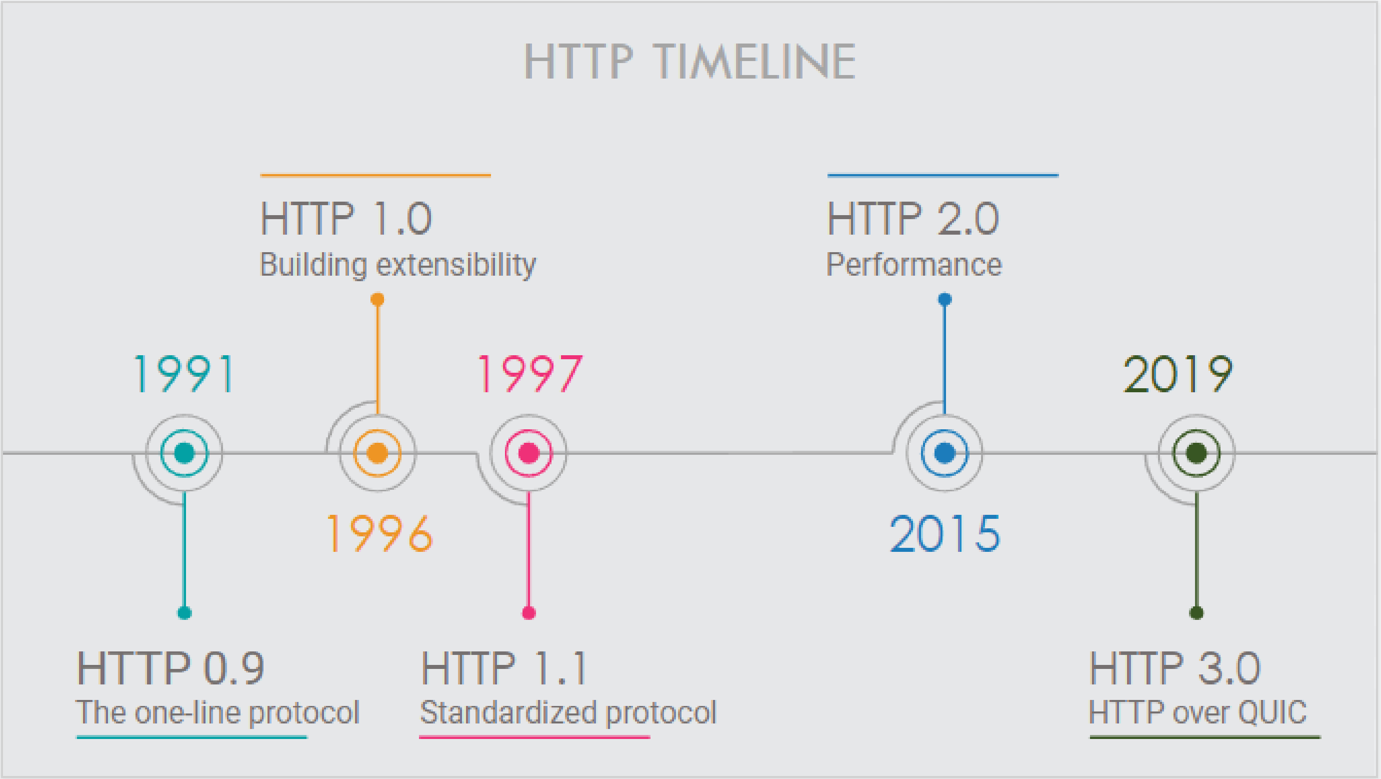
Difference between HTTP1.1 vs HTTP2

Acronym for Hypertext Transfer Protocol, HTTP is the protocol used to send and receive information on the web, and it's based on requests and responses between clients and servers. The basic operation of HTTP goes as follows: the client — a browser or device making a request — requests a certain resource by sending an information packet containing some headers to a URL. The server receives this information and returns a response.

