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
    storage
Server <-----Data
    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 and 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
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

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

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/], [Mozilla]

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

Goals: Improve performance on transport layer and solve application layer problems
- 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);

[developers.google]

[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 Control, 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 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

<table>
<thead>
<tr>
<th></th>
<th>Message Format</th>
</tr>
</thead>
</table>
| **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 |
GET REQUEST: HTTP/1.1 to HTTP/2
GET /resource HTTP/1.1
Host: example.org
Accept: image/jpeg
|
POST REQUEST HTTP/1.1 to HTTP/2
POST /resource HTTP/1.1
Host: example.org
Content-Type: image/jpeg
Content-Length: 123
|
| CONTINUATION |
| content-type = image/jpeg |
| host = example.org |
| content-length = 123 |
| DATA |
| {binary data} |
### Example of HTTP/3 Handshake:

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial[0]: CRYPTO[CH] -&gt;</td>
<td>Initial[0]: CRYPTO[SH] ACK[0]</td>
</tr>
<tr>
<td>Handshake[0]: CRYPTO[EE, CERT, CV, FIN]</td>
<td>&lt;- 1-RTT[0]: STREAM[1, &quot;...&quot;]</td>
</tr>
<tr>
<td>Initial[1]: ACK[0]</td>
<td></td>
</tr>
<tr>
<td>Handshake[0]: CRYPTO[FIN], ACK[0]</td>
<td>1-RTT[0]: STREAM[0, &quot;...&quot;], ACK[0] -&gt;</td>
</tr>
<tr>
<td>1-RTT[1]: STREAM[55, &quot;...&quot;], ACK[0]</td>
<td>&lt;- Handshake[1]: ACK[0]</td>
</tr>
</tbody>
</table>

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

- After the handshake, HTTP/2 message can be sent
<table>
<thead>
<tr>
<th>Transport Mechanism</th>
<th>Security</th>
</tr>
</thead>
</table>
| 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 |
<table>
<thead>
<tr>
<th>Connection Management: Estab, Persistence &amp; Closure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection Establishment</strong></td>
</tr>
</tbody>
</table>
| **HTTP/1.1** | • Client initiates TCP connection  
• Multiple simultaneous TCP connections allowed | • Persistent By default  
• Recipient determines the status based on protocol version of most 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

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

- **HTTP/3**

  - **Connection Establishment**
    - 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
  - **Closure**
    - 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
  - **Persistence**
    - Persistent By default
      - use QUIC PING frames to keep it open
      - Closes if idle
<table>
<thead>
<tr>
<th></th>
<th>Message Ordering</th>
<th>Multiplexing</th>
<th>Concurrency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HTTP/1.1</strong></td>
<td>● Queued</td>
<td>● Not Multiplexed</td>
<td>● Parallel sessions can be run via parallel</td>
</tr>
<tr>
<td></td>
<td>● Head-of-line</td>
<td></td>
<td>independent but simultaneous connections</td>
</tr>
<tr>
<td></td>
<td>blocking i.e.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>one request</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>serviced at a</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>time</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>HTTP/2</strong></td>
<td>● Absolute</td>
<td>● Multiplexed across</td>
<td>● Several streams can be open concurrently and</td>
</tr>
<tr>
<td></td>
<td>ordering of</td>
<td>streams</td>
<td>frames from multiple streams can be interleaved</td>
</tr>
<tr>
<td></td>
<td>frames spanning</td>
<td></td>
<td>● &quot;stream&quot; is an independent,</td>
</tr>
<tr>
<td></td>
<td>across all</td>
<td></td>
<td>bidirectional sequence of frames exchanged</td>
</tr>
<tr>
<td></td>
<td>streams</td>
<td></td>
<td>between the client and server</td>
</tr>
<tr>
<td></td>
<td>● Each stream</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>has integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>identifier</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Frame sending</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>order determines</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>receive order.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message Ordering</td>
<td>Multiplexing</td>
<td>Concurrency</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><strong>HTTP/3</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>● Separate frame ordering per each stream</td>
<td>● Allows multiplexing with no head-of-line blocking</td>
<td>● Multiple concurrent streams can be open</td>
<td></td>
</tr>
<tr>
<td>● Guarantees in-order delivery within each stream but not across all streams</td>
<td>● Per-stream flow control plus connection-wide flow control</td>
<td>● Parallel - due to correct out-of-order stream delivery.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>● Messages on different streams do not block each other i.e. if packet is lost on one stream, other streams can go on.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Flow Control</td>
<td>Congestion Control</td>
<td>Prioritization</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td><strong>HTTP/1.1</strong></td>
<td>• No flow control</td>
<td>• No congestion control</td>
<td>• No prioritization</td>
</tr>
<tr>
<td></td>
<td>• Relies on TCP</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>HTTP/2</strong></td>
<td>• Flow control provided for entire connection i.e. across streams but not per</td>
<td>• Provided by TCP</td>
<td>• Client can assign priority</td>
</tr>
<tr>
<td></td>
<td>stream</td>
<td></td>
<td>status for a new stream via the</td>
</tr>
<tr>
<td></td>
<td>• Only data frames subject to flow control</td>
<td></td>
<td>HEADERS frame</td>
</tr>
<tr>
<td></td>
<td>• Any algorithm</td>
<td></td>
<td>• Can update it later using a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PRIORITY frame</td>
</tr>
<tr>
<td><strong>HTTP/3</strong></td>
<td>• Per-Stream Flow Control in addition to connection-level flow control</td>
<td>• Uses mechanism similar to</td>
<td>• Using PRIORITY frames sent on</td>
</tr>
<tr>
<td></td>
<td>• Advertises max data to be received on each stream and aggregate buffer size</td>
<td>TCP NewReno (RFC6582):</td>
<td>control streams</td>
</tr>
<tr>
<td></td>
<td>for all</td>
<td>Congestion avoidance</td>
<td>• Can also be done by assigning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>→ additive increase</td>
<td>others as dependents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>multiplicative decrease</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(AIMD)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Upgrading</td>
<td>Read/Reply Lower Version</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------</td>
<td>---------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>HTTP/1.1</strong></td>
<td>● Start a connection using HTTP/1.1&lt;br&gt;● Request upgrade to HTTP/2 using upgrade header&lt;br&gt;● Can only upgrade to h2c → HTTP/2 Cleartext&quot;&lt;br&gt;● Initiated by client but a server can require it</td>
<td>● compatible with HTTP/0.9, 1.0&lt;br&gt;● can recognize the request line and any valid request&lt;br&gt;● respond appropriately with a message in the same version used by the client.&lt;br&gt;● recognize the status line in HTTP/1.0</td>
<td></td>
</tr>
<tr>
<td><strong>HTTP/2</strong></td>
<td>● No upgrade mechanism</td>
<td>● Fully compatible with HTTP/1.1,</td>
<td></td>
</tr>
<tr>
<td><strong>HTTP/3</strong></td>
<td>● N/A</td>
<td>● HTTP/3 is compatible with previous versions</td>
<td></td>
</tr>
</tbody>
</table>
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://www.w3.org/Protocols/HTTP/1.0/spec.html#Purpose
- https://hpbn.co/http2/
- https://http2.github.io/
- https://www.w3.org/Protocols/rfc2616/rfc2616-sec4.html#sec4.4
- https://hpbn.co/brief-history-of-http/
- https://www.chromium.org/quic
- 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