Applying, Observing, and Debugging QUIC

Lucas Pardue

QUIC Deep Dive, IETF 115, Tuesday 8 November 2022

QUIC is <u>not</u> TCP

QUIC is <u>not</u> TLS

QUIC is not HTTP

QUIC is <u>not</u> the web over UDP

QUIC is QUIC

QUIC is a secure transport protocol

QUIC is what you make it

Ain't got the time?

It all starts with a handshake.

Then, application data can be sent using reliable streams or unreliable datagrams.

QUIC packets are protected. If you don't have the keys, you can't see contents.

Reliable data is retransmitted in new packets. Packets are <u>not</u> retransmitted

Applicability and Management

Want to send your application data over QUIC? Read [RFC 9308] - “Applicability of the QUIC Transport Protocol”

Operate a network and want to observe/manage QUIC? Read [RFC 9312] - “Manageability of the QUIC Transport Protocol”

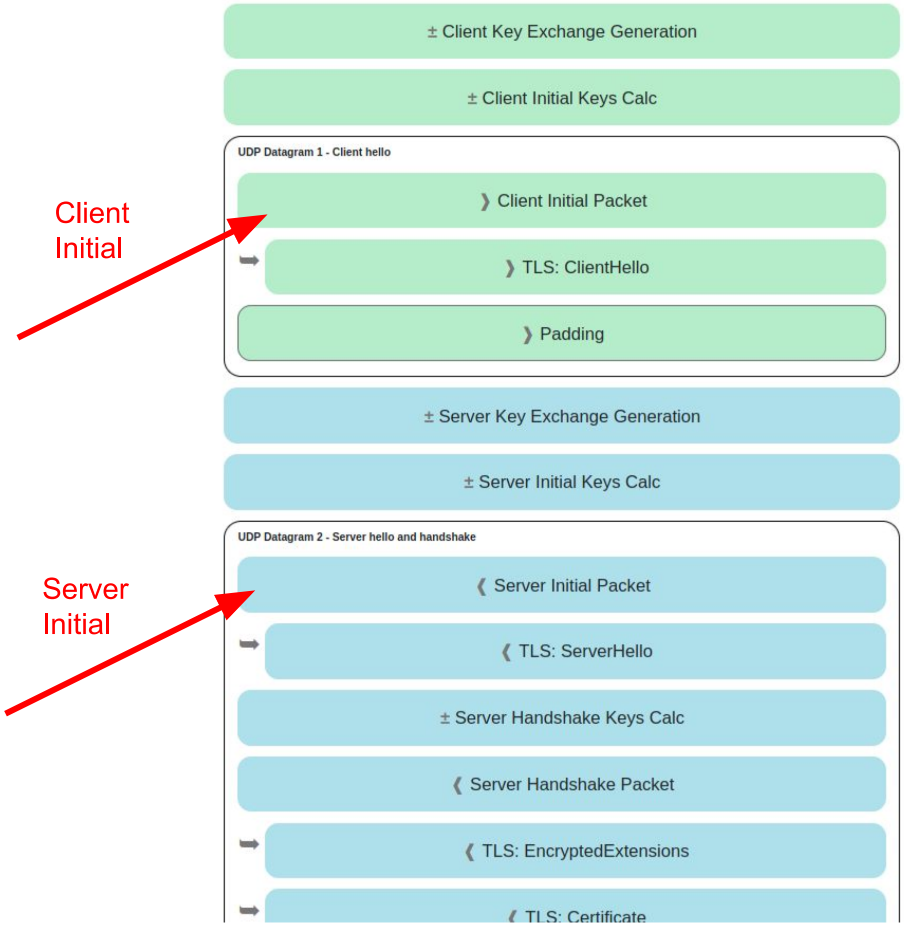
Everything starts with a handshake

- RFC 9000, Section 7 - Cryptographic and Transport Handshake
- RFC 9001 - Using TLS to Secure QUIC
- RFC 9312, Section 2.4 - The QUIC Handshake
- The specs detail it all
  - Jana and MT walked us through during Monday's session
- Key items: **Initial** and **Handshake** packets
  - Initial is a type, not an adjective
  - Easy to misinterpret "Initial packet" as "initial (first) packet" - that way leads to sadness

The illustrated guide

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

https://quic.xargs.org/
(source code: https://github.com/syncsynchalt/illustrated-quic)



Client Initial (expanded view)

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 6b 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 9b 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 a0 bd b2 9f 45 72 da 85 aa 3d ef
39 b7 ef af ff a0 74 b9 26 70 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
c0 e2 e6 e2 12 ce f0 f9 e3 f6 2e fd 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 00 18 00 16 00 00 13 65 78 61 6d 70 6c 65 2e 75 6c 66
68 65 69 6d 2e 6e 65 74 00 0a 00 08 00 06 00 1d 00 17 00 18 00 10 00 0b 00
09 08 70 69 6e 67 2f 31 2e 30 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 d0 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 08 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 00 18 00 16 00 00 13 65 78 61 6d 70 6c 65 2e 75 6c 66 68
65 69 6d 2e 6e 65 74 00 0a 00 08 00 06 00 1d 00 17 00 18 00 10 00 0b 00
09 08 70 69 6e 67 2f 31 2e 30 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 d0 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 08 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