How it all began



HTTP has been in use since 1991. The first version of the protocol, called HTTP/0.9, was a simple data transferring protocol in ASCII text format over the Internet. Then, the HTTP/1.0 version was released in 1996 to meet the need to transfer not only text. Richer data, request and response metadata, and content negotiation, for example, were now considered.

The third version, 1.1, was released in 1999. Its release is considered a milestone that set the Internet standard. The HTTP/1.1 protocol, besides solving numerous ambiguities of version 1.0, introduced critical performance improvements.

From the 2000s on, through the boom in Internet usage, the number of devices connected to it grew exponentially. With this exorbitant growth came the need for performance improvements. That's how SPDY was born in 2009, a protocol created by Google engineers to help overcome HTTP/1.1 performance problems.

These moves caught the attention of the HTTP Working Group (HTTP-WG). In 2015, 16 years after the release of HTTP/1.1, HTTP/2 was born. Without changing the protocol's semantics, significant improvements were achieved in information transport performance, as well as lower latency and higher throughput.

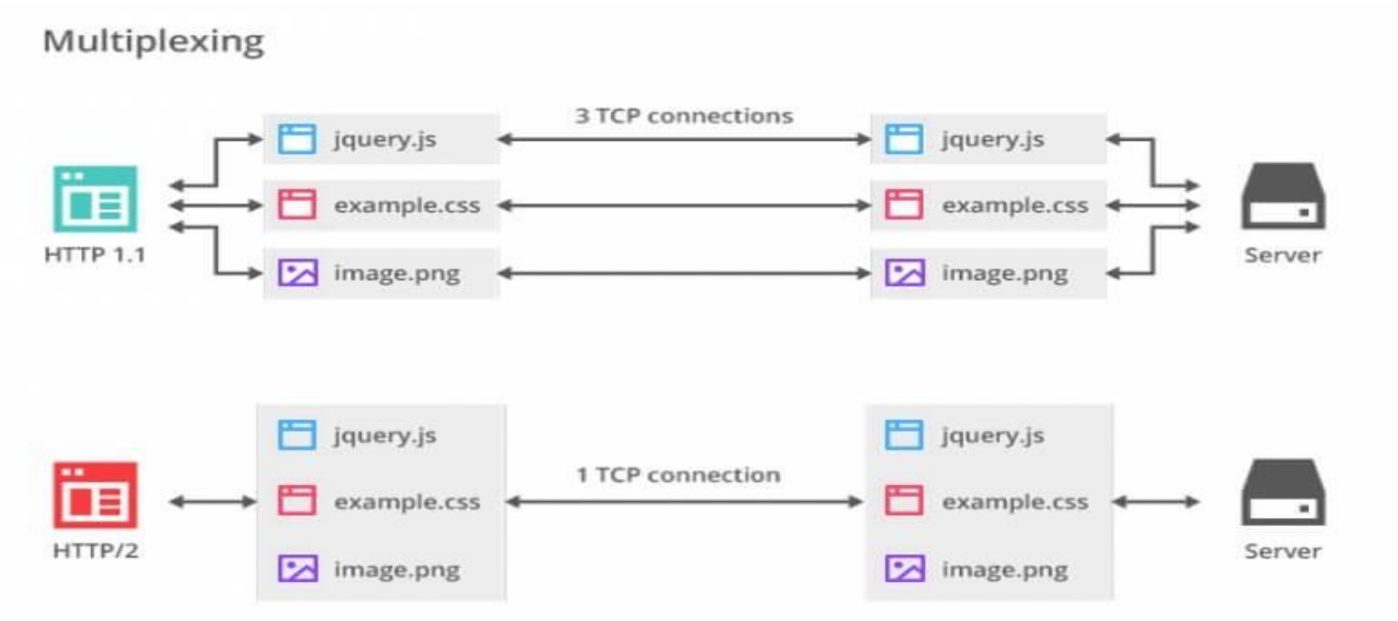
HTTP/2 Main Features

To better understand HTTP/2, take a look at the following characteristics.

