**HTTP1.1 vs HTTP2**

In the world of web development and internet communication, protocols play a crucial role in ensuring seamless data transfer between clients its such as web browsers and servers. HTTP, or Hypertext Transfer Protocol, is the foundation of data communication on the World Wide Web. Over the years, HTTP has evolved, and one of the significant advancements in this evolution is the transition from HTTP/1.1 to HTTP/2. In this blog, we'll delve into the differences between these two versions of the HTTP protocol and understand how HTTP/2 has addressed several limitations of its predecessor.

HTTP/1.1: The Old Reliable

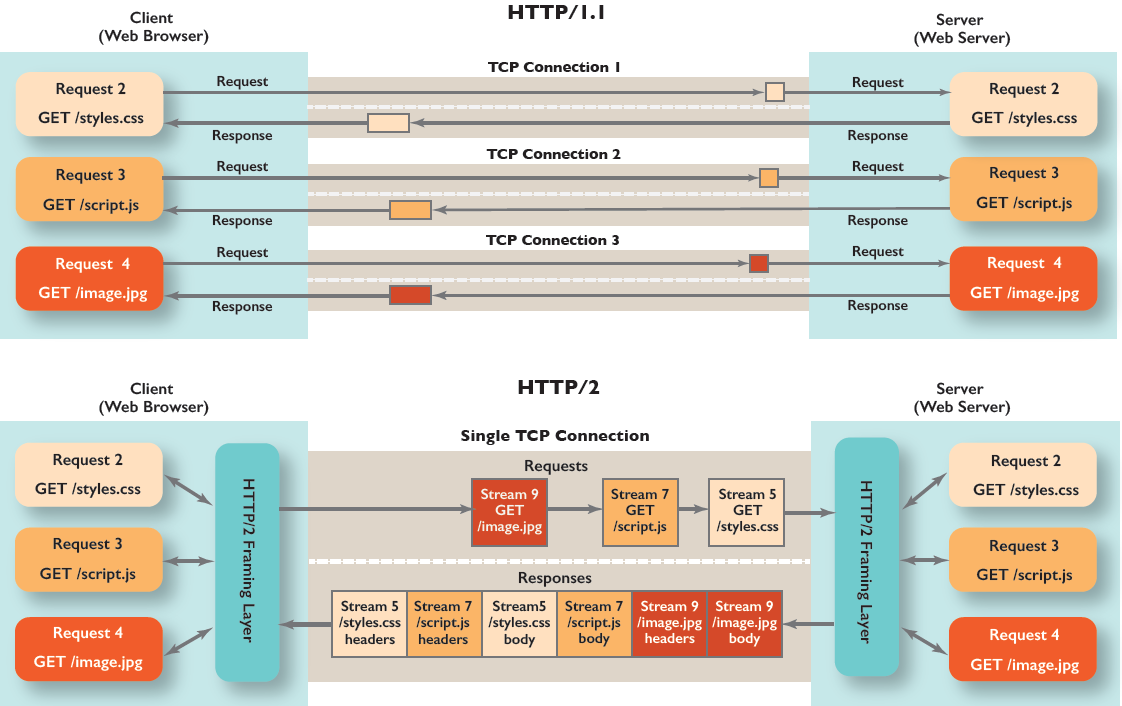
The first usable version of HTTP was created in 1997.HTTP/1.1 has been the workhorse of the internet for quite some time. It was standardized in 1999 and has powered the vast majority of web traffic since then. However, as the web grew more complex, certain limitations of HTTP/1.1 became apparent:

HTTP/2: A Modern Approach

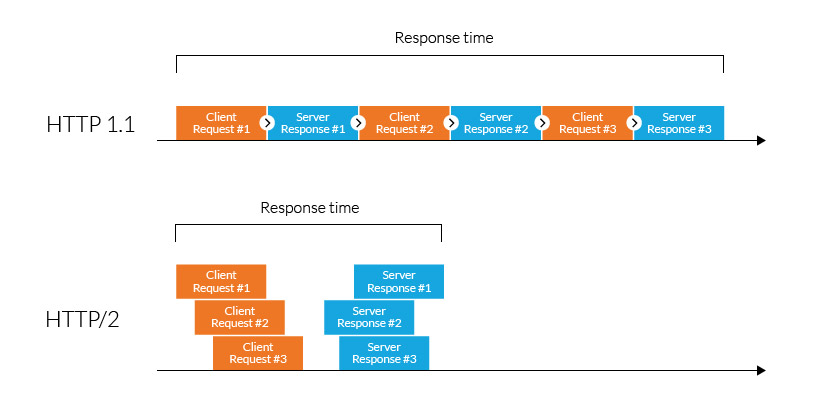
In 2015, a new version of HTTP called HTTP/2 was created. HTTP/2 solves several problems that the creators of HTTP/1.1 did not anticipate. In particular, HTTP/2 is much faster and more efficient than HTTP/1.1. One of the ways in which HTTP/2 is faster is in how it prioritizes content during the loading process. Recognizing these shortcomings, the need for a more efficient and capable protocol became evident. This led to the development of HTTP/2, which aimed to improve upon the limitations of HTTP/1.1:

| **HTTP/1.1** | **HTTP/2** |
| --- | --- |
| **Head of Line Blocking**:  In HTTP/1.1, only one request-response pair could be handled at a time over a single TCP connection. If a resource was requested, and another request was waiting for a response, the latter had to wait until the former was completely processed. This led to inefficiencies and slow page load times, especially on websites with many resources. | **Binary Protocol**:  While HTTP/1.1 used plain text for communication, HTTP/2 employs a binary protocol, which is more efficient for both machines to parse and reduces the overhead of sending text-based headers. |
| **Uncompressed Headers**:  HTTP/1.1 sent headers with each request and response, which were often redundant and consumed unnecessary bandwidth. This became more problematic as web pages started to consist of multiple resources, each with its own set of headers. | **Header Compression**:  HTTP/2 utilizes header compression techniques to significantly reduce the size of headers that are sent with each request and response. This results in reduced bandwidth consumption and faster data transmission. |
| **Lack of Multiplexing**:  Without the ability to send multiple requests and receive multiple responses concurrently on a single connection, the full potential of modern high-speed networks was not being realized. | **Multiplexing**:  HTTP/2 introduced a multiplexing feature that enables multiple requests and responses to be interleaved over a single TCP connection. This solves the head-of-line blocking issue and allows for more efficient use of network resources, reducing latency and improving page load times. |
| **Resource Inlining**:  To mitigate the problem of latency, developers resorted to techniques like resource inlining or spriting, where multiple resources were combined into a single one. While this reduced the number of requests, it also led to inefficient resource usage and made caching challenging. | **Server Push**:  One of the standout features of HTTP/2 is server push. This allows the server to proactively send resources to the client's cache before they are requested. It's particularly useful for sending critical resources, reducing the need for subsequent round-trip requests. |
|  | **Prioritization**:  HTTP/2 introduces stream prioritization, allowing developers to specify which resources should be loaded first. This ensures that more important resources are prioritized, further improving page load times. |

**Single TCP Connection :**



**HTTP/1.1 vs. HTTP/2 Protocol:**



**Multiplexing:**

