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

CLOUDFLARE

QUIC is not HTTP

HTTP/3 is not HTTP/2

QUIC is not just the web over UDP

CLOUDFLARE

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

CLOUDFLARE

QUIC is a secure transport protocol

CLOUDFLARE

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

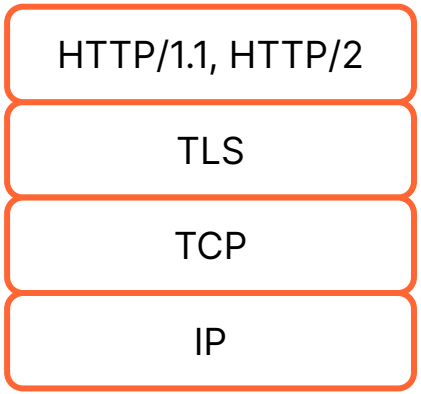
Applicability and Manageability

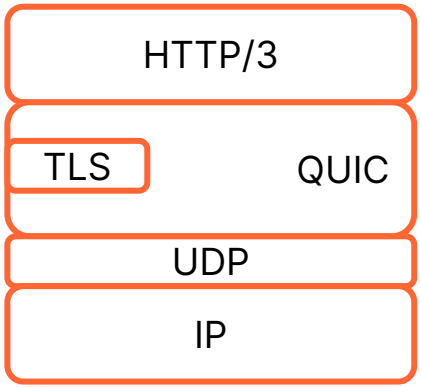
- RFC 9308 - Applicability of the QUIC Transport Protocol
- RFC 9312 - Manageability of the QUIC Transport Protocol

HTTP/3 and QPACK

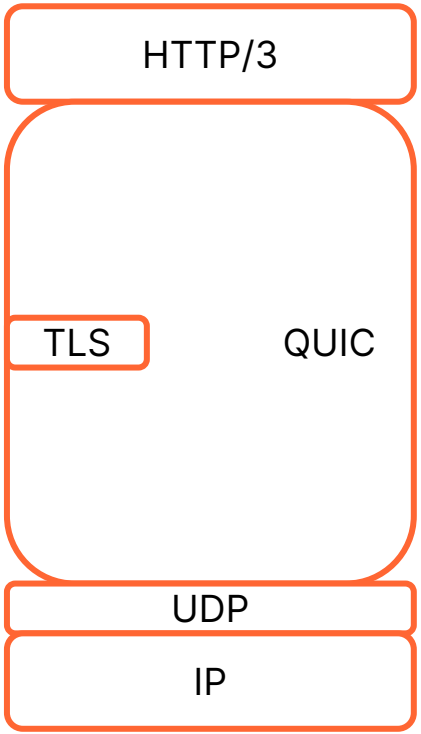
- RFC 9114 - HTTP/3
- RFC 9204 - QPACK

Obligatory Diagram





Interlocking Pieces



Application Mapping

Transport

Packets & Frames
Flow control
Reliable streams
Unreliable datagrams

Security

Authentication
Key exchange
Packet protection

Recovery

Loss detection and recovery
Congestion control

Who are you?

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

No action required. Stay happy.

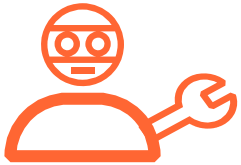
Unless you hit an issue you can't understand.

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

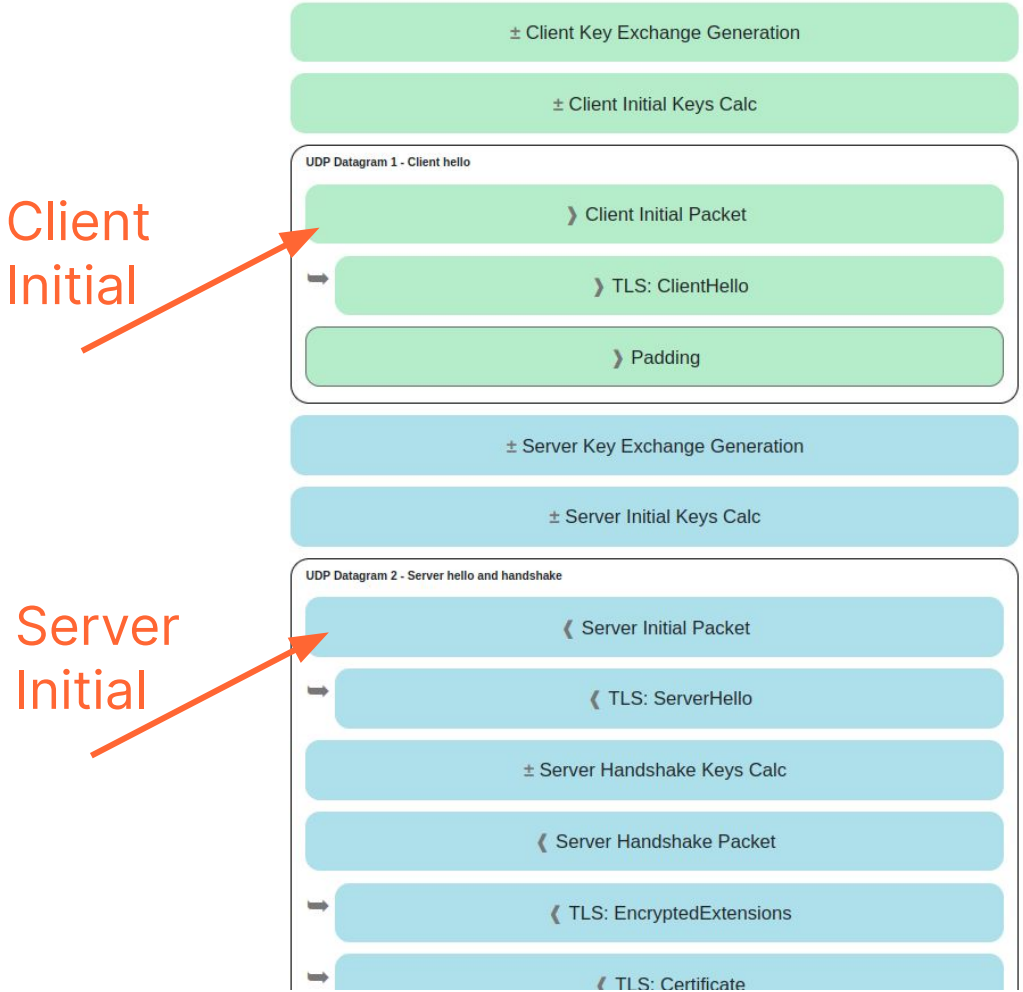
Don't mix up packets types with adjectives