Single connection and multiplexing

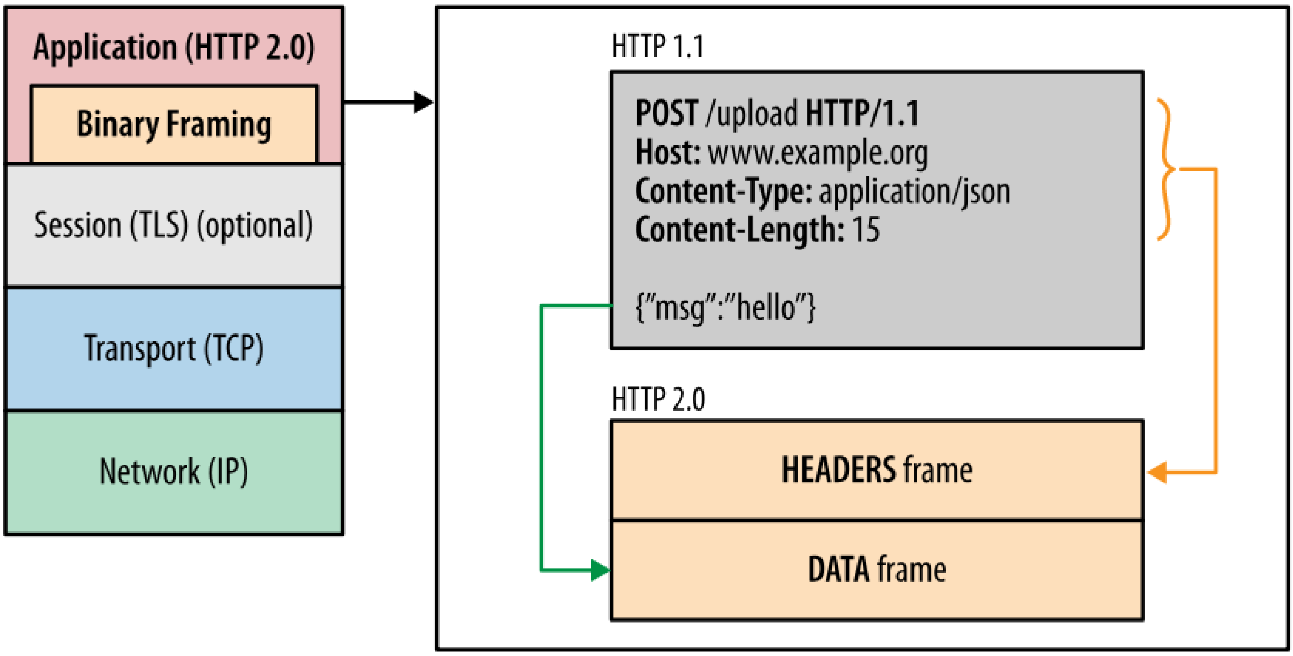
HTTP/1.1 is a sequential protocol, where the browser opens only one TCP connection, requests one file, and only after receiving the file it moves to the next one. If a file is too large, or the server processing is too slow, the page can crash. To minimize this issue, browsers usually open multiple connections, between 6 and 8, per server. In HTTP/2, on the other hand, a TCP connection will be persistent and only one per source is needed since parallel requests and responses can request/receive all the necessary files.

This brings a reduction in processing and memory consumption, a reduction in network operating cost, and increased usability. The result is reduced network latency and lower hardware and software costs.



Binary framing layer

This is the core of all HTTP/2 performance improvements, determining how HTTP messages are encapsulated and transferred between client and server. The encoding mechanism has been redesigned without changing the semantics of methods, verbs, and headers. Communication is broken into frames, over a single TCP connection.

