mdn web docs

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 gone through many changes that have helped maintain its simplicity while shaping its flexibility. Keep reading to learn how HTTP evolved from a protocol designed to exchange files in a semitrusted laboratory environment into a modern internet maze that carries images and videos in high resolution and 3D.

Invention of the World Wide Web

In 1989, while working at CERN, Tim Berners-Lee wrote a proposal to build a hypertext system over the internet. Initially called the *Mesh*, it was later renamed the *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* (HTML).
- A simple protocol to exchange these documents, the *HyperText Transfer Protocol* (HTTP).
- A client to display (and edit) these documents, the first web browser called the *WorldWideWeb*.
- 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 running outside of CERN by early 1991. On August 6, 1991, Tim Berners-Lee posted on the public *alt.hypertext* newsgroup. This is now considered to be the official start of the World Wide Web as a public project.

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

HTTP/0.9 – The one-line protocol

The initial version of HTTP had no version number; it was later called 0.9 to differentiate it from later versions. HTTP/0.9 was extremely simple: requests consisted of a single line and started with the only possible method `GET` followed by the path to the resource. The full URL wasn't included as the protocol, server, and port weren't necessary once connected to the server.

HTTP

```
GET /mypage.html
```

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

HTML

```
<html>
  A very simple HTML page
</html>
```

Unlike subsequent evolutions, there were no HTTP headers. This meant that only HTML files could be transmitted. There were no status or error codes. If there was a problem, a specific HTML file was generated and included a description of the problem for human consumption.

HTTP/1.0 – Building extensibility

HTTP/0.9 was very limited, but browsers and servers quickly made it more versatile:

- Versioning information was sent within each request (`HTTP/1.0` was appended to the `GET` line).
- A status code line was also sent at the beginning of a response. This allowed the browser itself to recognize the success or failure of a request and adapt its

behavior accordingly. For example, updating or using its local cache in a specific way.

- The concept of HTTP headers was introduced for both requests and responses. Metadata could be transmitted and the protocol became extremely flexible and extensible.
- Documents other than plain HTML files could be transmitted thanks to the `Content-Type` header.

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

```http
GET /mypage.html HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)

HTTP/1.0 200 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>
```

It was followed by a second connection and a request to fetch the image (with the corresponding response):

```http
GET /myimage.gif HTTP/1.0
User-Agent: NCSA_Mosaic/2.0 (Windows 3.1)

HTTP/1.0 200 OK
Date: Tue, 15 Nov 1994 08:12:32 GMT
Server: CERN/3.0 libwww/2.17
Content-Type: text/gif
(image content)
```

Between 1991-1995, these were introduced with a try-and-see approach. A server and a browser would add a feature and see if it got traction. Interoperability problems were common. In an effort to solve these issues, an informational document that described the common practices was published in November 1996. This was known as RFC 1945 and defined HTTP/1.0.

HTTP/1.1 – The standardized protocol

In the meantime, proper standardization was in progress. This happened in parallel to the diverse implementations of HTTP/1.0. 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 could be reused, which saved time. It no longer needed to be opened multiple times to display the resources embedded in the single original document.
- Pipelining was added. This allowed a second request to be sent before the answer to the first one was fully transmitted. This lowered the latency of the communication.
- Chunked responses were also supported.
- Additional cache control mechanisms were introduced.
- Content negotiation, including language, encoding, and type, was introduced. A client and a server could now agree on which content to exchange.
- Thanks to the `Host` header, the ability to host different domains from the same IP address allowed server collocation.

A typical flow of requests, all through one single connection, looked like this:

HTTP

```
GET /en-US/docs/Glossary/CORS-safelisted_request_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/CORS-
safelisted_request_header

HTTP/1.1 200 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/CORS-
safelisted_request_header

HTTP/1.1 200 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 in January 1997.

More than two decades of development

The extensibility of HTTP made it easy to create new headers and methods. Even though the HTTP/1.1 protocol was refined over two revisions, RFC 2616 published in June 1999 and RFC 7230 -RFC 7235 published in June 2014 before the release of HTTP/2, it was extremely stable for more than 15 years. HTTP/1.1 was updated again in 2022 with RFC 9110 . Not only was HTTP/1.1 updated, but all of HTTP was revised and is now split into the following documents: semantics (RFC 9110), caching (RFC 9111) applying to all HTTP versions, and HTTP/1.1 (RFC 9112), HTTP/2 (RFC 9113), and HTTP/3 (RFC 9114). In addition, the specification finally achieved the status of Internet Standard (STD 97), whereas before it was always a proposed/draft standard.

Using HTTP for secure transmissions

The largest change to HTTP was made at the end of 1994. Instead of sending HTTP over a basic TCP/IP stack, the computer-services company Netscape Communications created an additional encrypted transmission layer on top of it: SSL. SSL 1.0 was never released to the public, but SSL 2.0 and its successor SSL 3.0 allowed for the creation of e-commerce websites. To do this, they encrypted and guaranteed the authenticity of the messages exchanged between the server and client. SSL was eventually standardized and became TLS.

During the same time period, it became clear that an encrypted transport layer was needed. The web was no longer a mostly academic network, and instead became a jungle where advertisers, random individuals, and criminals competed for as much private data as possible. As the applications built over HTTP became more powerful and required access to private information like address books, email, and user location, TLS became necessary outside the e-commerce use case.

Using HTTP for complex applications

Tim Berners-Lee didn't originally envision HTTP as a read-only medium. He wanted to create a web where people could add and move documents remotely—a kind of distributed file system. Around 1996, HTTP was extended to allow authoring, and a standard called WebDAV was created. It grew to include specific applications like CardDAV for handling address book entries and CalDAV for dealing with calendars. But all these *DAV extensions had a flaw: they were only usable when implemented by the servers.

In 2000, a new pattern for using HTTP was designed: representational state transfer (or REST). The API wasn't based on the new HTTP methods, but instead relied on access to specific URIs with basic HTTP/1.1 methods. This allowed any web application to let an API retrieve and modify its data without having to update the browsers or the servers. All necessary information was embedded in the files that the websites served through standard HTTP/1.1. The drawback of the REST model was that each website defined its own nonstandard RESTful API and had total control of it. This differed from the *DAV extensions where clients and servers were interoperable. RESTful APIs became very common in the 2010s.

Since 2005, more APIs have become available to web pages. Several of these APIs create extensions to the HTTP protocol for specific purposes:

- Server-sent events, where the server can push occasional messages to the browser.
- WebSocket, a new protocol that can be set up by upgrading an existing HTTP connection.

Relaxing the security-model of the web

HTTP is independent of the web security model, known as the same-origin policy. In fact, the current web security model was developed after the creation of HTTP! Over the years, it proved useful to lift some restrictions of this policy under certain constraints. The server transmitted how much and when to lift such restrictions to

the client using a new set of HTTP headers. These were defined in specifications like Cross-Origin Resource Sharing (CORS) and the Content Security Policy (CSP).

In addition to these large extensions, many other headers were added, sometimes only experimentally. Notable headers are the Do Not Track (`DNT`) header to control privacy, `X-Frame-Options` , and `Upgrade-Insecure-Requests` but many more exist.

HTTP/2 – A protocol for greater performance

Over the years, web pages became more complex. Some of them were even applications in their own right. More visual media was displayed and the volume and size of scripts adding interactivity also increased. Much more data was transmitted over significantly more HTTP requests and this created more complexity and overhead for HTTP/1.1 connections. To account for this, Google implemented an experimental protocol SPDY in the early 2010s. This alternative way of exchanging data between client and server amassed interest from developers working on both browsers and servers. SPDY defined an increase in responsiveness and solved the problem of duplicate data transmission, serving as the foundation for the HTTP/2 protocol.

The HTTP/2 protocol differs from HTTP/1.1 in a few ways:

- It's a binary protocol rather than a text protocol. It can't be read and created manually. Despite this hurdle, it allows for the implementation of improved optimization techniques.
- It's a multiplexed protocol. Parallel requests can be made over the same connection, removing the constraints of the HTTP/1.x protocol.
- It compresses headers. As these are often similar among a set of requests, this removes the duplication and overhead of data transmitted.
- It allows a server to populate data in a client cache through a mechanism called the server push.

Officially standardized in May 2015, HTTP/2 use peaked in January 2022 at 46.9% of all websites (see these stats). High-traffic websites showed the most rapid adoption in an effort to save on data transfer overhead and subsequent budgets.

This rapid adoption was likely because HTTP/2 didn't require changes to websites and applications. To use it, only an up-to-date server that communicated with a recent browser was necessary. Only a limited set of groups was needed to trigger adoption, and as legacy browser and server versions were renewed, usage was naturally increased, without significant work for web developers.

Post-HTTP/2 evolution

HTTP's extensibility is still being used to add new features. Notably, we can cite new extensions of the HTTP protocol that appeared in 2016:

- Support for `Alt-Svc` allowed the dissociation of the identification and the location of a given resource. This meant a smarter CDN caching mechanism.
- The introduction of client hints allowed the browser or client to proactively communicate information about its requirements and hardware constraints to the server.
- The introduction of security-related prefixes in the `Cookie` header helped guarantee that secure cookies couldn't be altered.

HTTP/3 - HTTP over QUIC

The next major version of HTTP, HTTP/3 has the same semantics as earlier versions of HTTP but uses QUIC instead of TCP for the transport layer portion. By October 2022, 26% of all websites were using HTTP/3 .

QUIC is designed to provide much lower latency for HTTP connections. Like HTTP/2, it is a multiplexed protocol, but HTTP/2 runs over a single TCP connection, so packet loss detection and retransmission handled at the TCP layer can block all streams. QUIC runs multiple streams over UDP and implements packet loss detection and

retransmission independently for each stream, so that if an error occurs, only the stream with data in that packet is blocked.

Defined in RFC 9114 , HTTP/3 is supported by most major browsers including Chromium (and its variants such as Chrome and Edge) and Firefox.