- 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

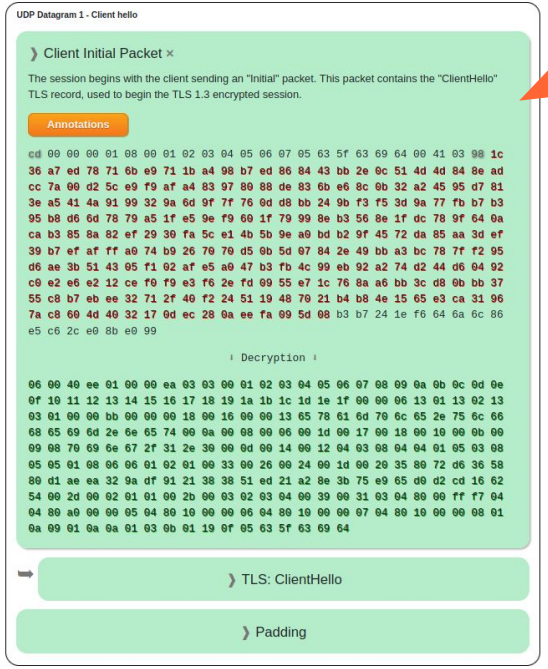
The Illustrated Guide

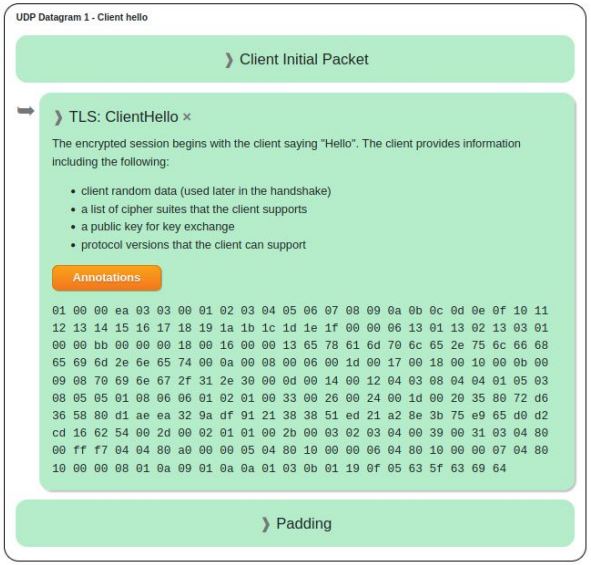
Sometimes it helps to look at things differently than the specs.

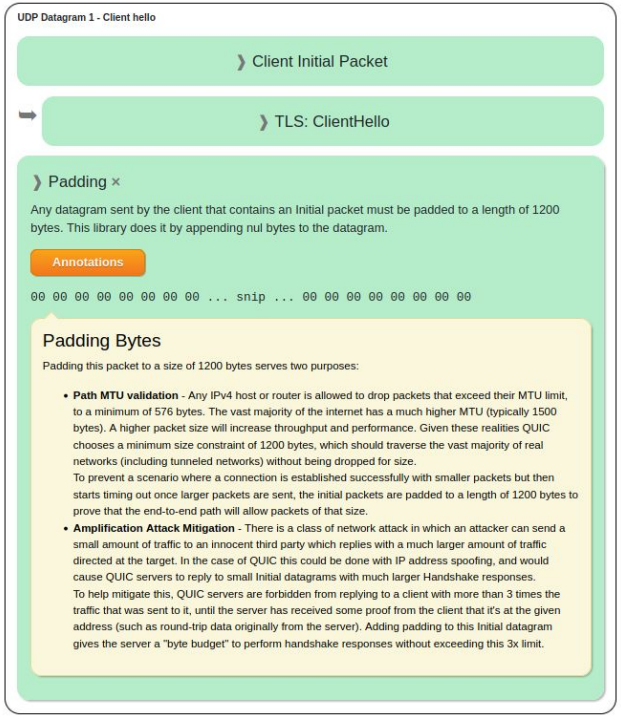
https://quic.xargs.org



Client Initial expanded







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.

