

# HTTP V1

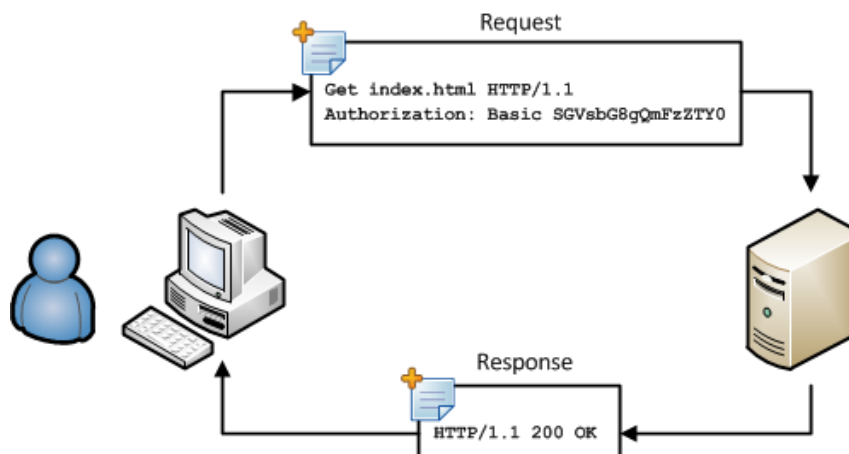
## The Beginnings of HTTP & The Internet

Our story begins in 1969, with a program called Advanced Research Projects Agency Network (ARPANET). ARPANET used packet switching and allowed multiple computers to communicate with each other on a single network. However, this was just a by-product. The original intention behind ARPANET was to design a time-sharing system that allowed research institutes to share their computer resources for effective utilization of processing power.

Before then, sometime during the 19th century, the seeds for the existence of the internet as we know it today had already been sown with the invention of electricity and the telegraph. With Morse sending the first telegraphic message in 1844 and the first cable being laid across Atlantic, the telegraph network infrastructure had spread its roots through continents and across oceans. In years to come, this would become the very foundation on which the internet was built. In 1973, Kahn and Cerf designed the TCP/IP protocol suite which was adopted by ARPANET a decade later, and from this point on, we witness the development of an interconnected network. The internet took a more recognizable form with the invention of the World Wide Web (that used HTTP as its underlying protocol) by Tim Berners-Lee and the Commercial Internet eXchange (CIX) that allowed a free exchange of TCP/IP traffic between ISPs.

## Evolution of HTTP

HTTP (Hypertext Transfer Protocol) is a set of rules that runs on top of the TCP/IP suite of protocols and defines how files are to be transferred between clients and servers on the world wide web.



## The Beginning of HTTP: Version 0.9 & 1.0

In the earliest phase (HTTP/0.9), the HTTP protocol did not use headers and only transmitted plain HTML files. It was a one-line protocol only supporting the GET method.

```

HTTP/0.9

$> telnet mysite.com 80
(Connection 1 Establishment - TCP Three-Way Handshake)
Connected to xxx.xxx.xxx.xxx

(Request)
GET /mypage.html

(Response in hypertext)
<HTML>
A very simple HTML page
</HTML>

(Connection 1 Closed - TCP Teardown)

```

As the need to exchange more than just plain HTML emerged along with the client and server applications becoming more mature, HTTP/1.0 (between 1991-1996) introduced several new features.

#### Key Features of HTTP/1.0:

- The concept of headers both for requests (from the client machine) as well as responses (from servers) was introduced. The use of headers such as GET, POST, HEAD added extended flexibility, none of which was possible with the earlier version.
- Version information was now included.
- It allowed a single request/response for every TCP connection.
- Status codes were used to indicate successful requests and to indicate transmission errors.
- The content-type header made it possible to send files other than plain HTML, including scripts and media.

#### Created for Added Security: HTTPS

In 1994, Netscape Communications created HTTPS (Hypertext Transfer Protocol Secure) to be used with SSL for its web browser, Netscape Navigator. The need for encrypted transmission channels emerged as the applications being designed shifted towards a more commercial market where advertisers, unknown individuals, and cybercriminals could have easy access to personal data. SSL evolved into TLS with TLS version 1.2 and 1.3 being used currently.

