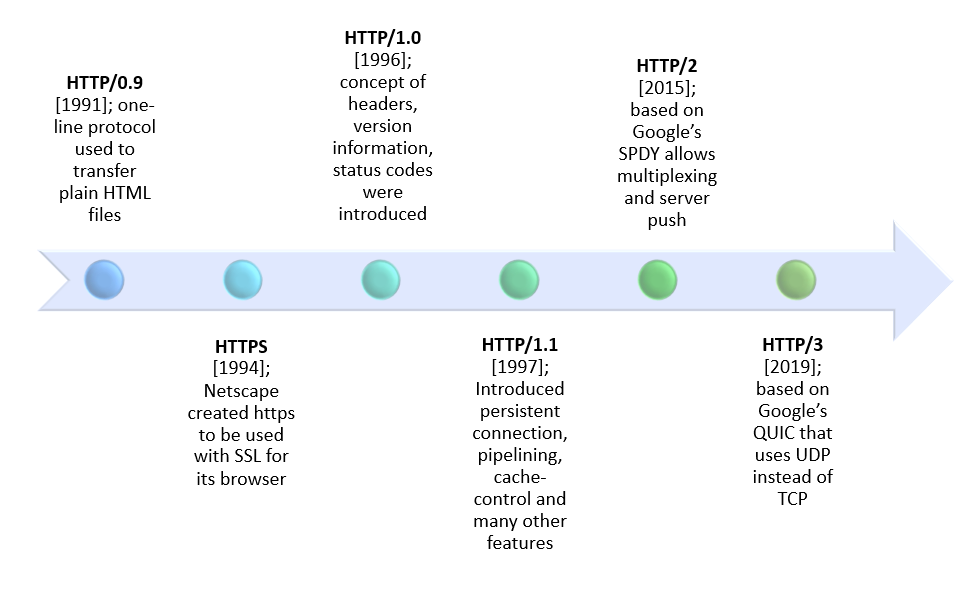
# **Evolution of HTTP**

****

## **HTTP/1.1 vs HTTP/2**

|  |  |  |
| --- | --- | --- |
| **Differentiator** | **HTTP/1.1** | **HTTP/2** |
| **Year** | 1997 | 2015 |
| **Protocol Type** | Text based protocol that is in the readable form | It is a **binary protocol** (HTTP requests are sent in the form of 0’s and 1’s). Needs to be converted back from binary in order to read it. |
| **Multiplexing with Stream Prioritization** | 1. Limitation of HTTP/1.1 is that it is only able to request files one by one on a single TCP connection — this means requesting the file, waiting for a response, downloading the file and then requesting the next one. (**Sequential**) 2. This creates the problem of **head-of-line blocking** **(i.e**, **“**If one of the transmitted packets has not received a response or acknowledgement, then the other packets to be transmitted in the queue will be blocked**”**) and inefficient use of the underlying TCP connection. | 1. Uses multiplexing, where over a single TCP connection resources to be delivered are interleaved and arrive at the client almost at the same time. (**Parallel**) 2. Server and client break down an HTTP message into independent frames, interleave them, and then reassemble them on the other end. 3. Reassembly of frames is done by **Stream Prioritization** |
| **Server Push** | In HTTP/1.1, when your browser connects to a web server requesting a web page, the HTML file is sent first and then your browser request for other resource files like CSS, javascript etc. | **Server push** that allows the server to send data that the client will need but has not yet requested, **(**i.e, **“**Client can get a response of JS, CSS and hTML files from server with a HTML request alone**”)** |
| **Security**  **&**  **Authentication Mechanism** | SSL is not required but recommended. Digest authentication (**NTLM Authentication**) used in HTTP1.1 | It has some minimum standards, such as minimum key size for encryption. TLS 1.2 features gives protection like connection error of type **Inadequate Security**. |
| **Header Compression** | 1. The header fields are transmitted after the request or response line. 2. HTTP/1.1 requests and responses are not compressed and transmitted in plain text which causes a lot of unnecessary bytes being downloaded. | 1. Header compression is included by default in HTTP/2 using **HPACK**, 2. The header fields are compressed and transmitted in binary codes making it more compact and efficient for the user computer to load. 3. Header compression decreases the overhead of client requests which lowers bandwidth and decreases page load time. |
| **Web Traffic** | 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. |
| **Caching** | Expands on the caching support by using additional headers like cache-control, conditional headers like If-Match and by using entity tags (**Etags**). | **Server push** feature enables the client to cancel the pushed stream from the server if it finds the resources are already present in the **cache**. |

