quic-http3-handshake-decrypted.annotated

Chrome | Cloudflare

[1] 🔗 TCP SYN

🎞️ Frame	1
📤 Src Port	57098
📥 Dst Port	443
🔢 TCP Stream	0
Sequence Number (raw)	2688906860
Calculated window size	65535
MSS Value	1440

[Frame 1] Phase 1: the first visit, over TCP. Chrome opens a TCP connection to port 443. On a first visit it has no evidence that the server speaks HTTP/3, so it starts with TCP, TLS and HTTP/2. Time zero for the TCP timings below.

[2] 🔗 TCP SYN-ACK

⏱️ Δ Frame	11.499000 milliseconds
⏱️ T+	11.499000 milliseconds
🎞️ Frame	2
📤 Src Port	443
📥 Dst Port	57098
🔢 TCP Stream	0
Sequence Number (raw)	3228380260
Acknowledgment number (raw)	2688906861
Calculated window size	64704
⏱️ Handshake RTT	11.560000 milliseconds

[Frame 2] The SYN-ACK arrives 11.5 ms later. Chrome's ACK (frame 3, not drawn) completes the three-way handshake. One round trip spent, and no application bytes have moved yet.

[4] 🤝 TLS Client Hello

🎞️ Frame	4
🔢 TCP Stream	0
Version	TLS 1.2 (0x0303)
Cipher Suites (16 suites)	
🌐 SNI	cloudflare-quic.com
Supported Groups (4 groups)	
🔒 JA3	cd08e31494f9531f560d64c695473da9

[Frame 4] TLS 1.3 Client Hello over TCP. SNI `cloudflare-quic.com`. ALPN offers h2 and `http/1.1`, not h3 : HTTP/3 cannot run over TCP. It lists 16 cipher suites: a GREASE value, the three TLS 1.3 suites, and TLS 1.2 suites for older servers. It also sends a 32-byte legacy session ID for TLS 1.3 middlebox compatibility mode. Compare this with the Client Hello inside the QUIC Initial in frame 47.

[6] 🤝 TLS Server Hello

🎞️ Frame	6
🔢 TCP Stream	0
Version	TLS 1.2 (0x0303)
Cipher Suite	TLS_AES_128_GCM_SHA256 (0x1301)
Supported Version	TLS 1.3 (0x0304)
JA3S	eb1d94daa7e0344597e756a1fb6e7054

[Frame 6] Server Hello: TLS 1.3 with `TLS_AES_128_GCM_SHA256` and an x25519 key share. From here on both sides derive handshake keys. The same TCP segment carries a dummy Change Cipher Spec record, part of the compatibility mode Chrome asked for.

[8.1] Handshake Protocol: Encrypted Extensions

🎞️ Frame	8
🔢 TCP Stream	0
Handshake Type	Encrypted Extensions (8)

[Frame 8.1] Encrypted Extensions, the first message protected with handshake keys. It confirms ALPN h2 . Wireshark can read it only because the capture embeds the TLS secrets.

[8.2] Handshake Protocol: Compressed Certificate

Handshake Type	Compressed Certificate (25)

[Frame 8.2] The server's certificate chain, compressed (Compressed Certificate).

[8.3] Handshake Protocol: Certificate Verify

Handshake Type	Certificate Verify (15)

[Frame 8.3] Certificate Verify: the server signs the handshake transcript, proving it holds the certificate's private key.

[8.4] Handshake Protocol: Finished

Handshake Type	Finished (20)

[Frame 8.4] Server Finished closes the server's side of the handshake.

