

# HTTP/2 y HTTP/3

Redes y Comunicaciones (2024)

# Qué es HTTP/2?

- Reemplazo de cómo HTTP se transporta.
- No es un reemplazo del protocolo completo.
- Se conservan métodos y semántica.
- Base del trabajo protocolo desarrollado por Google SPDY/2.
- Definido en:
  - RFC7540: Hypertext Transfer Protocol version 2.
  - RFC7541: HPACK - Header Compression for HTTP/2.

# Problemas con HTTP/1.0, HTTP/1.1

- Un request por conexión, por vez, muy lento.
- Alternativas (evitar HOL/HOLB Head of Line Blocking):
  - Conexiones persistentes y pipelining.
  - Generar conexiones paralelas.
- Problemas:
  - Pipelining requiere que los responses sean enviado en el orden solicitado, HOLB posible.
  - POST no siempre pueden ser enviados en pipelining.
  - Demasiadas conexiones genera problemas, control de congestión, mal uso de la red.
  - Muchos requests, muchos datos duplicados (headers).

# Diferencias principales con HTTP/1.1

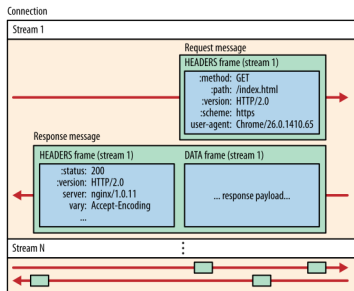
- Protocolo binario en lugar de textual(ASCII), binary framing: (más eficiente).
- Multiplexa varios request en una petición en lugar de ser una secuencia ordenada y bloqueante.
- Utilizar una conexión para pedir/traer datos en paralelos, agrega: datos fuera de orden, priorización, flow control por frame.
- Usa compresión de encabezado.
- Permite a los servidores “pushear” datos a los clientes.
- La mayoría de las implementaciones requieren TLS/SSL, no el estándar.

# HTTP/2 mux stream, framing

- Puede generar una o más conexiones TCP. Trata de aprovechar las que tiene establecidas.
- Un stream es como una sub-conexión (una “conexión” http2 dentro de una conexión TCP).
- Un stream tiene un ID y una prioridad(alternativa) y son bidireccionales.
- Sobre una conexión TCP multiplexa uno o más streams (“conexiones http2”).
- Los streams transportan mensajes.
- Los mensajes http2 (Request, Response) se envían usando un stream.
- Los mensajes http2 son divididos en frames dentro del mismo stream.
- Un frame es una porción de mensaje: header fijo+payload variable (unidad mínima).
- El mismo stream puede ser usado para llevar diferentes msj.

# HTTP/2 mux stream, framing (cont.)

- Los mensajes están compuestos por frames, que podrían ser de diferentes tipos.
- Los streams van en una misma conexión.
- Los streams son identificados y divididos en frames.
- Frame types: HEADERS, DATA, PUSH\_PROMISE, WINDOW\_UPDATE, SETTINGS, etc.



fuelle: <https://web.dev/performance-http2/#streams,-messages,-and-frames>

# HTTP/2 mux stream, framing (cont.)

http2\_native.pcapng

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http2

No.	Time	Source	Destination	Protocol	Length	Info
4	0.414129282	192.168.0.228	139.162.123.134	HTTP2	90	Magic
5	0.420956333	192.168.0.228	139.162.123.134	HTTP2	152	SETTINGS[0], WINDOW_UPDATE[0], HEADERS[1]: G
7	0.839101976	139.162.123.134	192.168.0.228	HTTP2	100	SETTINGS[0]
10	0.839139220	139.162.123.134	192.168.0.228	HTTP2	76	SETTINGS[0]
12	0.839167233	139.162.123.134	192.168.0.228	HTTP2	369	HEADERS[1]: 200 OK, DATA[1] (text/plain)
14	0.839280935	192.168.0.228	139.162.123.134	HTTP2	75	SETTINGS[0]
15	0.845040952	192.168.0.228	139.162.123.134	HTTP2	93	HEADERS[3]: GET /humans.txt

Transmission Control Protocol, Src Port: 38564, Dst Port: 80, Seq: 25, Ack: 1, Len: 86