Remember: QUIC Transport Parameters are a TLS extension

```
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"

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

A full listing and explanation of the bytes follows:

- `00 39` - assigned value for extension "QUIC Transport Parameters"
- `00 31` - 0x31 (49) bytes of "QUIC Transport Parameters" extension data follows
- `03` - assigned value for "max_udp_payload_size"
- `04` - 4 bytes of "max_udp_payload_size" data follows
- `80 00 ff f7` - a variable length integer with value 0xfff7 (65527)
- `04` - assigned value for "initial_max_data"
- `04` - 4 bytes of "initial_max_data" data follows
- `80 a0 00 00` - a variable length integer with value 0xa00000 (10485760)
- `05` - assigned value for "initial_max_stream_data_bidi_local"
- `04` - 4 bytes of "initial_max_stream_data_bidi_local" data follows
- `80 10 00 00` - a variable length integer with value 0x100000 (1048576)
- `06` - assigned value for "initial_max_stream_data_bidi_remote"
- `04` - 4 bytes of "initial_max_stream_data_bidi_remote" data follows
- `80 10 00 00` - a variable length integer with value 0x100000 (1048576)
- `07` - assigned value for "initial_max_stream_data_uni"
- `04` - 4 bytes of "initial_max_stream_data_uni" data follows
- `80 10 00 00` - a variable length integer with value 0x100000 (1048576)
- `08` - assigned value for "initial_max_streams_bidi"
- `01` - 1 bytes of "initial_max_streams_bidi" data follows
- `0a` - a variable length integer with value 0xA (10)
- `09` - assigned value for "initial_max_streams_uni"
- `01` - 1 bytes of "initial_max_streams_uni" data follows
- `0a` - a variable length integer with value 0xA (10)
- `0a` - assigned value for "ack_delay_exponent"
- `01` - 1 bytes of "ack_delay_exponent" data follows
- `03` - a variable length integer with value 3
- `0b` - assigned value for "GREASE", a technique for preventing middleboxes from disallowing new extensions, by pre-reserving extension values and injecting them randomly into connections
- `01` - 1 bytes of "GREASE" data follows
- `19` - a variable length integer with value 0x19 (25)
- `0f` - assigned value for "initial_source_connection_id"
- `05` - 5 bytes of "initial_source_connection_id" data follows
- `63 5f 63 69 64` - a copy of the source connection ID from the packet header: "`c_cid`"

Illustration on live connections

Our old friends pcap and Wireshark.

To successfully dissect QUIC packets, Wireshark 3.4.x and onwards. Examples use Cloudflare quiche - https://github.com/cloudflare/quiche.

Client: `quiche-client --no-verify --wire-version 1 https://127.0.0.1:4433/index.html`

Server: `quiche-server --no-retry`

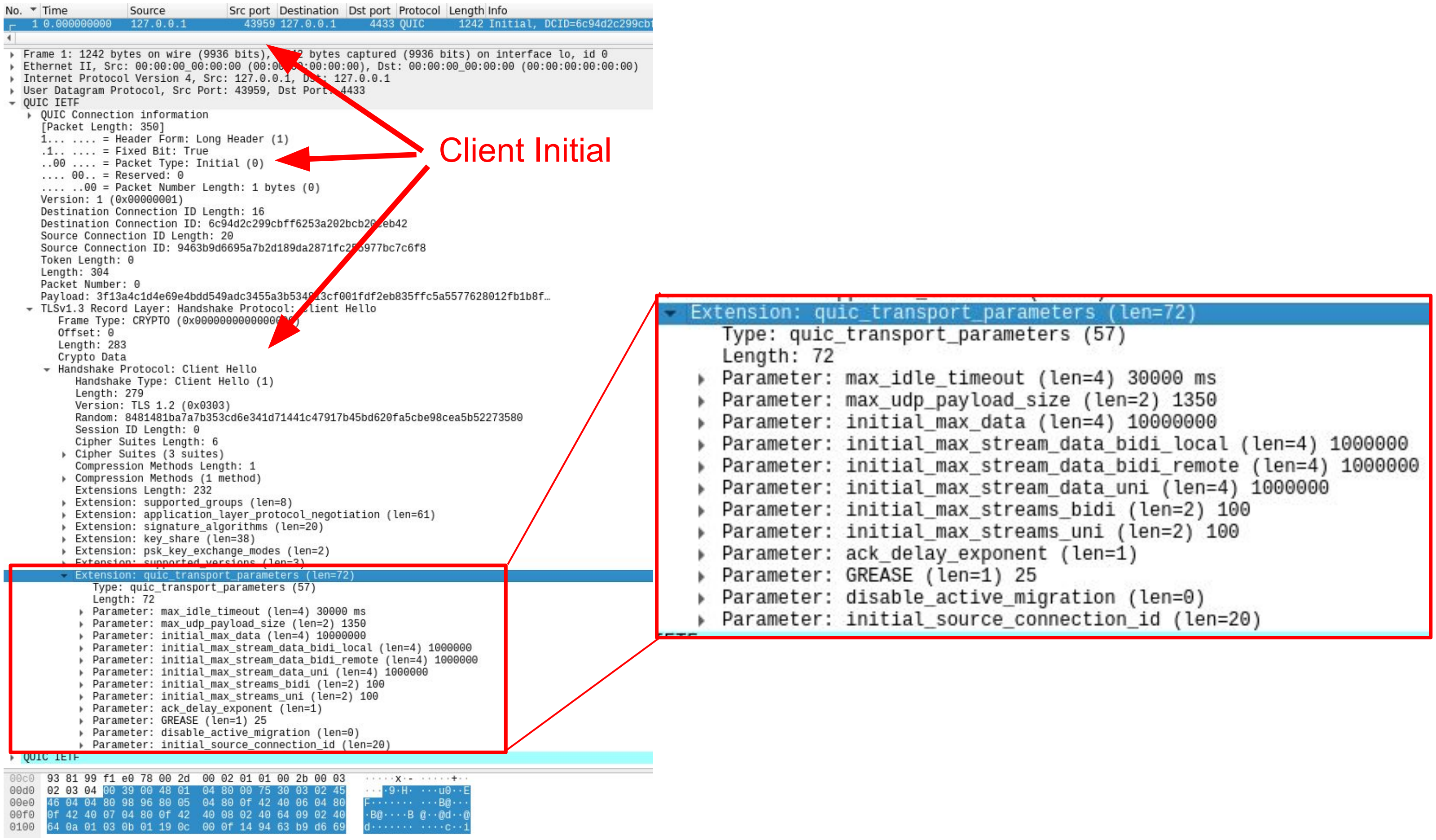
No.	Time	Source	Src port	Destination	Dst port	Protocol	Length	Info
1	0.000000000	127.0.0.1	43959	127.0.0.1	4433	QUIC	1242	Initial, DCID=6c94d2c299cbff6253a202bcb20ceb42, SCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8, PKN: 0, CRYPTO
2	0.003172573	127.0.0.1	4433	127.0.0.1	43959	QUIC	1242	Handshake, DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8, SCID=78015def011d1adf3af94c44067955dd4d52fc70
3	0.003941563	127.0.0.1	43959	127.0.0.1	4433	QUIC	1242	Handshake, DCID=78015def011d1adf3af94c44067955dd4d52fc70, SCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8
4	0.004213035	127.0.0.1	4433	127.0.0.1	43959	QUIC	491	Handshake, DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8, SCID=78015def011d1adf3af94c44067955dd4d52fc70
5	0.004963775	127.0.0.1	43959	127.0.0.1	4433	QUIC	256	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
6	0.005031516	127.0.0.1	43959	127.0.0.1	4433	QUIC	86	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
7	0.005082174	127.0.0.1	43959	127.0.0.1	4433	QUIC	86	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
8	0.005133409	127.0.0.1	43959	127.0.0.1	4433	QUIC	152	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
9	0.005183643	127.0.0.1	43959	127.0.0.1	4433	QUIC	111	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
10	0.006796488	127.0.0.1	4433	127.0.0.1	43959	QUIC	622	Protected Payload (KP0), DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8
11	0.007235573	127.0.0.1	43959	127.0.0.1	4433	QUIC	85	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
12	0.007425998	127.0.0.1	4433	127.0.0.1	43959	QUIC	86	Protected Payload (KP0), DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8
13	0.007554587	127.0.0.1	43959	127.0.0.1	4433	QUIC	85	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
14	0.007779237	127.0.0.1	4433	127.0.0.1	43959	QUIC	86	Protected Payload (KP0), DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8
15	0.007907760	127.0.0.1	43959	127.0.0.1	4433	QUIC	85	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
16	0.008091673	127.0.0.1	4433	127.0.0.1	43959	QUIC	150	Protected Payload (KP0), DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8
17	0.008407620	127.0.0.1	43959	127.0.0.1	4433	QUIC	90	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70

Ready-made examples

Follow along examples at https://github.com/LPardue/ietf-115-tdd

“localhost-good”

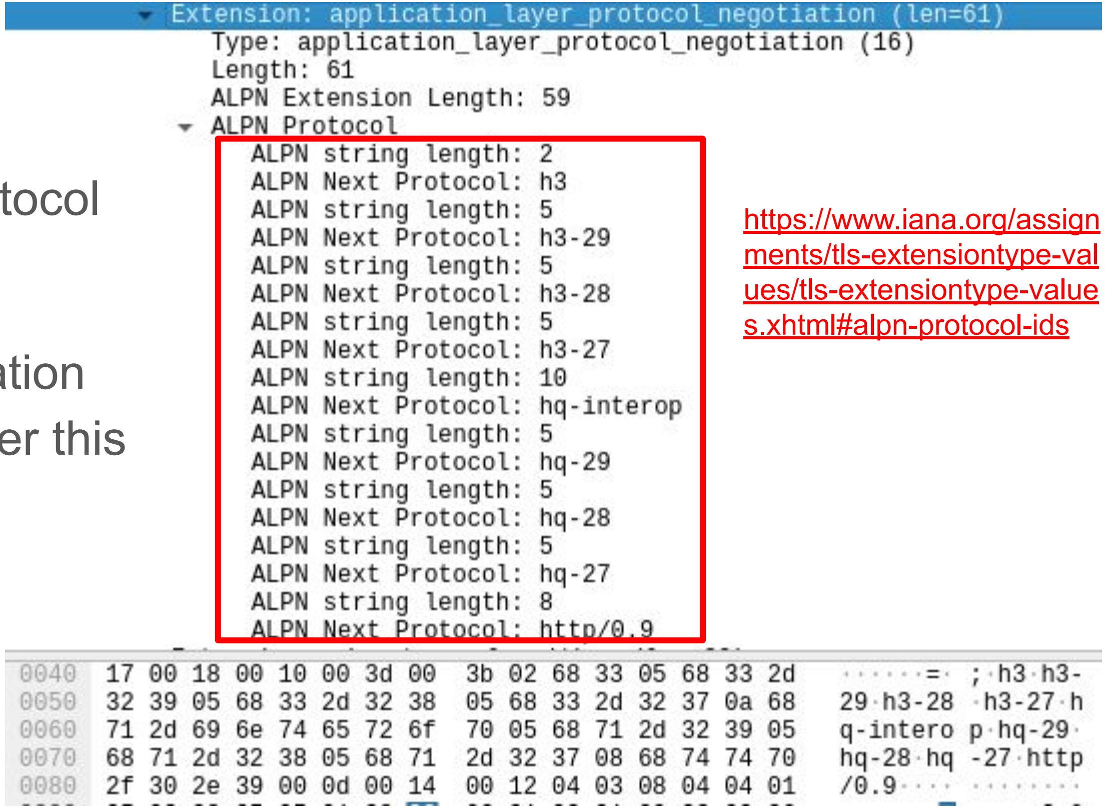
Client Initial



Client Initial - ALPN

[RFC 7301](https://www.rfc-editor.org/rfc/rfc7301) - Application-Layer Protocol Negotiation

Client offers a list of all the application protocols it would like to speak over this connection.



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

Server Initial and Handshake