[10] 🔒 TLS Change Cipher Spec

🎞️ Frame	10
🔢 TCP Stream	0

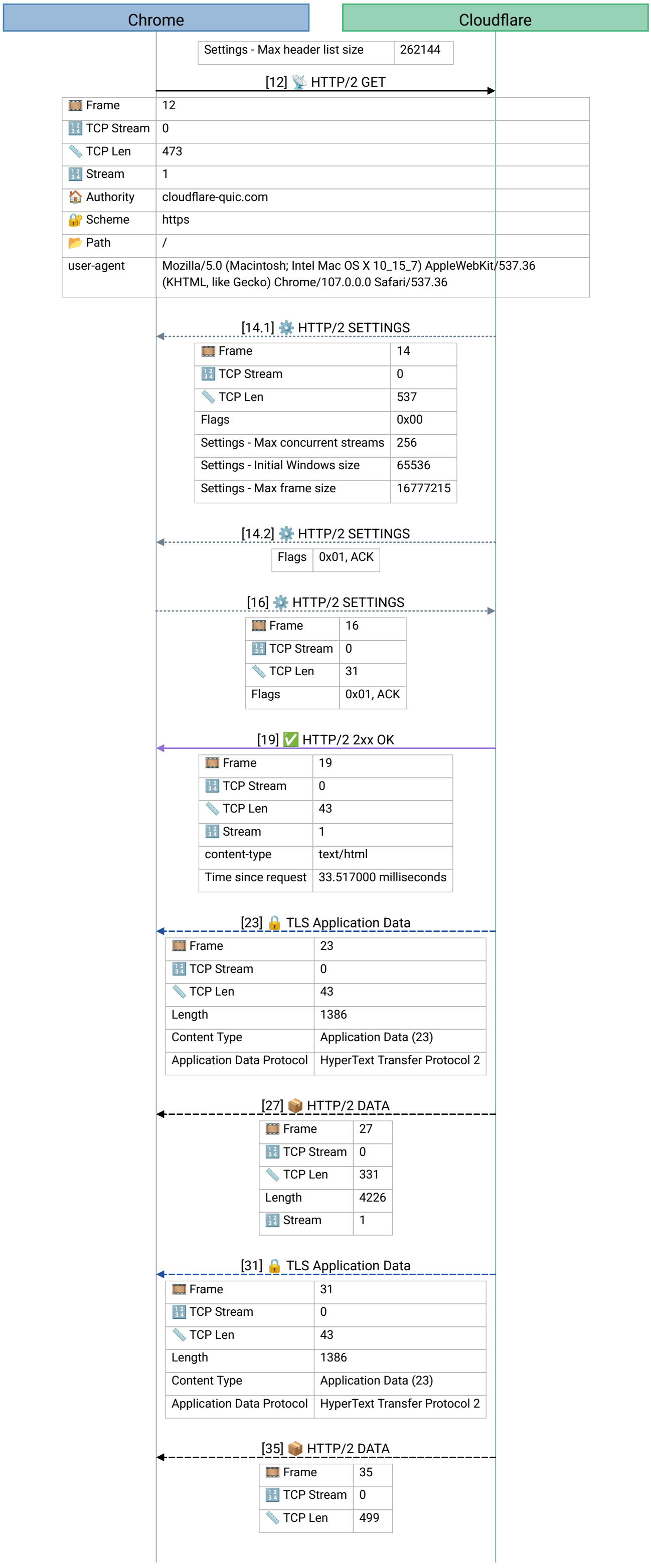
[Frame 10] Chrome's dummy Change Cipher Spec and its **Finished** share this segment. The TLS handshake is done 45.4 ms after the SYN: two round trips, one for TCP and one for TLS.

[11] ⚙️ HTTP/2 SETTINGS

🎞️ Frame	11
🔢 TCP Stream	0
📏 TCP Len	98
Flags	0x00
Settings - Header table size	65536
Settings - Enable PUSH	0
Settings - Max concurrent streams	1000
Settings - Initial Windows size	6291456

[Frame 11] HTTP/2 starts. The connection preface plus Chrome's SETTINGS: header table 65,536, server push disabled, initial stream window 6 MB.

quic-http3-handshake-decrypted.annotated



[Frame 12] GET / on stream 1, sent 45.7 ms after the first SYN. Two full round trips passed before the request could leave. Remember that number for frame 55.

[Frame 14.1] Cloudflare's SETTINGS: up to 256 concurrent streams, 64 KB initial window.

[Frame 14.2] Cloudflare acknowledges Chrome's SETTINGS.

[Frame 16] Chrome acknowledges Cloudflare's SETTINGS.

[Frame 19] 200 OK, 79.2 ms after the SYN. The response carries the header that changes everything: `alt-svc: h3=":443"; ma=86400`. Cloudflare is telling Chrome that HTTP/3 is available on UDP port 443 for the next 24 hours.

