



# Over, Under, Around, and Through: A Detailed Comparison of QUIC and HTTP/3 Application Mapping vs. Protocol Encapsulation

**Lucas Pardue**

Senior Software Engineer, Cloudflare  
QUIC Working Group Co-chair, IETF

QUIC is not TCP

QUIC is not TLS

QUIC is not HTTP

HTTP/3 is not  
HTTP/2



QUIC is not just  
the web over  
UDP



QUIC is QUIC

HTTP/3 is HTTP/3

# Thank you for your time

**Lucas Pardue**

Senior Software Engineer, Cloudflare  
QUIC Working Group Co-chair, IETF



QUIC is a secure transport protocol

QUIC is what you make it

# Where to start

<https://quicwg.org>

## Core Specifications

- [RFC 8999](#) - Version-Independent Properties of QUIC
- [RFC 9000](#) - QUIC: A UDP-Based Multiplexed and Secure Transport
- [RFC 9001](#) - Using TLS to Secure QUIC
- [RFC 9002](#) - QUIC Loss Detection and Congestion Control

## QUIC Extensions

- [RFC 9221](#) - An Unreliable Datagram extension to QUIC
- [RFC 9287](#) - Greasing the QUIC Bit
- [RFC 9368](#) - Compatible Version Negotiation of QUIC
- [RFC 9369](#) - QUIC Version 2

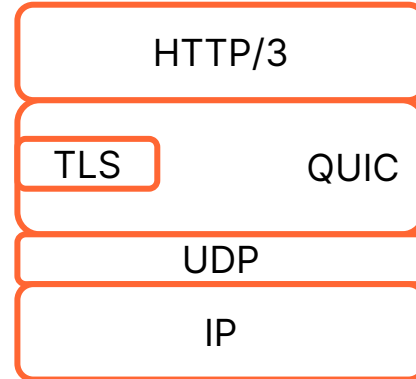
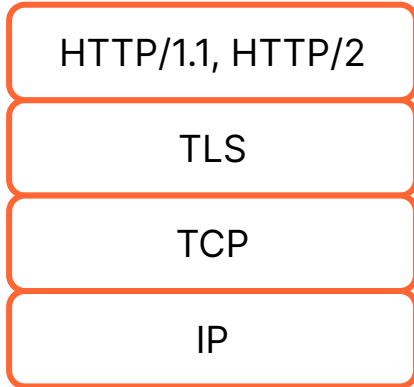
## Applicability and Manageability

- [RFC 9308](#) - Applicability of the QUIC Transport Protocol
- [RFC 9312](#) - Manageability of the QUIC Transport Protocol

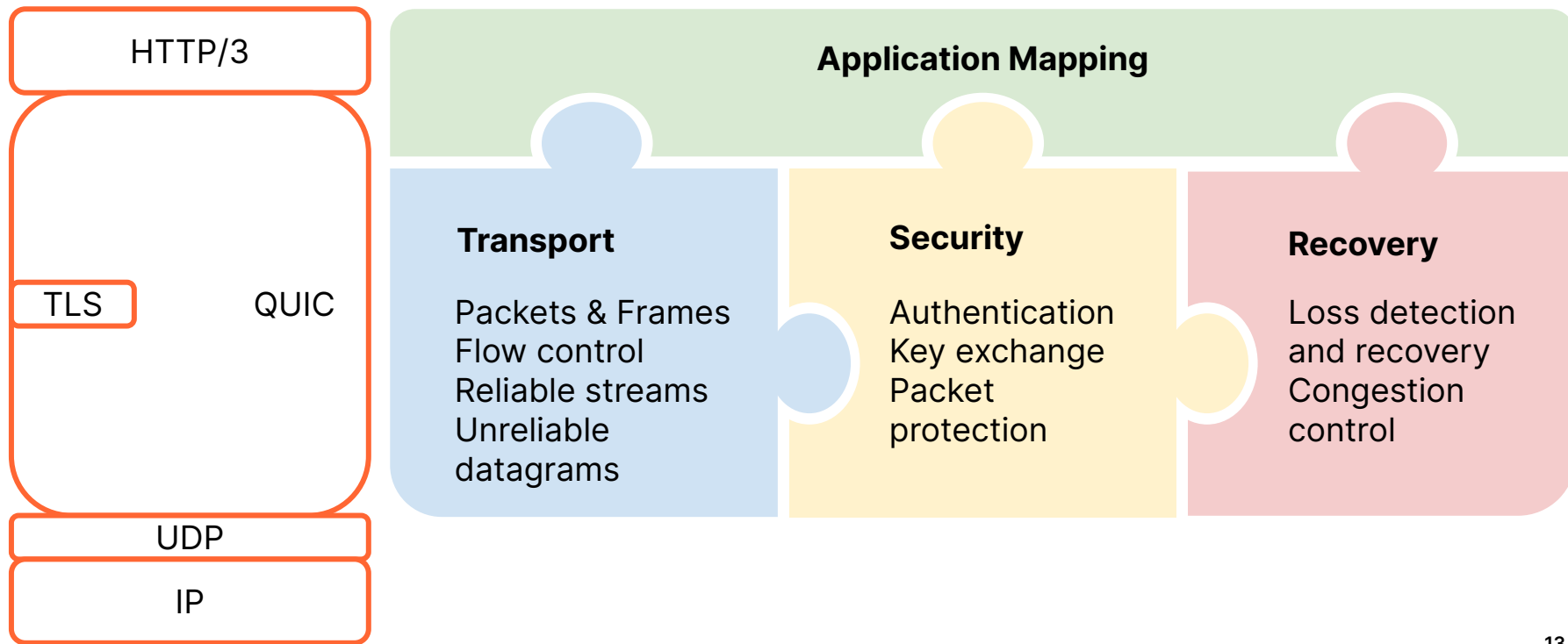
## HTTP/3 and QPACK

- [RFC 9114](#) - HTTP/3
- [RFC 9204](#) - QPACK

# Obligatory Diagram



# Interlocking Pieces



# Who are you?



**A happy person that doesn't need to worry about plumbing**

No action required. Stay happy.

Unless you hit an issue you can't understand.

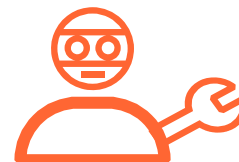
See my talk "[Layer 4 ¾: Fantastic quirks and where to find them](#)".



**Someone that operates systems and/or a network, that wants to observe/manage QUIC stuff.**

Read [RFC 9312](#) - "Manageability of the QUIC Transport Protocol".

See the next few slides...



**Someone that likes building new things, possibly using QUIC stuff**

Read [RFC 9308](#) - "Applicability of the QUIC Transport Protocol"

Stay tuned. But also pay attention to the "boring bits" next.

