A Comparative Review of HTTP/1.1, HTTP/2 & HTTP/3

December 3, 2018

Nancy Mogire

HTTP: What, Why & How Summary

- WHAT → **The Hypertext Transfer Protocol** (HTTP)

- WHAT → “a **stateless application-level** protocol for distributed, collaborative, hypertext **information exchange** ”

- WHY → **generic interface for communication on the internet** without regard to types of resources being exchanged or implementation of communicating HTTP clients.

- WHY → **enables communication of web resources between different user agents** and servers.

- HOW → **message sender lets a receiver know the format of data** representation **so as they can be able to appropriately parse** the exchanged web resource.

HTTP: Sequence

User --url--> HTTP Client

Client -----HTTP Request-------> Server

Server <----->Data
storage

Client <----HTTP Response--------Server

User <--parse & display--Client

HTTP Participants and Protocol Model

Participants

- **Client/User Agent**
  - **Initiator of the connection** - e.g. browser, command shell, mobile app, or any other end-user—facing application
- **Server**
  - The target host in a connection request
- **Intermediaries**
  - Virtual and physical components in between the two principals of a connection: server and client. Include:
    - **Proxies** - functions **such as caching, authentication a**nd content filtering
    - **Tunnels**- Blind relays which do not change the message e.g. TLS through a firewall
    - **Gateways** - Routers

Client - Server Model

- **Client sends a request to a server and the server responds with** the requested web resource
- **Intermediaries may(often) exist** between the two

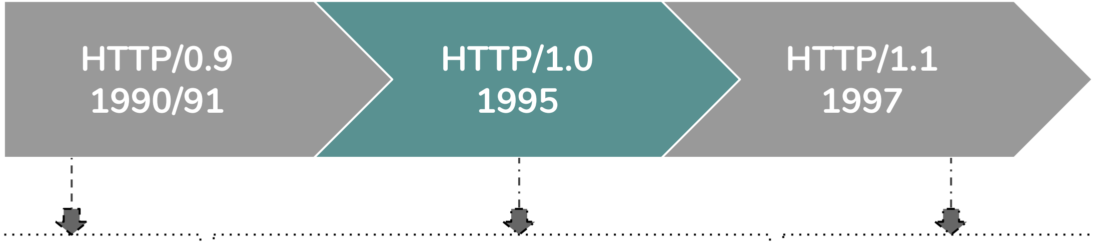
HTTP Components

- **The Resource**
  - Any **piece of data** identifiable by HTTP’s Uniform Resource Identifier(URI) scheme
  - E.g. **text, images, videos, scripts** ..etc
- **URI/URL**
  - **URL** → **resource identifier plus path** of getting to it i.e. its network location e.g. `https://en.wikipedia.org/wiki/Uniform_Resource_Identifier`
  - **URI** → **String that identifies** a specific resource e.g. `/wiki/Uniform_Resource_Identifier`
- **Request**
  - HTTP communication initiating message sent from client to server
- **Response**
  - Server reply to a request
- **Connection**
  - **Transport layer link between the client and the server.**
  - Protocols in/underlying a HTTP connection: **TCP, UDP**

HTTP Components

- ## Message
  - **Contents of the request or response**
  - Can be in the form of **plaintext characters** - HTTP/1.1 **or Frames** - HTTP/2 and HTTP/3
  - **Start Line**→ Request-Line/Status-Line, **Header**, **Message Body** → payload
- ## Message Header & Header Fields
  - Allows **client and server to exchange additional information with a request or response** → Information about resource involved in a connection or about the connection, or the participants
  - Carried **within the header fields e.g Content-Encoding, Content-Length**
  - Each field has a **name followed by a value separated by a colon**
  - Header Types:
  - Entity-header - about message body e.g. content length, Request header - about the requested resource or the client, Response header - about the response or the server, General header - about all except the entity
- ## Security
  - TLS - HTTPS

Version Overview, Background, Timeline



HTTP/0.9
1990/91

HTTP/1.0
1995

HTTP/1.1
1997

HTTP/0.9

- ➢ **Goal: Transfer html data online** - as **simplified prototype for full HTTP →** AKA, **One-line protocol**
- ➢ **Simple-request**: One line ASCII string e.g. **telnet google.com 80;** or **GET /mypage.html**
- ➢ **Simple - response:** ASCII character stream
- ➢ **HTML only**
- ➢ Over **TCP/IP**
- ➢ **Single Exchange** - Close Connection

