HTTP/1 → HTTP/2 → HTTP/3 with QUIC

Kristjon Ciko

IN3230/IN4230: Computer Networks

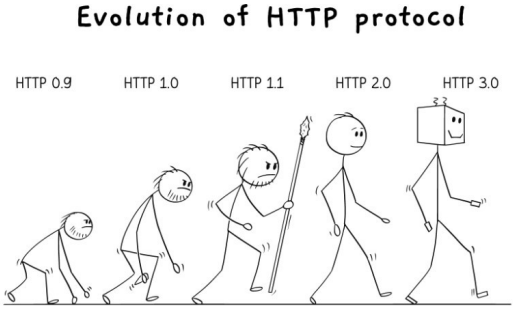
[A few slides from previous IN5150 lectures by Safiqul Islam and Kristian A. Hiorth]

[Many pictures from High Performance Browser Networking | Copyright © 2013 Ilya Grigorik]

October 22, 2026

HTTP/1 → HTTP/2 → HTTP/3 with QUIC

Kristjon Ciko
Department of Informatics
University of Oslo



[Photo: LinkedIn]

Evolution of HTTP: from legacy HTTP to HTTP/3 with QUIC



Legacy HTTP



HTTP/2



HTTP/3 and QUIC

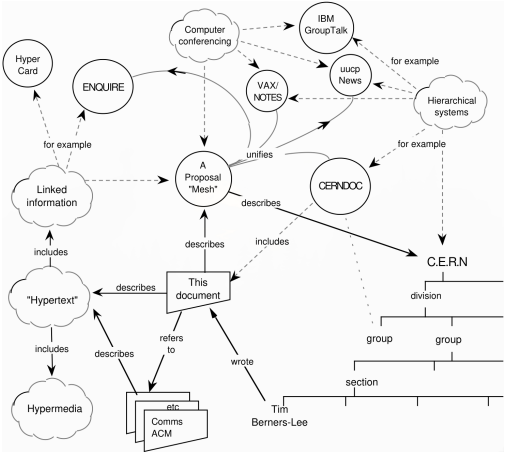
History

Legacy HTTP

HTTP/2

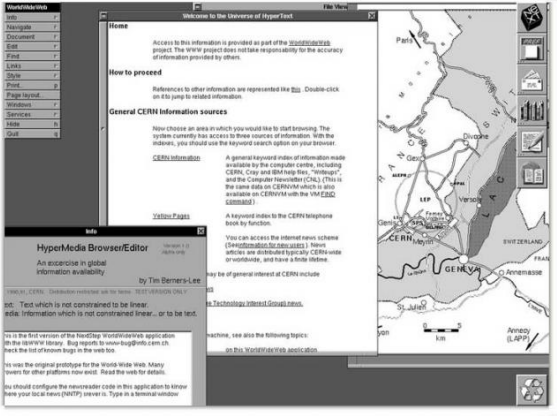
HTTP/3 and QUIC

In March 1989, Sir Tim Berners-Lee, while working at CERN, wrote a proposal to develop a distributed information system.



[URL: `https://cds.cern.ch/record/369245/files/dd-89-001.pdf`]

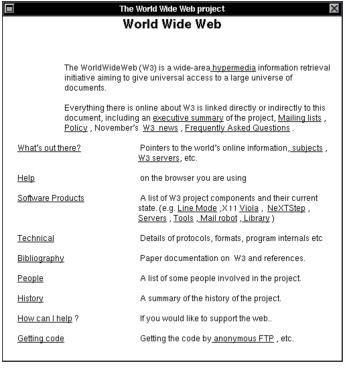
In December 1990, the world's first browser/editor, website and server go live at CERN.



[URL: `https://info.cern.ch/`]

In December 1990, the world's first browser/editor, website and server go live at CERN.

- `https://info.cern.ch/` — the first website



[`https://worldwideweb.cern.ch/`]

In December 1990, the world's first browser/editor, website and server go live at CERN.

- `https://info.cern.ch/` — the first website
- the first web server running on a NeXT machine



[`https://cds.cern.ch/record/42413`]

In December 1990, the world's first browser/editor, website and server go live at CERN.

- `https://info.cern.ch/` — the first website
- the first web server running on a NeXT machine
- the first web page: World Wide Web project



[cds.cern.ch/images/CERN-HOMEWEB-PHO-2019-004-1]

Word **W**ide **W**eb (**WWW**): a distributed collection of pages, built on four core building blocks.

Word Wide Web (WWW): a distributed collection of pages, built on four core building blocks.

1. HyperText Markup Language (HTML)

A textual format to represent hypertext documents

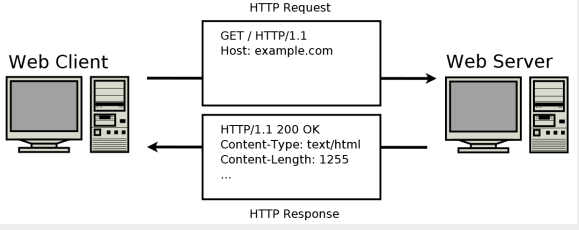
```
<!DOCTYPE html>
<html>
<body>
<h1>IN5150 - Recent Advancements in Internet Protocols</h1>
<p><a href="https://www.uio.no/">Visit UiO.no!</a></p>
</body>
</html>
```

Word Wide Web (WWW): a distributed collection of pages, built on four core building blocks.