```
  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: 1e8586cdeb719b35f6999cd9e10939fd3ecdee3de6ec324ce3dabcadb8[illegible]61847f7b67a8c…
  ▾ ACK
      Frame Type: ACK (0x0000000000000002)
      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: 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)
    Version: 1 (0x00000001)
    Destination Connection ID Length: 20
    Destination Connection ID: 9463b9d6695a7b2d189da2871fc255977bc7c6f8
    Source Connection ID Length: 20
    Source Connection ID: 78015def011d1adf3af94c44067955dd4d52fc70
    Length: 984
  ▸ [Expert Info (Warning/Decryption): Failed to create decryption context: Secrets are not available]
    Remaining Payload: 85aebbcd5e92bcc899017e280cf5706cea6976ccd9ef2f41269e7d94898071239d1b3226…
```

Server Initial

Server Handshake

Secrets not available!

Keys needed to see the full picture

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

SSLKEYLOGFILE is an approach used by many, but not all, implementations. Endpoints can be instructed to explicitly log their keys to the nominated file. draft-thomson-tls-keylogfile I-D seeking to formalise the format.

Session keys are symmetrical, either endpoint can log to enable packet decryption in both directions.

Client: `SSLKEYLOGFILE=mykeys.log quiche-client --no-verify --wire-version 1 https://127.0.0.1:4433/index.html`

Server Initial and Handshake (w/ keys)

https://wiki.wireshark.org/TLS

Server selects one ALPN, this is the application protocol that will be used over QUIC.

Applications need to agree on how QUIC streams are used.

Now we can dissect QUIC

Live illustration revisited

Dissection without keys

No.	Time	Source	Src port	Destination	Dst port	Protocol	Length	Info
1	0.000000000	127.0.0.1	43959	127.0.0.1	4433	QUIC	1242	Initial, DCID=6c94d2c299cbff6253a202bcb20ceb42, SCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8, PKN: 0, CRYPTO
2	0.003172573	127.0.0.1	4433	127.0.0.1	43959	QUIC	1242	Handshake, DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8, SCID=78015def011d1adf3af94c44067955dd4d52fc70
3	0.003941563	127.0.0.1	43959	127.0.0.1	4433	QUIC	1242	Handshake, DCID=78015def011d1adf3af94c44067955dd4d52fc70, SCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8
4	0.004213035	127.0.0.1	4433	127.0.0.1	43959	QUIC	491	Handshake, DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8, SCID=78015def011d1adf3af94c44067955dd4d52fc70
5	0.004963775	127.0.0.1	43959	127.0.0.1	4433	QUIC	256	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
6	0.005031516	127.0.0.1	43959	127.0.0.1	4433	QUIC	86	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
7	0.005082174	127.0.0.1	43959	127.0.0.1	4433	QUIC	86	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
8	0.005133409	127.0.0.1	43959	127.0.0.1	4433	QUIC	152	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
9	0.005183643	127.0.0.1	43959	127.0.0.1	4433	QUIC	111	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
10	0.006796488	127.0.0.1	4433	127.0.0.1	43959	QUIC	622	Protected Payload (KP0), DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8
11	0.007235573	127.0.0.1	43959	127.0.0.1	4433	QUIC	85	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
12	0.007425998	127.0.0.1	4433	127.0.0.1	43959	QUIC	86	Protected Payload (KP0), DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8
13	0.007554587	127.0.0.1	43959	127.0.0.1	4433	QUIC	85	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
14	0.007779237	127.0.0.1	4433	127.0.0.1	43959	QUIC	86	Protected Payload (KP0), DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8
15	0.007907760	127.0.0.1	43959	127.0.0.1	4433	QUIC	85	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
16	0.008091673	127.0.0.1	4433	127.0.0.1	43959	QUIC	150	Protected Payload (KP0), DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8
17	0.008407620	127.0.0.1	43959	127.0.0.1	4433	QUIC	90	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70

Dissection with keys

No.	Time	Source	Src port	Destination	Dst port	Protocol	Length	Info
1	0.000000000	127.0.0.1	43959	127.0.0.1	4433	QUIC	1242	Initial, DCID=6c94d2c299cbff6253a202bcb20ceb42, SCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8, PKN: 0, CRYPTO
2	0.003172573	127.0.0.1	4433	127.0.0.1	43959	QUIC	1242	Handshake, DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8, SCID=78015def011d1adf3af94c44067955dd4d52fc70, PKN: 0, CRYPTO
3	0.003941563	127.0.0.1	43959	127.0.0.1	4433	QUIC	1242	Handshake, DCID=78015def011d1adf3af94c44067955dd4d52fc70, SCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8, PKN: 0, ACK
4	0.004213035	127.0.0.1	4433	127.0.0.1	43959	QUIC	491	Handshake, DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8, SCID=78015def011d1adf3af94c44067955dd4d52fc70, PKN: 1, CRYPTO
5	0.004963775	127.0.0.1	43959	127.0.0.1	4433	HTTP3	256	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70, PKN: 0, NCI, STREAM(2), SETTINGS
6	0.005031516	127.0.0.1	43959	127.0.0.1	4433	HTTP3	86	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70, PKN: 1, STREAM(6)
7	0.005082174	127.0.0.1	43959	127.0.0.1	4433	HTTP3	86	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70, PKN: 2, STREAM(10)
8	0.005133409	127.0.0.1	43959	127.0.0.1	4433	HTTP3	152	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70, PKN: 3, STREAM(0), HEADERS
9	0.005183643	127.0.0.1	43959	127.0.0.1	4433	HTTP3	111	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70, PKN: 4, STREAM(14)
10	0.006796488	127.0.0.1	4433	127.0.0.1	43959	HTTP3	622	Protected Payload (KP0), DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8, PKN: 0, ACK, NCI, DONE, CRYPTO, STREAM(3), SETTINGS
11	0.007235573	127.0.0.1	43959	127.0.0.1	4433	QUIC	85	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70, PKN: 5, ACK
12	0.007425998	127.0.0.1	4433	127.0.0.1	43959	HTTP3	86	Protected Payload (KP0), DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8, PKN: 1, STREAM(7)
13	0.007554587	127.0.0.1	43959	127.0.0.1	4433	QUIC	85	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70, PKN: 6, ACK
14	0.007779237	127.0.0.1	4433	127.0.0.1	43959	HTTP3	86	Protected Payload (KP0), DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8, PKN: 2, STREAM(11)
15	0.007907760	127.0.0.1	43959	127.0.0.1	4433	QUIC	85	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70, PKN: 7, ACK
16	0.008091673	127.0.0.1	4433	127.0.0.1	43959	HTTP3	150	Protected Payload (KP0), DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8, PKN: 3, STREAM(0), HEADERS, DATA
17	0.008407620	127.0.0.1	43959	127.0.0.1	4433	QUIC	90	Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70, PKN: 8, CC

CIDs

Whether the packets are encrypted or not, connection IDs are visible. And they can be used for traffic steering / load balancing, as described by Ian and Martin.

