1. Write a blog on Difference between HTTP1.1 vs HTTP2

### **HTTP/1.1**

Developed by Timothy Berners-Lee in 1989 as a communication standard for the World Wide Web, HTTP 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.

**HTTP is based on** the Client/Server model. Client/Server model can be explained as two computers, Client (receiver of service) and Server (provider of service) that are communicating via requests and responses.

A simple and abstract example would be a **restaurant guest and a waiter**. The guest (**Client**) asks (**sends** **request**) waiter (**Server**) for a meal, then the waiter gets the meal from the restaurant chef (**your application logic**) and brings the meal to the guest.

This is a very simplistic example, but it is also the one that will help you understand the concept.

There are many more interesting HTTP concepts and utilities to discuss, but the star of this post is (not enough) famous **HTTP/2**.

For example, let’s say you are visiting a website at the domain www.example.com. When you navigate to this URL, the web browser on your computer sends an HTTP request in the form of a text-based message, similar to the one shown here:

GET /index.html HTTP/1.1

Host: 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. Note that not all of 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.

There is much to discuss about the lower levels of this stack, but in order to gain a high-level understanding of HTTP/2, you only need to know this abstracted layer model and where HTTP figures into it.

With this basic overview of HTTP/1.1 out of the way, we can now move on to recounting the early development of HTTP/2.

### **HTTP/2**

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.

**Main goals of developing HTTP/2 was:**

* 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

# **R**

HTTP/2 began as the SPDY protocol, developed primarily at Google with the intention of reducing web page load latency by using techniques such as compression, multiplexing, and prioritization. This protocol served as a template for HTTP/2 when the Hypertext Transfer Protocol working group httpbis of the [IETF (Internet Engineering Task Force)](https://www.ietf.org/) put the standard together, culminating in the publication of HTTP/2 in May 2015. From the beginning, many browsers supported this standardization effort, including Chrome, Opera, Internet Explorer, and Safari. Due in part to this browser support, there has been a significant adoption rate of the protocol since 2015, with especially high rates among new sites.

From a technical point of view, one of the most significant features that distinguishes HTTP/1.1 and HTTP/2 is the binary framing layer, which can be thought of as a part of the application layer in the internet protocol stack. As opposed to HTTP/1.1, which keeps all requests and responses in plain text format, 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. An 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 HTTP/2 to try new approaches to data delivery not available in HTTP/1.1, a contrast that is at the root of the practical differences between the two protocols. The next section will take a look at the delivery model of HTTP/1.1, followed by what new models are made possible by HTTP/2.

| **HTTP/1.1** |  | **HTTP/2** |  |
| --- | --- | --- | --- |
| It compresses data by itself. |  | It uses HPACK for data compression |  |
| It uses requests resource Inlining for use getting multiple pages |  | It uses PUSH frame by server that collects all multiple pages |  |
| It works on the textual format. |  | It works on the binary protocol. |  |
| There is head of line blocking that blocks all the requests behind it until it doesn’t get its all resources. |  | It allows multiplexing so one TCP connection is required for multiple requests. |  |
| SOURCE: google.com/class notes |  |  |  |