Deploying and Debugging HTTP/3

and other cautionary tales

Robin Marx

@programmingart
Shared state between Client and Server

Common occurrence:
- Client establishes a **first** connection to the server
- They exchange **some** state
- That state can be re-used to improve the next connection

Several use cases:
- Session resumption (e.g., session tickets)
- Authentication/SSO (e.g., JWT)
- Address validation
- Connection parameters
  - …
You don’t ACTUALLY want to keep state at server

Shared state explosion with distributed backends:
- All either need a copy
- OR need to query shared database = slow

Actual solution:
- Encrypt state on server with **private key**
- Send blob to **client to store**
- Client sends blob with next connection
- ⇒ Servers “only” need **shared private key**!
What happens…

if the blob generated by new server code, causes old server code to CRASH?
How would you handle this?

Rollback doesn’t “fix” it
- Stops handing out new Hungries, but existing ones will still be used

Just deploy new version to 100%?
- I see you like to live dangerously

QUIC Exit: A New class of Outage : https://www.youtube.com/watch?v=PLKNgvC5zbA
How would you handle this?

Rollback doesn’t “fix” it
- Stops handing out new Hungries, but existing ones will still be used

Just deploy new version to 100%?
- I see you like to live dangerously

Actual solution they used: BIG RED BUTTON!
- Disable QUIC on the fleet for x days

“New type of bug” : Contagion Bug
- Contaminated client state, impossible to purge

QUIC Exit: A new class of Outage: https://www.youtube.com/watch?v=PLKNgvC5zbA
Firewall returns 403 Forbidden for HTTP/3, on a domain that has H3 disabled...
Site Site, Different (sub)Domains

www.SRE.com

static.SRE.com
Different Domains, Different Connections

www.SRE.com

static.SRE.com
Different Domains, Same “Server”

www.SRE.com

static.SRE.com
Different Domains, Common Connection

Connection Coalescing / Re-use

Re-use existing connection if:
- DNS resolves to same IP *(optional)*
- **TLS** cert is authoritative for both (SAN)

www.SRE.com

static.SRE.com
Same Server, Different Protocols

www.SRE.com
HTTP/2 only!

static.SRE.com
HTTP/3 allowed
Different Protocols, Different Connections

HTTP/2 only!

HTTP/3 allowed
Different Protocols, Common Connection

Connection Re-use
Doesn’t care about protocol

HTTP/3 only!
www.SRE.com

HTTP/3 allowed
static.SRE.com
Different Protocols, Common Connection

Connection Re-use
Doesn’t care about protocol

STOP
403
FORBIDDEN

www.SRE.com
HTTP/2 only!

static.SRE.com
HTTP/3 allowed
Different Protocols, Common Connection

Possible “solutions”:
1. Disable HTTP/3 everywhere
2. Split TLS Certificates
Multi-CDN with fast failover OR live traffic split

SLOW.LY + Akamai

HTTP/2 + HTTP/3
Only on

November 2024

HBO Max™
HTTP/3 is 50% faster than HTTP/2, but it should be 33% or 66%...
QUIC hates wasting time on handshakes