# Back to the start



# We always start with a handshake

- RFC 9000, [Section 7](#) - Cryptographic and Transport Handshake
- RFC 9001 - Using TLS to Secure QUIC
- RFC 9312, [Section 2.4](#) - The QUIC Handshake

Recommend Martin Thomson's talk at the [IETF 115 Tech Deep Dive](#)

# Don't mix up packets types with adjectives

- QUIC has several packet types
- A QUIC connection handshake requires exchanging
  - **Initial** packets
    - Not the first packet!
  - **Handshake** packets
- Post-handshake steady state
  - Short-header packets
    - a.k.a short packets
    - a.k.a 1-RTT packets

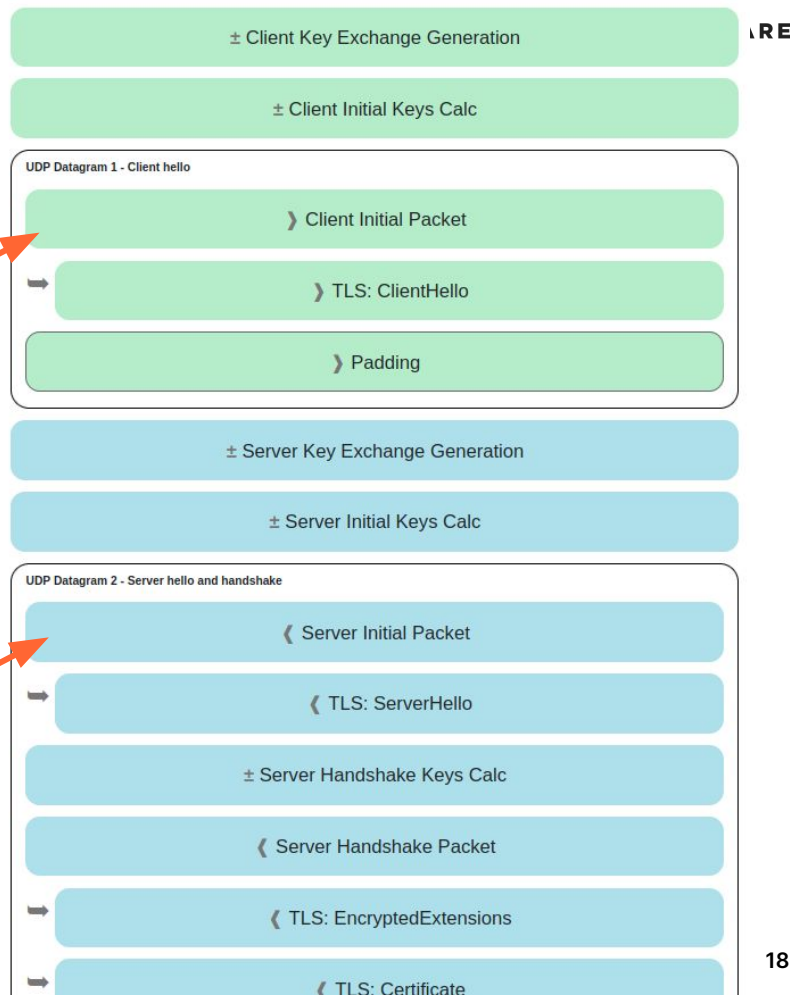
# The Illustrated Guide

Sometimes it helps to look at things differently than the specs.

<https://quic.xargs.org>

Client Initial

Server Initial



# Client Initial expanded

UDP Datagram 1 - Client hello

Client Initial Packet ×

The session begins with the client sending an "Initial" packet. This packet contains the "ClientHello" TLS record, used to begin the TLS 1.3 encrypted session.

Annotations

```
cd 00 00 00 01 08 00 01 02 03 04 05 06 07 05 63 5f 63 69 64 00 41 03 98 1c
36 a7 ed 78 71 6b e9 71 1b a4 98 b7 ed 86 84 43 bb 2e 0c 51 4d 4d 84 8e ad
cc 7a 00 d2 5c e9 f9 af a4 83 97 80 88 de 83 0b e6 8c 0b 32 a2 45 95 d7 81
3e a5 41 4a 91 99 32 9a 6d 9f 7f 76 0d d8 bb 24 0b f3 f5 3d 9a 77 fb b7 b3
95 b8 d6 6d 78 79 a5 1f e5 9e f9 60 1f 79 99 8e b3 56 8e 1f dc 78 9f 64 0a
ca b3 85 8a 82 ef 29 30 fa 5c e1 4b 5b 9e a9 bd b2 9f 45 72 da 85 aa 3d ef
39 b7 ef af ff a0 74 b9 26 79 70 d5 0b 5d 07 84 2e 49 bb a3 bc 78 7f f2 95
d6 ae 3b 51 43 05 f1 02 af e5 a0 47 b3 fb 4c 99 eb 92 a2 74 d2 44 d6 04 92
c9 e2 e6 e2 12 ce f0 f9 e3 f6 2e df 09 55 e7 1c 76 8a a6 bb 3c d8 0b bb 37
55 c8 b7 eb ee 32 71 2f 40 f2 24 51 19 48 70 21 b4 b8 4e 15 65 e3 ca 31 96
7a c8 60 4d 40 32 17 0d ec 28 0a ee fa 09 5d 08 b3 b7 24 1e f6 64 6a 6c 86
e5 c6 2c e0 8b e0 99
```

Decryption ↓

```
06 00 40 ee 01 00 00 ea 03 03 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e
0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 00 00 06 13 01 13 02 13
03 01 00 00 bb 00 00 18 00 16 00 00 13 65 78 61 6d 70 6c 05 2e 75 6c 66
68 65 69 6d 2e 6e 65 74 00 0a 00 08 06 06 0d 10 07 17 08 18 00 10 00 0b 00
09 08 76 69 6e 67 2f 31 2e 39 00 0d 00 14 00 12 04 03 08 04 04 01 05 03 08
05 05 01 08 06 06 01 02 01 00 33 00 26 00 24 00 1d 00 28 35 80 72 06 36 58
00 d1 ae ea 32 9a df 91 21 38 38 51 ed 21 a2 8e 3b 75 e9 65 00 d2 cd 16 62
54 00 2d 06 02 01 01 00 2b 00 03 02 03 04 00 39 00 31 03 04 80 00 ff f7 04
04 00 a0 00 00 05 04 80 10 00 00 06 04 80 10 00 00 07 04 80 10 00 00 00 01
0a 09 01 0a 0a 01 03 0b 01 19 0f 05 63 5f 63 69 64
```

→ TLS: ClientHello

→ Padding

UDP Datagram 1 - Client hello

Client Initial Packet

→ TLS: ClientHello ×

The encrypted session begins with the client saying "Hello". The client provides information including the following:

- client random data (used later in the handshake)
- a list of cipher suites that the client supports
- a public key for key exchange
- protocol versions that the client can support

Annotations

```
01 00 00 ea 03 03 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11
12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f 00 00 06 13 01 13 02 13 03 01
00 00 bb 00 00 18 00 16 00 00 13 65 78 61 6d 70 6c 05 2e 75 6c 66 68
65 69 6d 2e 6e 65 74 00 0a 00 08 06 06 0d 10 07 17 08 18 00 10 00 0b 00
09 08 76 69 6e 67 2f 31 2e 39 00 0d 00 14 00 12 04 03 08 04 04 01 05 03
08 05 05 01 08 06 06 01 02 01 00 33 00 26 00 24 00 1d 00 20 35 80 72 d6
36 58 80 d1 ae ea 32 9a df 91 21 38 38 51 ed 21 a2 8e 3b 75 e9 65 00 d2
cd 16 62 54 00 2d 00 02 01 01 00 2b 00 03 02 03 04 00 39 00 31 03 04 80
00 ff f7 04 04 80 a0 00 00 05 04 80 10 00 00 06 04 80 10 00 00 07 04 80
10 00 00 00 01 0a 09 01 0a 0a 01 03 0b 01 19 0f 05 63 5f 63 69 64
```

→ Padding

UDP Datagram 1 - Client hello

Client Initial Packet

→ TLS: ClientHello

→ Padding ×

Any datagram sent by the client that contains an Initial packet must be padded to a length of 1200 bytes. This library does it by appending nul bytes to the datagram.

