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

HTTP/1.1 and HTTP/2 share semantics, ensuring that the requests and responses traveling between the server and client in both protocols reach their destinations as traditionally formatted messages with headers and bodies, using familiar methods like GET and POST. But while HTTP/1.1 transfers these in plain-text messages, HTTP/2 encodes these into binary, allowing for significantly different delivery model possibilities. In this section, we will first briefly examine how HTTP/1.1 tries to optimize efficiency with its delivery model and the problems that come up from this, followed by the advantages of the binary framing layer of HTTP/2 and a description of how it prioritizes requests.

HTTP/1.1 takes care of this problem by introducing persistent connections and pipelining. With persistent connections, HTTP/1.1 assumes that a TCP connection should be kept open unless directly told to close. This allows the client to send multiple requests along the same connection without waiting for a response to each, greatly improving the performance of HTTP/1.1 over HTTP/1.0.

In HTTP/2, the binary framing layer encodes requests/responses and cuts them up into smaller packets of information, greatly increasing the flexibility of data transfer.

### **HTTP/1.1**

In HTTP/1.1, flow control relies on the underlying TCP connection. When this connection initiates, both client and server establish their buffer sizes using their system default settings. If the receiver’s buffer is partially filled with data, it will tell the sender its receive window, i.e., the amount of available space that remains in its buffer. This receive window is advertised in a signal known as an ACK packet, which is the data packet that the receiver sends to acknowledge that it received the opening signal. If this advertised receive window size is zero, the sender will send no more data until the client clears its internal buffer and then requests to resume data transmission. It is important to note here that using receive windows based on the underlying TCP connection can only implement flow control on either end of the connection.

Because HTTP/1.1 relies on the transport layer to avoid buffer overflow, each new TCP connection requires a separate flow control mechanism. HTTP/2, however, multiplexes streams within a single TCP connection, and will have to implement flow control in a different manner.

### **HTTP/2**

HTTP/2 multiplexes streams of data within a single TCP connection. As a result, receive windows on the level of the TCP connection are not sufficient to regulate the delivery of individual streams. HTTP/2 solves this problem by allowing 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. This fine-scale flow control can be modified or maintained after the initial connection via a WINDOW\_UPDATE frame.

**2)http version history**

## HTTP/0.9 — The One-line Protocol

* Initial version of HTTP — a simple client-server, request-response, telenet-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

## HTTP/1.0 — Building extensibility

* 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

Both HTTP/0.9 and HTTP/1.0 required to open up a new connection for each request (and close it immediately after the response was sent). Each time a new connection establishes, a TCP three-way handshake should also occur. For better performance, it was crucial to reduce these round-trips between client and server. HTTP/1.1 solved this with persistent connections.

# **HTTP/1.1 — The standardized protocol**

* 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

# **HTTPS**

* Hyper Text Transfer Protocol Secure (HTTPS) is the secure version of HTTP. It uses SSL/TLS for secure encrypted communications.
* Originally developed by Netscape in mid-1990s, SSL (Secure Socket Layer) is a cryptographic protocol enhancement to HTTP, which defines how client and server should communicate with each other securely. TLS (Transport Layer Security) is the successor of SSL.
* 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 serve
* long-lived

# **HTTP/2: Improving Transport Performance**

Since its publication, RFC 2616 has served as a foundation for the unprecedented growth of the Internet: billions of devices of all shapes and sizes, from desktop computers to the tiny web devices in our pockets, speak HTTP every day to deliver news, video, and millions of other web applications we have all come to depend on in our lives.

What began as a simple, one-line protocol for retrieving hypertext quickly evolved into a generic hypermedia transport, and now a decade later can be used to power just about any use case you can imagine. Both the ubiquity of servers that can speak the protocol and the wide availability of clients to consume it means that many applications are now designed and deployed exclusively on top of HTTP.

**3)List 5 difference between Browser JS(console) vs Nodejs**

| S.No | Javascript | NodeJS |
| --- | --- | --- |
| 1. | Javascript is a programming language that is used for writing scripts on the website. | NodeJS is a Javascript runtime environment. |
| 2. | Javascript can only be run in the browsers. | NodeJS code can be run outside the browser. |
| 3. | It is basically used on the client-side. | It is mostly used on the server-side. |
| 4. | Javascript is capable enough to add HTML and play with the DOM. | Nodejs does not have capability to add HTML tags. |
| 5. | Javascript can run in any browser engine as like JS core in safari and Spidermonkey in Firefox. | Nodejs can only run in V8 engine of google chrome. |
| 6. | Javascript is used in frontend development. | Nodejs is used in server-side development. |
| 7. | Some of the javascript frameworks are RamdaJS, TypedJS, etc. | Some of the Nodejs modules are Lodash, express etc. These modules are to be imported from npm. |
| 8. | It is the upgraded version of ECMA script that uses Chrome’s V8 engine written in C++. | Nodejs is written in C, C++ and Javascript. |

**4)what happens when you type a URL in the address bar in the browser?**

**Steps for what happens when we enter a URL :**

1. Browser checks cache for DNS entry to find the corresponding [IP address](https://www.geeksforgeeks.org/introduction-of-classful-ip-addressing/) of website.  
   It looks for following cache. If not found in one, then continues checking to the next until found.
   * Browser Cache
   * Operating Systems Cache
   * Router Cache
   * ISP Cache
2. If not found in cache, ISP’s (Internet Service Provider) DNS server initiates a DNS query to find IP address of server that hosts the domain name.  
   The requests are sent using small data packets that contain information content of request and IP address it is destined for.
3. Browser initiates a [TCP (Transfer Control Protocol)](https://www.geeksforgeeks.org/tcp-and-udp-in-transport-layer/) connection with the server using synchronize(SYN) and acknowledge(ACK) messages.
4. Browser sends an [HTTP](https://www.geeksforgeeks.org/http-non-persistent-persistent-connection/) request to the web server. GET or POST request.
5. Server on the host computer handles that request and sends back a response. It assembles a response in some format like JSON, [XML](https://www.geeksforgeeks.org/xml-basics/) and HTML.
6. Server sends out an HTTP response along with the status of response.
7. Browser displays [HTML](https://www.geeksforgeeks.org/html-tutorials/) content
8. Finally, Done.