HyperText Transfer Protocol 2

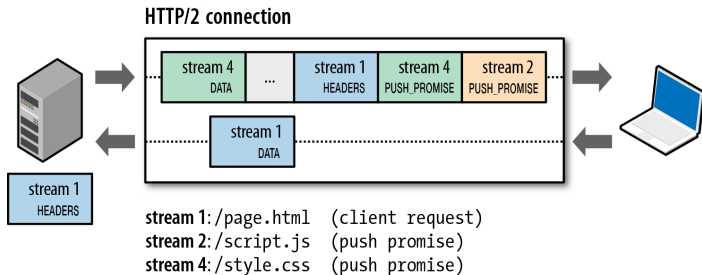
- Stream: SETTINGS, Stream ID: 0, Length 18
- Stream: WINDOW\_UPDATE, Stream ID: 0, Length 4
- Stream: HEADERS, Stream ID: 1, Length 37, GET /robots.txt
  - Length: 37
  - Type: HEADERS (1)
  - Flags: 0x05
  - 0... .. = Reserved: 0x0
  - .000 0000 0000 0000 0000 0000 0000 0001 = Stream Identifier: 1
  - [Pad Length: 0]
  - Header Block Fragment: 82048862c3c674a174f94f864188aa69d29ac4b9ec9b7a88...
  - [Header Length: 136]
  - [Header Count: 6]
  - Header: :method: GET
  - Header: :path: /robots.txt
  - Header: :scheme: http
  - Header: :authority: nghttp2.org
  - Header: user-agent: curl/7.61.0
  - Header: accept: \*/\*

HyperText Transfer Protocol 2: Protocol

Packets: 21 · Displayed: 8 (38.1%)

Profile: Classic

# HTTP/2 mux stream, framing (cont.)

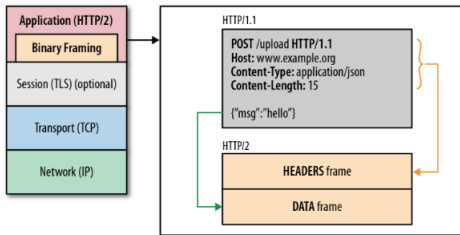


fuelle: <https://docs.google.com/presentation/d/1r7QXGYOLCh4fcUq0jDdDwKJWNqWK1o4xMtYpKZCJYjM/present?slide=id.p19>



# HTTP/2 mux stream, framing (Cont.)

- Streams codificados en binario y cada frame con header común fijo(9B).



fuelle: <https://docs.google.com/presentation/d/1r7QXGYOLCh4fcUq0jDdDwKJWNqWK1o4xMtYpKZCJYjM/present?slide=id.p19>

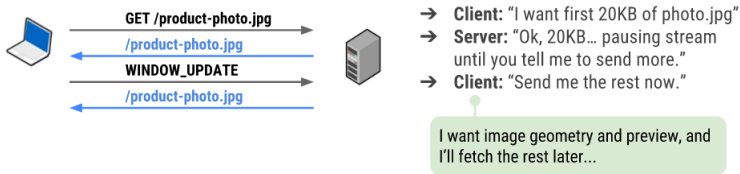
# HTTP/2 HEADERS

- Se mantienen casi todos los HEADERS de HTTP/1.1.
- No se codifican más en ASCII.
- Surgen nuevos pseudo-headers que contiene información que estaba en el método y otros headers.
- Por ejemplo: `HEAD /algo HTTP/1.1` se reemplaza con `http2:`
  - `:method: head`
  - `:path: /algo`
  - `:scheme: https o http`
  - `:authority: www.site.com` reemplaza al header `Host:`.

Para las respuestas: `:status:` códigos de retornos 200, 301, 404, etc.

# HTTP/2 priorización y flow-control

- Los streams dentro de una misma conexión tienen flow-control individual.
- Los streams pueden tener un weight (prioridad).
- Los streams pueden estar asociados de forma jerárquica, dependencias.



fuelle: <https://docs.google.com/presentation/d/1r7QXGYOLCh4fcUq0jDdDwKJWNqWK1o4xMtYpKZCJYjM/present?slide=id.p19>

# HTTP/2 inline vs. push

- Cuando el cliente solicita una página, “parsea” el primer response HTML luego solicita el resto.
- El server puede enviar el HTML más otros datos, por ejemplo CSS o Javascript.
- No siempre es lo que necesita el cliente, depende de que funcionalidad ofrece.