Annotations

```
00 00 00 00 00 00 00 00 ... snip ... 00 00 00 00 00 00 00 00
```

Padding Bytes

Padding this packet to a size of 1200 bytes serves two purposes:

- **Path MTU validation** - Any IPv4 host or router is allowed to drop packets that exceed their MTU limit, to a minimum of 576 bytes. The vast majority of the internet has a much higher MTU (typically 1500 bytes). A higher packet size will increase throughput and performance. Given these realities QUIC chooses a minimum size constraint of 1200 bytes, which should traverse the vast majority of real networks (including tunneled networks) without being dropped for size. To prevent a scenario where a connection is established successfully with smaller packets but then starts timing out once larger packets are sent, the initial packets are padded to a length of 1200 bytes to prove that the end-to-end path will allow packets of that size.
- **Amplification Attack Mitigation** - There is a class of network attack in which an attacker can send a small amount of traffic to an innocent third party which replies with a much larger amount of traffic directed at the target. In the case of QUIC this could be done with IP address spoofing, and would cause QUIC servers to reply to small Initial datagrams with much larger Handshake responses. To help mitigate this, QUIC servers are forbidden from replying to a client with more than 3 times the traffic that was sent to it, until the server has received some proof from the client that it's at the given address (such as round-trip data originally from the server). Adding padding to this Initial datagram gives the server a "byte budget" to perform handshake responses without exceeding this 3x limit.

# Transport parameters

QUIC Transport Parameters are a TLS extension sent in ClientHello and ServerHello.

They communicate the capabilities of the endpoint that sends them.

Registry is

<https://www.iana.org/assignments/quic/quic.xhtml>

```
00 39 00 31 03 04 80 00 ff f7 04 04 80 a0 00 00 05 04 80 10 00 00 06 04
80 10 00 00 07 04 80 10 00 00 08 01 0a 09 01 0a 0a 01 03 0b 01 19 0f 05
63 5f 63 69 64
```

## Extension - QUIC Transport Parameters

The client's configuration values for the QUIC connection are given here. They are put into this record instead of the headers of the Initial packet because all data in TLS records is protected from tampering by malicious actors.

The following QUIC parameters are set in the data below:

- max\_udp\_payload\_size: 65527
- initial\_max\_data: 10485760
- initial\_max\_stream\_data\_bidi\_local: 1048576
- initial\_max\_stream\_data\_bidi\_remote: 1048576
- initial\_max\_stream\_data\_uni: 1048576
- initial\_max\_streams\_bidi: 10
- initial\_max\_streams\_uni: 10
- ack\_delay\_exponent: 3
- initial\_source\_connection\_id: "c\_cid"

# Follow-along examples



<https://github.com/LPardue/sreconemea2023>

"localhost-good.pcapng"

To successfully dissect QUIC packets, Wireshark 3.4.x and onwards.

No.	Time	Source	Src port	Destination	Dst port	Protocol	Length	Info
1	0.000000000	127.0.0.1	44746	127.0.0.1	4433	QUIC	1242	Initial, DCID=f7cc19997f578525a476d84d1395c6a1, SCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4, PKN: 0, CRYPTO
2	0.003059492	127.0.0.1	4433	127.0.0.1	44746	QUIC	1242	Handshake, DCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4, SCID=770f4e294bf6006863eca225ce89c6c1f72d0963
3	0.003870636	127.0.0.1	44746	127.0.0.1	4433	QUIC	1242	Handshake, DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, SCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4
4	0.004193870	127.0.0.1	4433	127.0.0.1	44746	QUIC	491	Handshake, DCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4, SCID=770f4e294bf6006863eca225ce89c6c1f72d0963
5	0.005004163	127.0.0.1	44746	127.0.0.1	4433	QUIC	256	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963
6	0.005077476	127.0.0.1	44746	127.0.0.1	4433	QUIC	86	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963
7	0.005130464	127.0.0.1	44746	127.0.0.1	4433	QUIC	86	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963
8	0.005179016	127.0.0.1	44746	127.0.0.1	4433	QUIC	157	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963
9	0.005225358	127.0.0.1	44746	127.0.0.1	4433	QUIC	111	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963
10	0.006516110	127.0.0.1	4433	127.0.0.1	44746	QUIC	622	Protected Payload (KP0), DCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4
11	0.006960566	127.0.0.1	44746	127.0.0.1	4433	QUIC	85	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963
12	0.007254285	127.0.0.1	4433	127.0.0.1	44746	QUIC	86	Protected Payload (KP0), DCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4
13	0.007468118	127.0.0.1	44746	127.0.0.1	4433	QUIC	85	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963
14	0.007688664	127.0.0.1	4433	127.0.0.1	44746	QUIC	86	Protected Payload (KP0), DCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4
15	0.007884861	127.0.0.1	44746	127.0.0.1	4433	QUIC	85	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963
16	0.008144120	127.0.0.1	4433	127.0.0.1	44746	QUIC	150	Protected Payload (KP0), DCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4
17	0.008559762	127.0.0.1	44746	127.0.0.1	4433	QUIC	90	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963





```
No. Time Source Src port Destination Dst port Protocol Length Info
1 0.00000000 127.0.0.1 43959 127.0.0.1 4433 QUIC 1242 Initial, DCID=6c94d2c299cb
4)
  Frame 1: 1242 bytes on wire (9936 bits), 1242 bytes captured (9936 bits) on interface lo, id 0
  Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)
  Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
  User Datagram Protocol, Src Port: 43959, Dst Port: 4433
  QUIC IETF
    QUIC Connection information
      [Packet Length: 350]
      1... .. = Header Form: Long Header (1)
      .1. .... = Fixed Bit: True
      ..00 .... = Packet Type: Initial (0)
      .... .. = Reserved: 0
      .... .. = Packet Number Length: 1 bytes (0)
      Version: 1 (0x00000001)
      Destination Connection ID Length: 16
      Destination Connection ID: 6c94d2c299cbff6253a202bcb20ceb42
      Source Connection ID Length: 20
      Source Connection ID: 9463b9d6695a7b2d189da2871fc255977bc7c6f8
      Token Length: 0
      Length: 304
      Packet Number: 0
      Payload: 3f13a4c1d4e69e4bdd549adc3455a3b534813cf001fdf2eb835ffc5a5577628012f1b1b8f.
  TLSv1.3 Record Layer: Handshake Protocol: Client Hello
    Frame Type: CRYPTO (0x0000000000000000)
      Offset: 0
      Length: 283
      Crypto Data
    Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 279
      Version: TLS 1.2 (0x0303)
      Random: 8481481ba7a7b353cd6e341d71441c47917b45d620fa5cbe98a5cb55273580
      Session ID Length: 0
      Cipher Suites Length: 6
      Cipher Suites (3 suites)
      Compression Methods Length: 1
      Compression Methods (1 method)
      Extensions Length: 232
      Extension: supported_groups (len=8)
      Extension: application_layer_protocol_negotiation (len=61)
      Extension: signature_algorithms (len=20)
      Extension: key_share (len=38)
      Extension: psk_key_exchange_modes (len=2)
      Extension: supported_versions (len=2)
    Extension: quic_transport_parameters (len=72)
      Type: quic_transport_parameters (57)
      Length: 72
      Parameter: max_idle_timeout (len=4) 30000 ms
      Parameter: max_udp_payload_size (len=2) 1350
      Parameter: initial_max_data (len=4) 10000000
      Parameter: initial_max_stream_data_bidi_local (len=4) 1000000
      Parameter: initial_max_stream_data_bidi_remote (len=4) 1000000
      Parameter: initial_max_stream_data_uni (len=4) 10000000
      Parameter: initial_max_streams_bidi (len=2) 100
      Parameter: initial_max_streams_uni (len=2) 100
      Parameter: ack_delay_exponent (len=1)
      Parameter: GREASE (len=1) 25
      Parameter: disable_active_migration (len=0)
      Parameter: initial_source_connection_id (len=20)
  QUIC IETF
00c0 93 81 99 f1 e0 78 00 2d 00 02 01 01 00 2b 00 03 .....X-.-.....+
00d0 02 03 04 00 39 00 48 01 04 80 00 75 30 03 02 45 ...-9.H-...u@-E
00e0 46 04 04 80 98 96 80 05 04 80 0f 42 40 06 04 80 ..B.....B@...
00f0 0f 42 40 0f 04 80 0f 42 40 08 02 40 04 09 02 40 ..B@...B@...@...@
0100 04 0a 01 03 00 01 19 0c 00 0f 14 94 03 b9 d6 69 d.....c@...1
```

Client Initial

ClientHello

ALPN

Transport Parameters

```
Extension: quic_transport_parameters (len=72)
Type: quic_transport_parameters (57)
Length: 72
  Parameter: max_idle_timeout (len=4) 30000 ms
  Parameter: max_udp_payload_size (len=2) 1350
  Parameter: initial_max_data (len=4) 10000000
  Parameter: initial_max_stream_data_bidi_local (len=4) 1000000
  Parameter: initial_max_stream_data_bidi_remote (len=4) 1000000
  Parameter: initial_max_stream_data_uni (len=4) 10000000
  Parameter: initial_max_streams_bidi (len=2) 100
  Parameter: initial_max_streams_uni (len=2) 100
  Parameter: ack_delay_exponent (len=1)
  Parameter: max_ack_delay (len=1) 25
  Parameter: disable_active_migration (len=0)
  Parameter: initial_source_connection_id (len=20)
```



# Application-Layer Protocol Negotiation ALPN

## RFC 7301

Client and server negotiate what Application Protocol to use across the secure transport.