TCP
TLS 1.3
HTTP/2

QUIC
TLS 1.3
HTTP/3

QUIC
TLS 1.3
HTTP/3

with 0-RTT

## Measured Impact (2021)

### p75 Time to First Byte

<table>
<thead>
<tr>
<th>Country Code</th>
<th>Mobile</th>
<th>HTTP/2</th>
<th>HTTP/3</th>
</tr>
</thead>
<tbody>
<tr>
<td>AE</td>
<td>0.74 (-3.32ms, 34.63%)</td>
<td>1.13</td>
<td></td>
</tr>
<tr>
<td>AR</td>
<td>0.51 (-544ms, 51.66%)</td>
<td>1.05</td>
<td></td>
</tr>
<tr>
<td>AU</td>
<td>0.34 (-271ms, 44.28%)</td>
<td>0.61</td>
<td></td>
</tr>
<tr>
<td>BR</td>
<td>0.53 (-556ms, 51.06%)</td>
<td>1.09</td>
<td></td>
</tr>
<tr>
<td>CA</td>
<td>0.31 (-207ms, 39.81%)</td>
<td>0.52</td>
<td></td>
</tr>
<tr>
<td>CH</td>
<td>0.32 (-156ms, 32.50%)</td>
<td>0.48</td>
<td></td>
</tr>
<tr>
<td>CL</td>
<td>0.41 (-304ms, 42.64%)</td>
<td>0.71</td>
<td></td>
</tr>
<tr>
<td>ES</td>
<td>0.35 (-261ms, 42.65%)</td>
<td>0.61</td>
<td></td>
</tr>
<tr>
<td>FR</td>
<td>0.39 (-224ms, 36.78%)</td>
<td>0.61</td>
<td></td>
</tr>
<tr>
<td>GB</td>
<td>0.31 (-188ms, 37.98%)</td>
<td>0.50</td>
<td></td>
</tr>
<tr>
<td>IL</td>
<td>0.46 (-334ms, 42.07%)</td>
<td>0.79</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td>0.56 (-504ms, 47.50%)</td>
<td>1.06</td>
<td></td>
</tr>
<tr>
<td>IT</td>
<td>0.35 (-283ms, 44.92%)</td>
<td>0.63</td>
<td></td>
</tr>
<tr>
<td>JP</td>
<td>0.31 (-262ms, 45.49%)</td>
<td>0.58</td>
<td></td>
</tr>
<tr>
<td>MX</td>
<td>0.51 (-422ms, 45.43%)</td>
<td>0.93</td>
<td></td>
</tr>
<tr>
<td>NL</td>
<td>0.28 (-187ms, 40.39%)</td>
<td>0.46</td>
<td></td>
</tr>
<tr>
<td>NZ</td>
<td>0.47 (-203ms, 30.21%)</td>
<td>0.67</td>
<td></td>
</tr>
<tr>
<td>PH</td>
<td>0.55 (-678ms, 55.12%)</td>
<td>1.23</td>
<td></td>
</tr>
</tbody>
</table>

### India:
- 47.50% faster!
- 1000ms to 560ms!

### Philippines:
- 55% faster!
- 1230ms to 550ms!
Hold on a minute...

TCP
TLS 1.3
HTTP/2

100%

VS

QUIC
TLS 1.3
HTTP/3

~33% faster

Yet we see
~50%?

QUIC
TLS 1.3
HTTP/3

~66% faster

IT’S ALWAYS DNS
Unless it’s BGP...
Hold on a minute...

VS

100%

~50% faster

= unfair comparison

Browser only do HTTP/3 after *discovery*

For a new hostname browser doesn’t know yet:

1. Browser requests the page over HTTP/1 or 2

2. Server sends back **alternative services** header

<table>
<thead>
<tr>
<th>Response Headers</th>
</tr>
</thead>
<tbody>
<tr>
<td>alt-svc: h3=&quot;:443&quot;; ma=86400</td>
</tr>
<tr>
<td>cache-control: private, no-cache, no-store, must-revalidate</td>
</tr>
</tbody>
</table>

3. Browser stores alt-svc info in alt-svc “cache”

4. Browser also tries HTTP/3 from now on, *in parallel* with HTTP/1 and 2 ("free" fallback)
Hold on a minute...

VS

~50% faster

100%
## Measured Impact, Fair (2021)