Apply a display filter ... <Ctrl-/>

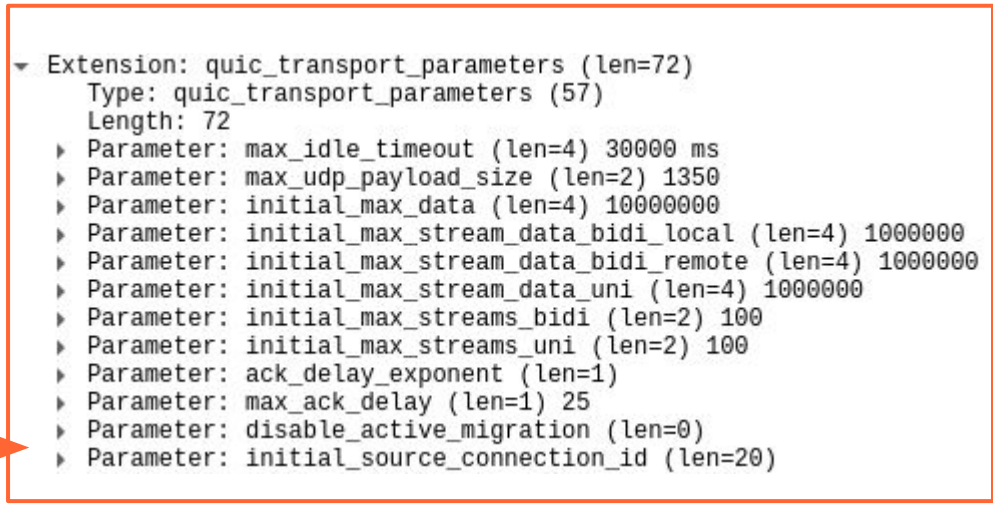
No.	Time	Source	Src port	Destination	Dst port	Protocol	Length	Info
1	0.000000000	127.0.0.1	44746	127.0.0.1	4433	QUIC	1242	Initial, DCID=f7cc19997f578525a476d84d1395c6a1, SCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4, PKN: 0, CRYPTO
2	0.003059492	127.0.0.1	4433	127.0.0.1	44746	QUIC	1242	Handshake, DCID=d24c3e72fee7cd93a210446c95fd1fd43cd53ff4, 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

Client Initial

ClientHello

ALPN

Transport Parameters



Application-Layer Protocol Negotiation ALPN

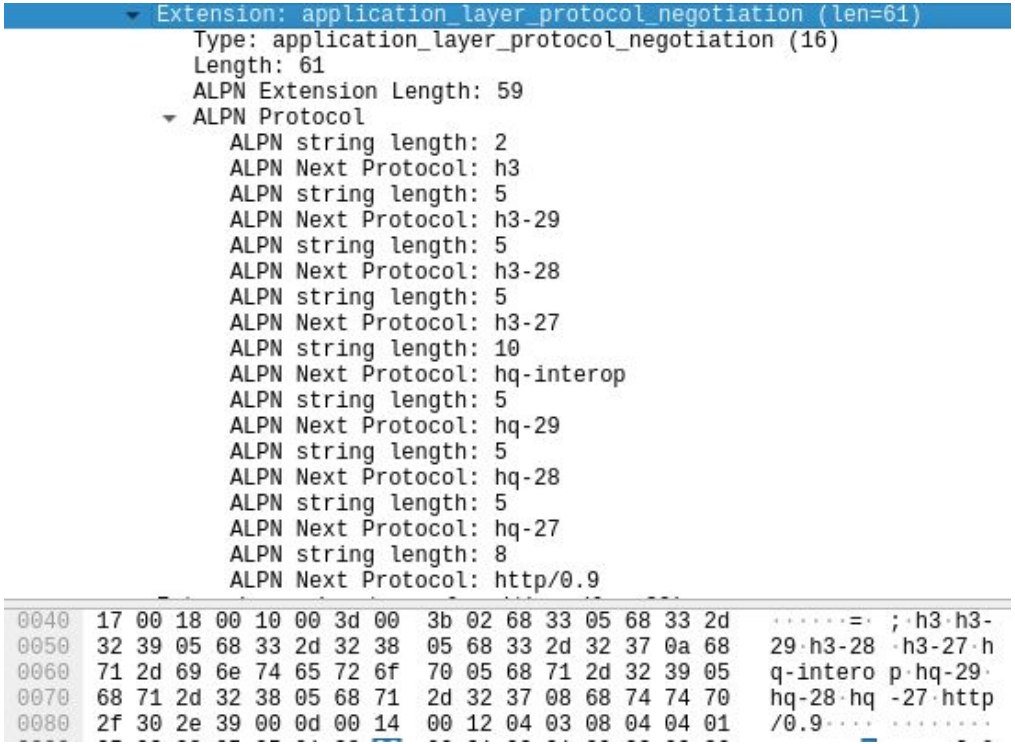
RFC 7301

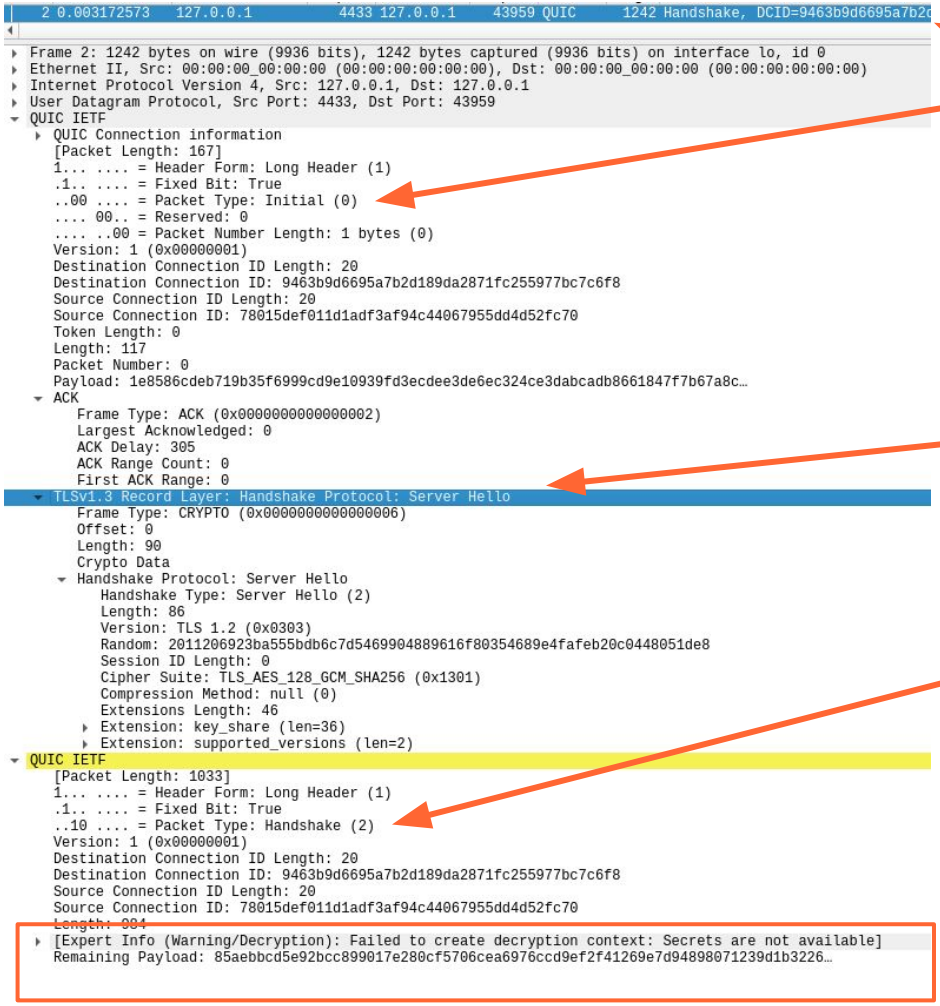
Client and server negotiate what Application Protocol to use across the secure transport.