[Frame 23] A TLS record carrying part of the response body. The HTTP/2 DATA frame inside it spans several TCP segments, so Wireshark decodes it in a later frame.

[Frame 27] HTTP/2 DATA on stream 1: 4,226 bytes of Brotli-compressed HTML.

[Frame 31] Another TLS record of response body, reassembled into the next DATA frame.

[Frame 35] HTTP/2 DATA, 3,163 bytes.

Chrome | Cloudflare

Length	3163
Stream	1

[39] TLS Application Data

Frame	39
TCP Stream	0
TCP Len	43
Length	1386
Content Type	Application Data (23)
Application Data Protocol	HyperText Transfer Protocol 2

[Frame 39] Another TLS record of response body.

[42] HTTP/2 DATA

Frame	42
TCP Stream	0
TCP Len	33
Length	2719
Stream	1

[Frame 42] HTTP/2 DATA, 2,719 bytes. The body totals about 10 KB.

[44] HTTP/2 DATA

Frame	44
TCP Stream	0
TCP Len	31
Length	0
Stream	1

[Frame 44] An empty DATA frame with END_STREAM ends the response, 81.6 ms after the SYN. The page is loaded. This TCP connection now sits idle until frame 80.

[47] QUIC Initial

Frame	47
Src Port	50280
Dst Port	443
Version	1 (0x00000001)
DCID	203f9e9f68698274
Pkt	1

[Frame 47] Phase 2: the reload, over QUIC. 5.4 s later Chrome loads the page again, and this time it uses the `alt-svc` hint from frame 19. Time zero for the QUIC timings below.

This **Initial** packet carries the TLS Client Hello. Initial packets are encrypted, but with keys derived from the Destination Connection ID (`203f9e9f68698274`, 8 random bytes Chrome chose) and a salt published in RFC 9001. Anyone on the path can decrypt them, so this Client Hello is readable even without the embedded secrets.

- ALPN h3 only, 3 cipher suites, empty session ID, and a `quic_transport_parameters` extension with Chrome's flow-control limits.
- Source Connection ID length 0: Chrome asks Cloudflare to send its packets with no connection ID at all.
- The 300-byte Client Hello is split into 9 CRYPTO frames in shuffled order, mixed with PING and PADDING frames. This is Chrome's deliberate scrambling, called chaos protection in its QUIC library.
- The datagram is 1,230 bytes: clients must pad Initial datagrams to at least 1,200 bytes.
- Packet number 1, not 0. See frame 52.

[48] QUIC Initial

Frame	48
Src Port	443
Dst Port	50280
Version	1 (0x00000001)
SCID	0130dfc5a047e6acd230b5c5e047ced9b0a6bbf0
Pkt	0

[Frame 48] Cloudflare's first Initial packet holds only an ACK of Chrome's packet 1. Its Source Connection ID, `0130dfc5...`, is 20 bytes; Chrome uses it as the destination from frame 52 on. The 53-byte packet is followed by random padding that fills the datagram to 1,200 bytes. The reply took 18.6 ms.

[49] QUIC Initial

Frame	49
Src Port	443
Dst Port	50280
Version	1 (0x00000001)
SCID	0130dfc5a047e6acd230b5c5e047ced9b0a6bbf0
Pkt	1

[Frame 49] Second Initial packet (Initial packet number 1): the **Server Hello,** TLS 1.3, `TLS_AES_128_GCM_SHA256`, x25519. No Retry and no version negotiation happened: QUIC version 1 was accepted on the first try.

[50] QUIC Handshake

Frame	50
Src Port	443
Dst Port	50280
Version	1 (0x00000001)
Source Connection ID	0130dfc5a047e6acd230b5c5e047ced9b0a6bbf0
Pkt	0

[Frame 50] Handshake packet space. Handshake packets use the handshake keys from the Server Hello, so only the endpoints, or someone holding the secrets, can read them. Handshake packet numbers restart at 0.

Inside: Encrypted Extensions with ALPN h3 and Cloudflare's transport parameters. `original_destination_connection_id` echoes Chrome's `203f9e9f68698274`, which confirms no Retry. The idle timeout is 180 s, versus 30 s from Chrome; the smaller one applies.

