# History of HTTP

Invented by Tim Berners-Lee at CERN in the years 1989–1991, HTTP (Hypertext Transfer Protocol) is the underlying communication protocol of World Wide Web. **HTTP functions as a request–response protocol in the client–server computing model.** HTTP standards are developed by the [Internet Engineering Task Force](https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force) (IETF) and the [World Wide Web Consortium](https://en.wikipedia.org/wiki/World_Wide_Web_Consortium) (W3C), culminating in the publication of a series of [Requests for Comments](https://en.wikipedia.org/wiki/Requests_for_Comments) (RFCs). HTTP has four versions — HTTP/0.9, HTTP/1.0, HTTP/1.1, and HTTP/2.0. Today the version in common use is HTTP/1.1 and the future will be HTTP/2.0.

## **HTTP/0.9 — The One-line Protocol**

* Initial version of HTTP — a simple client-server, request-response, telenet-friendly protocol
* Request nature: single-line (method + path for requested document)
* Methods supported: GET only
* Response type: hypertext only
* Connection nature: terminated immediately after the response
* No HTTP headers (cannot transfer other content type files), No status/error codes, No URLs, No versioning

$> telnet ashenlive.com 80**(Connection 1 Establishment - TCP Three-Way Handshake)**Connected to xxx.xxx.xxx.xxx**(Request)**  
GET /my-page.html**(Response in hypertext)**  
<HTML>  
A very simple HTML page  
</HTML>**(Connection 1 Closed - TCP Teardown)**

Popular web servers (Apache, Nginx) still supports HTTP/0/9. Try opening up a Telnet session and accessing google.com

## **HTTP/1.0 — Building extensibility**

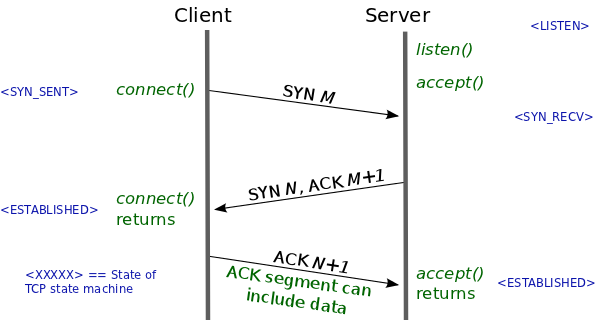
* Browser-friendly protocol
* Provided header fields including rich metadata about both request and response (HTTP version number, status code, content type)
* Response: not limited to hypertext (Content-Type header provided ability to transmit files other than plain HTML files — e.g. scripts, stylesheets, media)
* Methods supported: GET , HEAD , POST
* Connection nature: terminated immediately after the response

**connection 1 Establishment - TCP Three-Way Handshake)**  
Connected to xxx.xxx.xxx.xxx**(Request)**

GET /my-page.html HTTP/1.0   
User-Agent: NCSA\_Mosaic/2.0 (Windows 3.1)**(Response)**HTTP/1.0 200 OK   
Content-Type: text/html   
Content-Length: 137582  
Expires: Thu, 01 Dec 1997 16:00:00 GMT  
Last-Modified: Wed, 1 May 1996 12:45:26 GMT  
Server: Apache 0.84  
  
<HTML>   
A page with an image  
 <IMG SRC="/myimage.gif">  
</HTML>**(Connection 1 Closed - TCP Teardown)------------------------------------------(Connection 2 Establishment - TCP Three-Way Handshake)**  
Connected to xxx.xxx.xxx.xxx**(Request)**GET /myimage.gif HTTP/1.0  
User-Agent: NCSA\_Mosaic/2.0 (Windows 3.1)  
  
**(Response)**  
HTTP/1.0 200 OK   
Content-Type: text/gif   
Content-Length: 137582  
Expires: Thu, 01 Dec 1997 16:00:00 GMT  
Last-Modified: Wed, 1 May 1996 12:45:26 GMT  
Server: Apache 0.84[image content]**(Connection 2 Closed - TCP Teardown)**

## **Establishing a new connection for each request — major problem in both HTTP/0.9 and HTTP/1.0**

Both HTTP/0.9 and HTTP/1.0 required to open up a new connection for each request (and close it immediately after the response was sent). Each time a new connection establishes, a TCP three-way handshake should also occur. For better performance, it was crucial to reduce these round-trips between client and server. HTTP/1.1 solved this with persistent connections.