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





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.

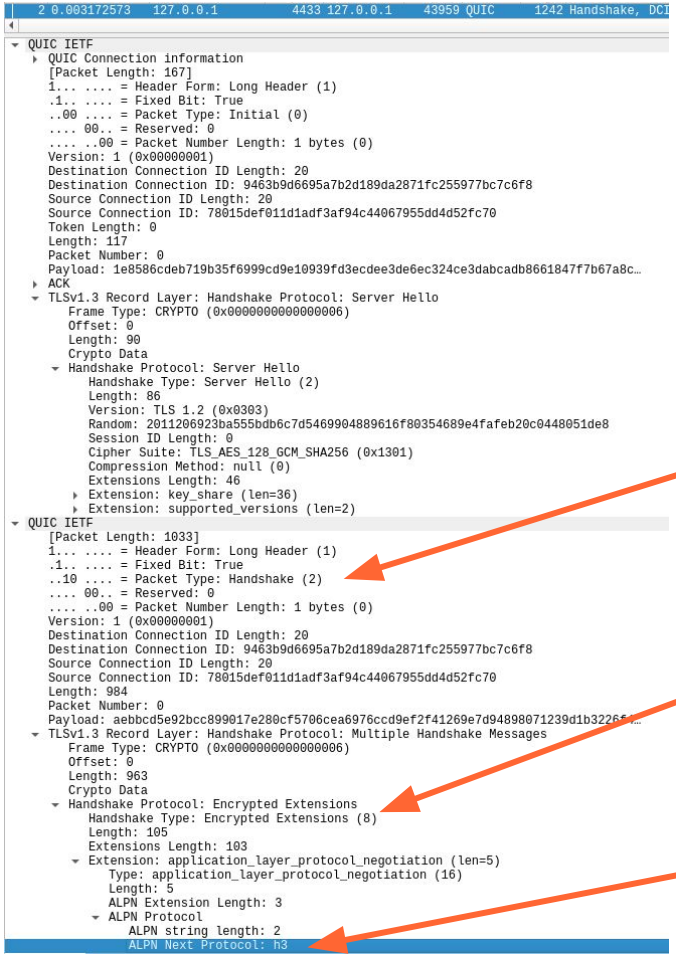
- Or use a new standard for endpoint logging - qlog.
  - Analyse it with the popular browser tool qvis.
  - Or build your own analysis tooling using libraries such as Cloudflare's qlog Rust crate.

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.



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: [illegible], 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: [illegible], CC

HTTP GET /index.html request

HTTP 404 response

CLOUDFLARE

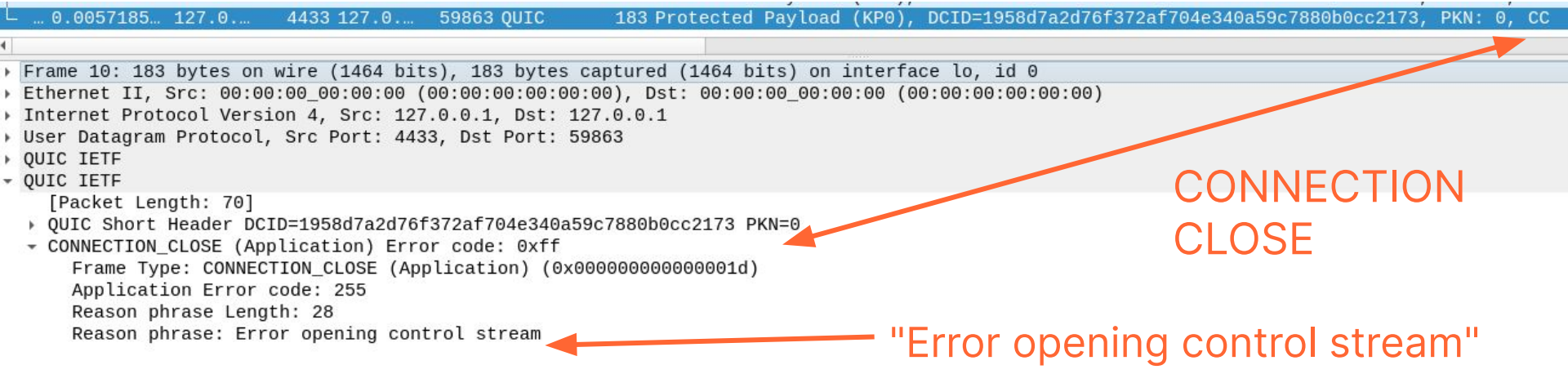
Something broke and I have no idea why”

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



CLOUDFLARE

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 globalling ordered. Loss or reording 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 (0×01) identifies the initiator of the stream - client or server.
- Second least significant bit (0×02) distinguishes between bidirectional and unidirectional.

