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

HTTP/1.1, HTTP/2
- TLS
- TCP
- IP

HTTP/3
- TLS
- QUIC
- UDP
- IP
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
Follow-along examples

https://github.com/LPardue/sreconemea2023

"localhost-good.pcapng"

To successfully dissect QUIC packets, Wireshark 3.4.x and onwards.
<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Src port</th>
<th>Destination</th>
<th>Dist port</th>
<th>Protocol</th>
<th>Length info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16:00:00</td>
<td>8999</td>
<td>127.0.0.1</td>
<td>43999</td>
<td>127.0.0.1</td>
<td>4429</td>
<td>QUIC 2247</td>
</tr>
</tbody>
</table>

Frame 1: 1424 bytes on wire (9396 bits), 1424 bytes captured (9396 bits) on interface lo, id 0
- Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dist: 00:00:00:00:00:00
- Internet Protocol Version 4, Src: 127.0.0.1, Dist: 127.0.0.1
- User Datagram Protocol, Src Port: 43999, Dist Port: 4429

**QUIC Header**
- Packet Length: 359
- Packet Number: 8
- Transport Parameters: ALPN
- Handshake Protocol: ClientHello
- Extension: quic_transport_parameters (len=72)

**Transport Parameters**
- max_idle_timeout (len=4) 30000 ms
- max_udp_payload_size (len=2) 1350
- initial_max_data (len=4) 10000000
- initial_max_stream_data_uni (len=4) 10000000
- initial_max_streams_bidi (len=2) 100
- initial_max_streams_uni (len=2) 100
- ack_delay_exponent (len=1) 0
- max_ack_delay (len=1) 25
- disable_active_migration (len=0)
- 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](https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml#alpn-protocol-ids)

---

<table>
<thead>
<tr>
<th>Extension: application_layer_protocol_negotiation (len=61)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: application_layer_protocol_negotiation (16)</td>
</tr>
<tr>
<td>Length: 61</td>
</tr>
<tr>
<td>ALPN Extension Length: 59</td>
</tr>
<tr>
<td>ALPN Protocol</td>
</tr>
<tr>
<td>ALPN string length: 2</td>
</tr>
<tr>
<td>ALPN Next Protocol: h3</td>
</tr>
<tr>
<td>ALPN string length: 5</td>
</tr>
<tr>
<td>ALPN Next Protocol: h3-29</td>
</tr>
<tr>
<td>ALPN string length: 5</td>
</tr>
<tr>
<td>ALPN Next Protocol: h3-28</td>
</tr>
<tr>
<td>ALPN string length: 5</td>
</tr>
<tr>
<td>ALPN Next Protocol: h3-27</td>
</tr>
<tr>
<td>ALPN string length: 10</td>
</tr>
<tr>
<td>ALPN Next Protocol: hq-interop</td>
</tr>
<tr>
<td>ALPN string length: 5</td>
</tr>
<tr>
<td>ALPN Next Protocol: hq-29</td>
</tr>
<tr>
<td>ALPN string length: 5</td>
</tr>
<tr>
<td>ALPN Next Protocol: hq-28</td>
</tr>
<tr>
<td>ALPN string length: 5</td>
</tr>
<tr>
<td>ALPN Next Protocol: hq-27</td>
</tr>
<tr>
<td>ALPN string length: 8</td>
</tr>
<tr>
<td>ALPN Next Protocol: http/0.9</td>
</tr>
</tbody>
</table>

---

### Example Values:

- h3: HTTP/2
- h2: HTTP/1.1
- h3-28: HTTP/3
- h2-25: HTTP/2.5
- q-h2: Quic HTTP/2
- q-oop: Quic HTTP/0.5
- q-h2p: Quic HTTP/2.0
- q-http: Quic HTTP/1.1
- h: HTTP/1.1
- h-28: HTTP/2.8
- h-27: HTTP/2.7
- h-26: HTTP/2.6
- h-25: HTTP/2.5
- h-24: HTTP/2.4
- h-23: HTTP/2.3
- h-22: HTTP/2.2
- h-21: HTTP/2.1
- h-20: HTTP/2.0
- h-19: HTTP/1.9
- h-18: HTTP/1.8
- h-17: HTTP/1.7
- h-16: HTTP/1.6
- h-15: HTTP/1.5
- h-14: HTTP/1.4
- h-13: HTTP/1.3
- h-12: HTTP/1.2
- h-11: HTTP/1.1
- h-10: HTTP/1.0
- h-9: HTTP/0.9
- h-8: HTTP/0.8
- h-7: HTTP/0.7
- h-6: HTTP/0.6
- h-5: HTTP/0.5
- h-4: HTTP/0.4
- h-3: HTTP/0.3
- h-2: HTTP/0.2
- h-1: HTTP/0.1
- h-0: HTTP/0.0
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.
Server picks on ALPN value. This is the negotiated application protocol.

