What is HTTP? Why is HTTP/2 faster than HTTP/1.1?

[HTTP](https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/) stands for hypertext transfer protocol, and it is the basis for almost all web applications. More specifically, HTTP is the method computers and servers use to request and send information. For instance, when someone navigates to cloudflare.com on their laptop, their web browser sends an HTTP request to the Cloudflare servers for the content that appears on the page. Then, Cloudflare servers send HTTP responses with the text, images, and formatting that the browser displays to the user.

The first usable version of HTTP was created in 1997. Because it went through several stages of development, this first version of HTTP was called HTTP/1.1. This version is still in use on the web. In 2015, a new version of HTTP called [HTTP/2](https://www.cloudflare.com/website-optimization/http2/what-is-http2/) was created.

HTTP/2 solves several problems that the creators of HTTP/1.1 did not anticipate. HTTP/2 is much faster and more efficient than HTTP/1.1. One of the ways in which HTTP/2 is faster is in how it prioritizes content during the loading process.

What is prioritization?

In the context of [web performance](https://www.cloudflare.com/learning/performance/why-site-speed-matters/), prioritization refers to the order in which pieces of content are loaded. Suppose a user visits a news website and navigates to an article. Should the photo at the top of the article load first? Should the text of the article load first? Should the banner ads load first?

Prioritization affects a webpage's load time. For example, certain resources, like large JavaScript files, may block the rest of the page from loading if they must load first. More of the page can load at once if these render-blocking resources load last.

In addition, the order in which these page resources load affects how the user perceives page load time. If only behind-the-scenes content (like a CSS file) or content the user can't see immediately (like banner ads at the bottom of the page) loads first, the user will think the page is not loading at all. If the content that is most important to the user loads first, such as the image at the top of the page, then the user will perceive the page as loading faster.

How does prioritization in HTTP/2 affect performance?

In HTTP/2, developers have hands-on, detailed control over prioritization. This allows them to maximize perceived and actual page load speed to a degree that was not possible in HTTP/1.1.

HTTP/2 offers a feature called weighted prioritization. This allows developers to decide which page resources will load first, every time. In HTTP/2, when a [client](https://www.cloudflare.com/learning/serverless/glossary/client-side-vs-server-side/) makes a request for a webpage, the server sends several streams of data to the client at once, instead of sending one thing after another. This method of data delivery is known as multiplexing. Developers can assign each of these data streams a different weighted value, and the value tells the client which data stream to render first.

Imagine that Alice wants to read a novel that her friend Bob wrote, but both Alice and Bob only communicate through the regular mail. Alice sends a letter to Bob and asks Bob to send her his novel. Bob decides to send the novel HTTP/1.1-style: He mails one chapter at a time, and he only mails the next chapter after receiving a reply letter from Alice confirming that she received the previous chapter. Using this method of content delivery, it takes Alice many weeks to read Bob's novel.

Now imagine that Bob decides to send Alice his novel HTTP/2-style: In this case, he sends each chapter of the novel separately (to stay within the postal service's size limits) but all at the same time. He also numbers each chapter: Chapter 1, Chapter 2, etc. Now, Alice receives the novel all at once and can assemble it in the correct order on her own time. If a chapter is missing, she may send a quick reply asking for that specific chapter, but otherwise the process is complete, and Alice can read the novel in just a few days.

In HTTP/2, data is sent all at once, much like Bob when he sends Alice multiple chapters at once. And just like Bob, developers get to number the chapters in HTTP/2. They can decide if the text of a webpage loads first, or the CSS files, or the JavaScript, or whatever they feel is most important for the user experience.

What are the other differences between HTTP/2 and HTTP/1.1 that impact performance?

**Multiplexing:** HTTP/1.1 loads resources one after the other, so if one resource cannot be loaded, it blocks all the other resources behind it. In contrast, HTTP/2 is able to use a single [TCP](https://www.cloudflare.com/learning/ddos/glossary/tcp-ip/) connection to send multiple streams of data at once so that no one resource blocks any other resource. HTTP/2 does this by splitting data into binary-code messages and numbering these messages so that the client knows which stream each binary message belongs to.

**Server push:** Typically, a server only serves content to a client device if the client asks for it. However, this approach is not always practical for modern webpages, which often involve several dozen separate resources that the client must request. HTTP/2 solves this problem by allowing a server to "push" content to a client before the client asks for it. The server also sends a message letting the client know what pushed content to expect – like if Bob had sent Alice a Table of Contents of his novel before sending the whole thing.