```
 9 0.005183643   127.0.0.1         43959 127.0.0.1      4433 QUIC       111 Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70
10 0.006796488   127.0.0.1          4433 127.0.0.1     43959 QUIC       622 Protected Payload (KP0), DCID=9463b9d6695a7b2d189da2871fc255977bc7c6f8
```

Client -> Server DCID: 78015def011d1adf3af94c44067955dd4d52fc70

Server-> Client DCID: 9463b9d6695a7b2d189da2871fc255977bc7c6f8

qlog - structured logging by endpoints

Implementations often have debugging that can enhance or augment packet captures.

A common logging format can encourage an ecosystem of analysis tools.
E.g. what is an endpoint producing and why is it doing that?

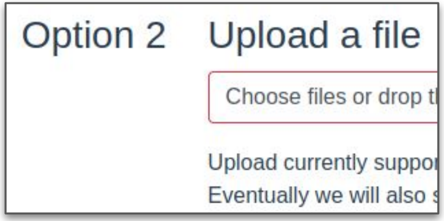
draft-ietf-quic-qlog-main-schema: a base schema defined in Concise Data Definition Language (CDDL; RFC 8610). Highly extensible. Many possible serialization formats.

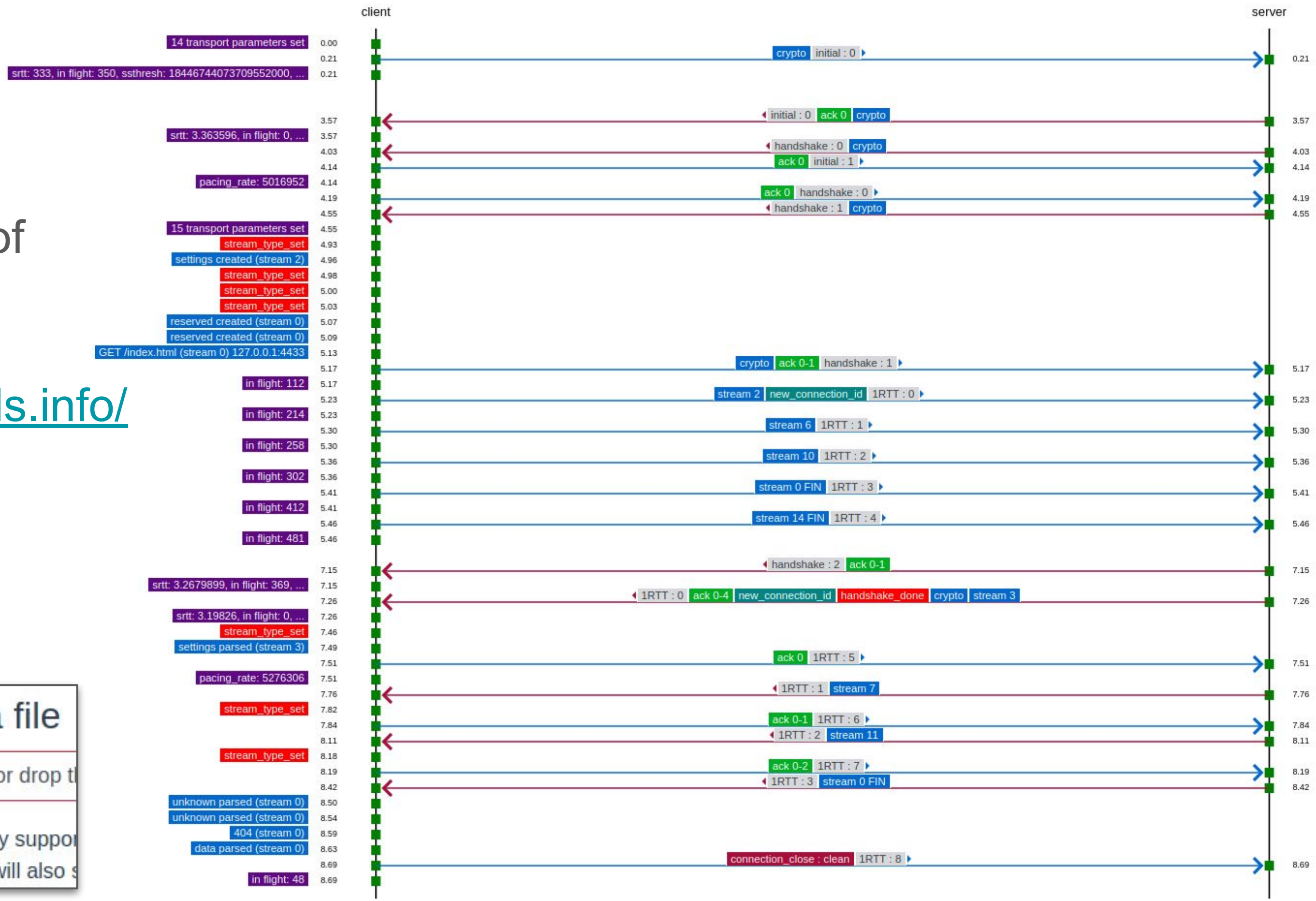
Draft-ietf-quic-qlog-quic-events, draft-ietf-quic-qlog-h3-events: concrete definitions to cover events related to packets and frames, security, congestion control etc.

qvis

Making sense out of oodles of data

https://qvis.quictools.info/





A real-world failure in wireshark -

"localhost-0-streams-uni"

Client: `SSLKEYLOGFILE=tdd.keys QLOGDIR=qlogs quiche-client --no-verify --wire-version 1 --max-streams-uni 0` https://127.0.0.1:4433/index.html

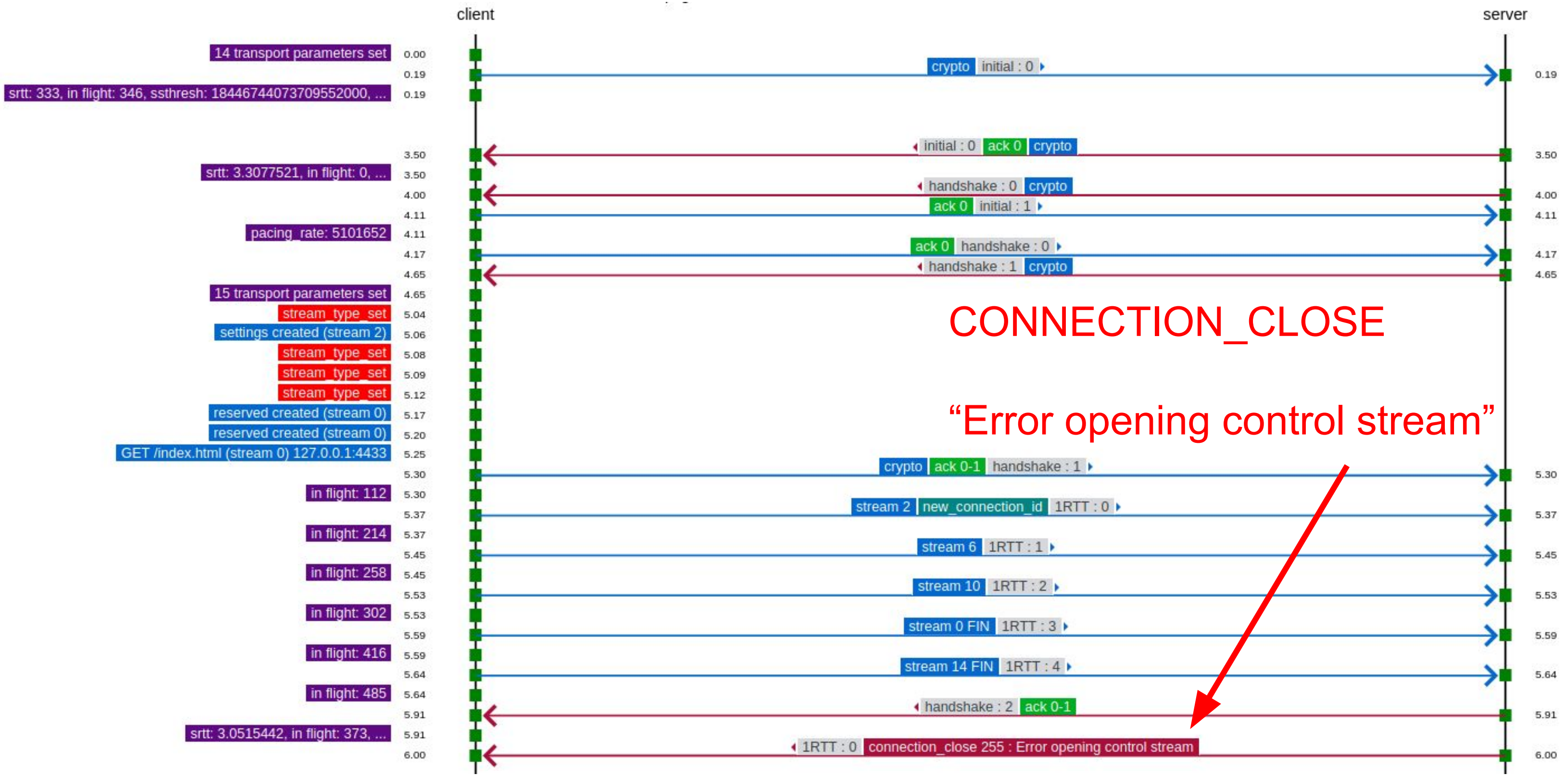
No.	Time	Source	Src port	Destination	Dst port	Protocol	Length	Info
1	0.000000000	127.0.0.1	56795	127.0.0.1	4433	QUIC	1242	Initial, DCID=7cefa90a37988cbd033262abb1a31183, SCID=2665e60d50be407f152fb5612d6f9bd816ab4724, PKN: 0, CRYPTO
2	0.003098500	127.0.0.1	4433	127.0.0.1	56795	QUIC	1242	Handshake, DCID=2665e60d50be407f152fb5612d6f9bd816ab4724, SCID=0a9e47e73eaf4c71affe685391fcc508f232e6f4, PKN: 0, CRYPTO
3	0.003933472	127.0.0.1	56795	127.0.0.1	4433	QUIC	1242	Handshake, DCID=0a9e47e73eaf4c71affe685391fcc508f232e6f4, SCID=2665e60d50be407f152fb5612d6f9bd816ab4724, PKN: 0, ACK
4	0.004270899	127.0.0.1	4433	127.0.0.1	56795	QUIC	491	Handshake, DCID=2665e60d50be407f152fb5612d6f9bd816ab4724, SCID=0a9e47e73eaf4c71affe685391fcc508f232e6f4, PKN: 1, CRYPTO
5	0.005129666	127.0.0.1	56795	127.0.0.1	4433	HTTP3	256	Protected Payload (KP0), DCID=0a9e47e73eaf4c71affe685391fcc508f232e6f4, PKN: 0, NCI, STREAM(2), SETTINGS
6	0.005208289	127.0.0.1	56795	127.0.0.1	4433	HTTP3	86	Protected Payload (KP0), DCID=0a9e47e73eaf4c71affe685391fcc508f232e6f4, PKN: 1, STREAM(6)
7	0.005270154	127.0.0.1	56795	127.0.0.1	4433	HTTP3	86	Protected Payload (KP0), DCID=0a9e47e73eaf4c71affe685391fcc508f232e6f4, PKN: 2, STREAM(10)
8	0.005321813	127.0.0.1	56795	127.0.0.1	4433	HTTP3	156	Protected Payload (KP0), DCID=0a9e47e73eaf4c71affe685391fcc508f232e6f4, PKN: 3, STREAM(0), HEADERS
9	0.005377782	127.0.0.1	56795	127.0.0.1	4433	HTTP3	111	Protected Payload (KP0), DCID=0a9e47e73eaf4c71affe685391fcc508f232e6f4, PKN: 4, STREAM(14)
10	0.005588657	127.0.0.1	4433	127.0.0.1	56795	QUIC	183	Protected Payload (KP0), DCID=2665e60d50be407f152fb5612d6f9bd816ab4724, 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: 56795
QUIC IETF
QUIC IETF
  [Packet Length: 70]
  QUIC Short Header DCID=2665e60d50be407f152fb5612d6f9bd816ab4724 PKN=0
    0... .... = Header Form: Short Header (0)
    .1.. .... = Fixed Bit: True
    ..0. .... = Spin Bit: False
    ...0 0... = Reserved: 0
    .... .0.. = Key Phase Bit: False
    .... ..00 = Packet Number Length: 1 bytes (0)
    Destination Connection ID: 2665e60d50be407f152fb5612d6f9bd816ab4724
    Packet Number: 0
    Protected Payload: 4a35e1d482752b08e16fb9bb046fc3c2aacfd15a2d2b437[illegible]75062e1b9318abd2cdb8936c…
  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 same real-world failure in qvis



Another real-world failure

“lucaspardue.com-0-streams-uni”

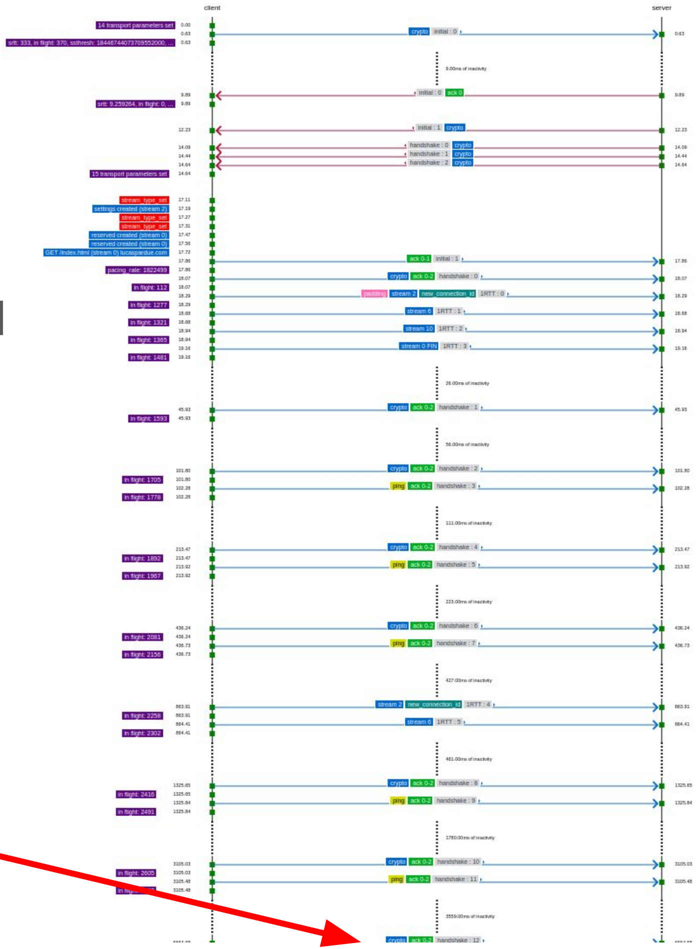
Client: `SSLKEYLOGFILE=tdd.keys QLOGDIR=qlogs quiche-client --wire-version 1 --max-streams-uni 0 https://lucaspardue.com/index.html`

1	0.000000000	1…	41130	104.19.21…	443	QUIC	1242	Initial, DCID=9f4dbec6d0216427d8d5a679a63dded8, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 0, CRYPTO