Meaning one port number can speak many protocols.

Client offers a list:

<https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml#alpn-protocol-ids>

```

    ▾ Extension: application_layer_protocol_negotiation (len=61)
      Type: application_layer_protocol_negotiation (16)
      Length: 61
      ALPN Extension Length: 59
      ▾ ALPN Protocol
        ALPN string length: 2
        ALPN Next Protocol: h3
        ALPN string length: 5
        ALPN Next Protocol: h3-29
        ALPN string length: 5
        ALPN Next Protocol: h3-28
        ALPN string length: 5
        ALPN Next Protocol: h3-27
        ALPN string length: 10
        ALPN Next Protocol: hq-interop
        ALPN string length: 5
        ALPN Next Protocol: hq-29
        ALPN string length: 5
        ALPN Next Protocol: hq-28
        ALPN string length: 5
        ALPN Next Protocol: hq-27
        ALPN string length: 8
        ALPN Next Protocol: http/0.9
  
```

0040	17 00 18 00 10 00 3d 00	3b 02 68 33 05 68 33 2d	.....= ; h3 h3-
0050	32 39 05 68 33 2d 32 38	05 68 33 2d 32 37 0a 68	29 h3-28 h3-27 h
0060	71 2d 69 6e 74 65 72 6f	70 05 68 71 2d 32 39 05	q-intero p hq-29
0070	68 71 2d 32 38 05 68 71	2d 32 37 08 68 74 74 70	hq-28 hq -27 http
0080	2f 30 2e 39 00 0d 00 14	00 12 04 03 08 04 04 01	/0.9.....

```
2 0.003172573 127.0.0.1 4433 127.0.0.1 43959 QUIC 1242 Handshake, DCID=9463b9d6695a7b2d
Frame 2: 1242 bytes on wire (9936 bits), 1242 bytes captured (9936 bits) on interface lo, id 0
Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 4433, Dst Port: 43959
QUIC IETF
  QUIC Connection information
  [Packet Length: 167]
  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: 20
  Destination Connection ID: 9463b9d6695a7b2d189da2871fc255977bc7c6f8
  Source Connection ID Length: 20
  Source Connection ID: 78015def011d1adf3af94c44067955dd4d52fc70
  Token Length: 0
  Length: 117
  Packet Number: 0
  Payload: 1e8586cdeb719b35f6999cd9e10939fd3ecdee3de6ec324ce3dabcadb8661847f7b67a8c...
ACK
  Frame Type: ACK (0x0000000000000000)
  Largest Acknowledged: 0
  ACK Delay: 305
  ACK Range Count: 0
  First ACK Range: 0
TLSv1.3 Record Layer: Handshake Protocol: Server Hello
  Frame Type: CRYPTO (0x0000000000000006)
  Offset: 0
  Length: 90
  Crypto Data
  Handshake Protocol: Server Hello
  Handshake Type: Server Hello (2)
  Length: 86
  Version: TLS 1.2 (0x0303)
  Random: 2011206923ba555bdb6c7d5469904889e16f80354689e4f4feb20c0448051de8
  Session ID Length: 0
  Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
  Compression Method: null (0)
  Extensions Length: 46
  Extension: key_share (len=36)
  Extension: supported_versions (len=2)
QUIC IETF
[Packet Length: 1033]
1... .. = Header Form: Long Header (1)
.1.. .. = Fixed Bit: True
..10 .. = Packet Type: Handshake (2)
Version: 1 (0x00000001)
Destination Connection ID Length: 20
Destination Connection ID: 9463b9d6695a7b2d189da2871fc255977bc7c6f8
Source Connection ID Length: 20
Source Connection ID: 78015def011d1adf3af94c44067955dd4d52fc70
Length: 904
[Expert Info (Warning/Decryption): Failed to create decryption context: Secrets are not available]
Remaining Payload: 85aebbc5e92bcc899017e280cf5706cea6976ccd9ef2f41269e7d94898071239d1b322e..
```

Server Initial

ServerHello

Server Handshake

Secrets not available!

## Keys or endpoint logging needed to see the full picture

From even a very early stage in a QUIC connection, packets are encrypted with a session key.

- SSLKEYLOGFILE is available in many, but not all, implementations.
  - Dump the symmetrical session keys to a file. Usable on client or server.
- Or use a new standard for endpoint logging - qlog.
  - Analyse it with the popular browser tool [qvis](#).
  - Or build your own analysis tooling using libraries such as Cloudflare's [qlog Rust crate](#).

# Decrypted traffic

<https://wiki.wireshark.org/TLS>

Server picks on ALPN value.  
This is the negotiated  
application protocol.

In this case - "h3" which is  
HTTP/3.

```
2 0.003172573 127.0.0.1 4433 127.0.0.1 43959 QUIC 1242 Handshake, DCI
└─ QUIC IETF
  └─ QUIC Connection Information
    [Packet Length: 167]
    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: 20
    Destination Connection ID: 9463b9d6695a7b2d189da2871fc255977bc7c6f8
    Source Connection ID Length: 20
    Source Connection ID: 78015def011d1adf3af94c44067955dd4d52fc70
    Token Length: 0
    Length: 117
    Packet Number: 0
    Payload: 1e0586cdeb719b35f6999cd9e10939fd3ecdee3de6ec324ce3dabcadb86618477b67a8c...
  └─ ACK
  └─ TLSv1.3 Record Layer: Handshake Protocol: Server Hello
    Frame Type: CRYPTO (0x0000000000000006)
    Offset: 0
    Length: 90
    Crypto Data
    └─ Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 86
      Version: TLS_1.2 (0x0303)
      Random: 2011206923ba555bdb6c7d5469904889616f80354689e4fafeb20c0448051de8
      Session ID Length: 0
      Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
      Compression Method: null (0)
      Extensions Length: 46
      └─ Extension: key_share (len=36)
        └─ Extension: supported_versions (len=2)
  └─ QUIC IETF
    [Packet Length: 1033]
    1... .. = Header Form: Long Header (1)
    1... .. = Fixed Bit: True
    ..10 ... = Packet Type: Handshake (2)
    .... 00.. = Reserved: 0
    .... 00.. = Packet Number Length: 1 bytes (0)
    Version: 1 (0x00000001)
    Destination Connection ID Length: 20
    Destination Connection ID: 9463b9d6695a7b2d189da2871fc255977bc7c6f8
    Source Connection ID Length: 20
    Source Connection ID: 78015def011d1adf3af94c44067955dd4d52fc70
    Length: 984
    Packet Number: 0
    Payload: aebbc5e92bcc899017e280cf5706cea6976ccd9ef2f41269e7d94898071239d1b32265...
  └─ TLSv1.3 Record Layer: Handshake Protocol: Multiple Handshake Messages
    Frame Type: CRYPTO (0x0000000000000006)
    Offset: 0
    Length: 963
    Crypto Data
    └─ Handshake Protocol: Encrypted Extensions
      Handshake Type: Encrypted Extensions (8)
      Length: 105
      Extensions Length: 103
      └─ Extension: application_layer_protocol_negotiation (len=5)
        Length: 5
        ALPN Extension Length: 3
        └─ ALPN Protocol
          ALPN string length: 2
          ALPN Next Protocol: h3
```