1. HyperText Markup Language (HTML)

2. HyperText Transfer Protocol (HTTP)

A protocol to exchange the documents



[www.packtpub.com]

Word **W**ide **W**eb (**WWW**): a distributed collection of pages, built on four core building blocks.

1. HyperText Markup Language (HTML)

2. HyperText Transfer Protocol (HTTP)

3. Web Server

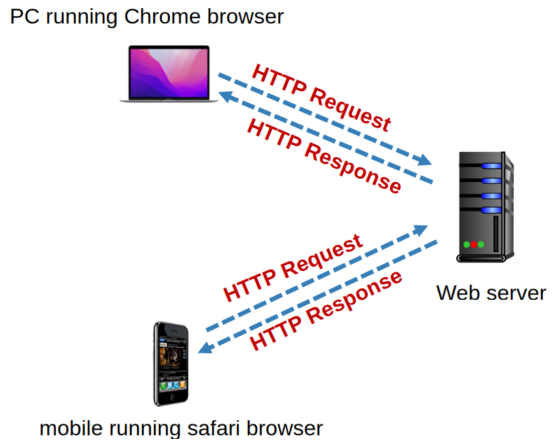
Stores and provides access to web pages

Word **W**ide **W**eb (**WWW**): a distributed collection of pages, built on four core building blocks.

1. HyperText Markup Language (HTML)
2. HyperText Transfer Protocol (HTTP)
3. Web Server
4. **Web Client**

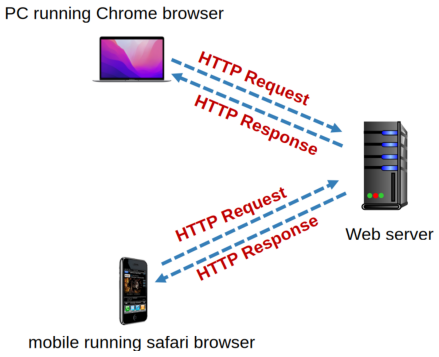
   Sends HTTP requests to servers and renders the web page responses

HyperText Transfer Protocol (HTTP)



HyperText Transfer Protocol (HTTP)

- Application layer protocol
- Client-server architecture
  - Client requests the page
  - Server sends objects in response
- Synchronous request-response protocol
- Reliable delivery (Runs on top of TCP)
- Stateless protocol
  - But not sessionless (HTTP cookies)



Evolution of HTTP

HTTP/0.9 (1991)

Evolution of HTTP

HTTP/0.9 (1991)

- One-line requests "GET /my-page.html"
- No HTTP headers
- No status or error codes
- Only HTML files could be transmitted
- Connection terminated upon document transfer completion

Evolution of HTTP

HTTP/0.9 (1991)

```
# telnet itcorp.com 80
Trying 134.173.42.59...
Connected to itcorp.com.
Escape character is '^]'.
GET /
<HTML>
<HEAD>
<TITLE>Nothing here</TITLE>
</HEAD>
<BODY>
<H1>Nothing here</H1>
There is nothing to see here.  Nor will there be.
</BODY>
</HTML>
Connection closed by foreign host.
```

Evolution of HTTP

HTTP/0.9 (1991)

HTTP/1.0 - RFC1945 (1992)

Evolution of HTTP

HTTP/1.0 - RFC1945 (1992)

- Versioning information was sent within each request.
- A status code line was also sent at the beginning of a response.
- The concept of HTTP headers was introduced for both requests and responses.
- Documents other than plain HTML files could be transmitted thanks to the Content-Type header.

Evolution of HTTP

HTTP/0.9 (1991)

HTTP/1.0 - RFC1945 (1992)

HTTP/1.1 - RFC2068, RFC9112 (1996)

Evolution of HTTP

HTTP/1.1 - RFC2068, RFC9112 (1996)

- A connection could be reused, which saved time.
- Pipelining was added.
- Chunked responses were also supported.
- Content negotiation, including language, encoding, and type, was introduced.

Evolution of HTTP

HTTP/0.9 (1991)

HTTP/1.0 - RFC1945 (1992)

HTTP/1.1 - RFC2068, RFC9112 (1996)

HTTP/2.0 - RFC9113 (2015)

Evolution of HTTP

HTTP/2.0 - RFC9113 (2015)

- It's a binary protocol rather than a text protocol.
- It's a multiplexed protocol. Parallel requests can be made over the same connection.
- It compresses headers.

Evolution of HTTP

- HTTP/0.9 (1991)
- HTTP/1.0 - RFC1945 (1992)
- HTTP/1.1 - RFC2068, RFC9112 (1996)
- HTTP/2.0 - RFC9113 (2015)
- **HTTP/3 - RFC9114 (2022)**

Evolution of HTTP

HTTP/3 - RFC9114 (2022)

- HTTP over QUIC (over UDP)
- Stream multiplexing and per-stream flow control
- Low latency connection establishment