2	0.008726182	1…	443	10.80.106…	41130	QUIC	1242	Initial, DCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, SCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, PKN: 0, ACK
3	0.011080403	1…	443	10.80.106…	41130	QUIC	1242	Initial, DCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, SCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, PKN: 1, CRYPTO
4	0.011506188	1…	443	10.80.106…	41130	QUIC	1242	Handshake, DCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, SCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, PKN: 0, CRYPTO
5	0.012029339	1…	443	10.80.106…	41130	QUIC	1242	Handshake, DCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, SCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, PKN: 1, CRYPTO
6	0.012029421	1…	443	10.80.106…	41130	QUIC	416	Handshake, DCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, SCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, PKN: 2, CRYPTO
7	0.017662672	1…	41130	104.19.21…	443	HTTP3	1392	Protected Payload (KP0), DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, PKN: 0, NCI, STREAM(2), SETTINGS, PADDING
8	0.017948873	1…	41130	104.19.21…	443	HTTP3	86	Protected Payload (KP0), DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, PKN: 1, STREAM(6)
9	0.018171389	1…	41130	104.19.21…	443	HTTP3	86	Protected Payload (KP0), DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, PKN: 2, STREAM(10)
10	0.018365091	1…	41130	104.19.21…	443	HTTP3	158	Protected Payload (KP0), DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, PKN: 3, STREAM(0), HEADERS
11	0.045350678	1…	41130	104.19.21…	443	QUIC	154	Handshake, DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 1, ACK, CRYPTO
12	0.101260282	1…	41130	104.19.21…	443	QUIC	154	Handshake, DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 2, ACK, CRYPTO
13	0.101543928	1…	41130	104.19.21…	443	QUIC	115	Handshake, DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 3, ACK, PING
14	0.212904589	1…	41130	104.19.21…	443	QUIC	156	Handshake, DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 4, ACK, CRYPTO
15	0.213180438	1…	41130	104.19.21…	443	QUIC	117	Handshake, DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 5, ACK, PING
16	0.435701313	1…	41130	104.19.21…	443	QUIC	156	Handshake, DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 6, ACK, CRYPTO
17	0.436014089	1…	41130	104.19.21…	443	QUIC	117	Handshake, DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 7, ACK, PING
18	0.863388215	1…	41130	104.19.21…	443	HTTP3	144	Protected Payload (KP0), DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, PKN: 4, NCI, STREAM(2), SETTINGS
19	0.863653593	1…	41130	104.19.21…	443	HTTP3	86	Protected Payload (KP0), DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, PKN: 5, STREAM(6)
20	1.324844771	1…	41130	104.19.21…	443	QUIC	156	Handshake, DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 8, ACK, CRYPTO
21	1.324923865	1…	41130	104.19.21…	443	QUIC	117	Handshake, DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 9, ACK, PING
22	3.104458245	1…	41130	104.19.21…	443	QUIC	156	Handshake, DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 10, ACK, CRYPTO
23	3.104730891	1…	41130	104.19.21…	443	QUIC	117	Handshake, DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 11, ACK, PING
24	6.664160214	1…	41130	104.19.21…	443	QUIC	156	Handshake, DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 12, ACK, CRYPTO
25	6.664598210	1…	41130	104.19.21…	443	QUIC	117	Handshake, DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 13, ACK, PING
26	13.782575565	1…	41130	104.19.21…	443	QUIC	156	Handshake, DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 14, ACK, CRYPTO
27	13.782937848	1…	41130	104.19.21…	443	QUIC	117	Handshake, DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, SCID=0999a3ac68efffce2b2c1dc0eb15ca30cdf8c92e, PKN: 15, ACK, PING
28	27.899369598	1…	41130	104.19.21…	443	HTTP3	144	Protected Payload (KP0), DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, PKN: 6, NCI, STREAM(2), SETTINGS
29	27.899551362	1…	41130	104.19.21…	443	HTTP3	86	Protected Payload (KP0), DCID=01d826ef907d6fc9efd819efbd7d6dbe083c7c98, PKN: 7, STREAM(6)

Where’s the connection CONNECTION_CLOSE?

Another real-world failure (2)