Chrome | Cloudflare

[51] 🤝 QUIC Handshake

Frame	51
Src Port	443
Dst Port	50280
Version	1 (0x00000001)
Source Connection ID	0130dfc5a047e6acd230b5c5e047ced9b0a6bbf0
Pkt	1

[Frame 51] The rest of the server's flight: the Brotli-compressed certificate for `*.cloudflare-quic.com`, Certificate Verify and server **Finished.** The server's handshake is complete, 22.6 ms after Chrome's first packet.

[52] 🤝 QUIC Handshake

Frame	52
Src Port	50280
Dst Port	443
Version	1 (0x00000001)
DCID	0130dfc5a047e6acd230b5c5e047ced9b0a6bbf0
Pkt	2

[Frame 52] Chrome ACKs Handshake packets 0-1, now addressing Cloudflare's 20-byte connection ID.

Packet numbers. QUIC keeps three separate number spaces: Initial, Handshake and application data (1-RTT). Each has its own ACKs. Cloudflare restarts at 0 in each space. Chrome's numbers run 1, then 2-3, then 4-14, as if from one counter. RFC 9000 says each space starts at 0, but receivers only need each space's numbers to increase, so the gaps cause no harm.

[53] 🤝 QUIC Handshake

Frame	53
Src Port	50280
Dst Port	443
Version	1 (0x00000001)
DCID	0130dfc5a047e6acd230b5c5e047ced9b0a6bbf0
Pkt	3

[Frame 53] Chrome's **Finished,** 24.0 ms after its first packet. The handshake took one round trip, against two for TCP plus TLS.

[54] ⚙️ HTTP/3 SETTINGS

Frame	54
Src Port	50280
Dst Port	443
DCID	0130dfc5a047e6acd230b5c5e047ced9b0a6bbf0
Stream	2
Max Table Capacity	65536
Max header list size	262144
Blocked Streams	100

[Frame 54] The first 1-RTT packet. From here on the packets use the short header, which carries only the destination connection ID and the packet number. The payload is encrypted with application keys, readable here only because the capture embeds the secrets.

Chrome opens its HTTP/3 **control stream** (unidirectional stream 2, type 0x00) and sends SETTINGS: QPACK table capacity 65,536, max field section 262,144, 100 blocked streams. It also adds a reserved GREASE setting and a GREASE frame, so servers keep ignoring unknown values as the RFC requires.

[55] 🔗 HTTP/3 HEADERS

Frame	55
Src Port	50280
Dst Port	443
DCID	0130dfc5a047e6acd230b5c5e047ced9b0a6bbf0
Stream	2
Stream	0
Method	GET
Authority	cloudflare-quic.com
Scheme	https
Path	/

[Frame 55] GET / on bidirectional stream 0, the first client request stream. The FIN flag is set because a GET has no body.

The same packet adds a PRIORITY_UPDATE on the control stream (`u=0`, the highest urgency, for stream 0). The headers `pragma: no-cache` and `cache-control: no-cache` mark a reload that bypasses the browser cache.

The request leaves 24.2 ms after Chrome's first QUIC packet, against 45.7 ms over TCP in frame 12.

[56] ⚙️ HTTP/3 SETTINGS

Frame	56
Src Port	443
Dst Port	50280
Stream	3
Max header list size	65536

[Frame 56] One UDP datagram holding two QUIC packets:

- A **Handshake** packet that ACKs Chrome's Handshake packets 2-3.
- A **1-RTT** packet (number 0) that ACKs Chrome's 1-RTT packets 4-5. It also carries HANDSHAKE_DONE, which confirms the handshake for Chrome, and two TLS NewSessionTickets. Each carries an `early_data` extension, which in QUIC means 0-RTT is allowed (RFC 9001 fixes its size field at 0xffffffff). Chrome could use a ticket for 0-RTT on its next visit; this capture has none. Last, Cloudflare's control stream (stream 3) with its SETTINGS: max field section 65,536 plus a GREASE setting.

Cloudflare's short-header packets carry no connection ID: Chrome asked for that in frame 47.