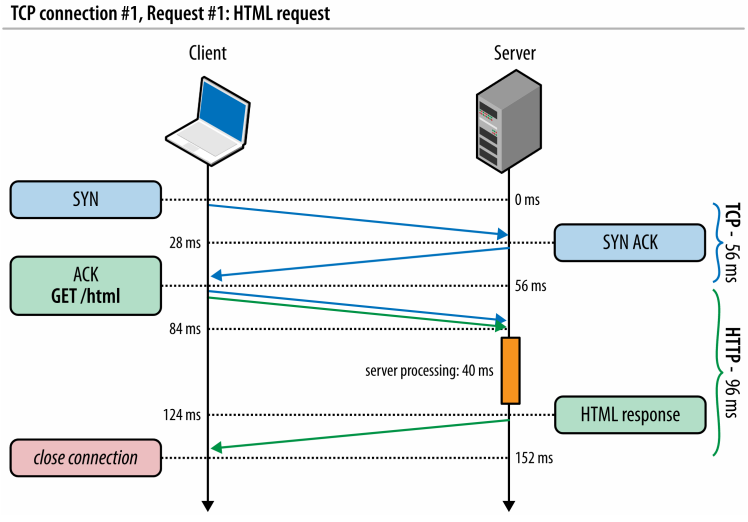
History

Legacy HTTP

HTTP/2

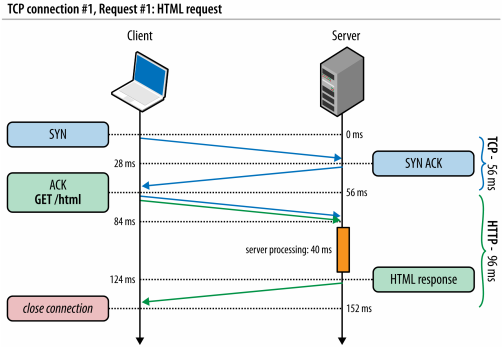
HTTP/3 and QUIC

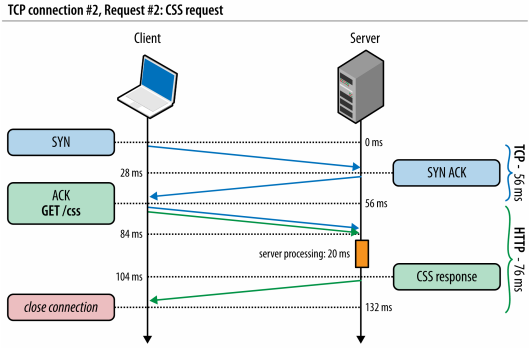
HTTP/1.0: one request ⇔ one TCP connection



[High Performance Browser Networking | © Ilya Grigorik]

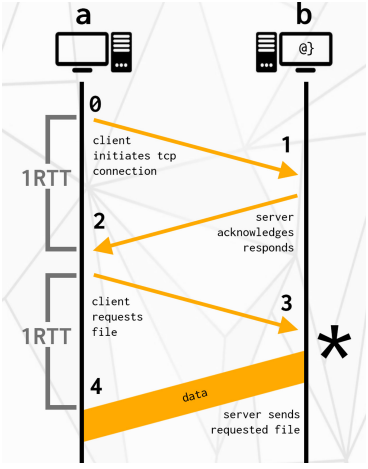
HTTP/1.0: one request ⇔ one TCP connection



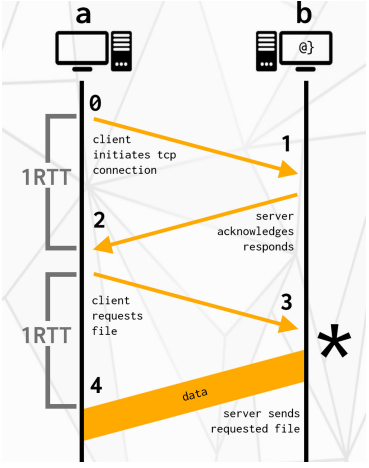


[High Performance Browser Networking | © Ilya Grigorik]

Requiring a new TCP connection per request imposes a significant performance penalty on HTTP/1.0.



Requiring a new TCP connection per request imposes a significant performance penalty on HTTP/1.0.



- one RTT to establish the TCP connection
- one RTT for the HTTP request and first byte of the HTTP response
- object/file transmission time
- HTTP/1.0 response time = 2 RTT + transmission time

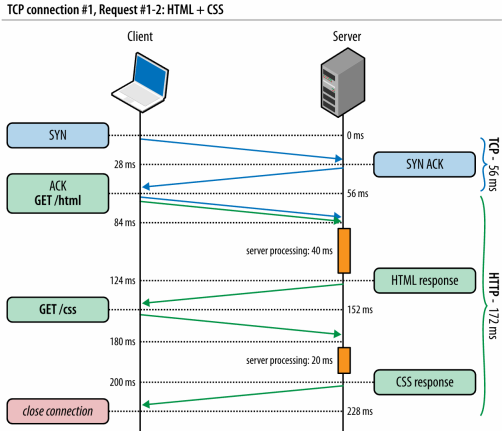
Case study from WWDC 2012: Delivering 3× Performance Improvement for iTunes Users with HTTP/1.1

Apple engineers saw a 300% performance improvement for users on slower networks once they made better reuse of existing TCP connections within iTunes, via **HTTP keepalive** and **pipelining**!