[https://hpbn.co/brief-history-of-http/], [Mozilla]

HTTP/1.0

- ➢ **Goals: Add functionality → transfer more than just HTML**; **provide metadata** on request & response; **format data** in internet mail format.
- ➢ **Added: headers** with header fields containing req/resp metadata e.g. version no.
- ➢ Over **TCP/IP**
- ➢ **Single Exchange per Connection** & close
- ➢ **Other content** types e.g img
- ➢ **Other capabilities**: e.g. content encoding & caching

[https://hpbn.co/brief-history-of-http/]

HTTP/1.1

- ➢ **Goals: Resolve ambiguities**; **performance optimization**
- ➢ Added: **Connection Persistence by default**;
- ➢ **Chunked transfer encoding(**message broken down and transferred in chunks- supports dynamic content generation)
- ➢ **Request pipelining**(send multiple requests without waiting for each response first - good use of persistent connection i.e. latency reduction)
- ➢ **Expanded caching** functionality

[https://hpbn.co/brief-history-of-http/] , [fir3net]

Version Overview, Background, Timeline

HTTP/2
2015

HTTP/3
2018/19

- ➢ Goal: Improve Performance from version 1 i.e. reduce latency; minimize protocol overhead; Enable request prioritization; Enable server push messages; Enhance other functions e.g. flow control & error handling.
- ➢ Left all HTTP semantics intact
- ➢ Changed: data formatting & transportation mechanism → ASCII to binary format
- ➢ Added: Binary framing layer & message framing; Transfers in bidirectional streams; Multiplexing(break msg into frames→ interleave in streams→reassemble at end) → allows parallel processing; stream prioritization; one connection per origin; server push; header compression
- ➢ Runs over TCP

[developers.google]

- ➢ **Goals: Improve performance on transport layer** and **solve application layer problem**s
- ➢ Leaves HTTP core intact
- ➢ Combines functionalities of **TCP+TLS+HTTP2 over UDP**
- ➢ **Additions include: Faster connection establishment**(Client uses cached server credentials from prev connection to send encrypted request right after hello → **one-way handshake** to start subseq) ; **Improved congestion control; Multiplexing with no head-of-line blocking**-(lost packets affect only that stream while streams without loss can go on);

[chromium.org]

Summing the HTTP Objectives

- **Correct Output**
  - Message **version specifies format for parsing**→ correct retrieval
  - Message **ordering for correct request/response matching** - e.g. head-of-line blocking or message IDs
  - **Server Push Messages -** responses needed to parse the ones requested
- **Reliable Delivery**
  - TCP reliability mechanisms, **Flow control**, **Congestion Contro**l, **Prioritization**
- **Fast Delivery** → Latency reduction
  - frame based transfer, compression, multiplexing, concurrency
- **Connection Management**
  - Set up, use, multiple uses(persistence) tear down
- **Resource management**
  - Reduce **header overhead** → **compression**, **session re-use -persistence**, **multiplexing - parallel processing**
- **Security**
  - **Confidentiality & Integrity** - **data encryption** in SHTTP, then **connection encryption**, then as **packet encryption** in version 3

Comparative View of Last 3 Versions

HTTP/1.1 , HTTP/2 , HTTP/3

- ➢ Header
- ➢ Message
- ➢ Transmission Format
- ➢ Transport and Security Mechanisms
- ➢ Connection Management: Establishment, Persistence, Closure
- ➢ Message Ordering, Multiplexing & Concurrency
- ➢ Flow Control, Congestion Control, Prioritization
- ➢ Cross-Version Compatibility

Header:

	Header Format, Compression and Transmission
HTTP/1.1	● ASCII /Plaintext ● No compression ● Header field names - case insensitive
HTTP/2	● HPACK compression of header into block ● Breaks header block into frames for transmission ● Huffman encoding + Static table of commonly used header fields + Dynamic table with fields specific to the session ● All field names lower case and request line is split into separate pseudo-header fields :method, :scheme, :authority, and :path.
HTTP/3	● Frames ● Lower case field names plus pseudoheaders as in version 2 ● QPACK compression ● Huffman encoding + Static table of commonly used header fields + Dynamic table with fields specific to the session

Message Transmission Format - Framing

	Message Format
HTTP/1.1	- ASCII /Plaintext - Header Section - Message Body - Separated by empty line
HTTP/2	- Frames - Headers Frame - Data Frame - Payload
HTTP/3	- Frames - Header block - message headers - Payload body - Data Frames - Optional Trailer Block - Additional Header information - dynamically generated while message sent