#### The Protocol Serving Netizens for Over 15 Years: HTTP/1.1

HTTP/1.1, the first standardized version of HTTP, was introduced in 1997. It presented significant performance optimizations (over HTTP/0.9 and HTTP/1.0) and transformed the way requests and responses were exchanged between clients and servers.

```
HTTP/1.1
(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: "547fa7e369ef56031dd3bffa2ace9fc0832eb251a"
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

[image content of 3077 bytes][Connection 1 Closed - TCP Teardown]
```

## Key Features of HTTP/1.1:

- It was no longer required for each connection to be terminated immediately after every request was served with a response; instead, with the keep-alive header, it was possible to have persistent connections. It allowed multiple requests/responses per TCP connection.
- The Upgrade header was used to indicate a preference from the client that made it possible to switch to a more preferred protocol if found appropriate by the server.
- HTTP/1.1 provided support for chunk transfers that allowed streaming of content dynamically as chunks and for additional headers to be sent after the message body. This enhancement was particularly useful in cases where values of a field remained unknown until the content had been produced. For example, when the content had to be digitally signed, it was not possible to do so before the entire content gets generated.
- Other features that reinforced its stability were introduced such as:
  - pipelining (the second request is sent before the response to the first is adequately served)
  - content negotiation (an exchange between client and server to determine the media type, it also provides the provision to serve different versions of a resource at the same URI)
  - cache control (used to specify caching policies in both requests and responses)

## The Protocol Designed to Speed Up Today's Complex Web pages: HTTP/2

At the beginning of 2010, Google introduced an experimental protocol, SPDY, which supported multiplexing (multiple requests/responses sent and received asynchronously over a single TCP connection) but as it gained traction IETF's HTTP Working Group came up with HTTP/2 in 2015, which is based on the SPDY protocol.

## Key Features of HTTP/2:

- It introduces the concept of a server push where the server anticipates the resources that will be required by the client and pushes them prior to the client making requests. The client retains the authority to deny the server push; however, in most cases, this feature adds a lot of efficiency to the process.
- Introduces the concept of multiplexing that interleaves the requests and responses without head-of-line blocking and does so over a single TCP connection.

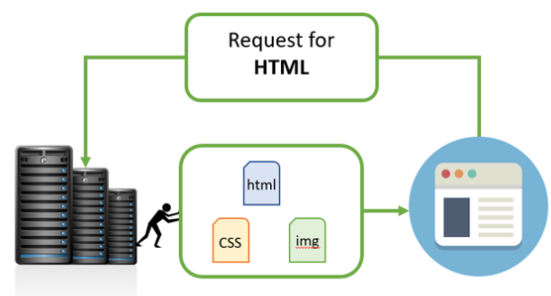


Figure 3: Server Push in HTTP/2

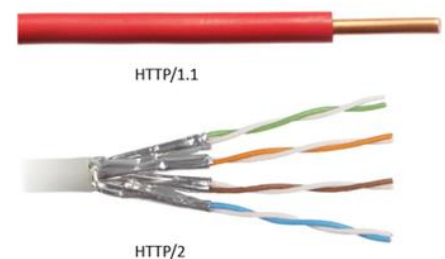
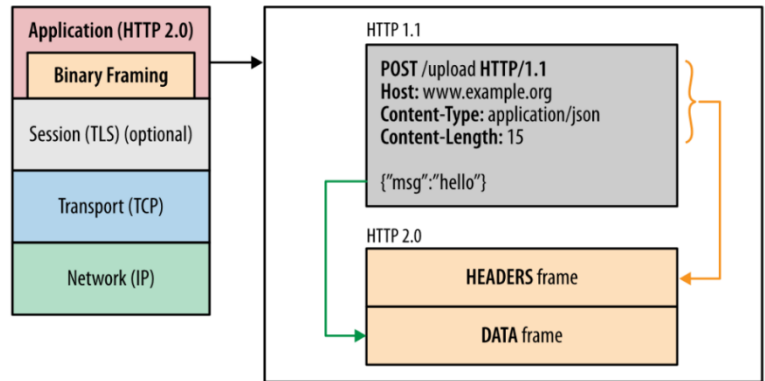


Figure 4: Multiplexing in HTTP/2

- It is a binary protocol i.e. only binary commands in the form of 0s and 1s are transmitted over the wire. The binary framing layer divides the message into frames that are segregated based on their type – Data or Header. This feature greatly increases efficiency in terms of security, compression and multiplexing.