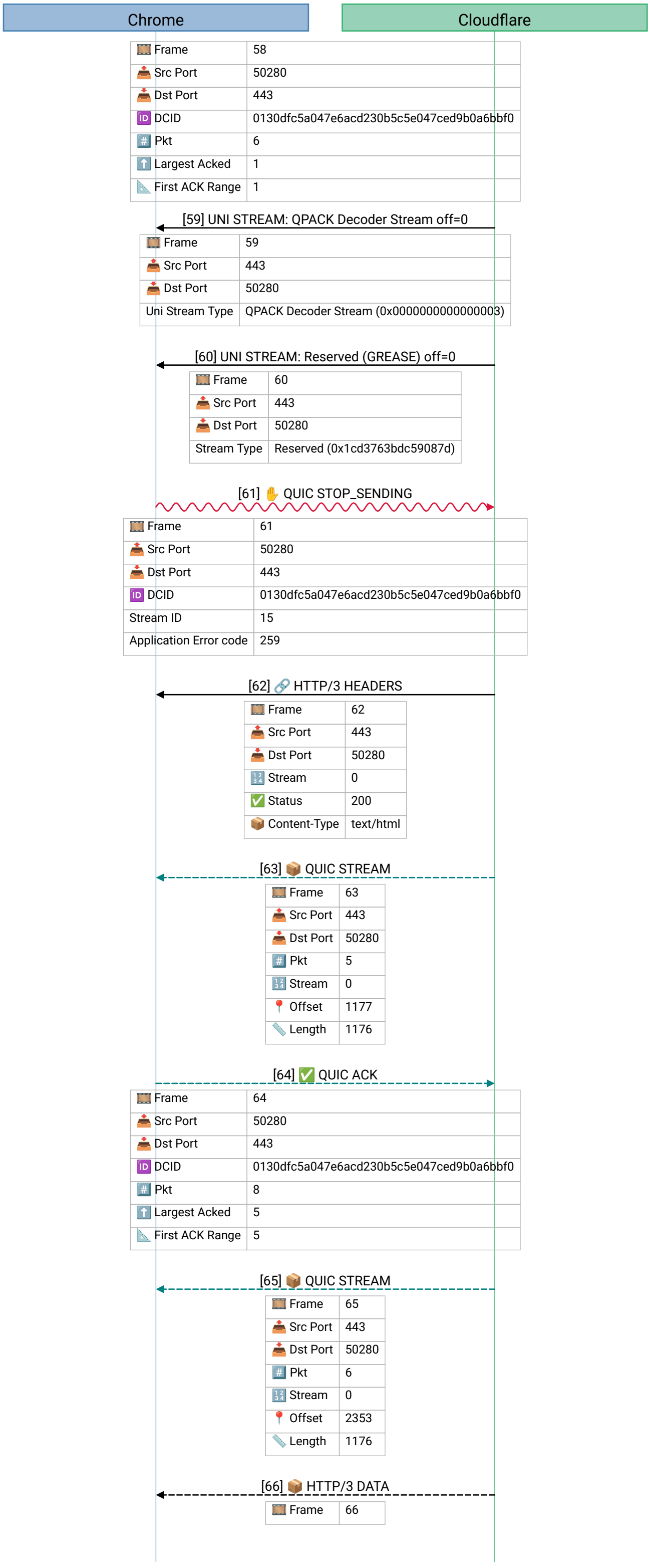
[57] UNI STREAM: QPACK Encoder Stream off=0

Frame	57
Src Port	443
Dst Port	50280
Uni Stream Type	QPACK Encoder Stream (0x0000000000000002)

[Frame 57] Cloudflare opens its **QPACK encoder stream** (stream 7, type 0x02). It sends only the stream type and no instructions: Cloudflare chooses not to use the 65,536-byte dynamic table Chrome offered in frame 54. Chrome can't use one in the other direction, because Cloudflare's SETTINGS offer no table capacity (default 0). Both header blocks in this connection use only QPACK's static table and literal fields.

[58] ✅ QUIC ACK

[Frame 58] 1-RTT ACK of Cloudflare's 1-RTT packets 0-1. Chrome's ACK-only packets also carry a couple of PADDING



bytes.

