HTTP (Hyper Text Transfer Protocol)

HTTP (Hyper Text Transfer Protocol) is an application layer protocol. It has been the standard for communication on the world wide web since its invention in 1989. From the release of HTTP 1.1 in 1997 until recently, there have been few revisions to the protocol. But in 2015 HTTP 2 was released and it offered several methods to deal with latency, especially when dealing with mobile platforms and server-intensive graphics and videos.

# **Difference between HTTP 1.1 and HTTP 2**

**Introduction:**

**HTTP 1.1** is a top-level application protocol that exchanges information between a client computer and a local or remote web server. In this process, a client sends a text-based request to a server by calling a method like GET or POST. In response, the server sends a resource like an HTML page back to the client. When we navigate to any website, The web browser on our computer sends an HTTP request in the form of text-based message like below.

**Example:  
 Get /index.html HTTP/1.1**

**Host:** [**www.example.com**](http://www.example.com)

This request uses the GET method, which asks for data from the host server listed after Host. In response to this request, the example.com web server returns an HTML page to the requesting client, in addition to any images, stylesheets, or other resources called for in the HTML. Not all the resources are returned to the client in the first call for data. The requests and responses will go back and forth between the server and client until the web browser has received all the resources necessary to render the contents of the HTML page on your screen.

**HTTP 2** uses the binary framing layer to encapsulate all messages in binary format, while still maintaining HTTP semantics, such as verbs, methods, and headers. The application-level API would still create messages in the conventional HTTP formats, but the underlying layer would then convert these messages into binary. This ensures that web applications created before HTTP/2 can continue functioning as normal when interacting with the new protocol. The conversion of messages into binary allows the HTTP 2 to try new approaches to data delivery that is not available in HTTP/1.1.

**Multiplexing:**

**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 other. Since multiple data packets cannot pass each other when traveling to the same destination, there are situations in which a request at the head of the queue that cannot retrieve its required resource will block all the requests behind it. This is known as head-of-line (HOL) blocking and is a significant problem with optimizing connection efficiency.

**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 2 establishes a single connection object between the two machines. Within this connection there are multiple streams of data. Each stream consists of multiple messages in the familiar request/response format. Finally, each of these messages split into smaller units called frames. The communication channel consists of a bunch of binary-encoded frames, each tagged to a particular stream. The identifying tags allow the connection to interleave these frames during transfer and reassemble them at the other end. The interleaved requests and responses can run in parallel without blocking the messages behind them it is called multiplexing.

Multiplexing allows the client to construct multiple streams in parallel, these streams only need to make use of a single TCP connection. Having a single persistent connection per origin improves upon HTTP/1.1 by reducing the memory and processing footprint throughout the network. This results in better network and bandwidth utilization and thus decreases the overall operational cost.

**Server Push:**

**HTTP 1.1** Typically, a server only serves content to a client device if the client asks for it. Whenever a client asks for a particular page, the server first send a html page and then the client will request for the c CSS, JavaScript file and images, logo etc. Which is required to render that HTML page. This approach is not always practical for modern webpages. Which always involves

**HTTP 2** solves this issue by allowing the server to push the content to a client before the client asks for it. Whenever the client request for a particular page the server not only sent the html document, it also sends the required CSS, JavaScript file and images and other required files to render that requested html page in the single request and the server also sends a message letting the client know what pushed content to expect.

**Compression:**

**HTTP 1.1** - Programs like gzip have long been used to compress the data sent in HTTP messages, especially to decrease the size of CSS and JavaScript files. The header component of a message, however, is always sent as plain text. The uncompressed data is heavier on the connection as more requests are made, particularly penalizing complicated, API-heavy web applications that require many different resources and thus many different resource requests. Additionally, the use of cookies can sometimes make headers much larger, increasing the need for some kind of compression.

**HTTP 2** can split headers from their data, resulting in a header frame and a data frame. The HTTP/2-specific compression program [HPACK](https://tools.ietf.org/html/draft-ietf-httpbis-header-compression-12) can then compress this header frame. This algorithm can encode the header metadata using Huffman coding, thereby greatly decreasing its size. The HPACK can keep track of previously conveyed metadata fields and further compress them according to a dynamically altered index shared between the client and the server.

**Conclusion:**

As we have seen features like multiplexing, stream prioritization, flow control, server push, and compression in HTTP 2 will affect the changing landscape of web development.