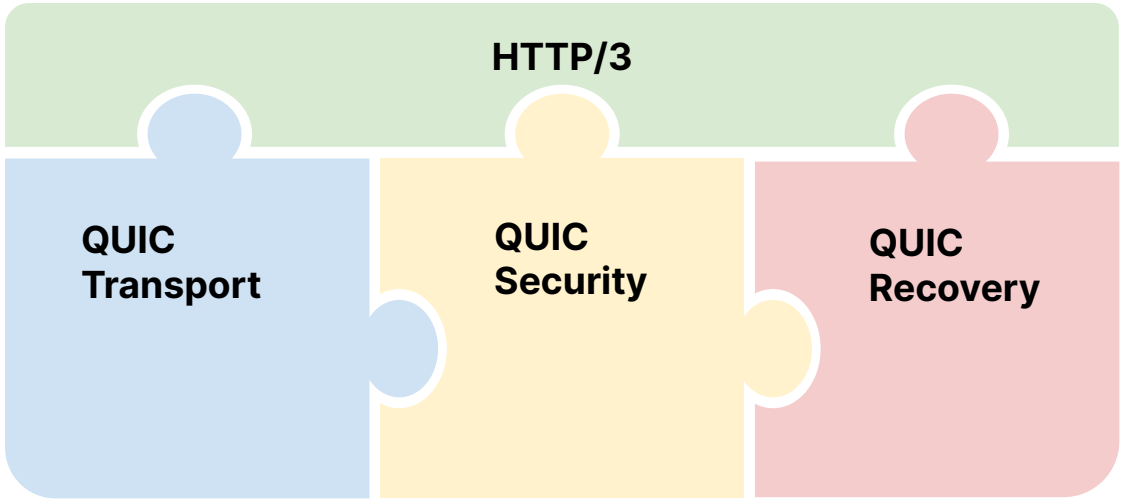
Bits	Stream Type
0x00	Client-Initiated, Bidirectional
0x01	Server-Initiated, Bidirectional
0x02	Client-Initiated, Unidirectional
0x03	Server-Initiated, Unidirectional

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.

CLOUDFLARE



HTTP/3 and its stream usage

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

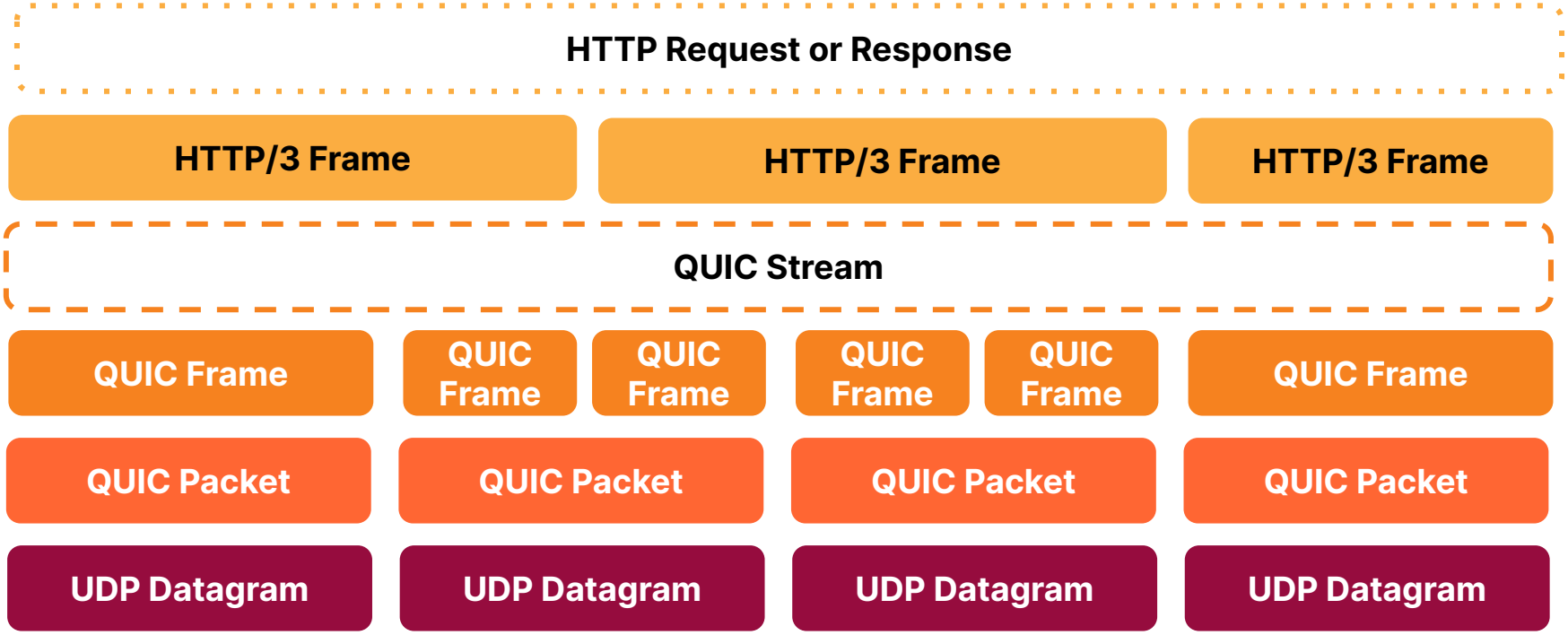
HTTP/3 is the **application-mapping** of HTTP semantics to QUIC transport services.