[Frame 59] Cloudflare's **QPACK decoder stream** (stream 11, type 0x03). With no dynamic table in use, it also carries only its type byte.

[Frame 60] Cloudflare opens a fourth unidirectional stream, 15, with a **reserved stream type** (`0x1cd3763bdc59087d`, of the form 0x1f x N + 0x21). It is GREASE: a deliberately unknown type that checks the client ignores stream types it doesn't know. 26 bytes, then FIN.

[Frame 61] Chrome passes the GREASE test. It ACKs Cloudflare's packets 0-3 and sends STOP_SENDING for stream 15 with error 0x103, `H3_STREAM_CREATION_ERROR`. RFC 9114 tells a receiver to abort reading an unknown stream type, preferably with that code, and never to treat it as a connection error. Cloudflare answers in frame 68.

[Frame 62] 200 OK, 56.6 ms after Chrome's first packet (79.2 ms over TCP). Stream 0 starts with two reserved GREASE frames, one carrying the ASCII text `GREASE is the word`, then the HEADERS frame. The same `alt-svc` header refreshes the HTTP/3 hint for another 24 hours. The body's first bytes follow in this packet.

[Frame 63] Packet 5 continues stream 0 at offset 1,177: response body bytes, with no HTTP/3 frame boundary in this packet.

[Frame 64] ACK of Cloudflare's packets 0-5.

[Frame 65] Packet 6, more of stream 0 (offset 2,353).

[Frame 66] The first HTTP/3 DATA frame, 4,226 bytes, completes here. Wireshark reassembled it from packets 62, 63, 65 and 66. Its length matches the first HTTP/2 DATA frame in frame 27.

Chrome | Cloudflare

Src Port	443
Dst Port	50280
Stream	0
Length	4226

[67] ✅ QUIC ACK

Frame	67
Src Port	50280
Dst Port	443
DCID	0130dfc5a047e6acd230b5c5e047ced9b0a6bbf0
Pkt	9
Largest Acked	7
First ACK Range	7

[Frame 67] ACK of Cloudflare's packets 0-7.

[68] Frame Type: ❌ HTTP/3 Stream Reset (0x0000000000000004)

Frame	68
Src Port	443
Dst Port	50280
Stream ID	15
Application Error code	259
Final Size	26

[Frame 68] Cloudflare ACKs Chrome's packets 6-7 and sends RESET_STREAM for stream 15, echoing error 0x103. Final size 26 matches the 26 bytes sent in frame 60. RFC 9000 makes STOP_SENDING trigger a RESET_STREAM, which SHOULD copy the error code. The GREASE stream is now closed in both directions.

[69] 📦 QUIC STREAM

Frame	69
Src Port	443
Dst Port	50280
Pkt	9
Stream	0
Offset	4390
Length	1176

[Frame 69] Packet 9, stream 0 at offset 4,390.

[70] 📦 HTTP/3 DATA

Frame	70
Src Port	443
Dst Port	50280
Stream	0
Length	2147

[Frame 70] HTTP/3 DATA, 2,147 bytes.

[71] ✅ QUIC ACK

Frame	71
Src Port	50280
Dst Port	443
DCID	0130dfc5a047e6acd230b5c5e047ced9b0a6bbf0
Pkt	10
Largest Acked	9
First ACK Range	6

[Frame 71] ACK of packets 3-9. Chrome no longer lists packets 0-2: frame 68 acknowledged the ACK that reported them, so Cloudflare already knows they arrived (RFC 9000 13.2.4).

[72] 📦 HTTP/3 DATA

Frame	72
Src Port	443
Dst Port	50280
Stream	0
Length	1019

[Frame 72] HTTP/3 DATA, 1,019 bytes.