**Header compression:** Small files load more quickly than large ones. To speed up web performance, both HTTP/1.1 and HTTP/2 compress HTTP messages to make them smaller. However, HTTP/2 uses a more advanced compression method called HPACK that eliminates redundant information in HTTP header packets. This eliminates a few bytes from every HTTP packet. Given the volume of HTTP packets involved in loading even a single webpage, those bytes add up quickly, resulting in faster loading.

**What is 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 were:**

* Protocol negotiation mechanism — protocol electing, e.g., 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.

**Request multiplexing**

HTTP/2 can send **multiple requests** for data in parallel over a **single** TCP connection. This is **the most** **advanced** **feature** of the HTTP/2 protocol because it **allows you to download web files asynchronously from one server**. Most modern browsers limit TCP connections to one server.

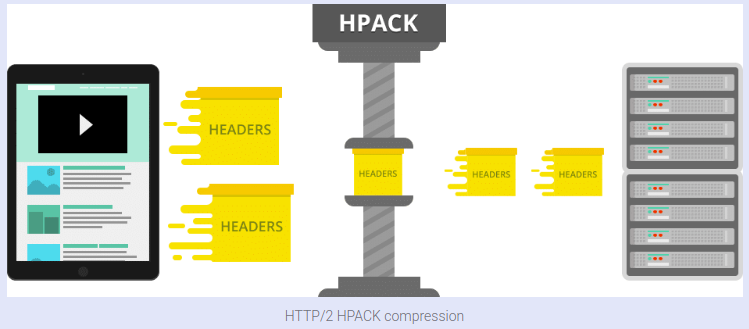


This reduces additional round trip time (RTT), **making your website load faster** without any optimization, and makes domain sharding unnecessary.

**Header compression**

HTTP/2 compress many redundant header frames. It uses the HPACK specification as a simple and secure approach to header compression. Both client and server maintain a list of headers used in previous client-server requests.

HPACK compresses the individual value of each header before it is transferred to the server, which then looks up the encoded information in a list of previously transferred header values to reconstruct the full header information.



**Binary protocol**

The latest HTTP version has evolved significantly in terms of capabilities and attributes such as transforming from a text protocol to a binary protocol. HTTP1.x used to process text commands to complete request-response cycles. HTTP/2 will use binary commands (in 1s and 0s) to execute the same tasks. This attribute eases complications with framing and simplifies implementation of commands that were confusingly intermixed due to commands containing text and optional spaces.

Browsers using HTTP/2 implementation will convert the same text commands into binary before transmitting it over the network.

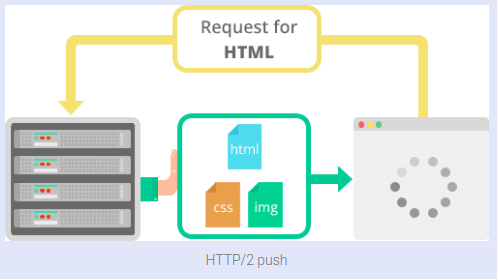


**Benefits:**

* Low overhead in parsing data — a critical value proposition in HTTP/2 vs HTTP1.
* Less prone to errors.
* Lighter network footprint.
* Effective network resource utilization.
* Eliminating security concerns associated with the textual nature of HTTP1.x such as response splitting attacks.
* Enables other capabilities of the HTTP/2 including compression, multiplexing, prioritization, flow control and effective handling of TLS.
* Compact representation of commands for easier processing and implementation.
* Efficient and robust in terms of processing of data between client and server.
* Reduced network latency and improved throughput.

**HTTP/2 Server Push**

This capability allows the server to send additional cacheable information to the client that isn’t requested but is anticipated in future requests. For example, if the client requests for the resource X and it is understood that the resource Y is referenced with the requested file, the server can choose to push Y along with X instead of waiting for an appropriate client request.



**Benefits:**

* The client saves pushed resources in the cache.
* The client can reuse these cached resources across different pages.
* The server can multiplex pushed resources along with originally requested information within the same TCP connection.
* The server can prioritize pushed resources — a key performance differentiator in HTTP/2 vs HTTP1.
* The client can decline pushed resources to maintain an effective repository of cached resources or disable Server Push entirely.
* The client can also limit the number of pushed streams multiplexed concurrently.

