1. Difference between HTTP1.1 vs HTTP2?

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.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.

In HTTP1.1 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*. In 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. 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, a process called *multiplexing*. Multiplexing resolves the head-of-line blocking issue in HTTP/1.1 by ensuring that no message has to wait for another to finish.

HTTP1.1 uses multiple connections while HTTP2 uses the single TCP connections.

HTTP/1.1 relies on the transport layer to avoid buffer overflow, each new TCP connection requires a separate flow control mechanism. HTTP/2, however, multiplexes streams within a single TCP connection, and will have to implement flow control in a different manner. HTTP/2 multiplexes streams of data within a single TCP connection. As a result, receive windows on the level of the TCP connection are not sufficient to regulate the delivery of individual streams. HTTP/2 solves this problem by allowing the client and server to implement their own flow controls, rather than relying on the transport layer. The application layer communicates the available buffer space, allowing the client and server to set the receive window on the level of the multiplexed streams

In HTTP1.1 the resource inlinig is used where the inlined resources are not separate from the HTML document, there is no mechanism for the client to decline resources that it already has, or to place a resource in its cache. If multiple pages require the resource, each new HTML document will have the same resource inlined in its code, leading to larger HTML documents and longer load times than if the resource were simply cached in the beginning.

HTTP/2 enables multiple concurrent responses to a client’s initial GET request, a server can send a resource to a client along with the requested HTML page, providing the resource before the client asks for it. This process is called *server push*. In this way, an HTTP/2 connection can accomplish the same goal of resource inlining while maintaining the separation between the pushed resource and the document. This means that the client can decide to cache or decline the pushed resource separate from the main HTML document

In HTTP1.1 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. Although each header is quite small, the burden of this uncompressed data weighs heavier and 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.

HTTP/2 can split headers from their data, resulting in a header frame and a data frame. The HTTP/2-specific compression program HPACK can then compress this header frame. This algorithm can encode the header metadata using Huffman coding, thereby greatly decreasing its size.

1. History of HTTP version

In 1989, while he was working at CERN, Tim Berners-Lee wrote a proposal to build a hypertext system over the Internet. Initially calling it the Mesh, it was later renamed to World Wide Web during its implementation in 1990. Built over the existing TCP and IP protocols, it consisted of 4 building blocks:

* A textual format to represent hypertext documents, the [HyperText Markup Language](https://developer.mozilla.org/en-US/docs/Web/HTML) (HTML).
* A simple protocol to exchange these documents, the HypertText Transfer Protocol (HTTP).
* A client to display (and accidentally edit) these documents, the first Web browser called WorldWideWeb.
* A server to give access to the document, an early version of httpd.

[HTTP/0.9 – The one-line protocol](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP" \l "http0.9_%E2%80%93_the_one-line_protocol" \o "Permalink to HTTP/0.9 – The one-line protocol)

The initial version of HTTP had no version number; it has been later called 0.9 to differentiate it from the later versions. HTTP/0.9 is extremely simple: requests consist of a single line and start with the only possible method [GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET) followed by the path to the resource (not the URL as both the protocol, server, and port are unnecessary once connected to the server).

[HTTP/1.0 – Building extensibility](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP" \l "http1.0_%E2%80%93_building_extensibility" \o "Permalink to HTTP/1.0 – Building extensibility)

HTTP/0.9 was very limited and both browsers and servers quickly extended it to be more versatile:

* Versioning information is now sent within each request (HTTP/1.0 is appended to the GET line)
* A status code line is also sent at the beginning of the response, allowing the browser itself to understand the success or failure of the request and to adapt its behavior in consequence (like in updating or using its local cache in a specific way)
* The notion of HTTP headers has been introduced, both for the requests and the responses, allowing metadata to be transmitted and making the protocol extremely flexible and extensible.
* With the help of the new HTTP headers, the ability to transmit other documents than plain HTML files has been added (thanks to the [Content-Type](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type) header).

[HTTP/1.1 – The standardized protocol](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP" \l "http1.1_%E2%80%93_the_standardized_protocol" \o "Permalink to HTTP/1.1 – The standardized protocol)

In parallel to the somewhat chaotic use of the diverse implementations of HTTP/1.0, and since 1995, well before the publication of HTTP/1.0 document the next year, proper standardization was in progress. The first standardized version of HTTP, HTTP/1.1 was published in early 1997, only a few months after HTTP/1.0.

HTTP/1.1 clarified ambiguities and introduced numerous improvements:

* A connection can be reused, saving the time to reopen it numerous times to display the resources embedded into the single original document retrieved.
* Pipelining has been added, allowing to send a second request before the answer for the first one is fully transmitted, lowering the latency of the communication.
* Chunked responses are now also supported.
* Additional cache control mechanisms have been introduced.
* Content negotiation, including language, encoding, or type, has been introduced, and allows a client and a server to agree on the most adequate content to exchange.
* Thanks to the [Host](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Host) header, the ability to host different domains at the same IP address now allows server colocation.

[HTTP/2 – A protocol for greater performance](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP" \l "http2_%E2%80%93_a_protocol_for_greater_performance" \o "Permalink to HTTP/2 – A protocol for greater performance)

In the first half of the 2010s, Google demonstrated an alternative way of exchanging data between client and server, by implementing an experimental protocol SPDY. This amassed interest from developers working on both browsers and servers. Defining an increase in responsiveness, and solving the problem of duplication of data transmitted, SPDY served as the foundations of the HTTP/2 protocol.

The HTTP/2 protocol has several prime differences from the HTTP/1.1 version:

* It is a binary protocol rather than text. It can no longer be read and created manually. Despite this hurdle, improved optimization techniques can now be implemented.
* It is a multiplexed protocol. Parallel requests can be handled over the same connection, removing the order and blocking constraints of the HTTP/1.x protocol.
* It compresses headers. As these are often similar among a set of requests, this removes duplication and overhead of data transmitted.
* It allows a server to populate data in a client cache, in advance of it being required, through a mechanism called the server push.

1. List 5 difference between Browser JS(console) vs Nodejs

|  |  |
| --- | --- |
| Browser JS | Node JS |
| Javascript is a programming language that is used for writing scripts on the website. | NodeJS is a Javascript runtime environment. |
| Javascript can only be run in the browsers. | It is mostly used on server-side. |
| Javascript is used in frontend development. | Node JS is used in server-side development. |
| It is the upgraded version of ECMA script that uses Chrome’s V8 engine written in C++. | Node JS is written in C, C++ and Javascript. |
| Javascript is capable enough to add HTML and play with the DOM. | Nodejs does not have capability to add HTML tags. |

1. What happens when you type a URL in the address bar in the browser?

When a url is typed for example [www.example.com](http://www.example.com) the steps followed are as follows:

1. Type www.example.com into the address bar of your browser.
2. The browser checks the cache for a DNS record to find the corresponding IP address of [www.example.com](http://www.example.com.)
3. If the requested URL is not in the cache, ISP’s DNS server initiates a DNS query to find the IP address of the server that hosts [www.example.com](http://www.example.com)
4. The browser initiates a TCP connection with the server.
5. The browser sends an HTTP request to the web server.
6. The server handles the request and sends back a response.
7. The server sends out an HTTP response.
8. The browser displays the HTML content.