[73] 📦 QUIC STREAM

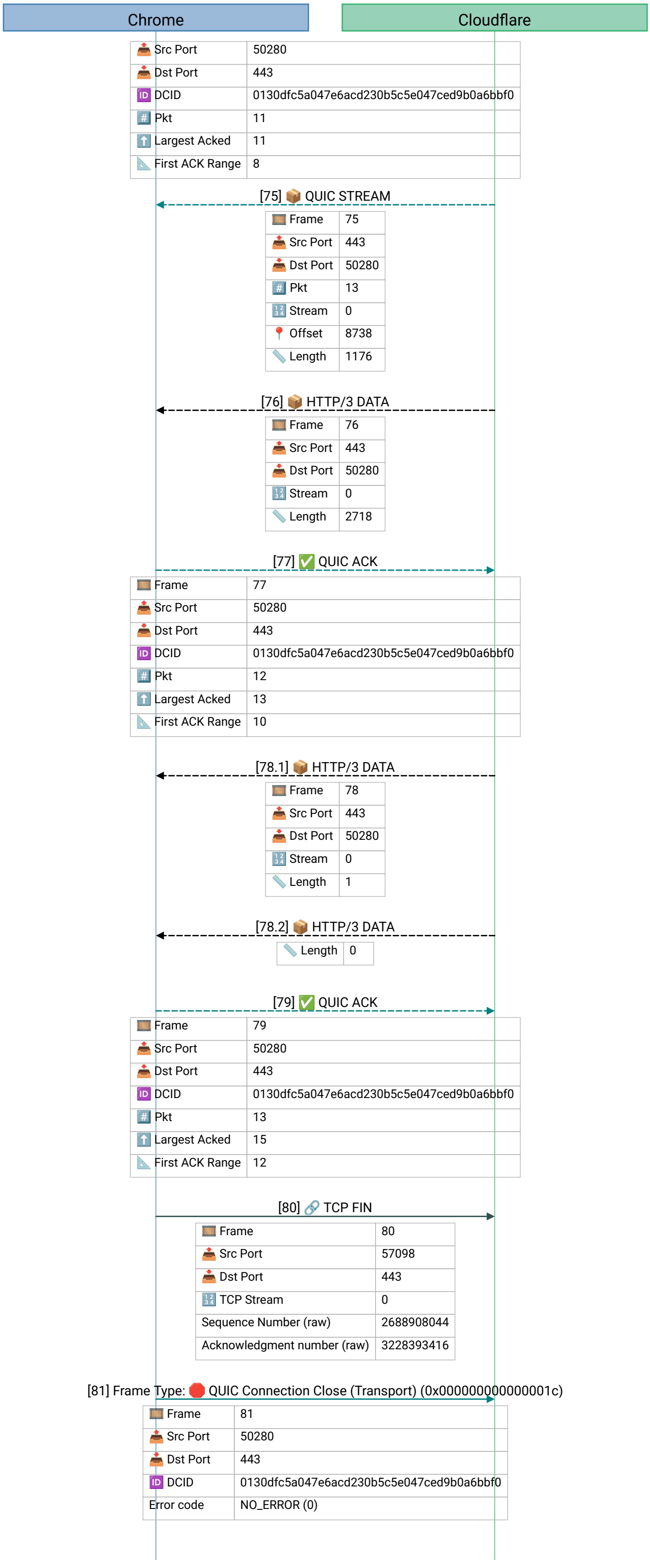
Frame	73
Src Port	443
Dst Port	50280
Pkt	12
Stream	0
Offset	7562
Length	1176

[Frame 73] Packet 12, stream 0 at offset 7,562.

[74] ✅ QUIC ACK

Frame	74

[Frame 74] ACK of packets 3-11.



[Frame 75] Packet 13, stream 0 at offset 8,738.

[Frame 76] HTTP/3 DATA, 2,718 bytes.

[Frame 77] ACK of packets 3-13.