Server  
Handshake

Encrypted  
Extensions

ALPN

# Decrypted post-handshake QUIC and HTTP/3

Protocol	Length	Info
QUIC	1242	Initial, DCID=f7cc19997f578525a476d84d1395c6a1, SCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4, PKN: 0, CRYPTO
QUIC	1242	Handshake, DCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4, SCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 0, CRYPTO
QUIC	1242	Handshake, DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, SCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4, PKN: 0, ACK
QUIC	491	Handshake, DCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4, SCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 1, CRYPTO
HTTP3	256	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 0, NCI, STREAM(2)
HTTP3	86	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 1, STREAM(6)
HTTP3	86	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 2, STREAM(10)
HTTP3	157	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 3, STREAM(0), HEADERS: GET /index.html
HTTP3	111	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 4, STREAM(14)
HTTP3	622	Protected Payload (KP0), DCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4, PKN: 0, ACK, NCI, DONE, CRYPTO, STREAM(3)
QUIC	85	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 5, ACK
HTTP3	86	Protected Payload (KP0), DCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4, PKN: 1, STREAM(7)
QUIC	85	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 6, ACK
HTTP3	86	Protected Payload (KP0), DCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4, PKN: 2, STREAM(11)
QUIC	85	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 7, ACK
HTTP3	150	Protected Payload (KP0), DCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4, PKN: 3, STREAM(0), HEADERS: 404 Not Found, DATA
QUIC	90	Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 8, CC

HTTP GET /index.html request

HTTP 404 response

“

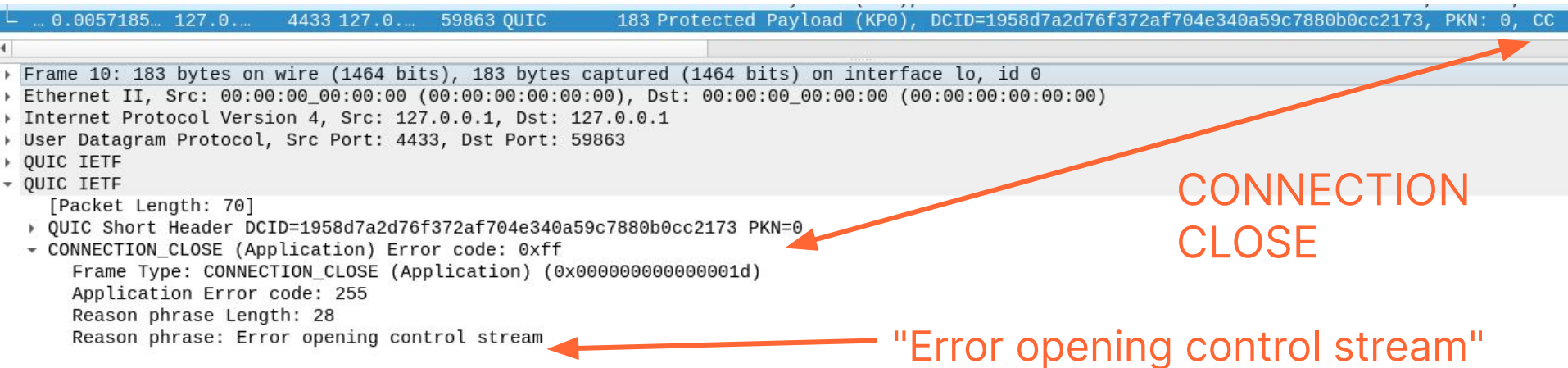
Something broke and I have no idea why”

Firstname Lastname  
Position Title, Company Name

# CONNECTION\_CLOSE and error codes

QUIC provides explicit close using a CONNECTION\_CLOSE frame.

<https://github.com/LPardue/sreconemea2023/blob/main/localhost-0-streams-uni.pcapng>



... 0.0057185... 127.0... 4433 127.0... 59863 QUIC 183 Protected Payload (KP0), DCID=1958d7a2d76f372af704e340a59c7880b0cc2173, PKN: 0, CC

Frame 10: 183 bytes on wire (1464 bits), 183 bytes captured (1464 bits) on interface lo, id 0

Ethernet II, Src: 00:00:00\_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00\_00:00:00 (00:00:00:00:00:00)

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

User Datagram Protocol, Src Port: 4433, Dst Port: 59863

QUIC IETF

QUIC IETF

[Packet Length: 70]

- QUIC Short Header DCID=1958d7a2d76f372af704e340a59c7880b0cc2173 PKN=0
- CONNECTION\_CLOSE (Application) Error code: 0xff
  - Frame Type: CONNECTION\_CLOSE (Application) (0x000000000000001d)
  - Application Error code: 255
  - Reason phrase Length: 28
  - Reason phrase: Error opening control stream

CONNECTION CLOSE

"Error opening control stream"



**The best thing about QUIC...**

**Connection Identifiers!!!11!**

# CIDs

Whether the packets are encrypted or not, connection IDs are visible. And they can be used for traffic steering / load balancing.

```
Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 0, NCI, STREAM(2)
Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 1, STREAM(6)
Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 2, STREAM(10)
Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 3, STREAM(0), HEADERS: GET /index.html
Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 4, STREAM(14)
Protected Payload (KP0), DCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4, PKN: 0, ACK, NCI, DONE, CRYPTO, STREAM(3)
```

Client → Server DCID: 770f4e294bf6006863eca225ce89c6c1f72d0963

Server → Client DCID: d24c3e72fee7cd93a210446c95fd1fd43cd53ff4

<https://datatracker.ietf.org/doc/html/draft-ietf-quic-load-balancers>

# Streams (the second-best thing)

# QUIC Streams

One of the superpowers is stream multiplexing and concurrency:

- Streams are a lightweight, ordered, and reliable byte stream.
- Can be between 0 and  $2^{62}-1$  bytes long.
- Can be created by either client or server.
- Are flow controlled.
- Can be terminated without breaking the connection using RST\_STREAM.
- Can have termination requested using STOP\_SENDING.
- Are not globally ordered. Loss or reordering independence.
- Have concurrency limits placed on them by Transport Parameters and MAX\_STREAMS.

# QUIC Stream IDs and Types

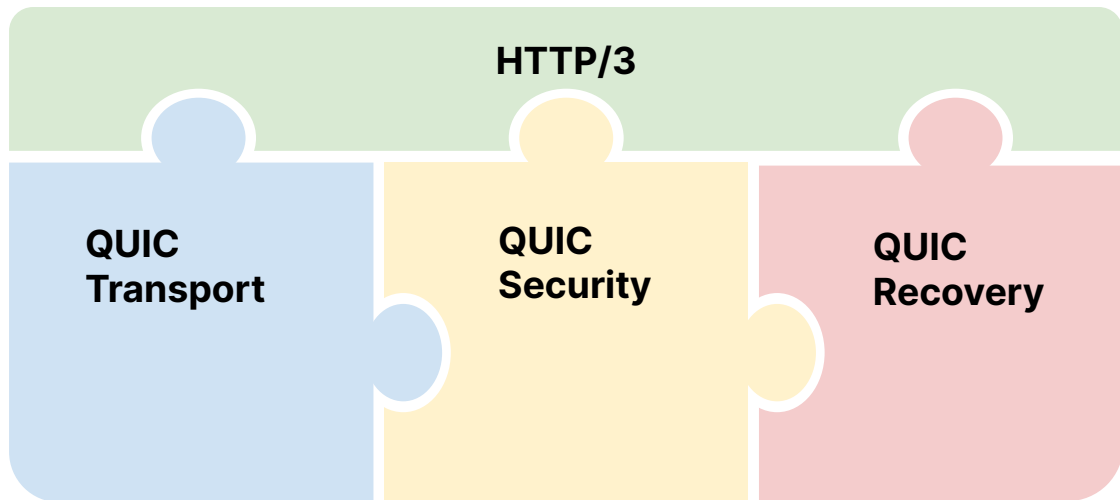
- Unidirectional or bidirectional.
- Each stream has a unique identifier within a connection.
  - 62-bit integer (0 to  $2^{62}-1$ )
- Least significant bit (0x01) identifies the initiator of the stream - client or server.
- Second least significant bit (0x02) distinguishes between bidirectional and unidirectional.