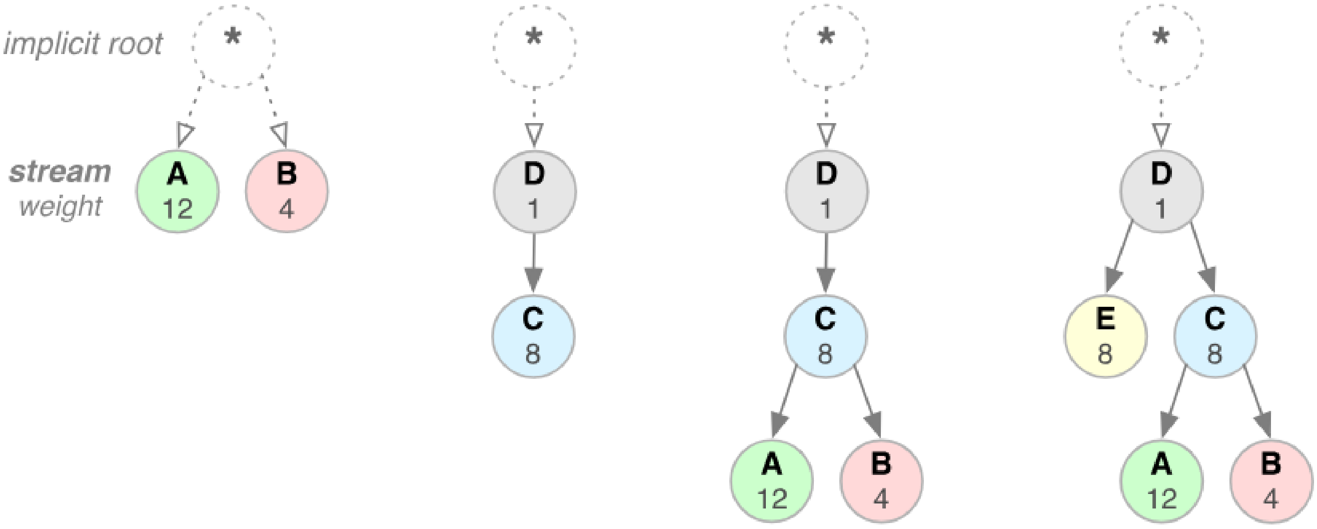


```
POST /upload HTTP/1.1
Host: www.example.org
Content-Type: application/json
Content-Length: 15

{"msg":"hello"}
```