The thing that sticks out is the trace is longer and there is no CONNECTION_CLOSE received by the server.



Where’s the connection CONNECTION_CLOSE?

Debugging the difference

1) Attempting to open a connection with initial_max_streams_uni = 0 to localhost elicits a CONNECTION_CLOSE from the server.
2) Attempting to open a connection with initial_max_streams_uni = 0 to lucaspardue.com causes no packets to be returned.
3) Different implementation? Not really, both servers powered by the same QUIC library.

So what could be the root cause?

Different types of CONNECTION_CLOSE

RFC 9000, [Section 10.2.3]

CONNECTION_CLOSE frame of type 0x1c is for transport layer.

CONNECTION_CLOSE frame of type 0x1d is for application layer.

"Sending a CONNECTION_CLOSE of type 0x1d in an Initial or Handshake packet could expose application state or be used to alter application state. A CONNECTION_CLOSE of type 0x1d MUST be replaced by a CONNECTION_CLOSE of type 0x1c when sending the frame in Initial or Handshake packets. Otherwise, information about the application state might be revealed. Endpoints MUST clear the value of the Reason Phrase field and SHOULD use the APPLICATION_ERROR code when converting to a CONNECTION_CLOSE of type 0x1c."

Trouble with timing, causing timeouts

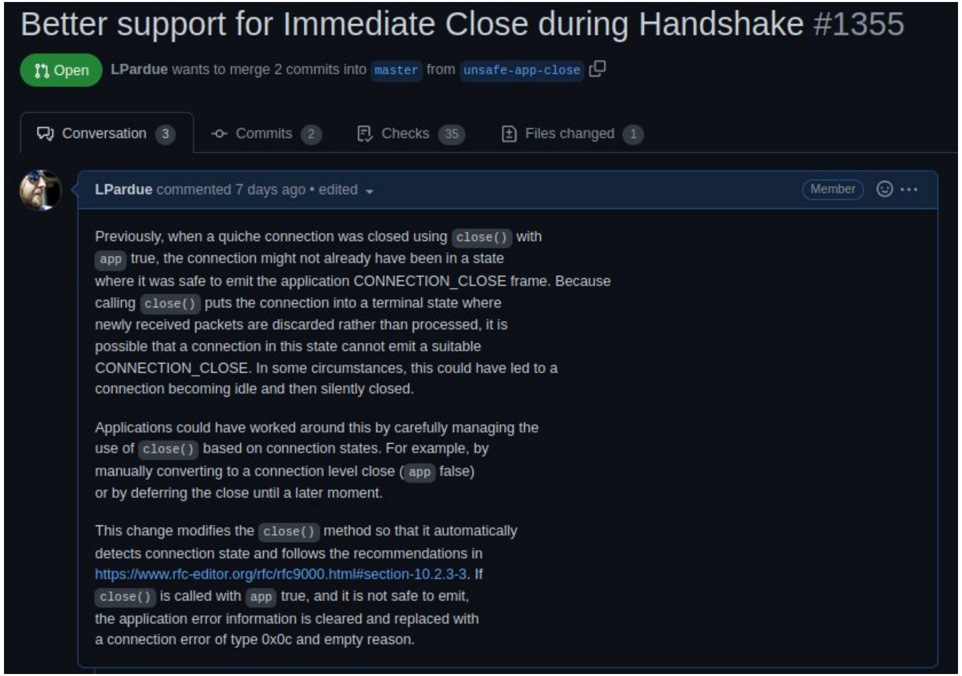
- Server, uses an HTTP/3 library.
- It sees initial_max_streams_uni = 0, it calls the QUIC library close() function, passing an error code and reason.
- Neither application nor HTTP/3 library check the transport state before closing it.
- Timing differences speaking to lucaspardue.com over the Internet.
- Handshake not complete when an application (0x1d) CONNECTION_CLOSE was triggered.
- Unsafe to send application errors => the server would not send a packet.
- After close(), server no longer processes client packets.
- Client retires, but eventually idle time out kicks in and it gives up.

Debugging leads to a fix

https://github.com/cloudflare/quiche/pull/1355

Automatically check the transport layer connection state and choose the most appropriate and safe type of error to emit.

Client always receives a timely close.



Better support for Immediate Close during Handshake #1355

Open LPardue wants to merge 2 commits into `master` from `unsafe-app-close`

Conversation 3 Commits 2 Checks 35 Files changed 1

LPardue commented 7 days ago • edited Member

Previously, when a quiche connection was closed using `close()` with `app` true, the connection might not already have been in a state where it was safe to emit the application CONNECTION_CLOSE frame. Because calling `close()` puts the connection into a terminal state where newly received packets are discarded rather than processed, it is possible that a connection in this state cannot emit a suitable CONNECTION_CLOSE. In some circumstances, this could have led to a connection becoming idle and then silently closed.

Applications could have worked around this by carefully managing the use of `close()` based on connection states. For example, by manually converting to a connection level close (`app` false) or by deferring the close until a later moment.

This change modifies the `close()` method so that it automatically detects connection state and follows the recommendations in https://www.rfc-editor.org/rfc/rfc9000.html#section-10.2.3-3. If `close()` is called with `app` true, and it is not safe to emit, the application error information is cleared and replaced with a connection error of type 0x0c and empty reason.

Summary

QUIC is not TCP, TLS, HTTP nor “the web over UDP”

QUIC is QUIC. It provides transport services for applications, such as multiplexed reliable byte streams. It doesn’t have much opinion about how these get used; see RFC 9308 for guidance and considerations for application protocols on top of QUIC. Define an ALPN identifier!

Minimal information in the wire image is observable; see RFC 9312. QUIC packets used in the handshake use a deterministic key. Once a secure connection is established, unique session keys are used.

Implementations and deployments can behave differently. Techniques that can decrypt (SSLKEYLOGFILE) or log plain text (qlog) can help analysis or debug.

Backup slides

qlog definition example