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)

CLOUDFLARE

QUIC reliability and Datagrams

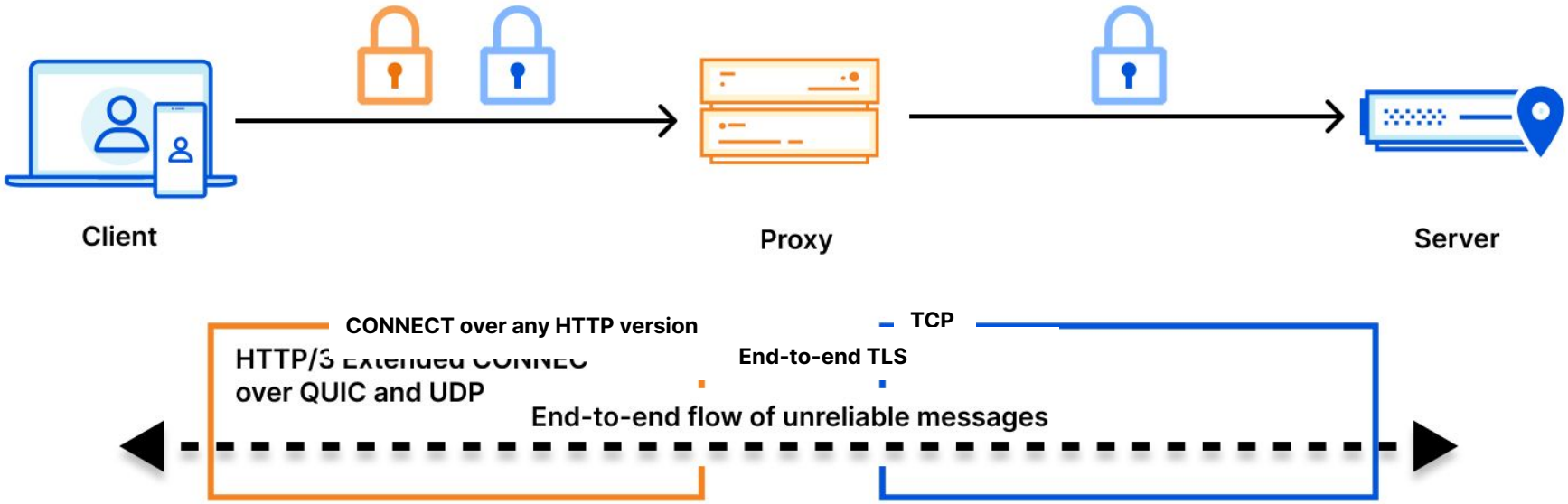
- QUIC can detect loss.
- Only reliable things are retransmitted.
- Packets are **never** retransmitted.
  - Restransmitted data is reframed and repacketized.
- Streams are always reliable.
- Sometimes application data doesn't need reliability.
- RFC 9221 - Datagram Extension.
- DATAGRAM frames **must** 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


Client
Proxy
Server
CONNECT over any HTTP version
HTTP/3 Extended CONNEC
over QUIC and UDP
TCP
End-to-end TLS
End-to-end flow of unreliable messages

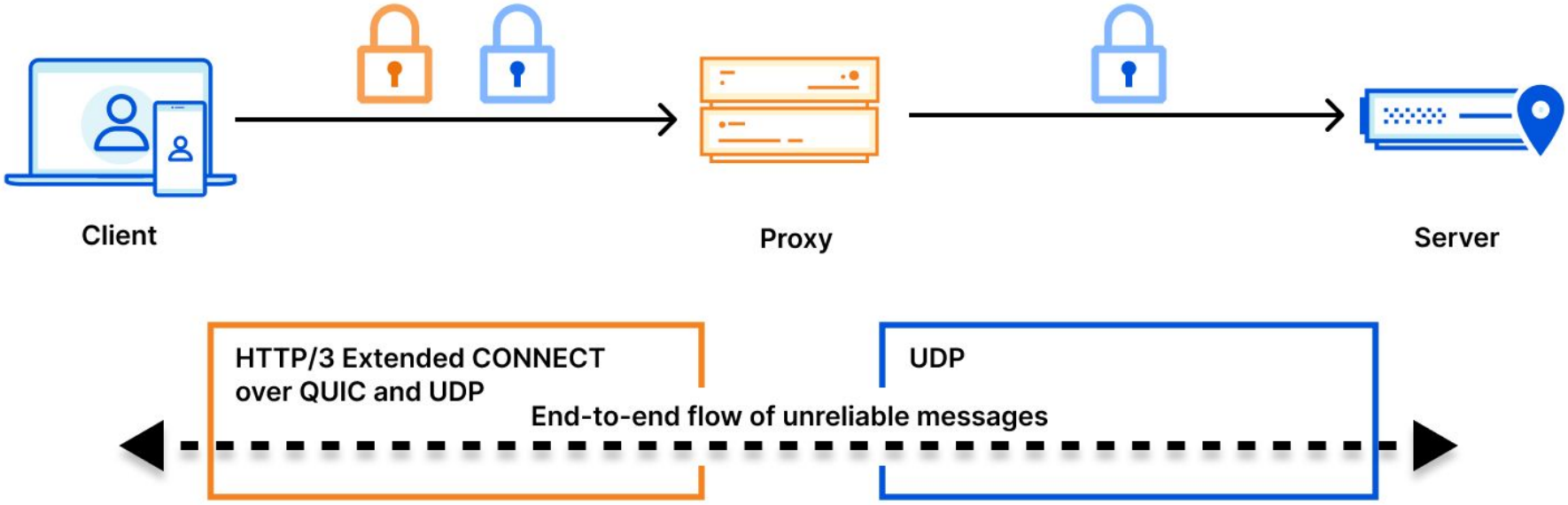
CONNECT(-UDP)