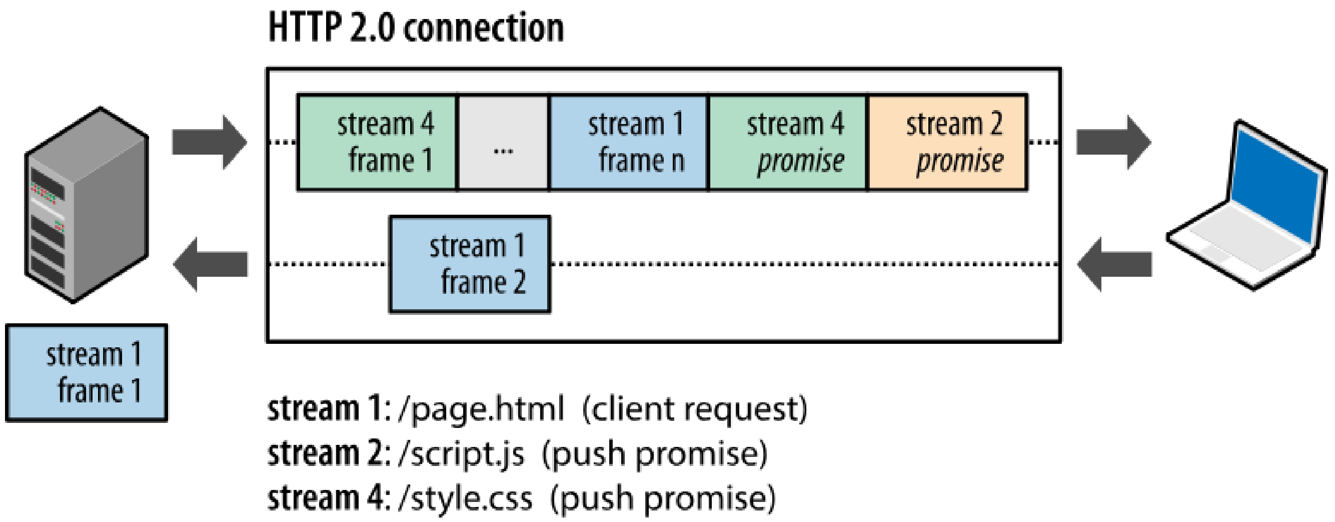
Request Prioritization

It enables the browser to request all elements when discovered, communicating to the server its intention to prioritize any of them. This is done through dependencies and stream weights. An example of this is when CSS files and JS files are requested, that with HTTP/2 the browser will prioritize CSS files first, even if its request in DOM order comes after the JS.