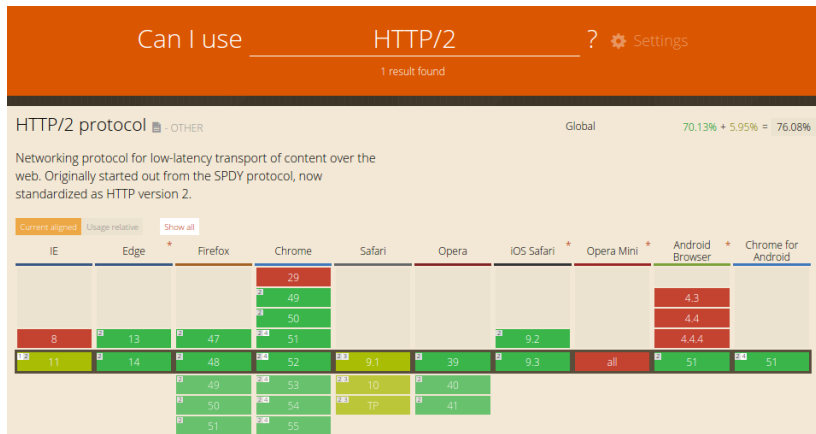
**Server:** "You asked for */product/123*, but you'll need *app.js*, *product-photo-1.jpg*, as well... I promise to deliver these to you. That is, unless you decline or cancel."

fuelle: <https://docs.google.com/presentation/d/1r7QXGYOLCh4fcUq0jDdDwKJWNqWK1o4xMtYpKZCJYjM/present?slide=id.p19>

# Compresión y Soporte

- Compresión de encabezados.
- SPDY/2 propone usar GZIP.
- GZIP + cifrado, tiene “bugs” utilizados por atacantes.
- Se crea un nuevo compresor de Headers: HPACK.
- H2 y SPDY, soportados en la mayoría de los navegadores.

# Soporte en clientes para 2016



fuelle: <http://caniuse.com/#search=HTTP%2F2>

# Otras Características

- HTTP/1.1, posibilidad de hacer un upgrade durante la conexión:  
Upgrade Header. `Connection: Upgrade, HTTP2-Settings`  
`Upgrade: h2c|h2.`
- http2: Negociar el protocolo de aplicación:  
ALPN: Application-Layer Protocol Negotiation.  
Se negocia como extensión de SSL en Hello (Anteriormente NPN). Se ofrece: h2, h3, http/1.1, ...
- Posibilidad de negociar protocolo alternativo:  
Alterantive Service: `alt-svc.`

# Application-Layer Protocol Neg.

Filter:  Expression... Clear Apply Save

No.	Time	Source	Destination	Length	Info
4	0.000114	127.0.0.1	127.0.0.1	56	[TCP Window Update] 8443-61946 [ACK] Seq=1
5	0.000285	127.0.0.1	127.0.0.1	461	Client Hello
6	0.000328	127.0.0.1	127.0.0.1	56	8443-61946 [ACK] Seq=1 Ack=406 Win=146576 L
7	0.001662	127.0.0.1	127.0.0.1	1122	Server Hello, Certificate, Server Hello Dor

- ▼ Extension: Application Layer Protocol Negotiation
  - Type: Application Layer Protocol Negotiation (0x0010)
  - Length: 52
  - ALPN Extension Length: 50
  - ▼ ALPN Protocol
    - ALPN string length: 17
    - ALPN Next Protocol: HTTP-draft-04/2.0
    - ALPN string length: 8
    - ALPN Next Protocol: spdy/4a2
    - ALPN string length: 8
    - ALPN Next Protocol: spdy/3.1
    - ALPN string length: 6
    - ALPN Next Protocol: spdy/3
    - ALPN string length: 6
    - ALPN Next Protocol: spdy/2
  - ▼ Extension: status\_request
    - Type: status\_request (0x0005)
    - Length: 5

File: "/home/andres/docs/mvdoc... Packets: 202 · Disp... Profile: Default



# Debugging (2016)

Google - Google Chrome

Google

https://www.google.com.ar/#gfe\_rd=cr

Argentina

Andres

Elements Console Sources Network Timeline Profiles Application Security Audits

View: [Icons] Preserve log [ ] Disable cache [ ] Offline No throttling [v]

