HTTP1.1

**1)HTTP/1.1** keeps all requests and responses in plain text format,

The first response that a client receives on an HTTP GET request is often not the fully rendered page. Instead, it contains links to additional resources needed by the requested page. The client discovers that the full rendering of the page requires these additional resources from the server only after it downloads the page. Because of this, the client will have to make additional requests to retrieve these resources. In HTTP/1.0, the client had to break and remake the TCP connection with every new request, a costly affair in terms of both time and resources.

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.

**HTTP/2**

**1)HTTP/2** uses the binary framing layer to encapsulate all messages in binary format,

In 2015, Internet Engineering Task Force (IETF) release HTTP/2, the second major version of the most useful internet protocol, HTTP. It was derived from the earlier experimental SPDY protocol.

* Protocol negotiation mechanism — protocol electing, eg. HTTP/1.1, HTTP/2 or other.
* High-level compatibility with HTTP/1.1 — methods, status codes, URIs and header fields.
* Page load speed improvements trough:
* Compression of request headers
* Binary protocol
* HTTP/2 Server Push
* Request multiplexing over a single TCP connection
* Request pipelining
* HOL blocking (Head-of-line) — Package blocking

Benefits:

* Low overhead in parsing data — a critical value proposition in HTTP/2 vs HTTP1.
* Less prone to errors.
* Lighter network footprint.
* Effective network resource utilization.
* Eliminating security concerns associated with the textual nature of HTTP1.x such as response splitting attacks.
* Enables other capabilities of the HTTP/2 including compression, multiplexing, prioritization, flow control and effective handling of TLS.
* Compact representation of commands for easier processing and implementation.
* Efficient and robust in terms of processing of data between client and server.
* Reduced network latency and improved throughput.

2)http version history:--

## HTTP 0.9: The One-Line Protocol

The original HTTP proposal by Tim Berners-Lee was designed with simplicity in mind as to help with the adoption of his other nascent idea: the World Wide Web. The strategy appears to have worked: aspiring protocol designers, take note.

In 1991, Berners-Lee outlined the motivation for the new protocol and listed several high-level design goals: file transfer functionality, ability to request an index search of a hypertext archive, format negotiation, and an ability to refer the client to another server. To prove the theory in action, a simple prototype was built, which implemented a small subset of the proposed functionality:

* Client request is a single ASCII character string.
* Client request is terminated by a carriage return (CRLF).
* Server response is an ASCII character stream.
* Server response is a hypertext markup language (HTML).
* Connection is terminated after the document transfer is complete.

However, even that sounds a lot more complicated than it really is. What

these rules enable is an extremely simple, Telnet-friendly protocol, which some web servers support to this very day:

## HTTP/1.1: Internet Standard

The work on turning HTTP into an official IETF Internet standard proceeded in parallel with the documentation effort around HTTP/1.0 and happened over a period of roughly four years: between 1995 and 1999. In fact, the first official HTTP/1.1 standard is defined in RFC 2068, which was officially released in January 1997, roughly six months after the publication of HTTP/1.0. Then, two and a half years later, in June of 1999, a number of improvements and updates were incorporated into the standard and were released as RFC 2616.

The HTTP/1.1 standard resolved a lot of the protocol ambiguities found in earlier versions and introduced a number of critical performance optimizations: keepalive connections, chunked encoding transfers, byte-range requests, additional caching mechanisms, transfer encodings, and request pipelining.

With these capabilities in place, we can now inspect a typical HTTP/1.1 session as performed by any modern HTTP browser and client:

| 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. |