⇒ check out WWDC archives on iTunes and look for Joshua Graessley's talk:
Session 706: Networking Best Practices

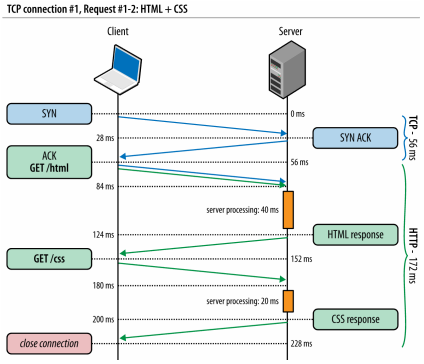
HTTP/1.1: Benefits of Keepalive Connections

HTTP/1.1: Benefits of Keepalive Connections



[High Performance Browser Networking | © Ilya Grigorik]

HTTP/1.1: Benefits of Keepalive Connections



[High Performance Browser Networking | © Ilya Grigorik]

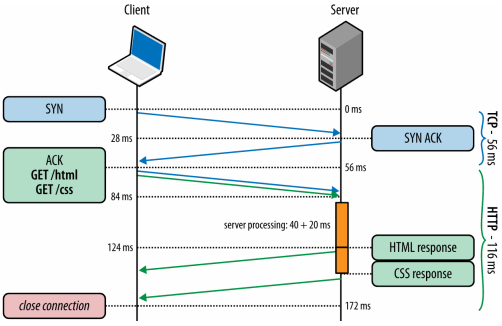
- No need for another TCP three-way handshake
- Avoids another round of TCP slow-start
- Saves a full roundtrip of network latency for each request
- Total latency savings for N requests is (N-1) x RTT

HTTP/1.1 Pipelining

Persistent HTTP lets clients reuse existing TCP connection for multiple requests.

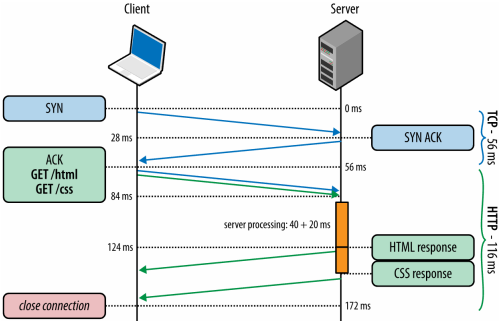
However, each request must wait until the previous response is fully received before it can be sent, enforcing a strict order.

HTTP/1.1 Pipelining with Server-Side FIFO queue



[High Performance Browser Networking | © Ilya Grigorik]

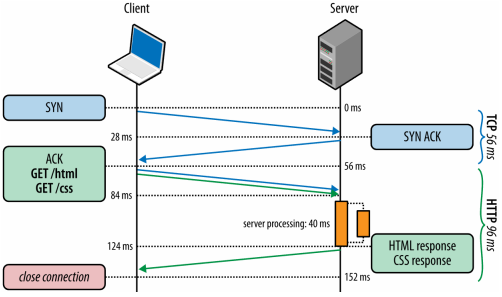
HTTP/1.1 Pipelining with Server-Side FIFO queue



[High Performance Browser Networking | © Ilya Grigorik]

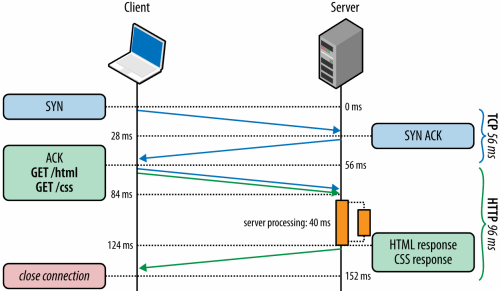
- Dispatches multiple requests without waiting for each response
- Eleminates another full roundtrip in between requests

HTTP/1.1 Pipelining with Parallel Processing



[High Performance Browser Networking | © Ilya Grigorik]

HTTP/1.1 Pipelining with Parallel Processing



[High Performance Browser Networking | © Ilya Grigorik]

- HTTP/1.x limitation: strict serialization of returned responses
- Data from multiple responses cannot be multiplexed on the same connection
- Each response needs to be returned in full before the bytes for the next response can be transferred
- **Head-of-line blocking** problem

Exercise: HTTP/1.1 with Multiple TCP connections???

History

Legacy HTTP

HTTP/2

HTTP/3 and QUIC

HTTP/2, the first major HTTP revision since 1999, originated as Google's SPDY experimental protocol in mid-2009.

> *"So far we have only tested SPDY in lab conditions. The initial results are very encouraging: when we download the top 25 websites over simulated home network connections, we see a significant improvement in performance—pages loaded up to 55% faster."*

Mike Belshe and Roberto Peon
Software Engineers at Google
Source: `https://hpbn.co/http2/`

Hello HTTP/2, Goodbye SPDY

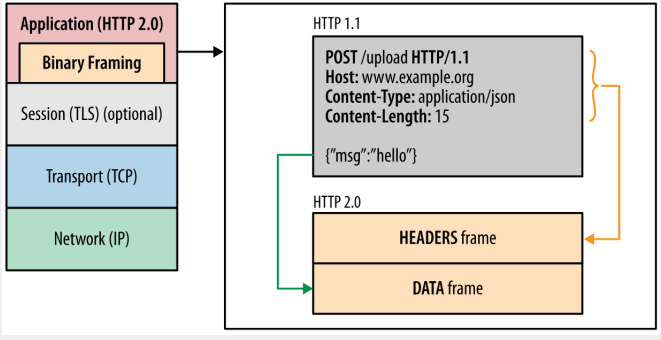
- March 2012: Call for proposals for HTTP/2
- November 2012: First draft of HTTP/2 (based on SPDY)
- August 2014: HTTP/2 draft-17 and HPACK draft-12 are published
- August 2014: Working Group last call for HTTP/2
- February 2015: IESG approved HTTP/2 and HPACK drafts
- May 2015: RFC 7540 (HTTP/2) and RFC 7541 (HPACK) are published

HTTP/2 low-level changes address the performance limitations of HTTP/1.x.