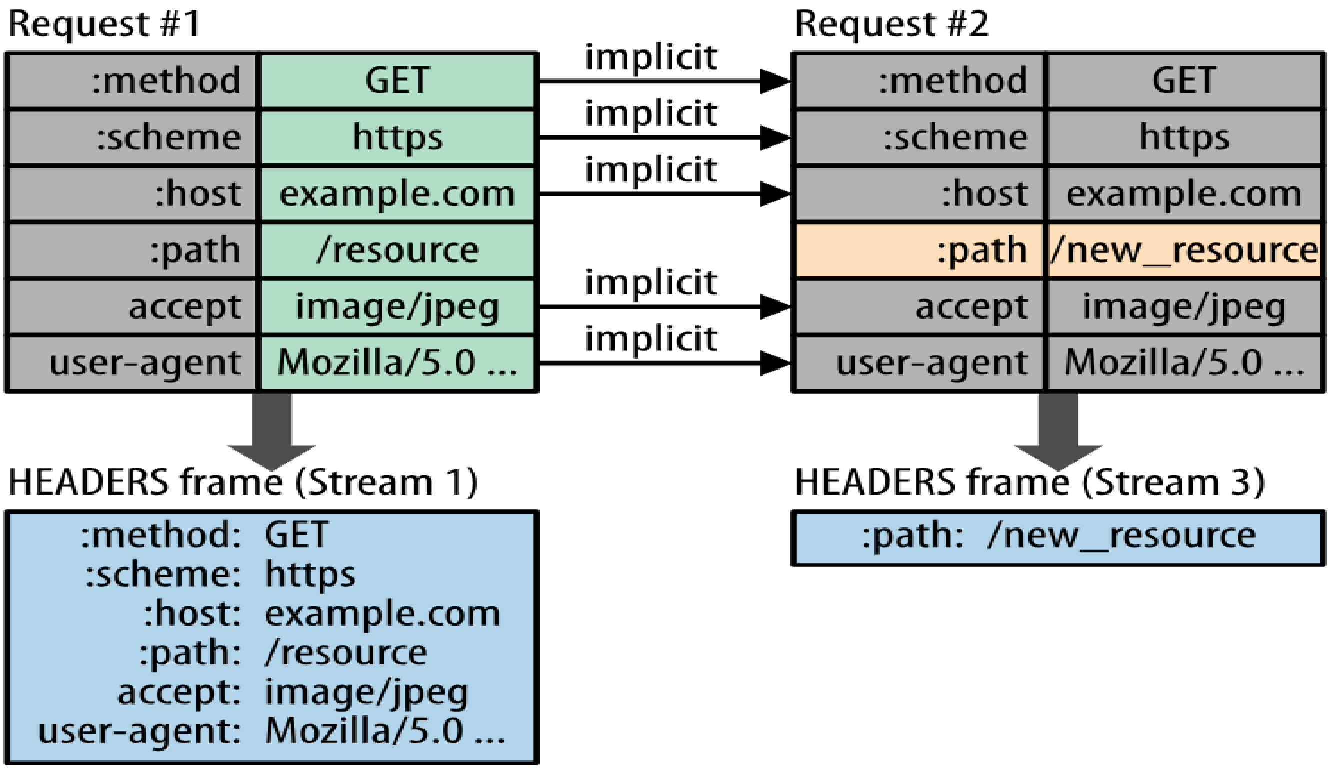
Server Push

It allows the server to send multiple responses to the client, from a single request, without the client explicitly asking for it. Imagine the following scenario: a web browser requests a page's HTML file; the server then responds with the requested file and also sends the CSS file, JavaScript, icons, and other things.



Automatic compression

HTTP requests have headers with important information about the resource and its properties. With HTTP/2 the headers are compressed using the HPACK algorithm, thus reducing the size of each transfer and maintaining and updating an indexed list of the header fields seen earlier. In addition, data compression via GZIP, which needs to be enabled in HTTP/1.1, became standard in version 2.



Request #1

:method	GET
:scheme	https
:host	example.com
:path	/resource
accept	image/jpeg
user-agent	Mozilla/5.0 ...

Request #2

:method	GET
:scheme	https
:host	example.com
:path	/new_resource
accept	image/jpeg
user-agent	Mozilla/5.0 ...

HEADERS frame (Stream 1)

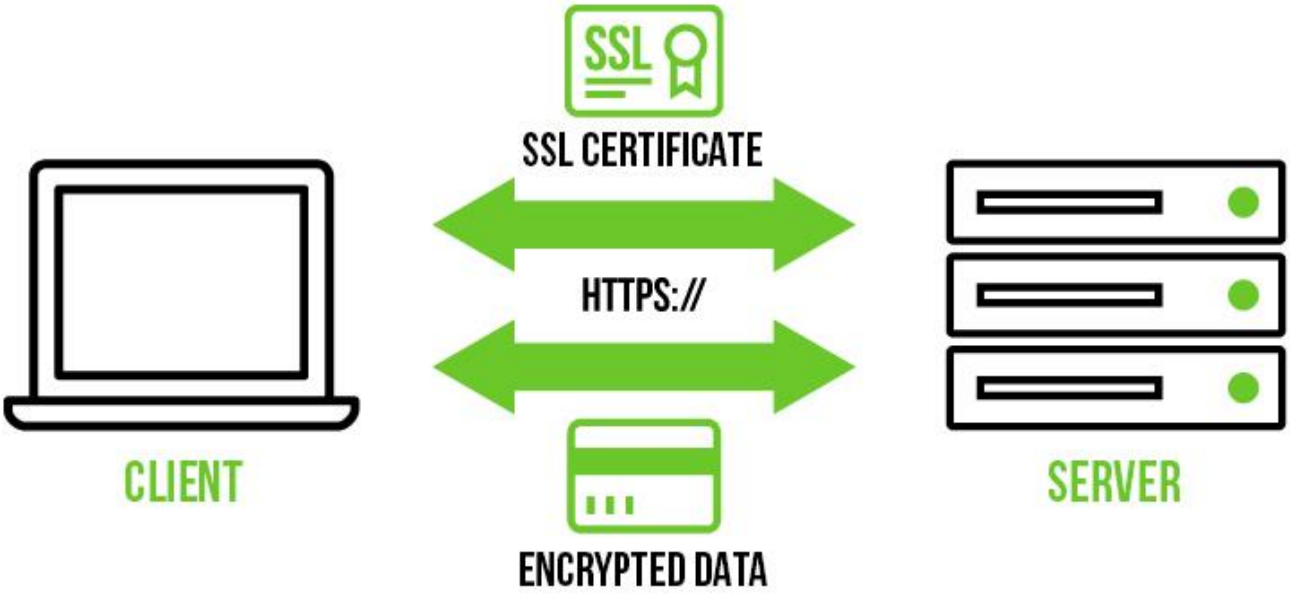
:method: GET
:scheme: https
:host: example.com
:path: /resource
accept: image/jpeg
user-agent: Mozilla/5.0 ...