Message: Examples

GET REQUEST: HTTP/1.1 to HTTP/2

```
GET /resource HTTP/1.1          HEADERS
Host: example.org        ==>   + END_STREAM
Accept: image/jpeg             + END_HEADERS
                                 :method = GET
                                 :scheme = https
                                 :path = /resource
                                 host = example.org
                                 accept = image/jpeg
```

POST REQUEST HTTP/1.1 to HTTP/2

```
POST /resource HTTP/1.1         HEADERS
Host: example.org        ==>     - END_STREAM
Content-Type: image/jpeg         - END_HEADERS
Content-Length: 123               :method = POST
                                 :path = /resource{binary data
                                 :scheme = https

                                  CONTINUATION
                                  + END_HEADERS
                                   content-type = mage/jpeg
                                   host = example.org
                                   content-length = 123

                                  DATA
                                   + END_STREAM
                                   {binary data}
```

Message: Examples

Example of HTTP/3 Handshake:

```
Client                                                  Server

Initial[0]: CRYPTO[CH] ->

                                 Initial[0]: CRYPTO[SH] ACK[0]
                      Handshake[0]: CRYPTO[EE, CERT, CV, FIN]
                                 <- 1-RTT[0]: STREAM[1, "..."]

Initial[1]: ACK[0]
Handshake[0]: CRYPTO[FIN], ACK[0]
1-RTT[0]: STREAM[0, "..."], ACK[0] ->

                          1-RTT[1]: STREAM[55, "..."], ACK[0]
                                     <- Handshake[1]: ACK[0]
```

Example of 1-RTT Handshake - source:[12]

- After the handshake, HTTP/2 message can be sent

Transport and Security Mechanisms

	Transport Mechanism	**Security**
HTTP/1.1	● TCP Session	● Transport Layer Security(TLS) ○ TLS 1.2 ○ Previously → SSL ● Hypertext Transfer Protocol Secure (HTTPS) ● Bi-directional encryption between client and server
HTTP/2	● TCP Session	● Same as in HTTP/1.1 i.e. optionally runs over TLS for encrypted connection
HTTP/3	● UDP Packet	● Packet level protection ● Runs TLS 1.3 at the transport layer ● Protects packets with keys from the TLS handshake under AEAD algorithm - Authentication Encryption with Associated Data (AEAD) ● All QUIC packets except Version Negotiation and Retry packets are protected with AEAD

Connection Management: Estab, Persistence & Closure

	Connection Establishment	Persistence	Closure
HTTP/1.1	● Client initiates TCP connection ● Multiple simultaneous TCP connections allowed	● Persistent By default ● Recipient determines the status based on protocol version of tmost recently received message or on connection header	● "close" connection header option to signal closing init ● sender or receiver ● Premature closing, re-open automatically, once
HTTP/2	● Client initiates TCP connection ● Single connection per host-port pair for each server, ● Multiple streams can be run	● Persistent By default ● Can be closed if idle	● Connection can be closed if idle ● Endpoints should send GOAWAY message to signal initiating graceful closing ● Can close without GOAWAY if misbehaving peer

Connection Management: Estab, Persistence & Closure

	Connection Establishment	**Persistence**	**Closure**
HTTP/3	● Quic Hello Handshake ○ Client sends ClientHello Msg, gets server hello with encryption credentials and sends settings frame e.g. Maximum stream ID ○ Then create streams by sending data	● Persistent By default ● use QUIC PING frames to keep it open ● Closes if idle	● Client can initiate close by not sending new messages i.e. staying idle ● Server sends GOAWAY message, clears any remaining requests it has and starts the shutdown

Message Ordering, Multiplexing & Concurrency

	Message Ordering	**Multiplexing**	**Concurrency**
HTTP/1.1	● Queued ● Head-of-line blocking i.e. one request serviced at a time	● Not Multiplexed	● Parallel sessions can be run via parallel independent but simultaneous connections
HTTP/2	● Absolute ordering of frames spanning across all streams ● Each stream has integer identifier ● Frame sending order determines receive order.	● Multiplexed across streams	● Several streams can be open concurrently and frames from multiple streams can be interleaved ● "stream" is an independent, bidirectional sequence of frames exchanged between the client and server

Message Ordering, Multiplexing & Concurrency

	Message Ordering	Multiplexing	Concurrency
HTTP/3	- Separate frame ordering per each stream - Guarantees in-order delivery within each stream but not across all streams	- Allows multiplexing with no head-of-line blocking - Per-stream flow control plus connection-wide flow control - Messages on different streams do not block each other i.e. if packet is lost on one stream, other streams can go on.	- Multiple concurrent streams can be open - Parallel - due to correct out-of-order stream delivery.