Binary Framing Layer

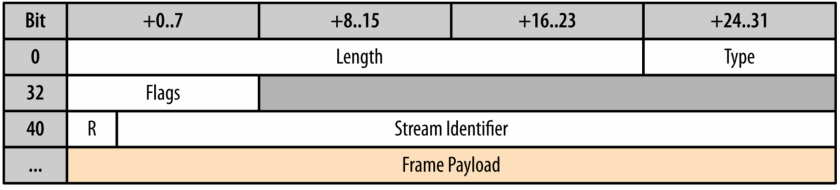
HTTP/2 low-level changes address the performance limitations of HTTP/1.x.

Binary Framing Layer



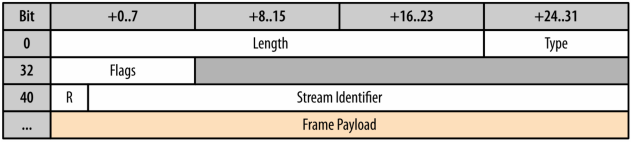
[High Performance Browser Networking | © Ilya Grigorik]

HTTP/2 Binary Framing Layer



[High Performance Browser Networking | © Ilya Grigorik]

HTTP/2 Binary Framing Layer



[High Performance Browser Networking | © Ilya Grigorik]

- HTTP semantics, such as methods, and headers, are unaffected, but are encoded in binary format.
- HTTP/1.x clients won't understand an HTTP/2 only server.
- Pros and Cons???

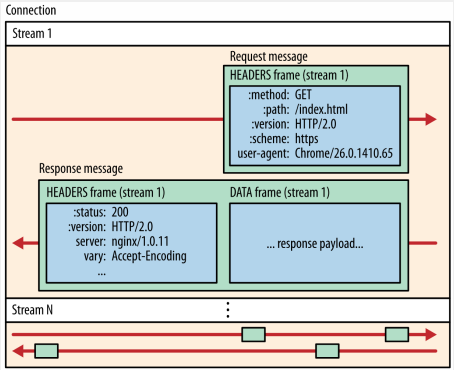
HTTP/2 low-level changes address the performance limitations of HTTP/1.x.

Binary Framing Layer

Streams, Messages, and Frames

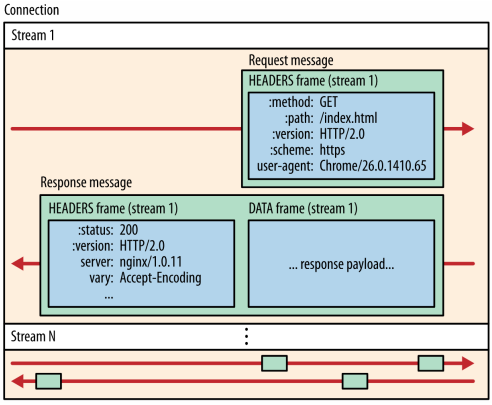
HTTP/2 low-level changes address the performance limitations of HTTP/1.x.

Streams, Messages, and Frames



[High Performance Browser Networking | © Ilya Grigorik]

HTTP/2 Streams, Messages, and Frames



[High Performance Browser Networking | © Ilya Grigorik]

- **Stream**: a bidirectional flow of bytes within an established connection, which may carry one or more messages
- **Message**: a complete sequence of frames that map to a logical request or response message
- **Frame**: The smallest unit of communication in HTTP/2, each containing a frame header, which at a minimum identifies the stream to which the frame belongs

HTTP/2 low-level changes address the performance limitations of HTTP/1.x.

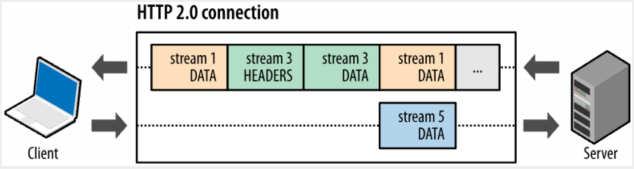
Binary Framing Layer

Streams, Messages, and Frames

Request and Response Multiplexing

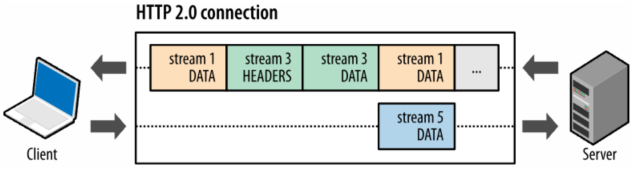
HTTP/2 low-level changes address the performance limitations of HTTP/1.x.

Request and Response Multiplexing



[High Performance Browser Networking | © Ilya Grigorik]

HTTP/2 Request and Response Multiplexing



[High Performance Browser Networking | © Ilya Grigorik]

The new binary framing layer enables HTTP/2 to:

- split HTTP messages into independent frames
- interleave frames from different streams within a single TCP connection
- reassemble frames on the other end via the embedded stream identifier in the header of each frame

HTTP/2 low-level changes address the performance limitations of HTTP/1.x.