## HTTP/2 Inside: binary

HTTP/2.0 request:

```

00 00 9D 01 25 00 00 00 01 00 00 00 00 B6 41 8A  ..%.  .A.
90 B4 9D 7A A6 35 5E 57 21 E9 82 00 84 B9 58 D3  ...z.5^W!  .X.
3F 85 61 09 1A 6D 47 87 53 03 2A 2F 2A 50 8E 9B  ?..a..mG.S.*/P..
D9 AB FA 52 42 CB 40 D2 5F A5 11 21 27 51 8B 2D  ...RB.@. _.'!Q.-
4B 70 DD F4 5A BE FB 40 05 DE 7A DA D0 7F 66 A2  Kp..Z..@...z...f.
81 B0 DA E0 53 FA D0 32 1A A4 9D 13 FD A9 92 A4  ...S...2.....
96 85 34 0C 8A 6A DC A7 E2 81 04 41 04 4D FF 6A  ..4..j.....A.M.j
43 5D 74 17 91 63 CC 64 B0 DB 2E AE CB 8A 7F 59  C]t..c.d.....Y
B1 EF D1 9F E9 4A 0D D4 AA 62 29 3A 9F FB 52 F4  ....J...b)...R.
F6 1E 92 B0 D3 AB 81 71 36 17 97 02 9B 87 28 EC  .......q6.....(
33 0D B2 EA EC B9

```

### HTTP/1.1 request:

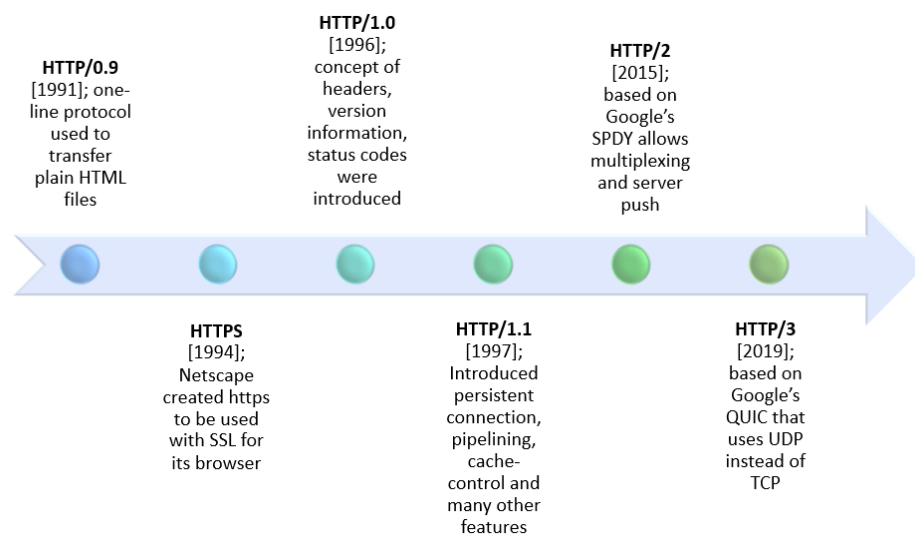
```

GET / HTTP/1.1
Host: demo.nginx.com
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Chrome/47.0.2518.0

```

- HTTP/2 uses HPACK header compression algorithm that is resilient to attacks like CRIME and utilizes static Huffman encoding.

HTTP/3, the next version in the series, is based on Google's QUIC which, unlike its precursors is a drastic shift to UDP. Given the gradual adoption rate of HTTP/2, HTTP/3 with its security challenges (that comes into play the moment we switch from TCP to UDP) is expected to face some difficulties.



## HTTP/1.x vs HTTP/2: A Comparative Study

HTTP2 Vs. HTTP1 is not a debate at all. HTTP2 is much faster and more reliable than HTTP1. HTTP1 loads a single request for every TCP connection, while HTTP2 avoids network delay by using multiplexing.



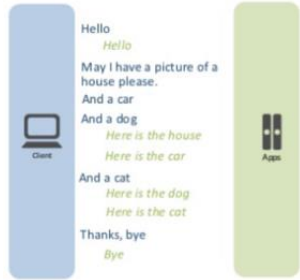
HTTP is a network delay sensitive protocol in the sense that if there is less network delay, then the page loads faster. However, an impressive increase in network bandwidth only slightly improves page load time. This is key to understanding the differences in performance efficiencies between the different versions of HTTP. Back in the day when people used dial up modems web pages were simple and it was the actual data transfer between the server and the client that contributed towards the largest chunk of the page load time. Today the actual downloading of resources from server takes a negligible portion of the total page load time due to the tremendous increase in bandwidth availability. It is the time taken to establish the TCP connection and making requests that impacts performance. It was initially recommended to use only two connections per hostname but today most browsers use six connections per hostname. When we talk about http vs http2 in terms of performance it is important to note that a lot of performance optimizations adopted by HTTP/1.1 introduced complexities in terms of developmental efforts as well as network congestion that HTTP/2 attempts to address.