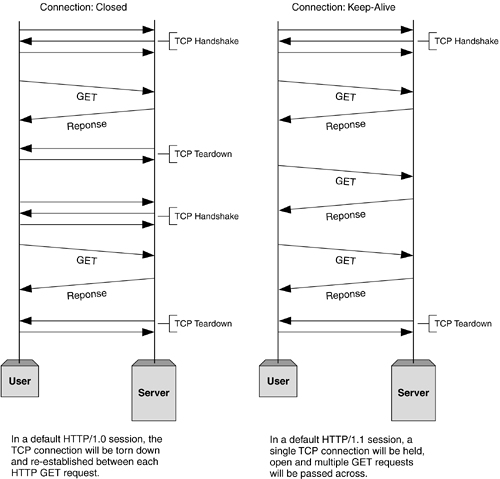


A typical TCP Three-way Handshake (see how the TCP state machine is changing its state) from [lwn.net](https://lwn.net/Articles/508865/)

# HTTP/1.1 — The standardized protocol

* This is the HTTP version currently in common use.
* Introduced critical performance optimizations and feature enhancements — persistent and pipelined connections, chunked transfers, compression/decompression, content negotiations, virtual hosting (a server with a single IP Address hosting multiple domains), faster response and great bandwidth savings by adding cache support.
* Methods supported: GET , HEAD , POST , PUT , DELETE , TRACE , OPTIONS
* Connection nature: long-lived

**(Connection 1 Establishment - TCP Three-Way Handshake)**  
Connected to xxx.xxx.xxx.xxx**(Request 1)**  
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  
  
**(Response 1)**  
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]  
  
**(Request 2)**  
GET /static/img/header-background.png HTTP/1.1  
Host: developer.cdn.mozilla.net  
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  
  
**(Response 2)**  
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



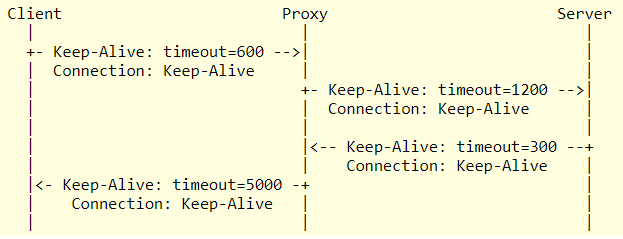
Before establishing any connection, a TCP three-way handshake happens. At the end, after sending all data to Client, Server sends a message saying there’s no more data to send. Then the client closes the connection (TCP teardown). The problem in HTTP/1.0 is, for each request-response cycle, a connection needs to be opened and closed. And the advantage of using HTTP/1.1 is, we can reuse the same open connection for multiple request-response cycles. (Image from [informit.com](http://www.informit.com/articles/article.aspx?p=169578))

# Keep-Alive and Upgrade headers

## **Keep-Alive header**

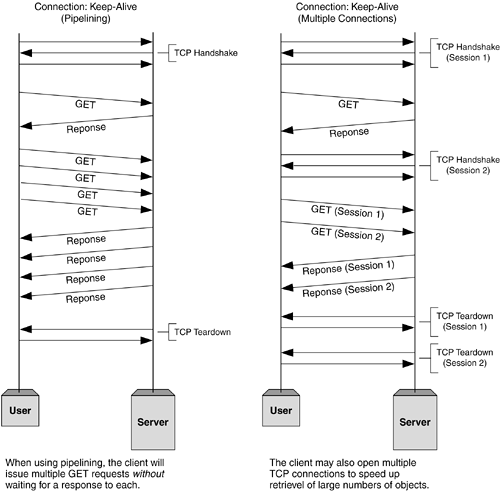
* The Keep-Alive header was used prior to HTTP/1.1 and was obsoleted by HTTP/1.1 making persistent connections the default behavior. Keep-Alive header can be used to define policies for long-lived communication between hosts (i.e. allows a connection to stay active until an event occurs). This laid foundation for persistence, reusable connections, pipelining, and many more enhanced capabilities in modern web communication protocols.
* Client, server, or any intermediary can provide information for Keep-Alive header independently. Also, a host can add timeout and max parameters in order to set a timeout or limit maximum request count per connection.