- Binary Framing Layer
- Streams, Messages, and Frames
- Request and Response Multiplexing
- **Stream Prioritization**

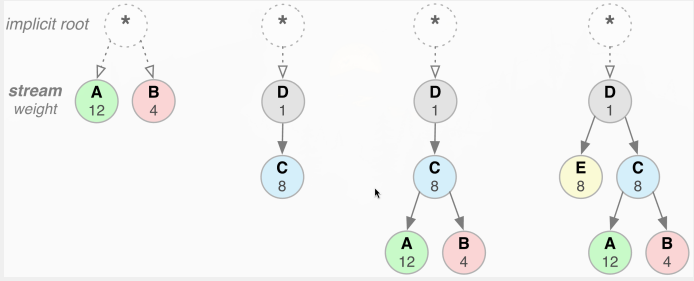
HTTP/2 low-level changes address the performance limitations of HTTP/1.x.

Stream Prioritization

- Each stream may be assigned an integer weight between 1 and 256.
- Each stream may be given an explicit dependency on another stream.

HTTP/2 Stream Prioritization

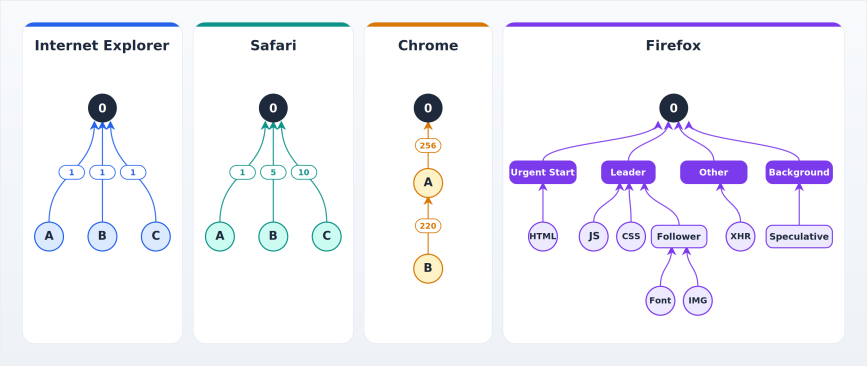
HTTP/2 stream dependencies and weights



[High Performance Browser Networking | © Ilya Grigorik]

HTTP/2 Stream Prioritization

HTTP/2 prioritization trees of different browsers



[https://doi.org/10.1145/3178876.3186181]

HTTP/2 Stream Prioritization

Check the following URL:

```
https://blog.cloudflare.com/
better-http-2-prioritization-for-a-faster-web/
```

HTTP/2 low-level changes address the performance limitations of HTTP/1.x.

Binary Framing Layer

Streams, Messages, and Frames

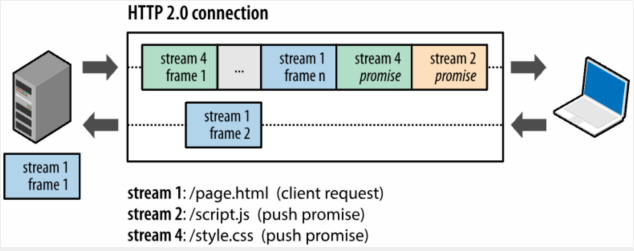
Request and Response Multiplexing

Stream Prioritization

Server Push

HTTP/2 low-level changes address the performance limitations of HTTP/1.x

Server Push: server sends multiple responses for a single request



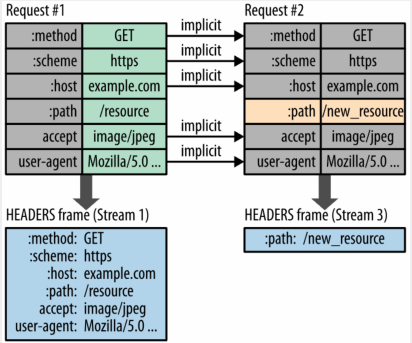
[High Performance Browser Networking | © Ilya Grigorik]

HTTP/2 low-level changes address the performance limitations of HTTP/1.x

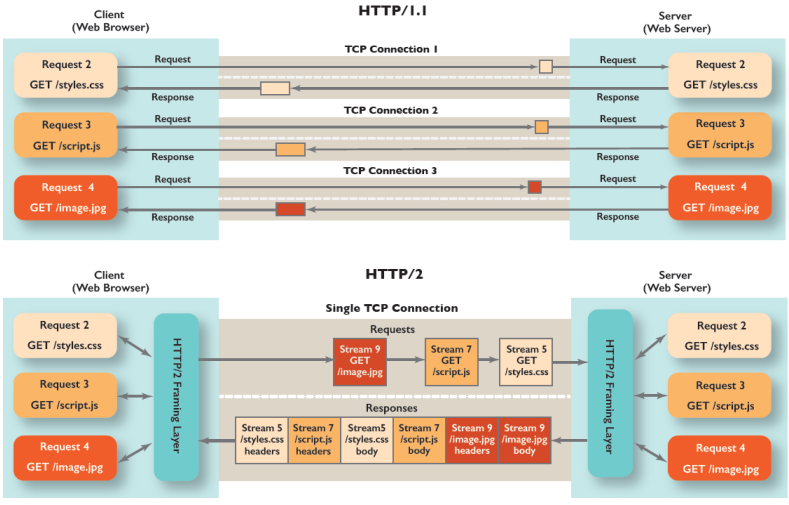
- Binary Framing Layer
- Streams, Messages, and Frames
- Request and Response Multiplexing
- Stream Prioritization
- Server Push
- **Header compression**

HTTP/2 low-level changes address the performance limitations of HTTP/1.x.