```
TransportPacketSent = {
    header: PacketHeader

    ? frames: [* $QuicFrame]
    ? is_coalesced: bool .default false
    ? retry_token: Token
    ? stateless_reset_token:
StatelessResetToken
    ? supported_versions: [+ QuicVersion]

    ? raw: RawInfo
    ? datagram_id: uint32

    ? is_mtu_probe_packet: bool .default false

    ? trigger:
      "retransmit_reordered" /
      "retransmit_timeout" /
      "pto_probe" /
      "retransmit_crypto" /
      "cc_bandwidth_probe"
}
```

```
; The QuicFrame is any key-value map (e.g., JSON object)
$QuicFrame /= {
    * text => any
}

$QuicFrame /= QuicBaseFrames

QuicBaseFrames /=
  PaddingFrame / PingFrame / AckFrame / ResetStreamFrame /
  StopSendingFrame / CryptoFrame / NewTokenFrame / StreamFrame /
  MaxDataFrame / MaxStreamDataFrame / MaxStreamsFrame /
  DataBlockedFrame / StreamDataBlockedFrame / StreamsBlockedFrame /
  NewConnectionIDFrame / RetireConnectionIDFrame /
  PathChallengeFrame / PathResponseFrame / ConnectionCloseFrame /
  HandshakeDoneFrame / UnknownFrame

PaddingFrame = {
    frame_type: "padding"

    ; total frame length, including frame header
    ? length: uint32
    payload_length: uint32
}

…
```

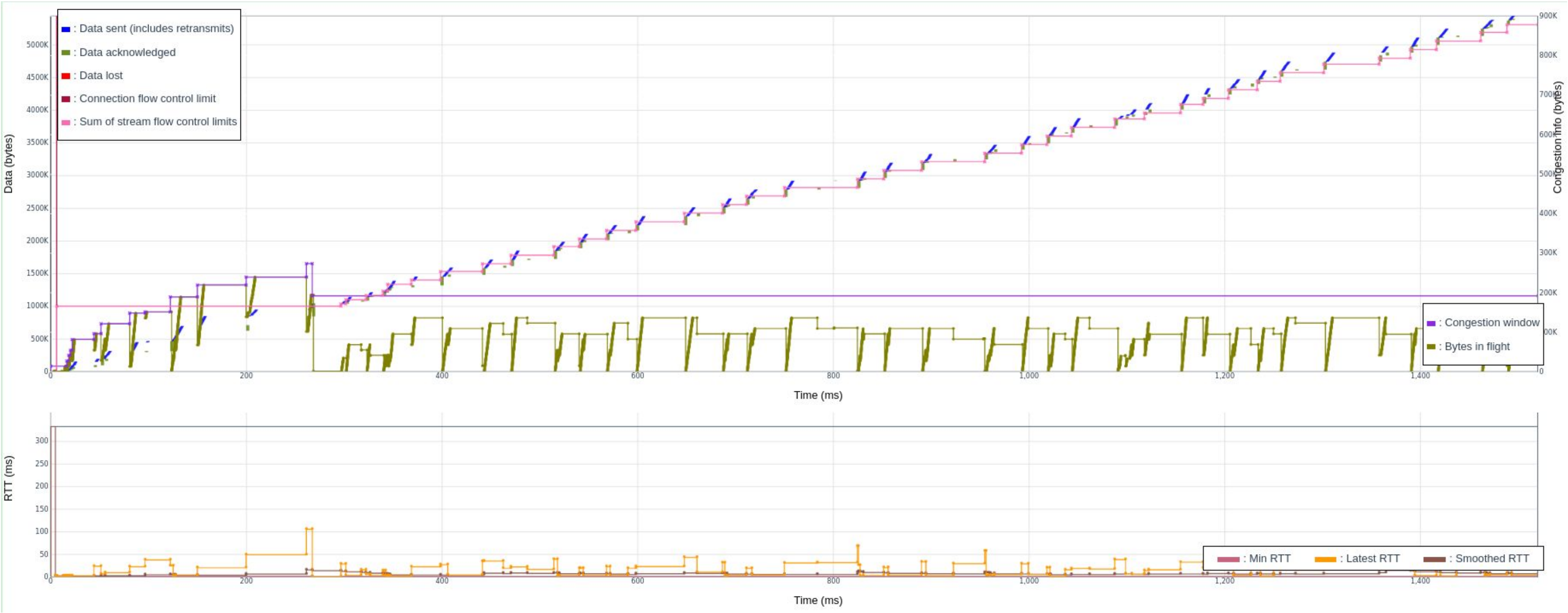
qlog example

Client: `QLOGDIR=qlogs quiche-client --no-verify --wire-version 1 https://127.0.0.1:4433/index.html`

Server: `QLOGDIR=qlogs quiche-server --no-retry`