**2.HTTP VERSION HISTORY**

# Evolution of HTTP

**HTTP** (Hypertext Transfer Protocol) is the underlying protocol of the World Wide Web. Developed by Tim Berners-Lee and his team between 1989-1991, HTTP has seen many changes, keeping most of the simplicity and further shaping its flexibility. HTTP has evolved from an early protocol to exchange files in a semi-trusted laboratory environment, to the modern maze of the Internet, now carrying images, videos in high resolution and 3D.

## [Invention of the World Wide Web](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP#invention_of_the_world_wide_web)

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 Hypertext Transfer Protocol (HTTP).
* A client to display (and accidentally edit) these documents, the first Web browser called Worldwide Web.
* A server to give access to the document, an early version of httpd.

These four building blocks were completed by the end of 1990, and the first servers were already running outside of CERN by early 1991. On August 6th 1991, Tim Berners-Lee's [post](https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt) on the public alt.hypertext newsgroup is now considered as the official start of the World Wide Web as a public project.

The HTTP protocol used in those early phases was very simple, later dubbed HTTP/0.9, and sometimes as the one-line protocol.

## [HTTP/0.9 – The one-line protocol](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP#http0.9_%E2%80%93_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).

GET /mypage.html

The response is extremely simple too: it only consisted of the file itself.

<HTML>

A very simple HTML page

</HTML>

Unlike subsequent evolutions, there were no HTTP headers, meaning that only HTML files could be transmitted, but no other type of documents. There were no status or error codes: in case of a problem, a specific HTML file was send back with the description of the problem contained in it, for human consumption.

## [HTTP/1.0 – Building extensibility](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP#http1.0_%E2%80%93_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).

At this point, a typical request and response looked like this:

GET /mypage.html HTTP/1.0

User-Agent: NCSA\_Mosaic/2.0 (Windows 3.1)

200 OK

Date: Tue, 15 Nov 1994 08:12:31 GMT

Server: CERN/3.0 libwww/2.17

Content-Type: text/html

<HTML>

A page with an image

<IMG SRC="/myimage.gif">

</HTML>

Followed by a second connection and request to fetch the image (followed by a response to that request):

GET /myimage.gif HTTP/1.0

User-Agent: NCSA\_Mosaic/2.0 (Windows 3.1)

200 OK

Date: Tue, 15 Nov 1994 08:12:32 GMT

Server: CERN/3.0 libwww/2.17

Content-Type: text/gif

(image content)

These novelties have not been introduced as concerted effort, but as a try-and-see approach over the 1991-1995 period: a server and a browser added one feature and it saw if it got traction. A lot of interoperability problems were common. In November 1996, in order to solve these annoyances, an informational document describing the common practices has been published, [RFC 1945](https://tools.ietf.org/html/rfc1945). This is the definition of HTTP/1.0 and it is notable that, in the narrow sense of the term, it isn't an official standard.

## [HTTP/1.1 – The standardized protocol](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP#http1.1_%E2%80%93_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.

A typical flow of requests, all through one single connection is now looking like this:

GET /en-US/docs/Glossary/Simple\_header HTTP/1.1

Host: developer.mozilla.org

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate, br

Referer: https://developer.mozilla.org/en-US/docs/Glossary/Simple\_header

200 OK

Connection: Keep-Alive

Content-Encoding: gzip

Content-Type: text/html; charset=utf-8

Date: Wed, 20 Jul 2016 10:55:30 GMT

Etag: "547fa7e369ef56031dd3bff2ace9fc0832eb251a"

Keep-Alive: timeout=5, max=1000

Last-Modified: Tue, 19 Jul 2016 00:59:33 GMT

Server: Apache

Transfer-Encoding: chunked

Vary: Cookie, Accept-Encoding

(content)

GET /static/img/header-background.png HTTP/1.1

Host: developer.mozilla.org

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0) Gecko/20100101 Firefox/50.0

Accept: \*/\*

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate, br

Referer: https://developer.mozilla.org/en-US/docs/Glossary/Simple\_header

200 OK

Age: 9578461

Cache-Control: public, max-age=315360000

Connection: keep-alive

Content-Length: 3077

Content-Type: image/png

Date: Thu, 31 Mar 2016 13:34:46 GMT

Last-Modified: Wed, 21 Oct 2015 18:27:50 GMT

Server: Apache

(image content of 3077 bytes)

HTTP/1.1 was first published as [RFC 2068](https://tools.ietf.org/html/rfc2068) in January 1997.