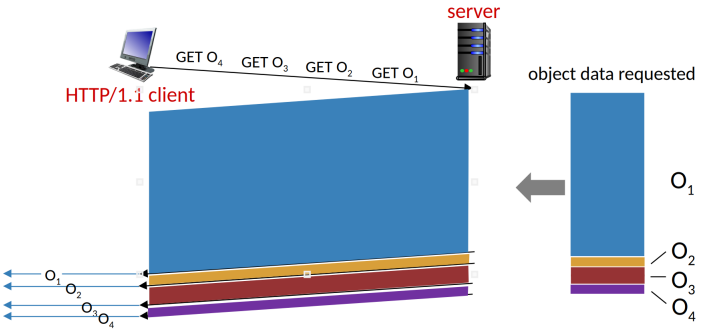
Header compression: both client and server maintain an indexed list of seen headers to efficiently encode previously transmitted values.



HTTP/2 resolves the head-of-line blocking problem found in HTTP/1.x.

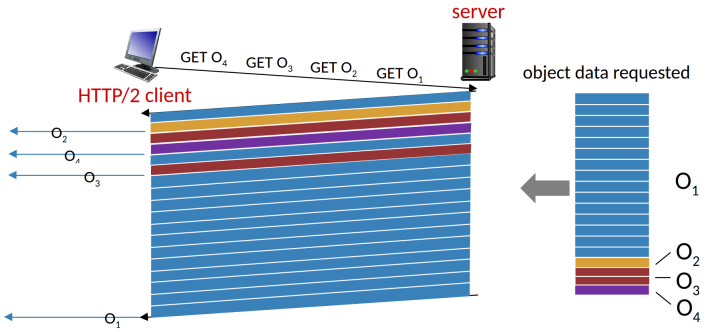


HTTP/2 resolves the head-of-line blocking problem found in HTTP/1.x.



objects delivered in order requested: O_2, O_3, O_4 wait behind O_1

HTTP/2 resolves the head-of-line blocking problem found in HTTP/1.x.



O_2, O_3, O_4 delivered quickly, O_1 slightly delayed

To properly resolve HOL blocking, HTTP/2 is not enough, because TCP itself suffers from HOL blocking.

History

Legacy HTTP

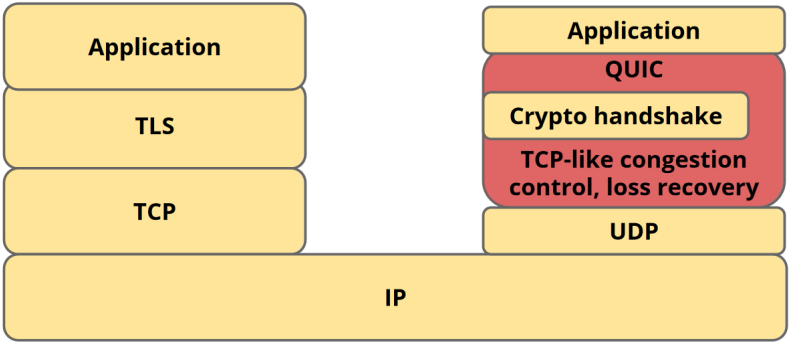
HTTP/2

HTTP/3 and QUIC

HTTP/3 has the same semantics as earlier versions of HTTP, but uses QUIC instead of TCP for the transport layer.

HTTP/3 = HTTP over QUIC (Quick UDP Internet Connections)

Quick UDP Internet Connections



[https://devopedia.org/quic]

Quick UDP Internet Connections

Designed by Google

- In June 2013, Google engineer Jim Roskind makes public QUIC's design document
- Based on lessons learned from SPDY and HTTP/2

Quick UDP Internet Connections

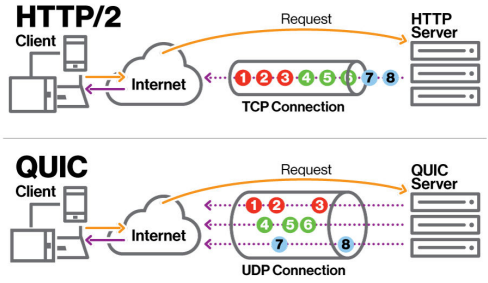
Designed by Google

- In June 2013, Google engineer Jim Roskind makes public QUIC's design document.
- Based on lessons learned from SPDY and HTTP/2

Runs in userspace over UDP

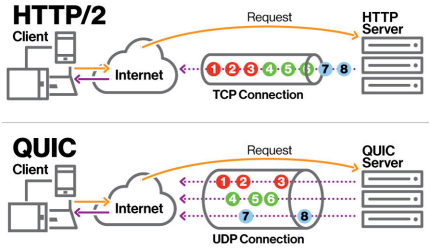
- By September 2025, 35.8% of all websites were using HTTP/3
- Overcomes HOL blocking issue by multiplexing QUIC streams over UDP
- Per-stream flow control: loss of a packet blocks only the stream with data in that packet
- Reduced connection establishment time - 0-RTT in the common case
- Seamless Connection Migration: when a device moves (eg., from cellular to WiFi), it can resume a connection by using the same QUIC Connection ID on a different IP address

QUIC streams do not suffer from Head-of-Line blocking.