HTTP/1.1 200 OK  
**Connection: Keep-Alive**  
Content-Encoding: gzip  
Content-Type: text/html; charset=utf-8  
Date: Thu, 11 Aug 2016 15:23:13 GMT  
**Keep-Alive: timeout=5, max=1000**  
Last-Modified: Mon, 25 Jul 2016 04:32:39 GMT  
Server: Apache  
  
[body]



This example from [ietf.org](https://tools.ietf.org/id/draft-thomson-hybi-http-timeout-01.html) shows how a Keep-Alive header could be used. **All connections are independently negotiated**. The client indicates a timeout of 600 seconds (10 minutes), but the proxy is only prepared to retain the connection for at least 120 seconds (2 minutes). On the link between proxy and server, the proxy requests a timeout of 1200 seconds and the server reduces this to 300 seconds. As this example shows, the timeout policies maintained by the proxy are different for each connection. **Each connection hop is independent**.

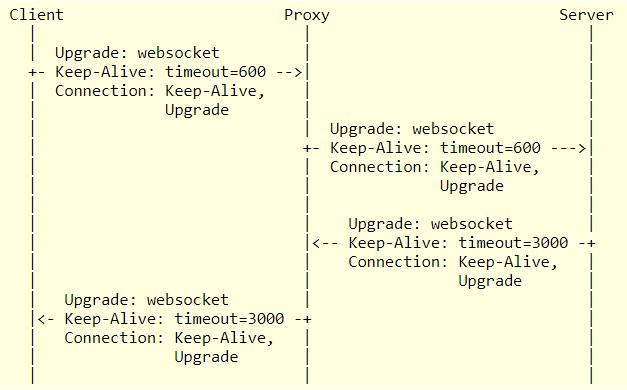
* HTTP pipelining, multiple connections, and many more improvements have been implemented, thanks to the Keep-Alive header’s behavior.



HTTP Pipelining and Multiple Parallel Connections (Image from [informit.com](http://www.informit.com/articles/article.aspx?p=169578))