The table below points out the differentiating factors between http2 vs http1:

**Header Compression** Headers are sent on every request leading to a lot of duplicate data being sent uncompressed across the wire. Header compression is included by default in HTTP/2 using HPACK. **Performance Optimization** Provides support for caching to deliver pages faster. Spriting, concatenating, inlining, domain sharding are some of the optimizations used as a workaround to the 'six connections per host' rule. Removes the need for unnecessary optimization hacks. **Protocol Type** Text based protocol that is in the readable form. It is a binary protocol (HTTP requests are sent in the form of 0s and 1s). Needs to be converted back from binary in order to read it. **Security** SSL is not required but recommended. Digest authentication used in HTTP1.1 is an improvement over HTTP1.0. HTTPS uses SSL/TLS for secure encrypted communication. Though security is still not mandatory, it is mostly encrypted (though it is not enforced) since almost all clients require traffic to be encrypted. It also has some minimum standards, such as minimum key size for encryption. TLS 1.2 etc.

**Header Compression** Headers are sent on every request leading to a lot of duplicate data being sent uncompressed across the wire. Header compression is included by default in HTTP/2 using HPACK. **Performance Optimization** Provides support for caching to deliver pages faster. Spriting, concatenating, inlining, domain sharding are some of the optimizations used as a workaround to the 'six connections per host' rule. Removes the need for unnecessary optimization hacks. **Protocol Type** Text based protocol that is in the readable form. It is a binary protocol (HTTP requests are sent in the form of 0s and 1s). Needs to be converted back from binary in order to read it. **Security** SSL is not required but recommended. Digest authentication used in HTTP1.1 is an improvement over HTTP1.0. HTTPS uses SSL/TLS for secure encrypted communication. Though security is still not mandatory, it is mostly encrypted (though it is not enforced) since almost all clients require traffic to be encrypted. It also has some minimum standards, such as minimum key size for encryption. TLS 1.2 etc.

**Header Compression** Headers are sent on every request leading to a lot of duplicate data being sent uncompressed across the wire. Header compression is included by default in HTTP/2 using HPACK. **Performance Optimization** Provides support for caching to deliver pages faster. Spriting, concatenating, inlining, domain sharding are some of the optimizations used as a workaround to the 'six connections per host' rule. Removes the need for unnecessary optimization hacks. **Protocol Type** Text based protocol that is in the readable form. It is a binary protocol (HTTP requests are sent in the form of 0s and 1s). Needs to be converted back from binary in order to read it. **Security** SSL is not required but recommended. Digest authentication used in HTTP1.1 is an improvement over HTTP1.0. HTTPS uses SSL/TLS for secure encrypted communication. Though security is still not mandatory, it is mostly encrypted (though it is not enforced) since almost all clients require traffic to be encrypted. It also has some minimum standards, such as minimum key size for encryption. TLS 1.2 etc.

Differentiator	HTTP/1.0	HTTP/1.1	HTTP/2
Year	1991	1997	2015
Key Features	<p>For every TCP connection there is only one request and one response.</p>  <p>HTTP/1.0</p>	<p>It supports connection reuse i.e. for every TCP connection there could be multiple requests and responses, and pipelining where the client can request several resources from the server at once. However, pipelining was hard to implement due to issues such as head-of-line blocking and was not a feasible solution.</p>  <p>HTTP/1.1</p>	<p>Uses multiplexing, where over a single TCP connection resources to be delivered are interleaved and arrive at the client almost at the same time. It is done using streams which can be prioritized, can have dependencies and individual flow control. It also provides a feature called server push that allows the server to send data that the client will need but has not yet requested.</p>  <p>HTTP/2</p>
Status Code	Can define 16 status codes; the error prompt is not specific enough.	Introduces a warning header field to carry additional information about the status of a message. Can define 24 status codes, error reporting is quicker and more efficient.	Underlying semantics of HTTP such as headers, status codes remains the same.
Authentication Mechanism	Uses basic authentication scheme which is unsafe since username and passwords are transmitted in clear text or base64 encoded.	It is relatively secure since it uses digest authentication, NTLM authentication.	Security concerns from previous versions will continue to be seen in HTTP/2. However, it is better equipped to deal with them due to new TLS features like connection error of type Inadequate_Security.
Caching	Provides support for caching via the If-Modified-Since header.	Expands on the caching support by using additional headers like cache-control, conditional headers like If-Match and by using entity tags.	HTTP/2 does not change much in terms of caching. With the server push feature if the client finds the resources are already present in the cache, it can cancel the pushed stream.
Web Traffic	HTTP/1.1 provides faster delivery of web pages and reduces web traffic as compared to HTTP/1.0. However, TCP starts slowly and with domain sharding (resources can be downloaded simultaneously by using multiple domains), connection reuse and pipelining, there is an increased risk of network congestion.		HTTP/2 utilizes multiplexing and server push to effectively reduce the page load time by a greater margin along with being less sensitive to network delays.