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

**Decrypted traffic**

https://wiki.wireshark.org/TLS

Server Handshake

Encrypted Extensions

ALPN
Decrypted post-handshake QUIC and HTTP/3

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Length Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUIC</td>
<td>1242 Initial, DCID=f7cc19997f578525a476d84d1395c6a1, SCID=d2c4c3e72f2ee7cd93a2104464c95fd1fd43cd53ff4, PKN: 0, CRYPTO</td>
</tr>
<tr>
<td>QUIC</td>
<td>1242 Handshake, DCID=d24c3e72f2ee7cd93a2104464c95fd1fd43cd53ff4, SCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 0, CRYPTO</td>
</tr>
<tr>
<td>QUIC</td>
<td>1242 Handshake, DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, SCID=d24c3e72f2ee7cd93a2104464c95fd1fd43cd53ff4, PKN: 0, ACK</td>
</tr>
<tr>
<td>QUIC</td>
<td>491 Handshake, DCID=d24c3e72f2ee7cd93a2104464c95fd1fd43cd53ff4, SCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 1, CRYPTO</td>
</tr>
<tr>
<td>HTTP/3</td>
<td>256 Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 0, NCI, STREAM(2)</td>
</tr>
<tr>
<td>HTTP/3</td>
<td>86 Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 1, STREAM(6)</td>
</tr>
<tr>
<td>HTTP/3</td>
<td>86 Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 2, STREAM(10)</td>
</tr>
<tr>
<td>HTTP/3</td>
<td>157 Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 3, STREAM(9), HEADERS: GET /index.html</td>
</tr>
<tr>
<td>HTTP/3</td>
<td>111 Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 4, STREAM(14)</td>
</tr>
<tr>
<td>HTTP/3</td>
<td>622 Protected Payload (KP0), DCID=d2c4c3e72f2ee7cd93a2104464c95fd1fd43cd53ff4, PKN: 0, ACK, NCI, DONE, CRYPTO, STREAM(3)</td>
</tr>
<tr>
<td>QUIC</td>
<td>85 Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 5, ACK</td>
</tr>
<tr>
<td>HTTP/3</td>
<td>86 Protected Payload (KP0), DCID=d2c4c3e72f2ee7cd93a2104464c95fd1fd43cd53ff4, PKN: 1, STREAM(7)</td>
</tr>
<tr>
<td>QUIC</td>
<td>85 Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 6, ACK</td>
</tr>
<tr>
<td>HTTP/3</td>
<td>86 Protected Payload (KP0), DCID=d2c4c3e72f2ee7cd93a2104464c95fd1fd43cd53ff4, PKN: 2, STREAM(11)</td>
</tr>
<tr>
<td>QUIC</td>
<td>85 Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 7, ACK</td>
</tr>
<tr>
<td>HTTP/3</td>
<td>150 Protected Payload (KP0), DCID=d2c4c3e72f2ee7cd93a2104464c95fd1fd43cd53ff4, PKN: 3, STREAM(9), HEADERS: 404 Not Found, DATA</td>
</tr>
<tr>
<td>QUIC</td>
<td>90 Protected Payload (KP0), DCID=770f4e294bf6006863eca225ce89c6c1f72d0963, PKN: 8, CC</td>
</tr>
</tbody>
</table>

HTTP GET /index.html request

HTTP 404 response
“Something broke and I have no idea why”
CONNECTION_CLOSE and error codes

QUIC provides explicit close using a CONNECTION_CLOSE frame.

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.

Client → Server DCID: 770f4e294bf6006863eca225ce89c6c1f72d0963
Server→ Client DCID: d24c3e72fee7cd93a210446c95fd1fd43cd53ff4

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 globallig 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 (0x01) identifies the initiator of the stream - client or server.
- Second least significant bit (0x02) distinguishes between bidirectional and unidirectional.

<table>
<thead>
<tr>
<th>Bits</th>
<th>Stream Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Client-Initiated, Bidirectional</td>
</tr>
<tr>
<td>0x01</td>
<td>Server-Initiated, Bidirectional</td>
</tr>
<tr>
<td>0x02</td>
<td>Client-Initiated, Unidirectional</td>
</tr>
<tr>
<td>0x03</td>
<td>Server-Initiated, Unidirectional</td>
</tr>
</tbody>
</table>
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](https://tools.ietf.org/html/rfc9114)) or DNS over QUIC ([RFC 9250](https://tools.ietf.org/html/rfc9250)) 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

HTTP Request or Response

HTTP/3 Frame

QUIC Stream

QUIC Frame

QUIC Packet

UDP Datagram
Unreliable datagrams (the third-best thing)
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

CONNECT over any HTTP version
HTTP/3 extended CONNECT over QUIC and UDP
End-to-end flow of unreliable messages

TCP
End-to-end TLS
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)

HTTP/3 Extended CONNECT over QUIC and UDP

End-to-end flow of unreliable messages

UDP
Anatomy of encapsulation

UDP datagram

QUIC Packet

QUIC DATAGRAM frame

Quarter Stream ID

UDP datagram payload
QUIC over QUIC framing and packetization
Nested tunneling
TLS over QUIC over QUIC framing and packetization

End-to-end TLS

STREAM Frame
QUIC Packet
UDP Datagram
DATAGRAM Frame
QUIC Packet
UDP Datagram
STREAM Frame
QUIC Packet
UDP Datagram
DATAGRAM Frame
QUIC Packet
UDP Datagram
STREAM Frame
QUIC Packet
UDP Datagram
DATAGRAM Frame
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