10000 ms 20000 ms 30000 ms 40000 ms 50000 ms 60000 ms 70000 ms 80000 ms 90000 ms 100000 ms 110000 ms

Name	Method	Status	Protocol	Type	Initiator	Size	Time	Timeline - Start Time	4.00 s	6.00 s	▲
www.google.com	GET	302	h2	text/html	Other	399 B	33 ms				
?gfe_rd=cr&ei=6XPEV9nsCcOgxgS5...	GET	200	quic/1+spdy/3	document	https://www.goog...	62.7 KB	335 ms				
nav_logo242.png	GET	200	quic/1+spdy/3	png	?gfe_rd=cr&ei=6X...	(from cac...)	2 ms				
googlelogo_color_272x92dp.png	GET	200	quic/1+spdy/3	png	?gfe_rd=cr&ei=6X...	(from cac...)	3 ms				
i1_1967ca6a.png	GET	200	quic/1+spdy/3	png	?gfe_rd=cr&ei=6X...	(from cac...)	5 ms				
photo.jpg	GET	200	quic/1+spdy/3	png	?gfe_rd=cr&ei=6X...	(from cac...)	4 ms				
data:image/gif;base...	GET	200	data	gif	?gfe_rd=cr&ei=6X...	(from cac...)	0 ms				
data:image/gif;base...	GET	200	data	gif	?gfe_rd=cr&ei=6X...	(from cac...)	0 ms				

25 requests | 64.3 KB transferred | Finish: 7.34 s | DOMContentLoaded: 536 ms | Load: 535 ms

# Debugging (Cont.)

The screenshot shows the Google Chrome browser with the address bar displaying `https://www.google.com.ar/#gfe_rd=cr`. The page content shows the Google logo with 'Argentina' text below it and a search bar. The DevTools Network tab is open, showing a list of requests on the left and the details of the first request on the right.

**Network Tab Details:**

- Name:** www.google.com
- General:**
  - Request URL: `https://www.google.com/`
  - Request Method: GET
  - Status Code: 302
  - Remote Address: `[2800:3f0:4003:c01:63]:443`
- Response Headers:**
  - `alt-svc: quic=":443"; ma=2592000; v="35,34,33,32,31,30"`
  - `alternate-protocol: 443:quic`
  - `cache-control: private`
  - `content-length: 263`
  - `content-type: text/html; charset=UTF-8`
  - `date: Mon, 29 Aug 2016 17:42:01 GMT`
  - `location: https://www.google.com.ar/7gfe_rd=cr&ei=6XPEV9nsCc0gxg55_ZuICg`
  - `status: 302`

The bottom of the Network tab shows `41 requests | 177 KB transfer...`.

# Debugging (Cont.)

chrome://net-internals/#http2

Capturing halted

Export  
Import  
Proxy  
Events  
Timeline  
DNS  
Sockets  
Alt-Svc  
HTTP/2  
QUIC  
SDCH  
Cache  
Modules  
HSTS  
Bandwidth  
Prerender

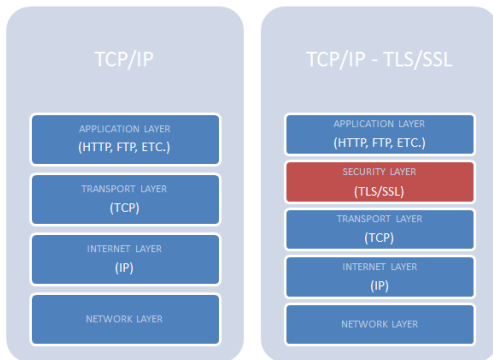
- HTTP/2 Enabled: true
- SPDY/3.1 Enabled: false
- Use Alternative Service: true
- ALPN Protocols: h2,http/1.1
- NPN Protocols: undefined

**HTTP/2 sessions**

[View live HTTP/2 sessions](#)