Flow Control, Congestion Control, Prioritization

	Flow Control	Congestion Control	Prioritization
HTTP/1.1	● No flow control ● Relies on TCP	● No congestion control	● No prioritization
HTTP/2	● Flow control provided for entire connection i.e. across streams but not per stream ● Only data frames subject to flow control ● Any algorithm	● Provided by TCP	● Client can assign priority status for a new stream via the HEADERS frame ● Can update it later using a PRIORITY frame
HTTP/3	● Per-Stream Flow Control in addition to connection-level flow control ● Advertises max data to be received on each stream and aggregate buffer size for all	● Uses mechanism similar to TCP NewReno (RFC6582): Congestion avoidance → additive increase multiplicative decrease (AIMD)	● Using PRIORITY frames sent on control streams ● Can also be done by assigning others as dependents

Cross-Version Compatibility - Upward & Downward

	Upgrading	**Read/Reply Lower Version**
HTTP/1.1	● Start a connection using HTTP/1.1 ● Request upgrade to HTTP/2 using upgrade header ● Can only upgrade to h2c → HTTP/2 Cleartext" ● Initiated by client but a server can require it	● compatible with HTTP/0.9, 1.0 ● can recognize the request line and any valid request ● respond appropriately with a message in the same version used by the client. ● recognize the status line in HTTP/1.0
HTTP/2	● No upgrade mechanism	● Fully compatible with HTTP/1.1,
HTTP/3	● N/A	● HTTP/3 is compatible with previous versions

Conclusion

- Hypertext Transfer Protocol (HTTP) has undergone **numerous changes since it was first adopted** → Now multiple versions of HTTP exist
- **Each version filling in gaps** that existed in the previous one
- **HTTP/0.x got the core concept up** and running—a stateless application-level protocol for distributed, collaborative, hypertext information exchange.
- **HTTP/1.x solved details** such as the **need for persistent connections** and name-based virtual hosts. Security Introduced here SSL → TLS
- **HTTP/2 introduced binary message framing, multiplexing** and other extensions to optimize performance
- **HTTP/3—the latest version— adds per-stream multiplexing and flow control plus packet-level security** → adds reliability, reduces latency and improves security

References

https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

https://www.w3.org/Protocols/HTTP/1.0/spec.html#Purpose

https://tools.ietf.org/html/rfc7230

https://tools.ietf.org/html/rfc7231

https://tools.ietf.org/html/rfc1945

https://hpbn.co/http2/

https://http2.github.io/

https://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00

https://quicwg.org/base-drafts/draft-ietf-quic-http.html

https://en.wikipedia.org/wiki/Text-based_protocol

https://en.wikipedia.org/wiki/Binary_protocol

https://tools.ietf.org/html/draft-ietf-quic-http-16#page-4

https://tools.ietf.org/html/draft-ietf-quic-http-16#page-10

https://www.w3.org/Protocols/rfc2616/rfc2616-sec4.html#sec4.4

https://tools.ietf.org/html/draft-ietf-quic-qpack-03

https://tools.ietf.org/html/rfc7540

https://tools.ietf.org/html/draft-ietf-quic-transport-16

https://tools.ietf.org/html/draft-ietf-quic-transport-16#ref-QUIC-TLS

https://www.ietf.org/rfc/rfc2660.txt

https://www.pcmag.com/encyclopedia/term/51302/shttp

https://hpbn.co/brief-history-of-http/

https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP

https://www.fir3net.com/Networking/Protocols/http-caching-http-1-0-vs-http-1-1.html

https://developers.google.com/web/fundamentals/performance/http2/#push_promise_101

https://www.chromium.org/quic

https://tools.ietf.org/html/rfc7540#section-5

https://tools.ietf.org/html/draft-ietf-quic-transport-13#section-4.4.1

https://developers.google.com/web/fundamentals/performance/http2/#streams_messages_and_frames

https://quicwg.org/base-drafts/draft-ietf-quic-http.html#rfc.section.5.3

https://tools.ietf.org/html/draft-tsvwg-quic-protocol-00

https://developer.mozilla.org/en-US/docs/Web/HTTP/Protocol_upgrade_mechanism