**HTTP**

The Hypertext Transfer Protocol (HTTP) is one of the most ubiquitous and widely adopted application protocols on the Internet: it is the common language between clients and servers, enabling the modern web

**HTTP 0.9: The One-Line Protocol(1991)**

Tim Berners-Lee was originally designed HTTP with Simplictity.Later, it developed with file transfer functionality, ability to request an index search of a hypertext archive, format negotiation, and an ability to refer the client to another server. Only **Get & Put** request will support. Only the file path / Name will be appear in the GET request. Server will send the content of the file as a response.

**Flow:**

* Client creates a TCP connection to host
* **Request**

Get <file name/Path>/index.html // Only Get & Put request will support. Only the file path / Name will be appear in the GET request

* **Response**

<html>……………</html> // Server will send the content of the file as a response.

* The server terminates the TCP Connection

**Example:**

**$> telnet google.com 80**

Connected to 74.125.xxx.xxx

GET /about/

(hypertext response)

(connection closed)

**HTTP/1.0: Rapid Growth and Informational RFC(1996)**

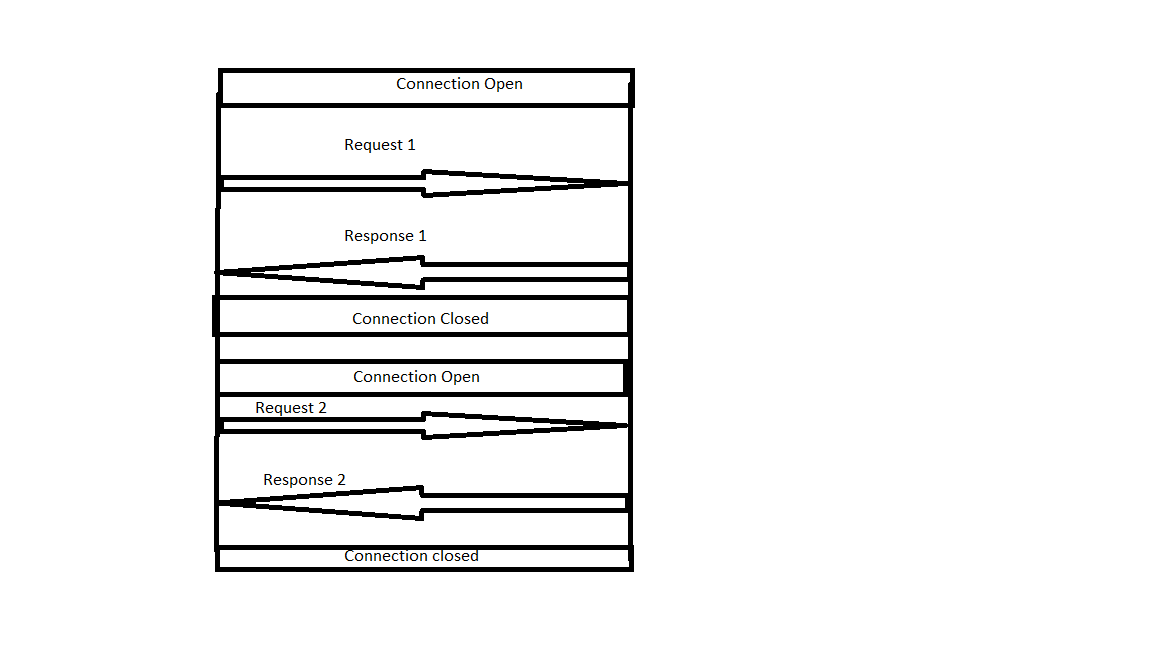
Connections cannot be reused, and a TCP connection (three-way handshake and four waved hands) is required for each request sent, which makes the network utilization rate very low.

Format of request:

1. There is a first request line (from the client) or response line (from the server)
2. Then there are a few header lines (e.g. to give information about the client or server, to indicate following contents). The header always ends with a blank line.
3. Finally there can be a ``body'' section containing contents (user data from the client or response from the server).

Client Request Contains:

1. **GET** is a request for information located at a specific place (i.e. a Universal Ressource Identifier, URI). URIs are basically URLs.
2. The **POST** method allows client request data to be sent to the server. Note that a client can also send data to the server by appending it to the URL of a GET request (you will see that later).
3. The **HEAD** method works like GET but is only used to retrieve document or ressource header information (e.g. contents inside the <HEAD>....</HEAD> of a HTML document.

****

**Flow:**

In GET /index.html HTTP/1.0

* /index.html is te resource that the client want to get.
* HTTP/1.0, tells the server that the client is using HTTP 1.0 protocol.

In each request, the server response by sending a status line with a number of response, header and the content of requester resource.