Host	Proxy	ID	Protocol Negotiated	Active streams	Unclaimed pushed	Max	Initiated	Pushed	Pushed and claimed	Abandoned	Received frames	Secure	Sent settings	Received settings
accounts.google.com:443	direct://	231	h2	0	0	100	0	0	0	0	0	true	true	true
s2.googleusercontent.com:443	direct://	370	h2	0	0	100	0	0	0	0	0	true	true	true
ssl.google-analytics.com:443	direct://	265	h2	0	0	100	0	0	0	0	0	true	true	true
ssl.gstatic.com:443	direct://	450	h2	0	0	100	0	0	0	0	0	true	true	true
www.google-analytics.com:443	direct://	339	h2	0	0	100	0	0	0	0	0	true	true	true
www.google.com:443	direct://	516	h2	0	0	100	0	0	0	0	0	true	true	true
www.google.com.ar:443	direct://	335	h2	0	0	100	0	0	0	0	0	true	true	true
accounts.google.com:443	direct://	328	h2	0	0	100	0	0	0	0	0	true	true	true
clients2.google.com:443	direct://	654	h2	0	0	100	0	0	0	0	0	true	true	true
www.googleapis.com:443	direct://	120	h2	0	0	100	0	0	0	0	0	true	true	true

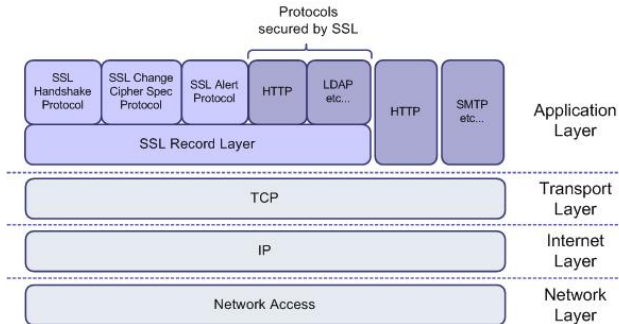
# SSL/TLS



**Network Layer = Network Access Layer = Network Interface Layer = Link + Física**

fuelle: <https://www.simple-talk.com/dotnet/net-framework/tlsssl-and-net-framework-4-0/>

# SSL/TLS (Cont.)

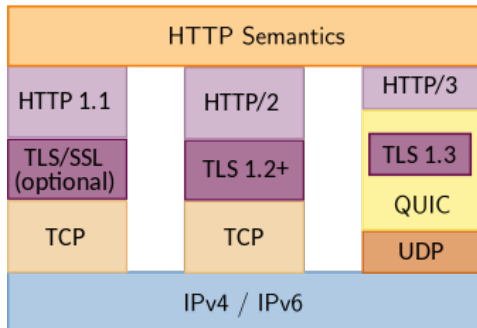


fuelle: [http://nicolascormier.com/documentation/bin/apache/apache2\\_with\\_ssl\\_tls/part1.htm](http://nicolascormier.com/documentation/bin/apache/apache2_with_ssl_tls/part1.htm)

# Qué es HTTP/3?

- Transportar HTTP sobre nuevo protocolo: QUIC, que corre sobre UDP, default UDP:443.
- Igual que HTTP/2 se conservan métodos y semántica de HTTP/1.1.
- Base del trabajo protocolo desarrollado por Google, QUIC.
- Definido en:
  - RFC9000: QUIC: A UDP-Based Multiplexed and Secure Transport.
  - RFC9114: HTTP/3.

# HTTP/3



fuelle: <https://en.wikipedia.org/wiki/HTTP/3>

# Diferencias principales de HTTP/3-QUIC

- Reducción de latencia en conexiones (2 msj. de handshake).
- No depende del manejo de la conexión y controles de TCP (evita HOLB).
- Loss recovery/congestion control, sobre UDP, lo puede hacer como TCP-CUBIC o de otra forma.
- QUIC genere nuevos números de secuencia y re-cifra las retransmisiones, permitiendo mejor detección de pérdidas y medición de RTT.
- QUIC Paquetes cifrados de forma individual, no por conexión/bytestream.
- QUIC facilita movilidad, tiene ID de conexión independiente de IPs y ports.
- Solo trabaja seguro con cifrado.



## Diferencias principales de HTTP/3-QUIC (Cont.)

- Desventaja de QUIC: muchos middle-boxes filtran UDP, salvo port 53 y NAT.
- Protocolo de transporte QUIC: implementado usualmente en User-space vs Kernel-space.
- UDP facilita algunos ataques de seguridad.

