata can add up to 1KB or more of overhead per transfer, impacting the overall performance. For minimizing this overhead and boosting performance, compressions algorithms must be used to reduce the size of HTTP messages that travels between the machines.

**HTTP/1.1**

HTTP/1.x uses formats like gzip to compress the data transferred in the messages. However, the header component of the message is always sent as plain text. Though the header itself is small, it gets larger due to the use of cookies or an increased number of requests.

**HTTP/2**

To deal with this bottleneck, HTTP/2 uses HPACK compression to decrease the average size of the header. This compression program encodes the header metadata using Huffman coding, which significantly reduces its size as a result. In addition, HPACK keeps track of previously transferred header values and further compresses them as per a dynamically modified index shared between client and server.

**Buffer Overflow**

Server and client machine TCP connection requires both of these to have certain buffer space for holding incoming requests. Though these buffers can hold numerous or large requests, they may also lack space due to small or limited buffer size. It causes buffer overflow at receiver’s end, resulting in data packet lost. For example, packets received after the buffer is full, will be lost. To prevent it from happening, a flow control mechanism stops the sender from transmitting an overwhelming amount of data to the receiver side.

**HTTP/1.1**

The flow control mechanism in HTTP/1.1 relies on the basic TCP connection. In beginning itself, both the machines set their buffer sizes automatically. If the receiver’s buffer is full, it shares the receive window details, telling how much available space is left. The receiver acknowledges the same and sends an opening signal.

**HTTP/2**

It multiplexes data streams utilizing the same TCP connection. So, in this case, both machines can implement their flow controls instead of using the transport layer. The application layer shares the available buffer size data, after which, both machines set their receive window details on the multiplexed streams level. In addition, the flow control mechanism does not need to wait for the signal to reach its destination before modifying the receive window.

**2. Objects and its Internal Representation in JavaScript**

Objects, in JavaScript, is it’s most important data-type and forms the building blocks for modern JavaScript. These objects are quite different from JavaScript’s primitive data-types(Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).

**1.**Objects are more complex and each object may contain any combination of these primitive data-types as well as reference data-types.

**2.**An object, is a reference data type. Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the location in memory where the object is stored. The variables don’t actually store the value.

**3.**Loosely speaking, objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of “key: value” pairs. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.

An object can be created with figure brackets {…} with an optional list of properties. A property is a “key: value” pair, where a key is a string (also called a “property name”), and value can be anything.  
To understand this rather abstract definition, let us look at an example of a JavaScript Object :

let school = {

name : "Rvhss School",

location : "Perambalur",

established : "2000"

}

In the above example **“name”, “location”, “established”** are all **“keys”** and **“Rvhss School”, “Perambalu” and “2000”** are values of these keys respectively.

Each of these keys is referred to as **properties** of the object. An object in JavaScript may also have a function as a member, in which case it will be known as a **method** of that object.