Bits	Stream Type
0x00	Client-Initiated, Bidirectional
0x01	Server-Initiated, Bidirectional
0x02	Client-Initiated, Unidirectional
0x03	Server-Initiated, Unidirectional

# Applications have to decide how streams are used

QUIC provides streams as a **transport service** but has no opinion how they are used.

Application mappings like HTTP/3 ([RFC 9114](#)) or DNS over QUIC ([RFC 9250](#)) describe how application-level concepts messages utilise QUIC streams.



## HTTP/3 and its stream usage

HTTP is a request/response protocol that requires a reliable delivery beneath it.

HTTP/3 is the **application-mapping** of HTTP semantics to QUIC transport services.

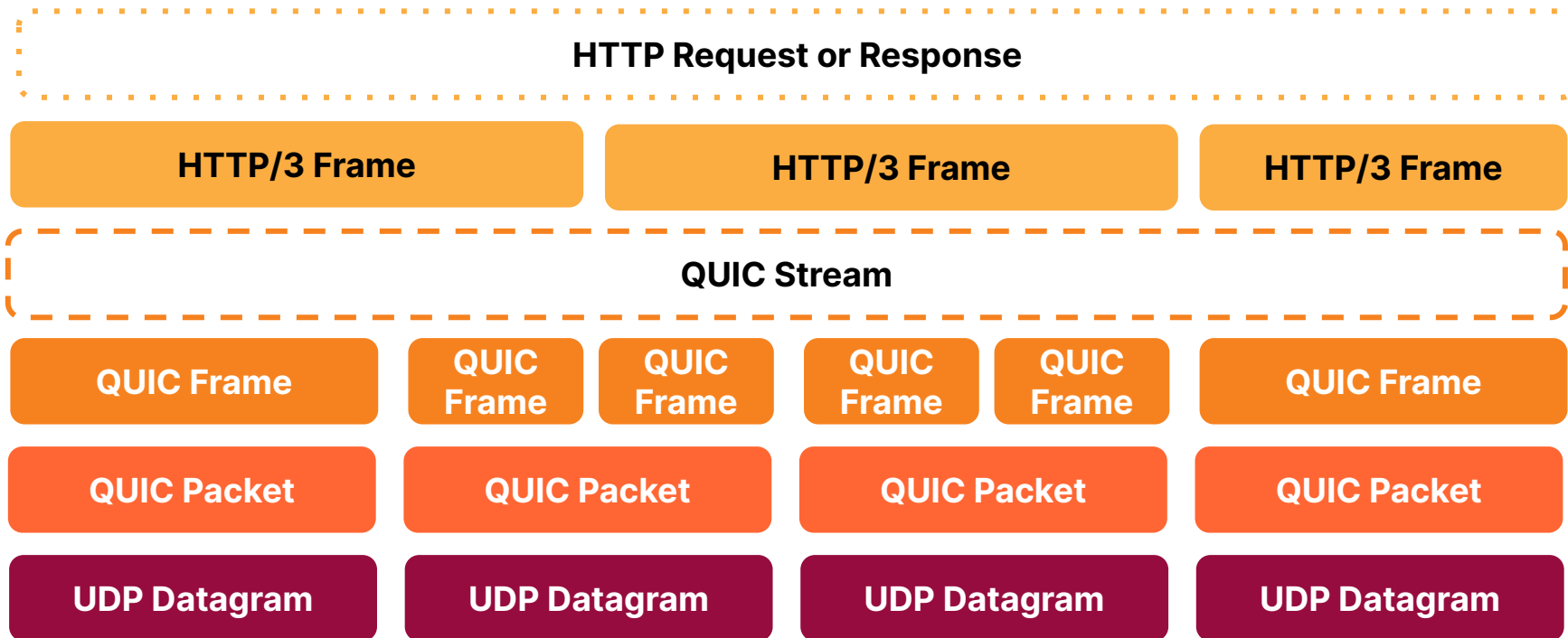
**Client-initiated bidirectional** streams are always used for request and response exchanges.

**Client- and server-initiated unidirectional** streams have a type, conveyed in the first byte(s) of the stream. Used for control channels.

HTTP/3 defines its own framing layer on top of QUIC. HTTP/3 frames are sent on QUIC streams.



# Framing and packetization



# Unreliable datagrams (the third-best thing)

# QUIC reliability and Datagrams

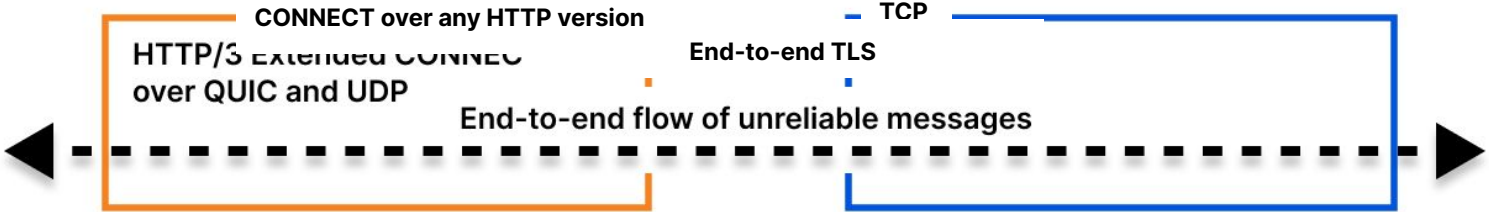
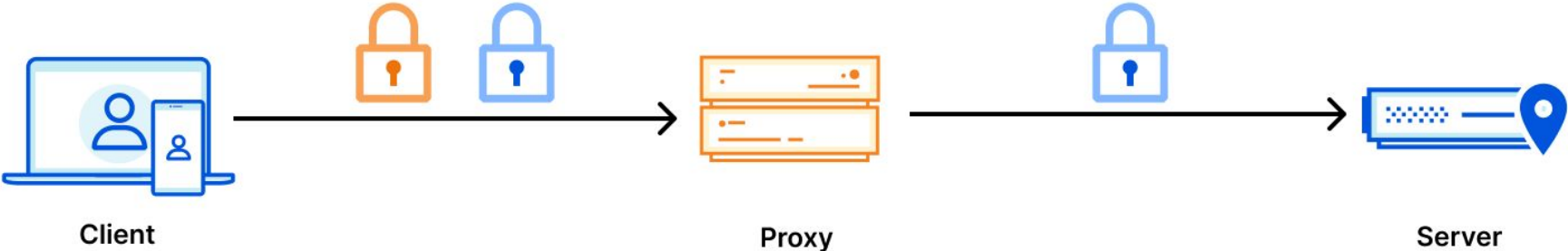
- QUIC can detect loss.
- Only reliable things are retransmitted.
- Packets are **never** retransmitted.
  - Retransmitted data is reframed and repacketized.
- Streams are always reliable.
- Sometimes application data doesn't need reliability.
- [RFC 9221](#) - Datagram Extension.
- DATAGRAM frames **must** fit entirely inside a QUIC packet.