```
{"qlog_version":"0.3","qlog_format":"JSON-SEQ","title":"quiche-client qlog","description":"quiche-client qlog
id=9463b9d6695a7b2d189da2871fc255977bc7c6f8","trace":{"vantage_point":{"type":"client"},"title":"quiche-client
qlog","description":"quiche-client qlog id=9463b9d6695a7b2d189da2871fc255977bc7c6f8","configuration":{"time_offset":0.0}}}
{"time":0.0,"name":"transport:parameters_set","data":{"owner":"local","tls_cipher":"None","disable_active_migration":true,"max_idle_t
imeout":30000,"max_udp_payload_size":1350,"ack_delay_exponent":3,"max_ack_delay":25,"active_connection_id_limit":2,"initial_max_data"
:10000000,"initial_max_stream_data_bidi_local":1000000,"initial_max_stream_data_bidi_remote":1000000,"initial_max_stream_data_uni":10
00000,"initial_max_streams_bidi":100,"initial_max_streams_uni":100}}
{"time":0.207949,"name":"transport:packet_sent","data":{"header":{"packet_type":"initial","packet_number":0,"version":"1","scil":20,"
dcil":16,"scid":"9463b9d6695a7b2d189da2871fc255977bc7c6f8","dcid":"6c94d2c299cbff6253a202bcb20ceb42"},"raw":{"length":350,"payload_le
ngth":287},"send_at_time":0.207949,"frames":[{"frame_type":"crypto","offset":0,"length":283}]}}
{"time":0.207949,"name":"recovery:metrics_updated","data":{"smoothed_rtt":333.0,"rtt_variance":166.5,"congestion_window":13500,"bytes
_in_flight":350,"ssthresh":18446744073709551615}}
{"time":3.5715451,"name":"transport:packet_received","data":{"header":{"packet_type":"initial","packet_number":0,"version":"1","scil"
:20,"dcil":20,"scid":"78015def011d1adf3af94c44067955dd4d52fc70","dcid":"9463b9d6695a7b2d189da2871fc255977bc7c6f8"},"raw":{"length":12
00,"payload_length":117},"frames":[{"frame_type":"ack","ack_delay":0.305,"acked_ranges":[[0,0]]},{"frame_type":"crypto","offset":0,"l
ength":90}]}}
```

Congestion control behavior



Applicability of QUIC - streams

Streams are a core capability of RFC 9000.

Streams in QUIC provide a lightweight, **ordered** byte-stream abstraction to an application.

Streams can be created by either endpoint, can concurrently send data interleaved with other streams, and can be canceled. QUIC **does not provide any means of ensuring ordering between bytes on different streams**.

QUIC allows for an arbitrary number of streams to operate concurrently and for an arbitrary amount of data to be sent on any stream, subject to flow control constraints and stream limits.

Applicability of QUIC - stream IDs

- Streams can be unidirectional or bidirectional.
- Unidirectional streams carry data in one direction: from the initiator of the stream to its peer.
- Bidirectional streams allow for data to be sent in both directions. Streams are identified within a connection by a numeric value, referred to as the stream ID.
- A stream ID is a 62-bit integer (0 to 2^{62}-1) that is unique for all streams on a connection.
- The least significant bit (0x01) of the stream ID identifies the initiator of the stream. The second least significant bit (0x02) of the stream ID distinguishes between bidirectional and unidirectional.

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

What does all that mean?

Applications have a toolkit of streams to use.

QUIC has no opinion how you use those streams, as long as the transport requirements on IDs and flow control are obeyed.

Application mappings like HTTP/3 (RFC 9114) or DNS over QUIC (RFC 9250) describe how application messages utilise QUIC streams.

Streams example: HTTP/3

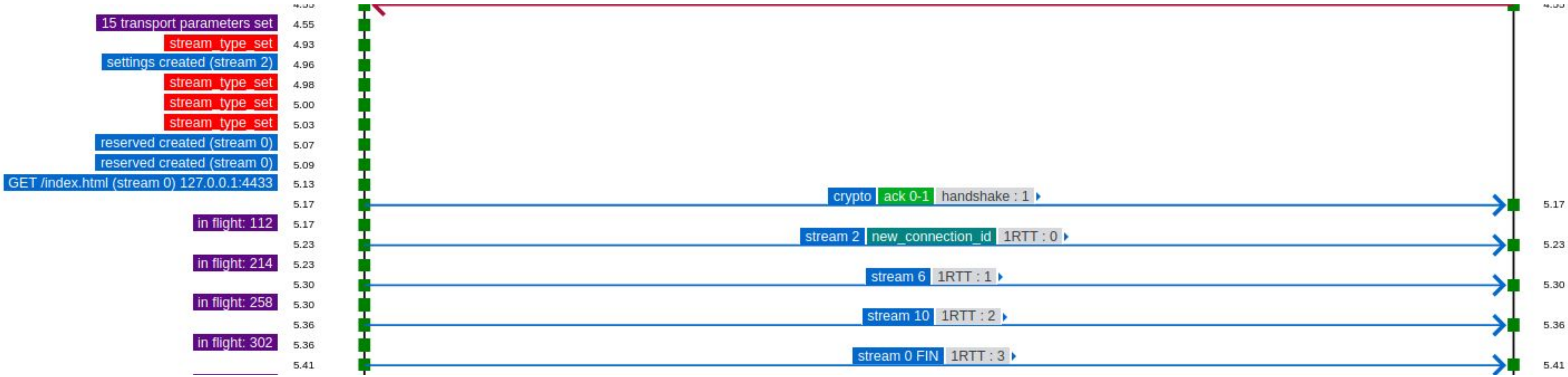
Client-initiated bidirectional streams are always used for request and response exchanges.

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

Each endpoint creates mandatory unidirectional control streams: Control, QPACK encoder, QPACK decoder.

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

Streams example: HTTP/3



Control stream on ID 2. QPACK streams on ID 6 and 10.

Request stream on ID 0. GET request for /index.html. Stream is FIN'd to indicate request message is complete

Streams example: HTTP/3

```
  5 0.004963775   127.0.0.1          43959 127.0.0.1      4433 HTTP3      256 Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70, PKN: 0, NCI, STREAM(2), SETTINGS

▸ Frame 5: 256 bytes on wire (2048 bits), 256 bytes captured (2048 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 IETF
    [Packet Length: 102]
  ▾ QUIC Short Header DCID=78015def011d1adf3af94c44067955dd4d52fc70 PKN=0
      0... .... = Header Form: Short Header (0)
      .1.. .... = Fixed Bit: True
      ..0. .... = Spin Bit: False
      ...0 0... = Reserved: 0
      .... .0.. = Key Phase Bit: False
      .... ..00 = Packet Number Length: 1 bytes (0)
      Destination Connection ID: 78015def011d1adf3af94c44067955dd4d52fc70
      Packet Number: 0
      Protected Payload: 03b7d8dfe40be2186a8251313d79001ec5d1d0e10dc73ae1213658fe7cfa6292b991553f…
  ▾ NEW_CONNECTION_ID
      Frame Type: NEW_CONNECTION_ID (0x0000000000000018)
      Sequence: 1
      Retire Prior To: 0
      Connection ID Length: 20
      Connection ID: 5a5896ac2c7ba6d164c6b616bd6409af74edd55f
      Stateless Reset Token: 86a804a6b016cc69312dce777734425b
  ▾ STREAM id=2 fin=0 off=0 len=19 uni=1
    ▸ Frame Type: STREAM (0x000000000000000e)
      Stream ID: 2
      Offset: 0
      Length: 19
      Stream Data: 000410e0b9395476f5e936ef7147d23285d941
▾ Hypertext Transfer Protocol Version 3
    Stream Type: Control Stream (0x0000000000000000)
    Type: SETTINGS (0x0000000000000004)
    Length: 16
    Frame Payload: e0b9395476f5e936ef7147d23285d941
```

Stream gotchas for applications 1

Concurrency and flow control have limits.

An endpoint tells its peer the initial limits using Transport Parameters in the QUIC handshake.

QUIC control frames like MAX_STREAMS, MAX_DATA, MAX_STREAM_DATA can be used to update limits during the connection lifetime.

QUIC doesn’t have an opinion. This is an application matter. There is no universal default. Implementations of applications probably have an opinion on defaults and behaviours.

Stream gotchas for applications 2

Transport Parameters apply to a QUIC connection, they affect applications.

Clients can offer many types of application protocols in their ALPN.

Servers can only pick one.

Applications might have specifications that disagree on suitable Transport Parameters.

For example, HTTP/3 control streams are mandatory. If an endpoint never gives credit to its peer to allow these streams to be opened, the peer might get upset.

Stream multiplexing

Unlike TCP, QUIC offers multiplexing of byte streams within the connection. This offers fruitful capability and fertile ground for new behaviours that might be hard to observe or debug.

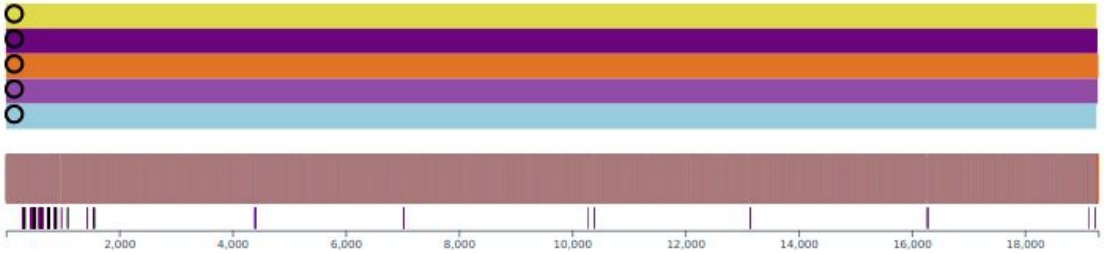
Streams compete for connection bandwidth. Not all streams are equal. E.g., streams for a control channel might be more important that bulk data.

QUIC does not provide global ordering of stream data in a connection. Stream IDs indicate stream creation order but data from different streams can arrive at any time. Applications that depend on ordering across streams need to implement application-layer synchronization.

Example: HTTP/3 prioritization shown in qvis

5 concurrent transfers of 5 MB, all urgency=1

quiche (before priorities)
round-robin



quiche (now)
FIFO