HEADERS frame (Stream 3)

:path: /new_resource

Data security and encryption

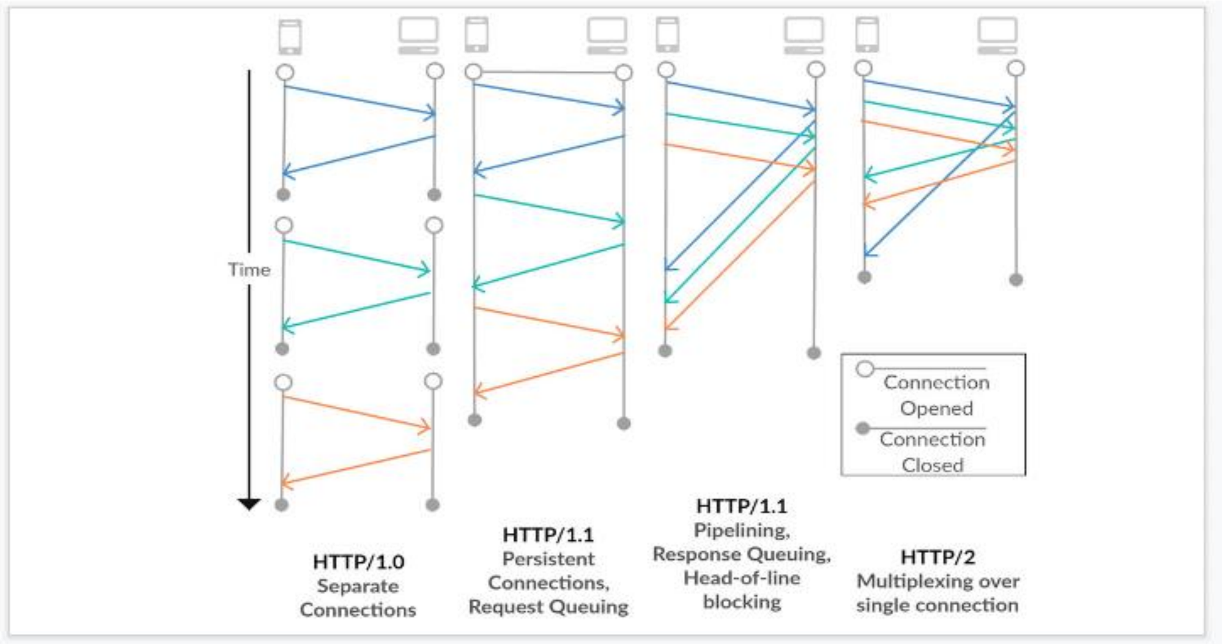
The HTTP/2 protocol has been implemented to work with or without encryption. However, all major browsers declared that they will only support HTTP/2 with encryption, requiring the use of an SSL certificate.



[IMAGEM: Client, SSL Certification, Encrypted Data, Server]

Conclusion

Web applications like Fusion Cloud, through the implementation of HTTP/2, are showing significant performance improvements, with a big difference in page load times, thus enabling a reduction in hardware costs and a better browsing experience for users.


Time
Connection Opened
Connection Closed
HTTP/1.0
Separate Connections
HTTP/1.1
Persistent Connections, Request Queuing
HTTP/1.1
Pipelining, Response Queuing, Head-of-line blocking
HTTP/2
Multiplexing over single connection

Objects And Its Internal Representation In JavaScript

Objects, in JavaScript, is it’s 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, 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.

For Eg. If your object is a student, it will have properties like name, age, address, id, etc and methods like **updateAddress, updateNam,** etc.