**$> telnet website.org 80**

Connected to xxx.xxx.xxx.xxx

**GET** /rfc/rfc1945.txt HTTP/1.0

User-Agent: CERN-LineMode/2.15 libwww/2.17b3

Accept: \*/\*

HTTP/1.0 200 **OK**

Content-Type: text/plain

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

(plain-text response)

(connection closed)

**HTTP/1.1: Internet Standard(1997)**

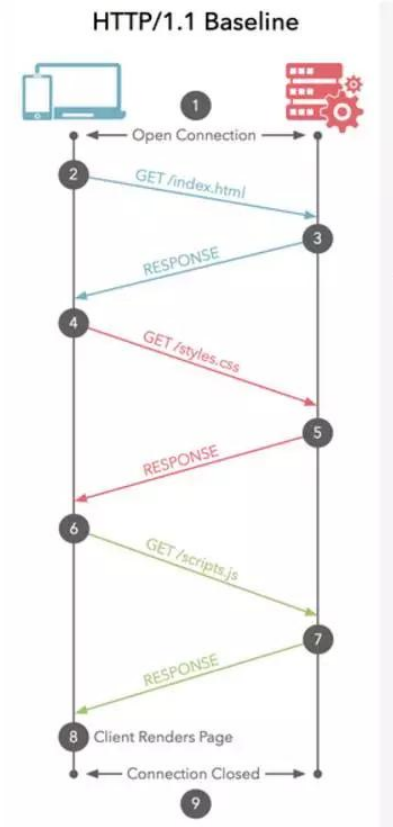
HTTP 1.1 is the latest version of Hypertext Transfer Protocol ([HTTP](https://whatis.techtarget.com/definition/HTTP-Hypertext-Transfer-Protocol)), the World Wide Web application [protoco](https://searchnetworking.techtarget.com/definition/protocol)l that runs on top of the Internet's [TCP/IP](https://searchnetworking.techtarget.com/definition/TCP-IP) suite of protocols. HTTP 1.1 provides faster delivery of Web pages than the original HTTP and reduces Web traffic.

Instead of opening and closing a connection for each application request, HTTP 1.1 provides a persistent connection that allows multiple requests to be batched or pipelined to an output [buffer](https://whatis.techtarget.com/definition/buffer) .

* When a browser supporting HTTP 1.1 indicates it can decompress HTML files, a server will compress them for transport across the Internet, providing a substantial aggregate savings in the amount of data that has to be transmitted. (Image files are already in a compressed format so this improvement applies only to HTML and other non-image data types.)
* HTTP/1.1 changed the semantics of the HTTP protocol to use connection keepalive by default. Meaning, unless told otherwise (via Connection: close header), the server should keep the connection open by default.

**Features:**

Persistent Connection.Pipelining,Expires,Entity tags, Max-Age, Max-Stale,Range Request,Chunked Encoding, Expect/Continue,Host Header.



**$> telnet website.org 80**

Connected to xxx.xxx.xxx.xxx

GET /index.html HTTP/1.1

Host: website.org

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_7\_4)... (snip)

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

Accept-Encoding: gzip,deflate,sdch

Accept-Language: en-US,en;q=0.8

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.3

Cookie: \_\_qca=P0-800083390... (snip)

HTTP/1.1 200 OK

Server: nginx/1.0.11

Connection: keep-alive

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

Via: HTTP/1.1 GWA

Date: Wed, 25 Jul 2012 20:23:35 GMT

Expires: Wed, 25 Jul 2012 20:23:35 GMT

Cache-Control: max-age=0, no-cache

Transfer-Encoding: chunked

100

<!doctype html>

(snip)

100

(snip)

0

GET /favicon.ico HTTP/1.1

Host: www.website.org

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_7\_4)... (snip)

Accept: \*/\*

Referer: http://website.org/

Connection: close

Accept-Encoding: gzip,deflate,sdch

Accept-Language: en-US,en;q=0.8

Accept-Charset: ISO-8859-1,utf-8;q=0.7,\*;q=0.3

Cookie: \_\_qca=P0-800083390... (snip)

HTTP/1.1 200 OK

Server: nginx/1.0.11

Content-Type: image/x-icon

Content-Length: 3638

Connection: close

Last-Modified: Thu, 19 Jul 2012 17:51:44 GMT

Cache-Control: max-age=315360000

Accept-Ranges: bytes

Via: HTTP/1.1 GWA

Date: Sat, 21 Jul 2012 21:35:22 GMT

Expires: Thu, 31 Dec 2037 23:55:55 GMT

Etag: W/PSA-GAu26oXbDi

(icon data)