# HTTP/3 captura

\*enp0s31f6

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

quic

No.	Time	Source	Destination	Protocol
21	9.202350845		2800:3f0:4002:809::2003	QUIC
22	9.202542084		2800:3f0:4002:809::2003	QUIC

Internet Protocol Version 6, Src: 2800:340:..., Dst: 2800:3f0:4002:809::2003

User Datagram Protocol, Src Port: 56546, Dst Port: 443

QUIC IETF

- QUIC Connection information
  - [Packet Length: 1230]
  - 1... .... = Header Form: Long Header (1)
  - .1... .... = Fixed Bit: True
  - ..00 .... = Packet Type: Initial (0)
  - ... 00... = Reserved: 0
  - .... ..00 = Packet Number Length: 1 bytes (0)
  - Version: 1 (0x00000001)
  - Destination Connection ID Length: 8
  - Destination Connection ID: 5b8c72711860d7e2
  - Source Connection ID Length: 0
  - Token Length: 70
  - Token: 00a724a2bbb3734560b9583c50a85edabdd33ef2a8f75a68ae4ec82ff9041010b80dbdf3...
  - Length: 1141
  - Packet Number: 1
  - Payload: d7b50a500c2fdf8454cd5916cc5100508a1778eae56de71f0604326db821547415afac4d...
- PING
  - Frame Type: PING (0x0000000000000001)
- PING
- PING
- PING
- PADDING Length: 18
- CRYPTO
  - Frame Type: CRYPTO (0x0000000000000006)
  - Offset: 312
  - Length: 746
  - Crypto Data
  - TLV1.3 Record Layer: Handshake Protocol: Encrypted Handshake Message
- PING
- CRYPTO

Destination Connection ID Length (quic.dci), 1 byte

Packets: 2015 · Displayed: 670 (33.3%) Profile: Classic

# HTTP/3 Debugging (2024)

The screenshot shows a web browser window displaying the YouTube homepage. The address bar shows the URL `https://www.youtube.com`. The Network DevTools panel is open at the bottom, showing a list of requests. The selected request is a GET request to `https://www.youtube.com/?themeRefresh=1`.

St...	Meth...	Domain	File	Protocol	Type	Transferred	Size
280	GET	www.y...	/?themeRefresh=1	HTTP/3	html	41.93 kB	475.20 kB
206	GET	www.y...	failure.mp3	HTTP/3	mp4	7.20 kB	6.53 kB
206	GET	www.y...	no_input.mp3	HTTP/3	mp4	7.63 kB	6.95 kB
206	GET	www.y...	open.mp3	HTTP/3	mp4	6.84 kB	6.17 kB
206	GET	www.y...	success.mp3	HTTP/3	mp4	7.31 kB	6.64 kB
	POST	www.yout...	log_event?alt=json&key=		NS_BINDI...		
200	GET	fonts.g...	24px.svg	HTTP/2	svg	997 B	146 B
200	GET	www.g...	24px.svg	HTTP/2	svg	2.92 kB	5.60 kB
204	GET	lytimg...	generate_204	HTTP/3	plain	172 B	0 B
200	GET	www.g...	24px.svg	HTTP/2	svg	997 B	146 B

The selected request details are as follows:

- Status:** 200
- Version:** HTTP/3
- Transferred:** 41.93 kB (475.20 kB size)
- Referrer Policy:** strict-origin-when-cross-origin
- Request Priority:** Highest
- DNS Resolution:** System

The response headers (1.204 kB) are:

- `alt-svc: h3="";ma=2592000,h3-29="";ma=2592000`
- `cache-control: no-cache, no-store, max-age=0, must-revalidate`

[HTTP/2] <https://http2.github.io/>.

[Ilya Grigorik] HTTP/2 is here, let's optimize!

<https://docs.google.com/presentation/d/1r7QXGYOLCh4fcUq0jDdDwKJWNqWK1o4xMtYpKZCJYjM/present?slide=id.p19>.

[HTTP/2-dev] <https://web.dev/performance-http2/#streams,-messages,-and-frames>

[HTTP/2-undertow] <https://undertow.io/blog/2015/04/27/An-in-overview-of-HTTP2.html>

[HTTP/3-cloudflare] <https://www.cloudflare.com/learning/performance/what-is-http3/>

[HTTP/3-wikipedia] <https://en.wikipedia.org/wiki/HTTP/3>

[QUIC-wikipedia] <https://en.wikipedia.org/wiki/QUIC>

[QUIC-google] <https://peering.google.com/#/learn-more/quic>