**Over, under, around,  
and through**

# Tunneling other protocols

- HTTP traditionally runs over TCP
- CONNECT method
  - End-to-end TLS over a forward proxy (e.g. corporate proxy)

# CONNECT



# CONNECT(-UDP)

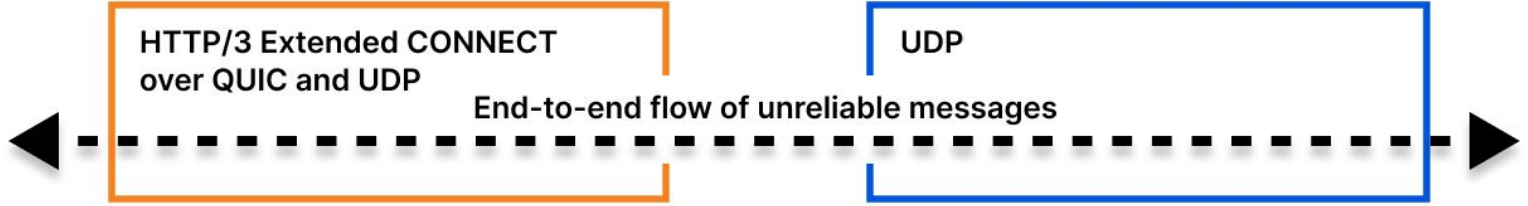
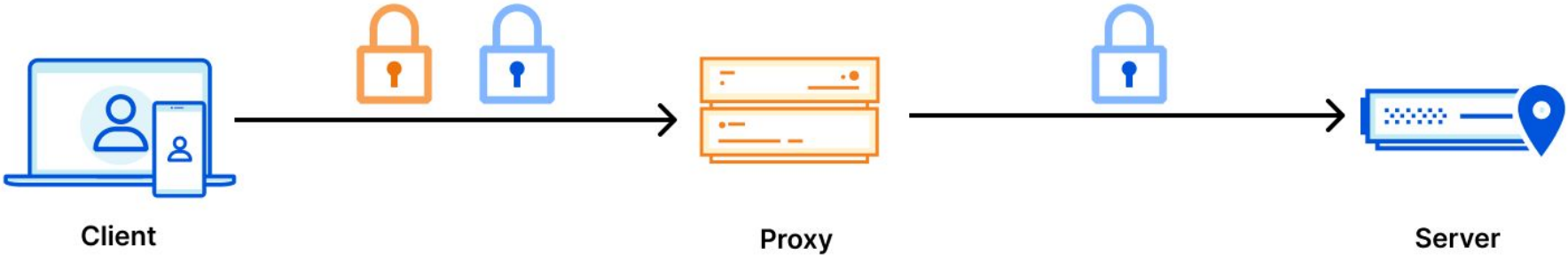
- We put HTTP over UDP
- So why not put UDP over HTTP?
  - IETF MASQUE Working Group chartered
- But HTTP doesn't have the notion of unreliable data.
- So first, HTTP datagrams - [RFC 9297](#)
  - How to use QUIC DATAGRAMS for HTTP
- Then extend CONNECT - [RFC 9298](#)
  - How to associate a logical flow of atomic messages with an HTTP request

## CONNECT(-UDP)

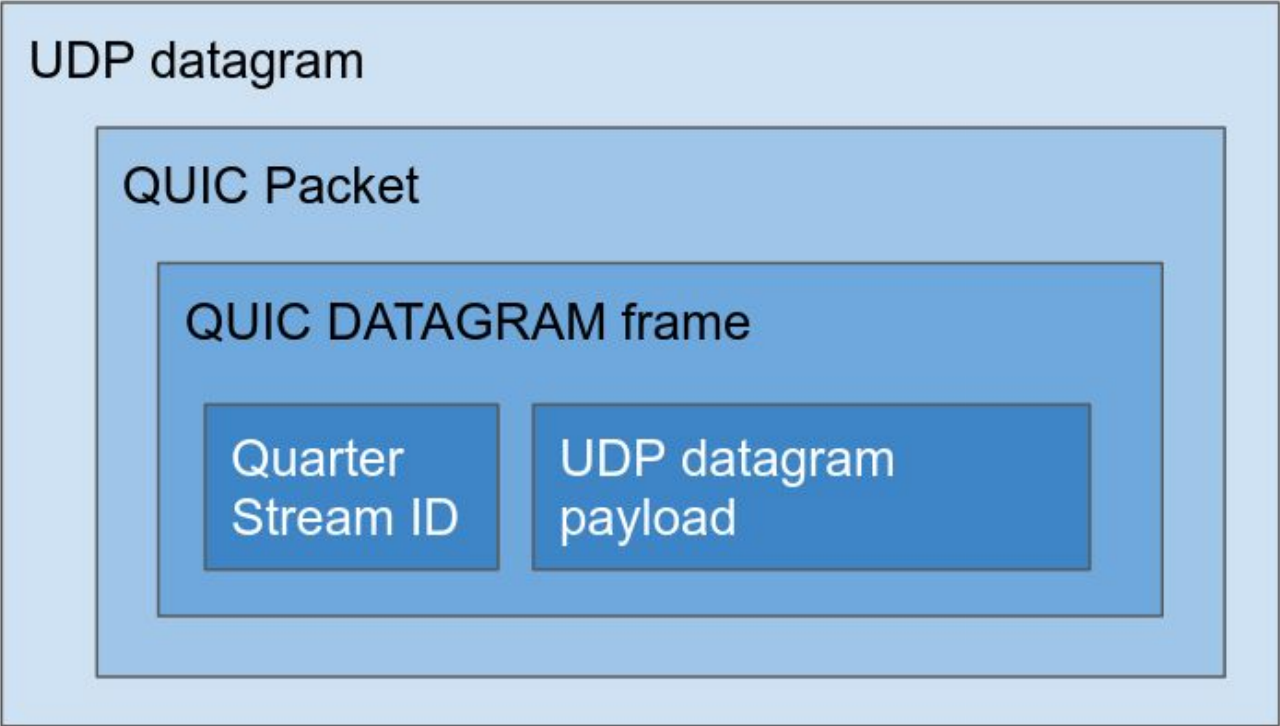
```
:method = CONNECT  
:protocol = connect-udp  
:scheme = https  
:path = /target.example.com/443/  
:authority = proxy.example.com
```



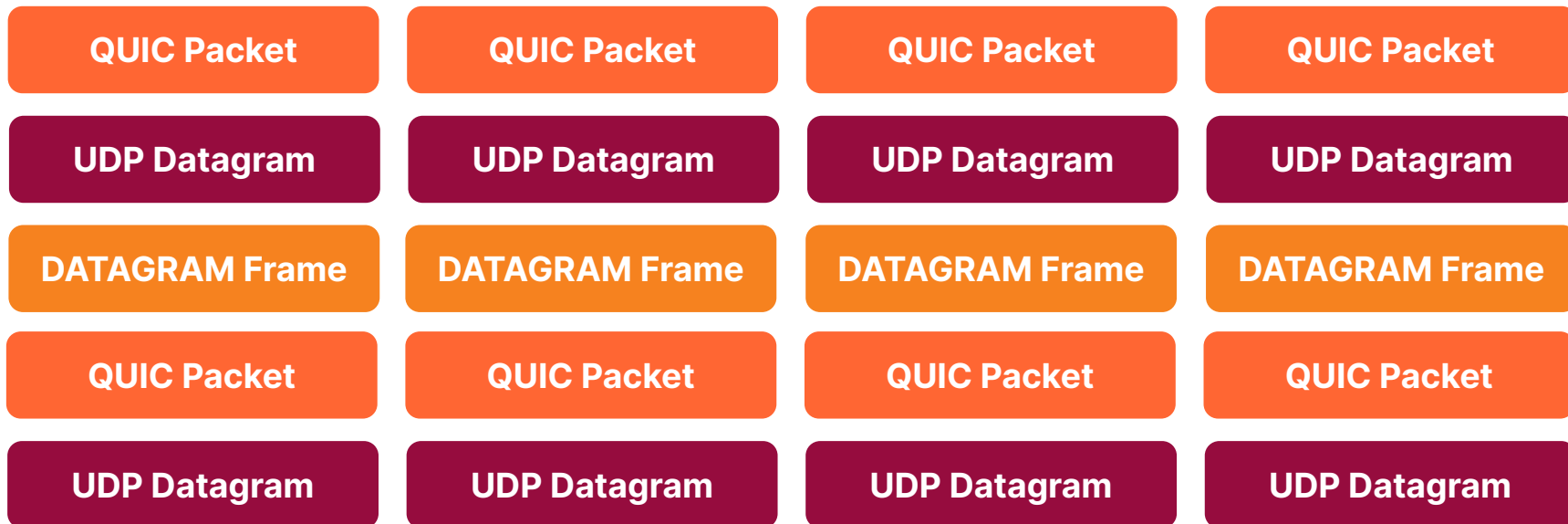
# CONNECT(-UDP)