[https://devopedia.org/quic]

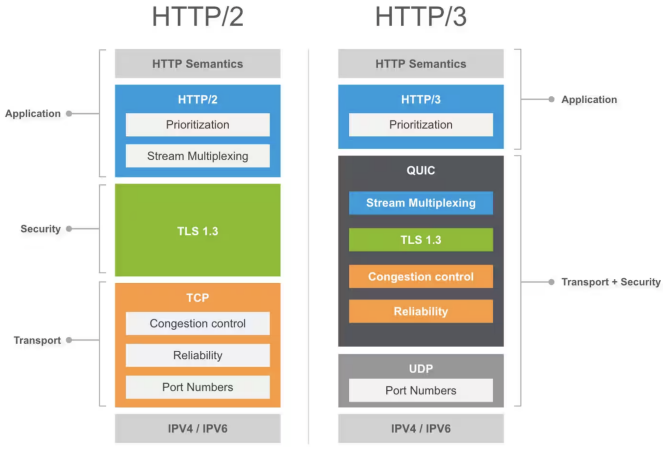
QUIC streams do not suffer from Head-of-Line blocking.



[https://devopedia.org/quic]

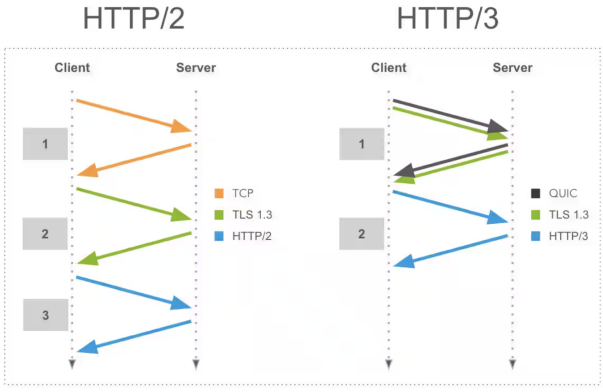
- Stream multiplexing over UDP
- Per-stream flow control (in-order reliable delivery per stream)
- Stream prioritization as part of QUIC
- Congestion control e.g. CUBIC, BBR as part of QUIC

QUIC integrates most features of TLS 1.3, enabling 0-RTT on prior connections.



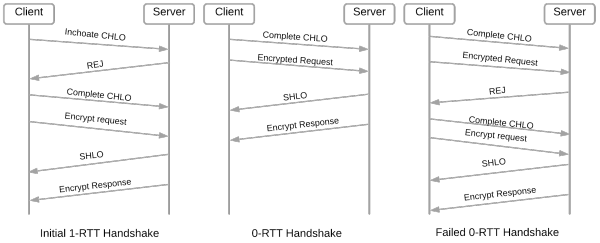
[Akamai]

QUIC integrates most features of TLS 1.3, enabling 0-RTT on prior connections.



[Akamai]

QUIC integrates most features of TLS 1.3, enabling 0-RTT on prior connections.

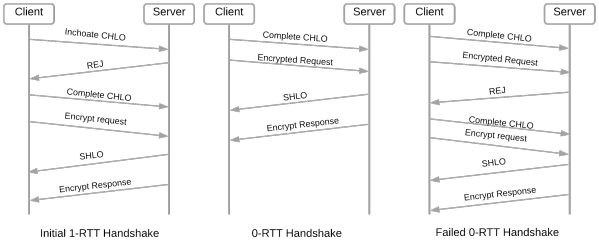


[https://www.cse.wustl.edu/~jain/]

First-time Connection Establishment (1-RTT)

- Combines TCP + TLS 1.3 handshakes
- Client and server exchange public keys and encryption parameters in 1-RTT

QUIC integrates most features of TLS 1.3, enabling 0-RTT on prior connections.



[https://www.cse.wustl.edu/~jain/]

Subsequent Connection Establishment (0-RTT)

- Remember long-lived public key from previous connection
- Client sends encrypted request along with its public key

QUIC: key features

- Payload and packet numbers are encrypted; only a few header fields stay visible (Connection IDs, spin bit) $\Rightarrow$ hard for middleboxes
- TLS 1.3 built in: 1-RTT handshake, 0-RTT on prior connections
- Connection migration: connection tied to Connection ID, not the 4-tuple, so it survives IP changes; fresh ID per path, validated via `PATH_CHALLENGE`
- Packet numbers are never reused, even for retransmissions
- Precise RTT: unique packet numbers + ACK Delay field

QUIC and HTTP/3 standards are now complete.

- QUIC: A UDP-Based Multiplexed and Secure Transport – RFC9000
- Version-Independent Properties of QUIC - RFC8999
- Using TLS to Secure QUIC - RFC9001
- QUIC Loss Detection and Congestion Control - RFC9002
- QPACK: Field Compression for HTTP/3 - RFC9204

QUIC is increasingly deployed in the wild.

Questions?



All properties use QUIC



Majority of traffic is QUIC (app, website)



...

CDNs support QUIC, but clients don't necessarily enable it

UiO : Department of Informatics
University of Oslo