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

|  |  |  |
| --- | --- | --- |
| **S.NO** | **HTTP1.1** | **HTTP2** |
| 1 | It supports connection reuse i.e. for every TCP connection there could be multiple requests and responses, and pipelining where the client can request several resources from the server at once. However, pipelining was hard to implement due to issues such as head-of-line blocking and was not a feasible solution. | Uses multiplexing, where over a single TCP connection resources to be delivered are interleaved and arrive at the client almost at the same time. It is done using streams which can be prioritized, can have dependencies and individual flow control. It also provides a feature called server push that allows the server to send data that the client will need but has not yet requested. |
| 2 | Introduces a warning header field to carry additional information about the status of a message. Can define 24 status codes, error reporting is quicker and more efficient. | Underlying semantics of HTTP such as headers, status codes remains the same. |
| 3 | It is relatively secure since it uses digest authentication, NTLM authentication. | Security concerns from previous versions will continue to be seen in HTTP/2. However, it is better equipped to deal with them due to new TLS features like connection error of type Inadequate Security. |
| 4 | Expands on the caching support by using additional headers like cache-control, conditional headers like If-Match and by using entity tags. | HTTP/2 does not change much in terms of caching. With the server push feature if the client finds the resources are already present in the cache, it can cancel the pushed stream. |
| 5 | HTTP/1.1 provides faster delivery of web pages and reduces web traffic as compared to HTTP/1.0. However, TCP starts slowly and with domain sharding (resources can be downloaded simultaneously by using multiple domains), connection reuse and pipelining, there is an increased risk of network congestion. | HTTP/2 utilizes multiplexing and server push to effectively reduce the page load time by a greater margin along with being less sensitive to network delays. |
| 6 | Headers are sent on every request leading to a lot of duplicate data being sent uncompressed across the wire. | Header compression is included by default in HTTP/2 using HPACK. |
| 7 | Spriting, concatenating, inlining, domain sharding are some of the optimizations used as a workaround to the ‘six connections per host’ rule. | Removes the need for unnecessary optimization hacks. |
| 8 | Text based protocol that is in the readable form. | It is a binary protocol (HTTP requests are sent in the form of 0s and 1s). Needs to be converted back from binary in order to read it. |
| 9 | SSL is not required but recommended. Digest authentication used in HTTP1.1 is an improvement over HTTP1.0. HTTPS uses SSL/TLS for secure encrypted communication. | Though security is still not mandatory, it is mostly encrypted (though it is not enforced) since almost all clients require traffic to be encrypted. It also has some minimum standards, such as minimum key size for encryption. TLS 1.2 etc |

**2. HTTP version history**

**Evolution of HTTP**

HTTP (HyperText Transfer Protocol) is the underlying protocol of the World Wide Web. Developed by Tim Berners-Lee and his team between 1989-1991, HTTP has seen many changes, keeping most of the simplicity and further shaping its flexibility. HTTP has evolved from an early protocol to exchange files in a semi-trusted laboratory environment, to the modern maze of the Internet, now carrying images, videos in high resolution and 3D.

[**Invention of the World Wide Web**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP#invention_of_the_world_wide_web)

Invented by Tim Berners-Lee at CERN in the years 1989–1991, HTTP (Hypertext Transfer Protocol) is the underlying communication protocol of World Wide Web. HTTP functions as a request–response protocol in the client–server computing model. HTTP standards are developed by the [Internet Engineering Task Force](https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force) (IETF) and the [World Wide Web Consortium](https://en.wikipedia.org/wiki/World_Wide_Web_Consortium) (W3C), culminating in the publication of a series of [Requests for Comments](https://en.wikipedia.org/wiki/Requests_for_Comments) (RFCs). HTTP 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 the future will be HTTP/2.0.

## 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

Popular web servers (Apache, Nginx) still supports HTTP/0/9.

## 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

## 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

# 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
* Connection nature: long-lived

# Keep-Alive and Upgrade headers

## Keep-Alive header

* The Keep-Alive header was used prior to HTTP/1.1 and was obsoleted by HTTP/1.1 making persistent connections the default behavior. Keep-Alive header can be used to define policies for long-lived communication between hosts (i.e. allows a connection to stay active until an event occurs). This laid foundation for persistence, reusable connections, pipelining, and many more enhanced capabilities in modern web communication protocols.
* Client, server, or any intermediary can provide information for Keep-Alive header independently. Also, a host can add timeout and max parameters in order to set a timeout or limit maximum request count per connection.
* HTTP pipelining, multiple connections, and many more improvements have been implemented, thanks to the Keep-Alive header’s behavior.

## Upgrade header

* With Upgrade header introduced in HTTP/1.1, it is possible to start a connection using a commonly-used protocol, such as HTTP/1.1, then request that the connection switch to an enhanced protocol type like HTTP/2.0 or WebSockets.
* In an upgraded protocol connection, max parameter (maximum request count) is not present. The upgraded protocol can provide new policies for timeout parameter (if not specifically defined, it uses default timeout value in underlying protocol).

# 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 server.

## SSL/TLS Handshake — major problem in HTTPS

* Although HTTPS is secure by its design, the SSL/TLS handshake process consumes a significant time before establishing an HTTPS connection. It normally costs 1–2 seconds and drastically slows down the startup performance of a website.

# HTTP/2.0

* It introduces the concept of a server push where the server anticipates the resources that will be required by the client and pushes them prior to the client making requests. The client retains the authority to deny the server push; however, in most cases, this feature adds a lot of efficiency to the process.
* Introduces the concept of multiplexing that interleaves the requests and responses without head-of-line blocking and does so over a single TCP connection.
* It is a binary protocol i.e. only binary commands in the form of 0s and 1s are transmitted over the wire. The binary framing layer divides the message into frames that are segregated based on their type – Data or Header. This feature greatly increases efficiency in terms of security, compression and multiplexing.
* HTTP/2 uses HPACK header compression algorithm that is resilient to attacks like CRIME and utilizes static Huffman encoding.

HTTP/3, the next version in the series, is based on Google’s QUIC which, unlike its precursors is a drastic shift to UDP. Given the gradual adoption rate of HTTP/2, HTTP/3 with its security challenges (that comes into play the moment we switch from TCP to UDP) is expected to face some difficulties.

**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 basically used on the client-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?**

**URL**

[URL](https://www.geeksforgeeks.org/url-full-form/) stands for Uniform Resource Locator. URL is the address of the website which you can find in the address bar of your web browser. It is a reference to a resource on the internet, be it images, hypertext pages, audio/video files, etc.

**DNS**

DNS is short for Domain Name System. Like a phonebook, DNS maintains and maps the name of the website, i.e. URL, and particular IP address it links to. Every URL on the internet has a unique IP address which is of the computer which hosts the server of the website requested.

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.