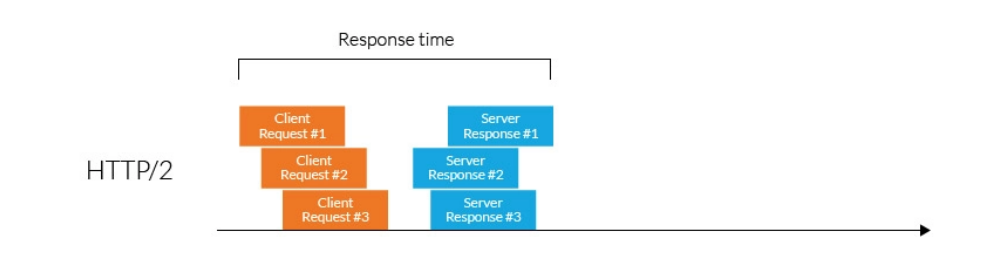
(connection closed)

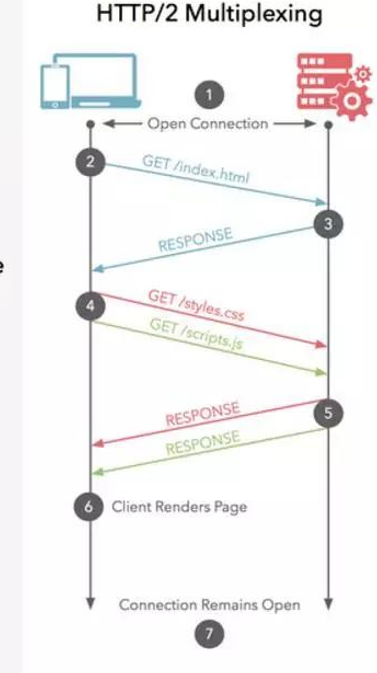
1. Request for HTML file, with encoding, charset, and cookie metadata
2. Chunked response for original HTML request
3. Number of octets in the chunk expressed as an ASCII hexadecimal number (256 bytes)
4. End of chunked stream response
5. Request for an icon file made on same TCP connection
6. Inform server that the connection will not be reused
7. Icon response, followed by connection close

**HTTP/2: Improving Transport Performance(2015)**

HTTP/2 will make our applications faster, simpler, and more robust

The primary goals for HTTP/2 are to reduce latency by enabling full request and response multiplexing, minimize protocol overhead via efficient compression of HTTP header fields, and add support for request prioritization and server push. To implement these requirements, there is a large supporting cast of other protocol enhancements, such as new flow control, error handling, and upgrade mechanisms, but these are the most important features that every web developer should understand and leverage in their applications





* **Binary protocols** – Binary protocols consume less bandwidth, are more efficiently parsed and are less error-prone than the textual protocols used by HTTP/1.1. Additionally, they can better handle elements such as whitespace, capitalization and line endings.
* **Multiplexing** – HTTP/2 is multiplexed, i.e., it can initiate multiple requests in parallel over a single TCP connection. As a result, web pages containing several elements are delivered over one TCP connection. These capabilities solve the head-of-line blocking problem in HTTP/1.1, in which a packet at the front of the line blocks others from being transmitted.
* **Header compression** – HTTP/2 uses header compression to reduce the overhead caused by TCP’s [slow-start](https://en.wikipedia.org/wiki/TCP_congestion_control#Slow_start) mechanism.
* **Server push** – HTTP/2 servers push likely-to-be-used resources into a browser’s cache, even before they’re requested. This allows browsers to display content without additional request cycles.
* **Increased security** – Web browsers only support HTTP/2 via encrypted connections, increasing user and application security.

# **HTTP3: the past, the present, and the future(2020)**

 HTTP/3 uses [QUIC](https://en.wikipedia.org/wiki/QUIC), a [transport layer](https://en.wikipedia.org/wiki/Transport_layer) [network protocol](https://en.wikipedia.org/wiki/Network_protocol) which uses [user space](https://en.wikipedia.org/wiki/User_space) [congestion control](https://en.wikipedia.org/wiki/Congestion_control) over the [User Datagram Protocol](https://en.wikipedia.org/wiki/User_Datagram_Protocol) (UDP). The switch to QUIC aims to fix a major problem of HTTP/2 called "[head-of-line blocking](https://en.wikipedia.org/wiki/Head-of-line_blocking)": because the parallel nature of HTTP/2's multiplexing is not visible to TCP's [loss recovery mechanisms](https://en.wikipedia.org/wiki/Packet_loss#Packet_recovery_for_reliable_delivery), a lost or reordered [packet](https://en.wikipedia.org/wiki/Network_packet) causes all active [transactions](https://en.wikipedia.org/wiki/Transaction_processing) to experience a stall regardless of whether that transaction was impacted by the lost packet. Because QUIC provides native multiplexing, lost packets only impact the streams where data has been lost.

instead of using TCP as the transport layer for the session, it uses [QUIC, a new Internet transport protocol](https://blog.cloudflare.com/the-road-to-quic/), which, among other things, introduces streams as first-class citizens at the transport layer. QUIC streams share the same QUIC connection, so no additional handshakes and slow starts are required to create new ones, but QUIC streams are delivered independently such that in most cases packet loss affecting one stream doesn't affect others. This is possible because QUIC packets are encapsulated on top of UDP datagrams.