## **Upgrade header**

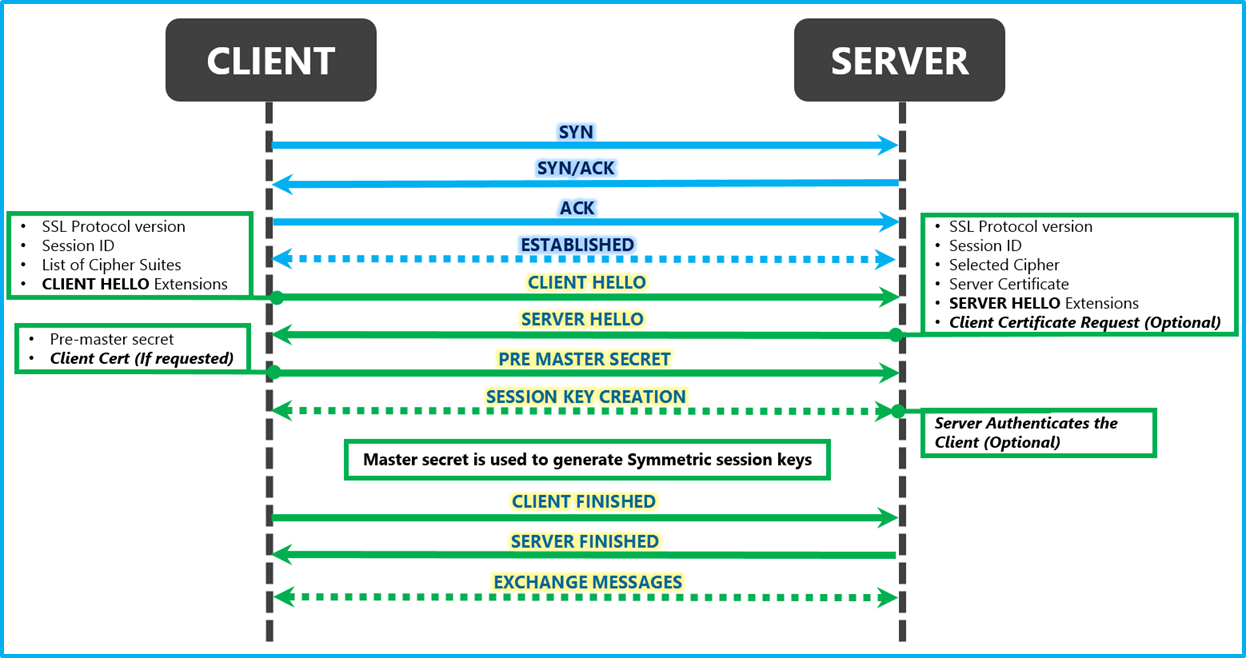
* With Upgrade header introduced in HTTP/1.1, it is possible to start a connection using a commonly-used protocol, such as HTTP/1.1, then request that the connection switch to an enhanced protocol type like HTTP/2.0 or WebSockets.
* In an upgraded protocol connection, max parameter (maximum request count) is not present. The upgraded protocol can provide new policies for timeout parameter (if not specifically defined, it uses default timeout value in underlying protocol).



This example from [ietf.org](https://tools.ietf.org/id/draft-thomson-hybi-http-timeout-01.html) shows the headers included in an upgrade from HTTP/1.1 to [WebSocket](https://tools.ietf.org/id/draft-thomson-hybi-http-timeout-01.html" \l "RFC6455)[RFC6455]. With a websocket upgrade, the connections on each hop cannot have independent lifecycles on either side of an intermediary. After the upgrade, timeout policies cannot be independent for each connection. The proxy adjusts the timeout value to reflect the lower of the values set by client and the proxy policies so that the server is aware of the connection characteristics; similarly, the value from the server is provided to the client. Upgrade is said to be a hop-by-hop header.

**HTTPS**

* Hyper Text Transfer Protocol Secure (HTTPS) is the secure version of HTTP. It uses SSL/TLS for secure encrypted communications.
* Originally developed by Netscape in mid-1990s, SSL (Secure Socket Layer) is a cryptographic protocol enhancement to HTTP, which defines how client and server should communicate with each other securely. TLS (Transport Layer Security) is the successor of SSL.
* An HTTPS connection can protect the data transfer from the man-in-the-middle attacks and common security threats by providing bidirectional encryption for communications between a client and server.



Diagrammatic representation of the SSL Handshake from [msdn.microsoft.com](https://blogs.msdn.microsoft.com/kaushal/2013/08/02/ssl-handshake-and-https-bindings-on-iis/) — TCP Connection > SSL/TLS Client Hello > SSL/TLS Server Hello > SSL/TLS Certificate > SSL/TLS Client Key Exchange > SSL/TLS New Session Ticket > HTTPS Encrypted Data Exchange

## **SSL/TLS Handshake — major problem in HTTPS**

* Although HTTPS is secure by its design, the SSL/TLS handshake process consumes a significant time before establishing an HTTPS connection. It normally costs 1–2 seconds and drastically slows down the startup performance of a website.

# HTTP/2.0 and the future

All above features are being used by major web servers and browsers today. But modern enhancements like HTTP/2.0, Server Side Events (SSE), and Websockets have changed the way that the traditional HTTP works. In my next article on [Web API Design with HTTP and Websockets](https://medium.com/platform-engineer/web-api-design-35df8167460), we will discuss how we should choose them in real-world projects.