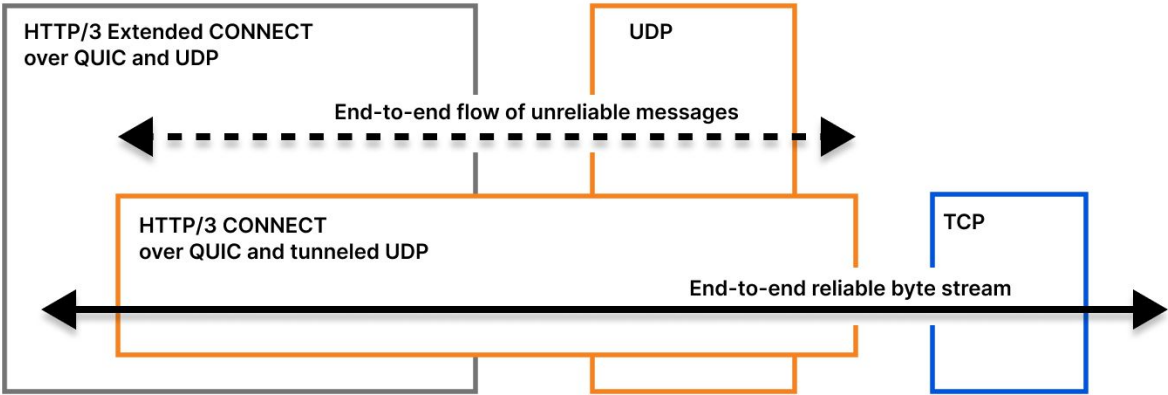
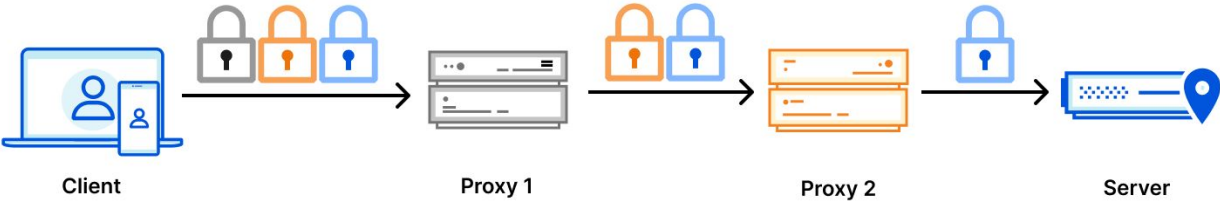
# Anatomy of encapsulation



# QUIC over QUIC framing and packetization

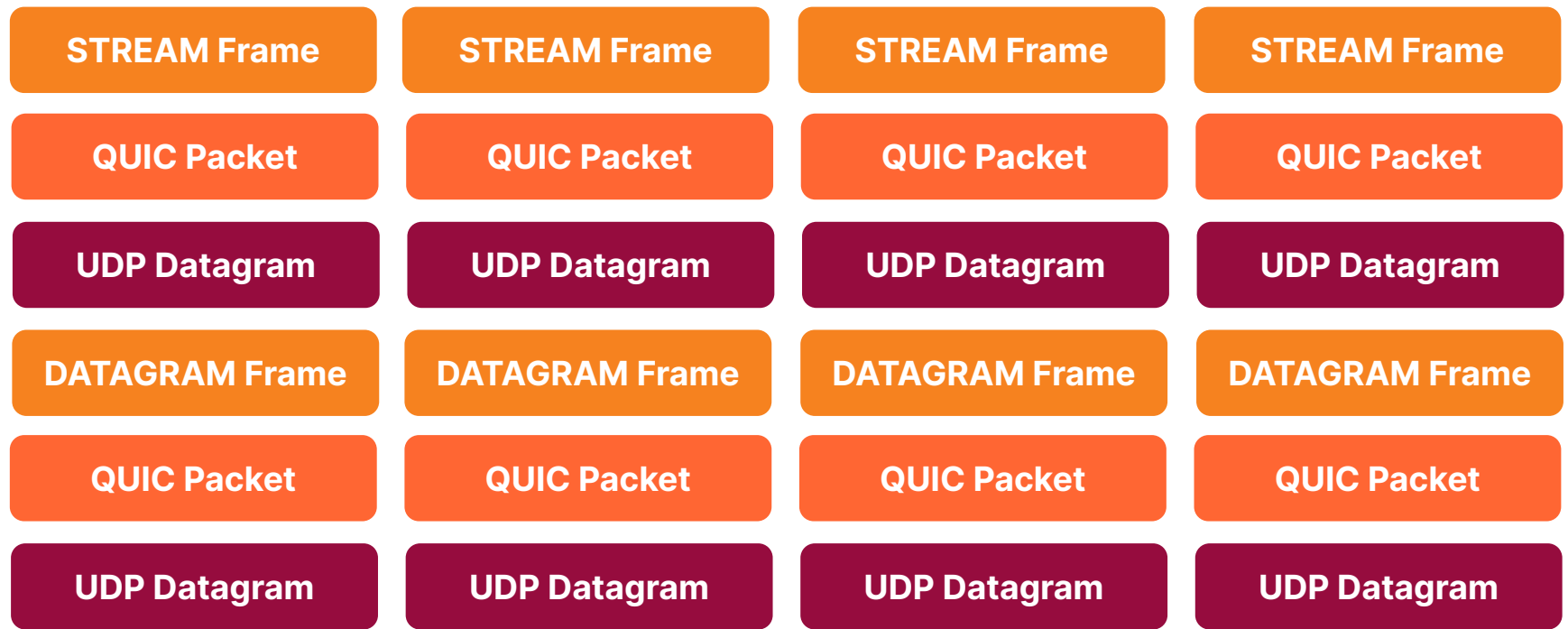


# Nested tunneling



# TLS over QUIC over QUIC framing and packetization

## End-to-end TLS



# Unpacking, unwrapping, and wrapping up

- QUIC is a secure transport protocol.
- A single connection can multiplex reliable streams and unreliable datagrams.
- Application mappings define how streams or datagrams get used.
  - You can define these fairly simply for own use cases.
- ALPN negotiates application protocol.

# Unpacking, unwrapping, and wrapping up

- Without keys or logs you can't see what is happening
- With keys or logs, you still need to understand the finer details
- Everything is fixable with a layer of abstraction.
  - Rather than define a new application mapping, define how to extend a protocol to carry your protocol.

QUIC is QUIC

HTTP/3 is HTTP/3

The power, is yours



# Thank you for your time

**Lucas Pardue**

Senior Software Engineer, Cloudflare  
QUIC Working Group Co-chair, IETF