## [More than 15 years of extensions](https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP#more_than_15_years_of_extensions)

Thanks to its extensibility – creating new headers or methods is easy – and even if the HTTP/1.1 protocol was refined over two revisions, [RFC 2616](https://tools.ietf.org/html/rfc2616) published in June 1999 and the series of [RFC 7230](https://tools.ietf.org/html/rfc7230)-[RFC 7235](https://tools.ietf.org/html/rfc7235) published in June 2014 in prevision of the release of HTTP/2, this protocol has been extremely stable over more than 15 years.

**3. DIFFRENCES BETWEEN NODE.JS AND JAVASSCRIPT(BROWSER)**

Difference between Node.JS and JavaScript

**1.**[NodeJS](https://www.geeksforgeeks.org/introduction-to-nodejs/)**:**  
NodeJS is a cross-platform and opensource JavaScript runtime environment that allows the JavaScript to be run on the server-side. Nodejs allows JavaScript code to run outside the browser. Nodejs comes with a lot of modules and mostly used in web development.

**2.**[JavaScript](https://www.geeksforgeeks.org/JavaScript-tutorial/)**:**  
JavaScript is a [Scripting language](https://www.geeksforgeeks.org/introduction-to-scripting-languages/). It is mostly abbreviated as JS. It can be said that JavaScript is the updated version of the ECMA script. JavaScript is a high-level programming language that uses the concept of OOP’s but it is based on prototype inheritance.

**Difference between Nodejs and JavaScript:**

| S.No | JavaScript | NodeJS |
| --- | --- | --- |
| 1. | JavaScript is a programming language that is used for writing scripts on the website. | NodeJS is a JavaScript runtime environment. |
| 2. | JavaScript can only be run in the browsers. | NodeJS code can be run outside the browser. |
| 3. | It is basically used on the client-side. | It is mostly used on the server-side. |
| 4. | JavaScript is capable enough to add HTML and play with the DOM. | Nodejs does not have capability to add HTML tags. |
| 5. | JavaScript can run in any browser engine as like JS core in safari and Spider monkey in Firefox. | Nodejs can only run in V8 engine of google chrome. |
| 6. | JavaScript is used in frontend development. | Nodejs is used in server-side development. |
| 7. | Some of the JavaScript frameworks are RamdaJS, TypedJS, etc. | Some of the Nodejs modules are Lodash, express etc. These modules are to be imported from npm. |
| 8. | It is the upgraded version of ECMA script that uses Chrome’s V8 engine written in C++. | Nodejs is written in C, C++, and JavaScript. |

**4.WHAT HAPPENS WHEN YOU TYPE URL IN THE BROWSER?**

[URL](https://www.geeksforgeeks.org/url-full-form/) stands for Uniform Resource Locator. URL is the address of the website which you can find in the address bar of your web browser. It is a reference to a resource on the internet, be it images, hypertext pages, audio/video files, etc.

**Example:**

https://practice.geeksforgeeks.org/

**What is**[DNS](https://www.geeksforgeeks.org/domain-name-server-dns-in-application-layer/)**:**  
DNS is short for Domain Name System. Like a phonebook, DNS maintains and maps the name of the website, i.e., URL, and particular IP address it links to. Every URL on the internet has a unique IP address which is of the computer which hosts the server of the website requested.

**Steps for what happens when we enter a URL:**

1. Browser checks cache for DNS entry to find the corresponding [IP address](https://www.geeksforgeeks.org/introduction-of-classful-ip-addressing/) of website.  
   It looks for following cache. If not found in one, then continues checking to the next until found.
   * Browser Cache
   * Operating Systems Cache
   * Router Cache
   * ISP Cache
2. If not found in cache, ISP’s (Internet Service Provider) DNS server initiates a DNS query to find IP address of server that hosts the domain name.  
   The requests are sent using small data packets that contain information content of request and IP address it is destined for.
3. Browser initiates a [TCP (Transfer Control Protocol)](https://www.geeksforgeeks.org/tcp-and-udp-in-transport-layer/) connection with the server using synchronize (SYN) and acknowledge(ACK) messages.
4. Browser sends an [HTTP](https://www.geeksforgeeks.org/http-non-persistent-persistent-connection/) request to the web server. GET or POST request.
5. Server on the host computer handles that request and sends back a response. It assembles a response in some format like JSON, [XML](https://www.geeksforgeeks.org/xml-basics/) and HTML.
6. Server sends out an HTTP response along with the status of response.
7. Browser displays [HTML](https://www.geeksforgeeks.org/html-tutorials/) content
8. Finally, Done.