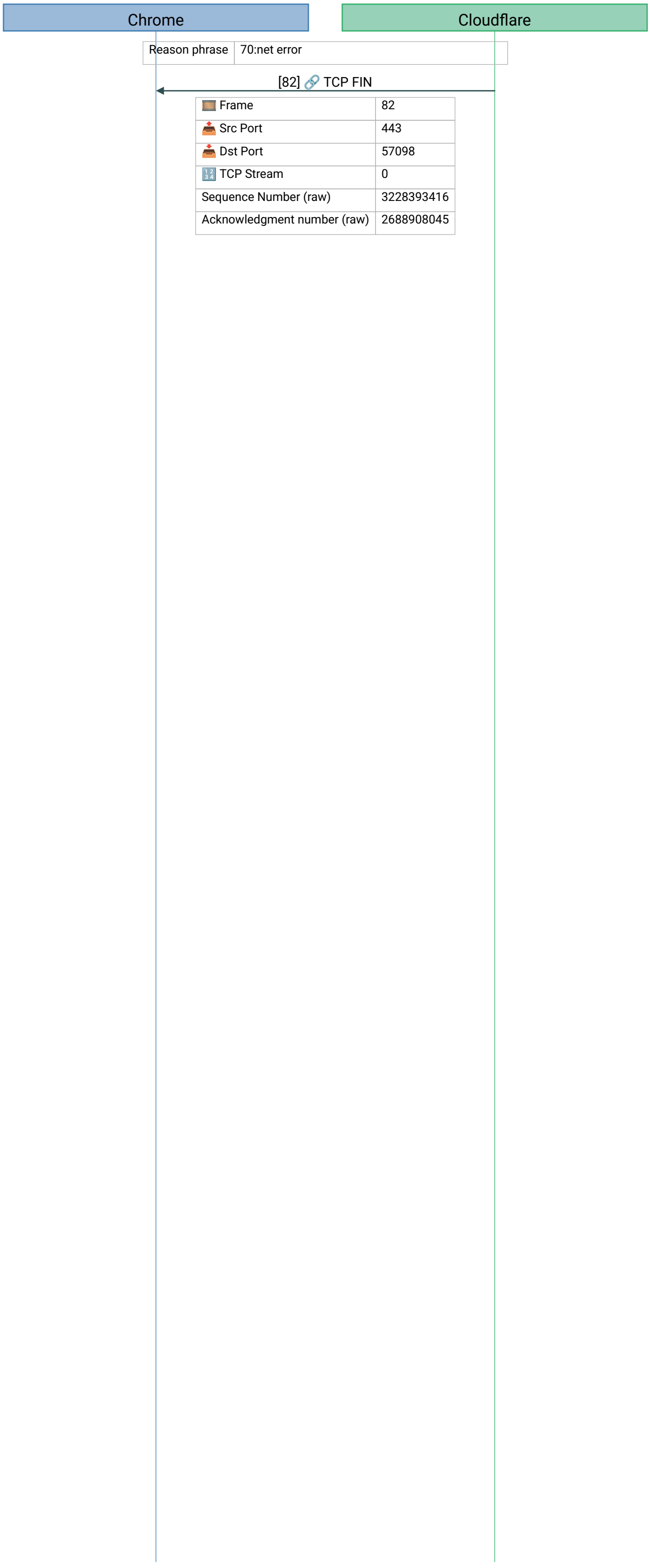
[Frame 78.1] A 1-byte DATA frame, the last of the Brotli-compressed body. Packet 15 sets FIN on stream 0 at final offset 10,288: the response is complete, 70.3 ms after Chrome's first QUIC packet (81.6 ms over TCP).

[Frame 78.2] An empty DATA frame in the same packet. It plays no role in HTTP/3; the stream's FIN is what ends the response. HTTP/2 also ended with an empty DATA frame in frame 44, but there it carried the END_STREAM flag.

[Frame 79] ACK of packets 3-15. Every packet Cloudflare sent has been acknowledged, and none were lost or retransmitted. Both connections now sit idle for 5.3 s.

[Frame 80] Phase 3: teardown. Chrome closes the idle TCP connection from phase 1 with a FIN. It sends the QUIC close 0.4 ms later: both connections end at once.

[Frame 81] CONNECTION_CLOSE, transport error `NO_ERROR`, with reason phrase `70:net error` from Chrome's network stack. The same packet repeats the ACK of packets 3-15. Its ACK Delay field reads 659,814; multiplied by 2^3 (the default exponent), that is 5.28 s, exactly the time since Cloudflare's last packet. The close came 10.8 s after the first packet, well inside the 30 s idle timeout.



[Frame 82] Cloudflare's FIN closes the TCP side. Chrome's final ACK (frame 83) ends the capture. QUIC needs no such exchange: one CONNECTION_CLOSE ends the connection.