**Difference between HTTP1.1 vs. HTTP2:**

In 1989, Tim Berners-Lee invented HTTP. HTTP/1.1 was its 1st standardized version that was available for use in the year 1997 for the end-users. This version presented considerable performance optimization over its precursors and changed how communication was handled between clients and servers. However, its key qualities opened the doors to many performance and API security loopholes.

HTTP/1 was known to have poor response time. With websites becoming more resource-intensive, the protocol was losing its efficiency. It progressively became essential to minimize latency and boost page load speeds.

HTTP/2, based on SPDY protocol, was developed to address the inherent limitations of HTTP/1.1 and further progress the Internet.



**HTTP/1.1 and HTTP/2 Main Differences**

* **The Background**

**HTTP/1.1**

HTTP protocol was developed in 1989 as the common language that enables client and server machines’ interaction. Process steps are as enlisted:

1. The client (browser) has to send a request to the server using the method (GET/POST).

2. Server responds with the demanded resource, for example – image, alongside the status of what it did to the client’s request.

Keep in mind that this is not a one-time process. Such requests and responses needs to be transferred between both these machines until the client receives all the resources, essential to load a web page on the end-user’s (your) screen.

**HTTP/2**

HTTP/2 was released at Google as the significant improvement of its predecessor. It was initially modeled after the SPDY protocol and went through significant changes to include features like multiplexing, header compression, and stream prioritization to minimize page load latency. After its release, Google announced that it would not provide support for SPDY in favor of HTTP/2.

The major feature that differentiates HTTP/2 from HTTP/1.1 is the binary framing layer. Unlike HTTP/1.1, HTTP/2 uses a binary framing layer.

* **Delivery Models**

**HTTP/1.1**

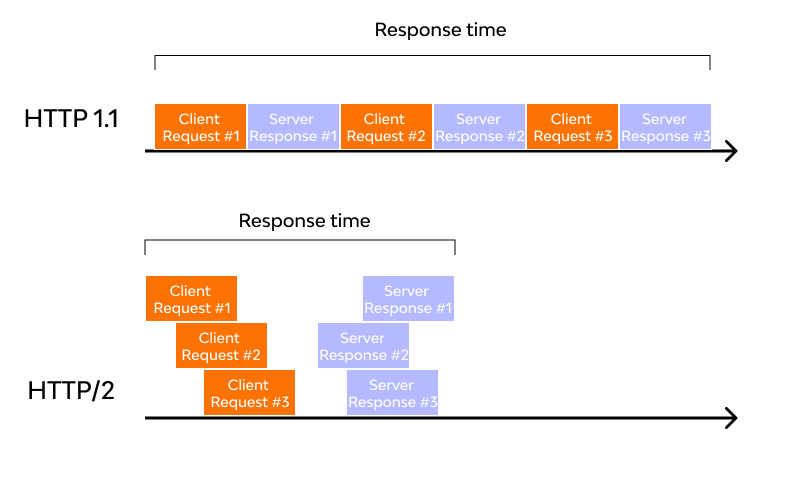
HTTP/1.1 addresses this problem by creating a persistent connection between server and client. Until explicitly closed, this connection will remain open. So, the client can use one TCP connection throughout the communication sans interrupting it again and again.

This approach surely ensures good performance, but it also is problematic.

**HTTP/2**

Considering the bottleneck in the previous scenario, the HTTP/2 developers introduced a binary framing layer. This layer partitions requests and responses in tiny data packets and encodes them. Due to this, multiple requests and responses becomes able to run parallel with HTTP/2 and chances of HOL blocking are bleak.

Not only has it solved HOL blocking problem in HTTP/1.1, but it also concurrent message exchange between the client and the server. This way, both of them can have more control while the connection management quality is boosted too.



**Predicting Resource Requests**

**HTTP/1.1**

To accomplish this, HTTP/1.1 has a different technique called resource in lining, wherein the server includes the required source within the HTML page in response to the initial GET request. Though this technique reduces the number of requests that the client must send, the larger, non-text format files increase the size of the page.

As a result, the connection speed decreases, and the primary benefit obtained from it also nullifies. Another drawback is the client cannot separate the in lined resources from the HTML page. For this, a deeper level of control is required for connection optimization – a need that HTTP/2 meets with server push.

**HTTP/2**

As HTTP/2 supports multiple simultaneous responses to the client’s initial GET request, the server provides the required resource along with the requested HTML page. This is called the server push process, which performs the resource in lining like its precursor while keeping the page and the pushed resource separate. This process fixes the main drawback of resource in lining by enabling the client machine to decide to cache/decline the pushed resource separate from the HTML page.

* **Buffer Overflow**

**HTTP/1.1**

The flow control mechanism in HTTP/1.1 relies on the basic TCP connection. In beginning itself, both the machines set their buffer sizes automatically. If the receiver’s buffer is full, it shares the receive window details, telling how much available space is left. The receiver acknowledges the same and sends an opening signal.

Note that flow control can only be implemented on either end of the connection. Moreover, since HTTP/1.1 uses a TCP connection, each connection demands an individual flow control mechanism.

**HTTP/2**

It multiplexes data streams utilizing the same (one) TCP connection. So, in this case, both machines can implement their flow controls instead of using the transport layer. The application layer shares the available buffer size data, after which, both machines set their receive window details on the multiplexed streams level. In addition, the flow control mechanism does not need to wait for the signal to reach its destination before modifying the receive window.

* **Compression**

Every HTTP transfer contains headers that describe the sent resource and its properties. This metadata can add up to 1KB or more of overhead per transfer, impacting the overall performance. For minimizing this overhead and boosting performance, compressions algorithms must be used to reduce the size of HTTP messages that travels between the machines.

**HTTP/1.1**

HTTP/1.x uses formats like grip to compress the data transferred in the messages. However, the header component of the message is always sent as plain text. Though the header itself is small, it gets larger due to the use of cookies or an increased number of requests.

**HTTP/2**

To deal with this bottleneck, HTTP/2 uses HPACK compression to decrease the average size of the header. This compression program encodes the header metadata using Huffman coding, which significantly reduces its size as a result. In addition, HPACK keeps track of previously transferred header values and further compresses them as per a dynamically modified index shared between client and server.

**Internal Representation of Objects in Java Script**

Objects, in JavaScript, is its most important data-type and forms the building blocks for modern JavaScript. These objects are quite different from JavaScript’s primitive data-types (Number, String, Boolean, null, and undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).

Objects are more complex and each object may contain any combination of these primitive data-types as well as reference data-types.  
An object, is a reference data type. Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the location in memory where the object is stored. The variables don’t actually store the value.

Loosely speaking, objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of “key: value” pairs. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.  
An object can be created with figure brackets {…} with an optional list of properties. A property is a “key: value” pair, where a key is a string (also called a “property name”), and value can be anything.

To understand this rather abstract definition, let us look at an example of a JavaScript Object:

let school = {  
name : “Vivekananda School”,  
location : “Delhi”,  
established : “1971”  
}  
In the above example “name”, “location”, “established” are all “keys” and “Vivekananda School”, “Delhi” and 1971 are values of these keys respectively.

Each of these keys is referred to as properties of the object. An object in JavaScript may also have a function as a member, in which case it will be known as a method of that object.

**Code kata practice:**

The code kata is very useful to know about the codes structure.

The code kata is given the conference about the codes.