- 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

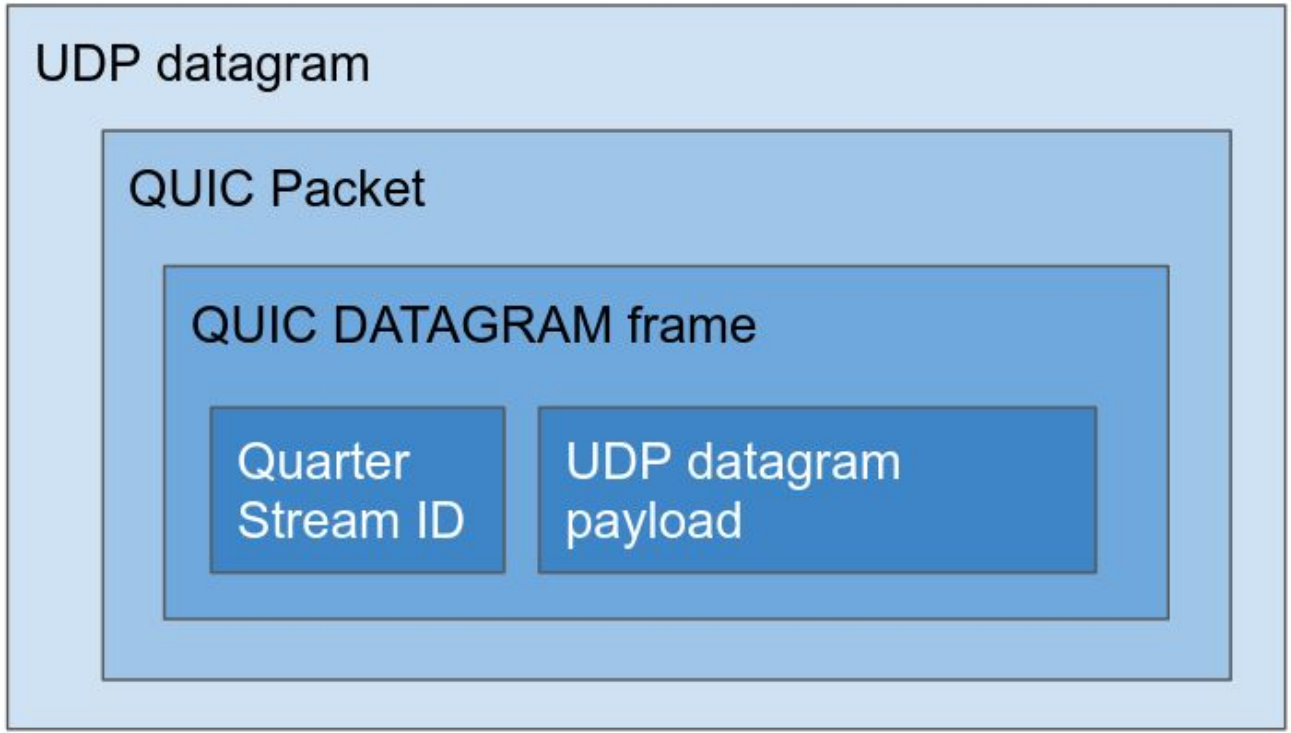
CONNECT(-UDP)

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

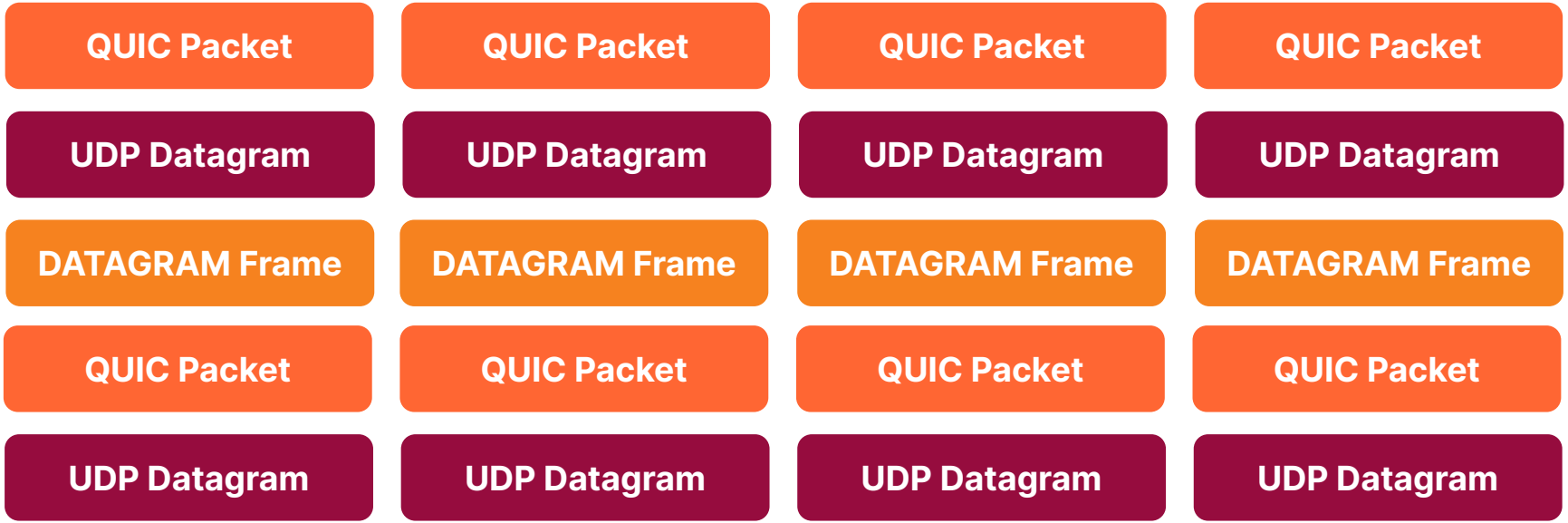
CONNECT(-UDP)


Client
Proxy
Server
HTTP/3 Extended CONNECT
over QUIC and UDP
UDP
End-to-end flow of unreliable messages