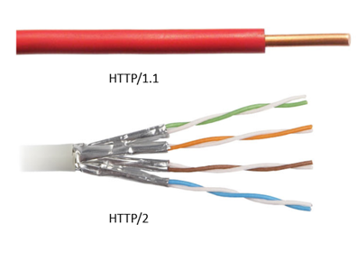
## 

## 

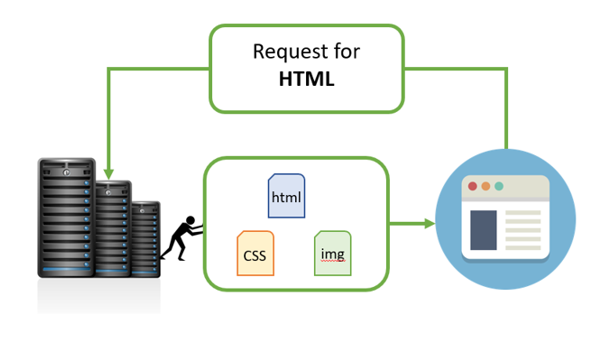
## 

## **Key Features of HTTP/2**

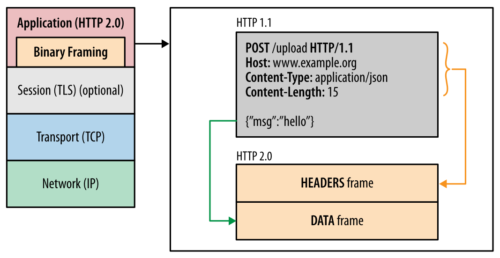
1. **Multiplexing**



2. **Server Push**

****

3. **Header Compression**



# 

# **GET vs POST**

## **The GET Method**

**1.** GET is used to request data from a specified resource.

**2.** GET method is used to append data to the URL in name or value pair.

|  |
| --- |
| **/test/demo\_form.php?name1=value1&name2=value2** |

#### **Features of GET**

* GET requests can be cached
* GET requests remain in the browser history
* GET requests can be bookmarked
* GET requests should never be used when dealing with sensitive data
* GET requests have length restrictions
* GET requests are only used to request data (not modify)

## **The POST Method**

**1.** POST is used to send data to a server to create/update a resource i.e. (depicts that a web server accepts the data).

**2.** The data sent to the server with POST is stored in the request body of the HTTP request.

|  |
| --- |
| **POST /test/demo\_form.php HTTP/1.1**  **Host: w3schools.com**  **name1=value1&name2=value2** |

### **Features of POST**

* POST requests are never cached
* POST requests do not remain in the browser history
* POST requests cannot be bookmarked
* POST requests have no restrictions on data length.

## **Differences of GET and POST**

|  |  |  |
| --- | --- | --- |
| **Differentiator** | **GET** | **POST** |
| BACK button/Reload | Harmless | Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted) |
| Bookmarked | Can be bookmarked | Cannot be bookmarked |
| Cached | Can be cached | Not cached |
| Encoding type | application/x-www-form-url encoded | application/x-www-form-ur lencoded or multipart/form-data. Use multipart encoding for binary data |
| History | Parameters remain in browser history | Parameters are not saved in browser history |
| Restrictions on data length | Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters) | No restrictions |
| Restrictions on data type | Only ASCII characters allowed (Only String) | No restrictions. Binary data is also allowed |
| Security | GET is less secure compared to POST because data sent is part of the URL  **Never use GET when sending passwords or other sensitive information**! | POST is a little safer than GET because the parameters are not stored in browser history or in web server logs |
| Visibility | Data is visible to everyone in the URL | Data is not displayed in the URL |

# **Browser JS vs Node JS**

|  |  |  |
| --- | --- | --- |
| **Differentiator** | **Browser Js** | **Node Js** |
| **Type** | It runs in any web browser with a proper browser engine. | It is an interpreter and environment for JavaScript with some specific useful libraries which JavaScript programming can use separately. |
| **Objects** | Window, Screen, Location, History, Navigator, Pop up, Timing, Cookies are present. | In this environment we don't deal with any DOM or Cookies, these objects are not present here. |
| **File System Access** | Not Permitted | APIs provide the privilege for access. |
| **Utility** | Mainly using for any client-side activity for a web application, like possible attribute validation or refreshing the page in a specific interval or providing some dynamic changes in web pages without refreshing the page. | It is mainly used for accessing or performing any non-blocking operation of any operating system, like creating or executing a shell script or accessing any hardware specific information or running any backend job. |
| **Running Engine** | JavaScript running any engine like Spider monkey (FireFox), JavaScript Core (Safari), V8 (Google Chrome). | Node JS only runs in a V8 engine which is mainly used by google chrome. And javascript programs which will be written under this Node JS will be always run in V8 Engine. |
| **Application** | Front End | Back End |