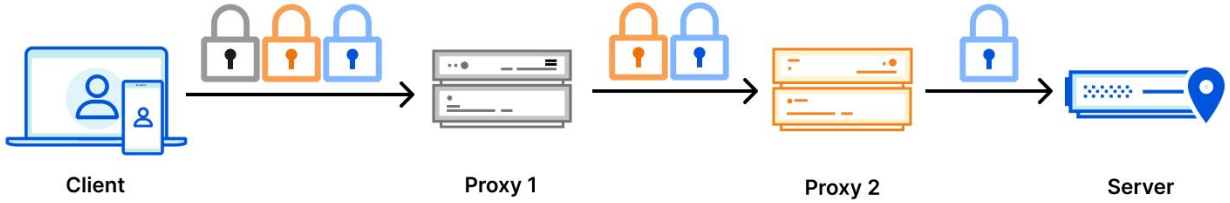
Anatomy of encapsulation

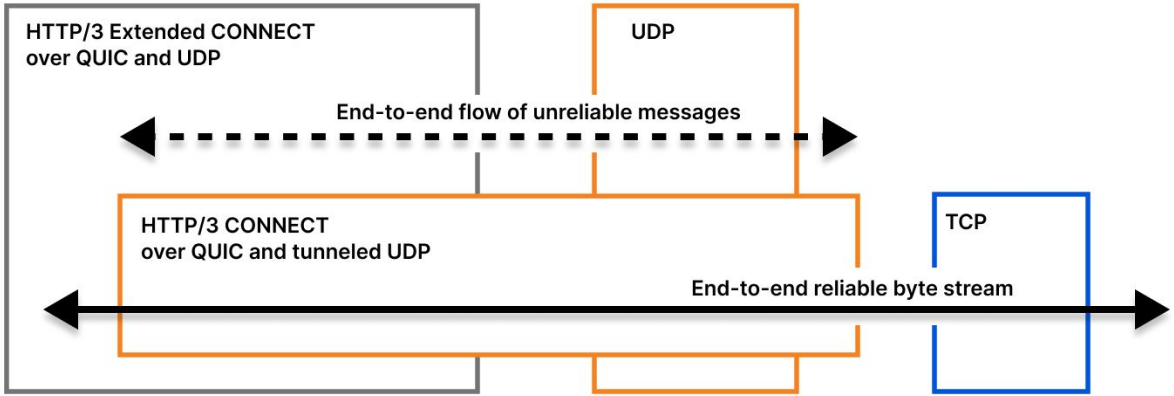


QUIC over QUIC framing and packetization

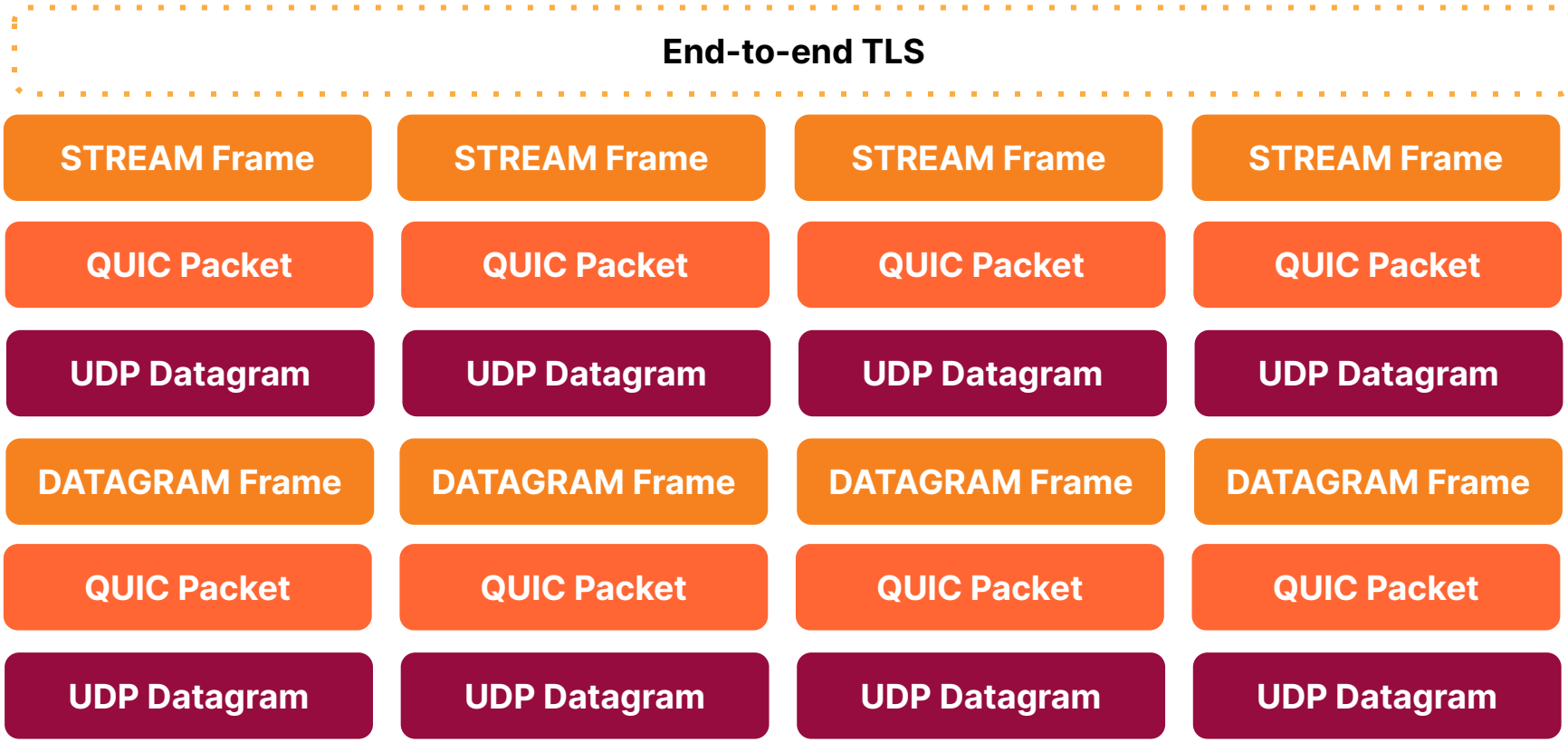


Nested tunneling





TLS over QUIC over QUIC framing and packetization



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.

CLOUDFLARE

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