### p75 Time to First Byte

<table>
<thead>
<tr>
<th>Country Code</th>
<th>Mobile Mean</th>
<th>HTTP/2 Mean (ms, % faster)</th>
<th>HTTP/3 Mean (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>0.22</td>
<td>(102ms, -31.38%)</td>
<td>0.33</td>
</tr>
<tr>
<td>AE</td>
<td>0.30</td>
<td>(252ms, -45.41%)</td>
<td>0.26</td>
</tr>
<tr>
<td>AR</td>
<td>0.26</td>
<td>(152ms, -36.63%)</td>
<td>0.42</td>
</tr>
<tr>
<td>AU</td>
<td>0.13</td>
<td>(64ms, +23.68%)</td>
<td>0.19</td>
</tr>
<tr>
<td>BR</td>
<td>0.23</td>
<td>(-102ms, -30.45%)</td>
<td>0.34</td>
</tr>
<tr>
<td>CA</td>
<td>0.17</td>
<td>(-77ms, -31.69%)</td>
<td>0.24</td>
</tr>
<tr>
<td>CH</td>
<td>0.16</td>
<td>(-71ms, -31.14%)</td>
<td>0.23</td>
</tr>
<tr>
<td>CL</td>
<td>0.19</td>
<td>(67ms, +25.77%)</td>
<td>0.26</td>
</tr>
<tr>
<td>ES</td>
<td>0.22</td>
<td>(-65ms, -22.89%)</td>
<td>0.28</td>
</tr>
<tr>
<td>FR</td>
<td>0.22</td>
<td>(-70ms, -24.48%)</td>
<td>0.29</td>
</tr>
<tr>
<td>GB</td>
<td>0.17</td>
<td>(-72ms, -30.25%)</td>
<td>0.24</td>
</tr>
<tr>
<td>IL</td>
<td>0.25</td>
<td>(-168ms, -33.51%)</td>
<td>0.38</td>
</tr>
<tr>
<td>IN</td>
<td>0.33</td>
<td>(95ms, +22.62%)</td>
<td>0.42</td>
</tr>
<tr>
<td>IT</td>
<td>0.20</td>
<td>(-93ms, -31.74%)</td>
<td>0.29</td>
</tr>
<tr>
<td>JP</td>
<td>0.15</td>
<td>(-77ms, -33.77%)</td>
<td>0.23</td>
</tr>
<tr>
<td>MX</td>
<td>0.30</td>
<td>(-132ms, -30.41%)</td>
<td>0.43</td>
</tr>
<tr>
<td>NL</td>
<td>0.15</td>
<td>(-52ms, -25.62%)</td>
<td>0.20</td>
</tr>
<tr>
<td>NZ</td>
<td>0.14</td>
<td>(-85ms, -37.78%)</td>
<td>0.23</td>
</tr>
<tr>
<td>PH</td>
<td>0.41</td>
<td>(-219ms, -34.60%)</td>
<td>0.63</td>
</tr>
</tbody>
</table>

**Mean:**
- 31% faster!
- 330ms to 220ms!

**Philippines:**
- 34% faster!
- 630ms to 410ms!
Visual tooling helps me a lot

https://qvis.quictools.info
Why can’t we just use **Wireshark**?

- Can’t just decrypt QUIC/H3 details: all or nothing
- Don’t always have TLS decryption keys
  - Facebook refused to even add SSL_KEYLOGFILE in their stack
- Some features not fully supported
  - HTTP/3 QPACK header decoding was added **just 3 weeks ago**!
- Wireshark JSON/XML output isn’t easy to use
  - by default, JSON even contains **duplicate keys**…
- Wire image does not contain all info
  - No **congestion window**, reasons why implementation made decision X
Parsing random application logs is “FUN”
qlog examples

```json
{
    "time": 15000,
    "name": "transport: packet_received",
    "data": {
        "header": {
            "packet_type": "1rtt",
            "packet_number": 25
        },
        "frames": [
            {
                "frame_type": "ack",
                "acked_ranges": [
                    [10,15],
                    [17,20]
                ]
            }
        ]
    }
}
```

```json
{
    "time": 15001,
    "name": "recovery: metrics_updated",
    "data": {
        "min_rtt": 25,
        "smoothed_rtt": 30,
        "latest_rtt": 25,
        "congestion_window": 60,
        "bytes_in_flight": 77000
    }
}
```
>70% of QUIC implementations have (partial) support:
- aioquic
- quic-go
- quiche
- mvfst
- picoquic
- haskell
- ngtcp2
- ...

Others do something similar:
- msquic
- google quiche

Standardization in-progress @ https://github.com/quicwg/qlog

@rmars we currently have qlog enabled in prod with similar amounts of events being recorded a day as I quoted before (dozens of billions).
Too many Selective Acknowledgements?

https://qvis.quictools.info
<qvis> Congestion diagram

https://qvis.quictools.info
<qvis> Congestion diagram

https://qvis.quictools.info
HTTP/3 not faster than HTTP/2 at all, we want our money back!
Faster Handshake Theory

TCP
TLS 1.3
HTTP/2

QUIC
TLS 1.3
HTTP/3

https://understanding-quic.net/
https://blog.apnic.net/2023/01/16/on-the-interplay-between-tls-certificates-and-quic-performance/
Faster Handshake in **Practice**...
UDP reflection / amplification attack

Memcrashed: $51000x$ amplification

5x bandwidth amplification towards victim
QUIC vulnerable **during** handshake: 0-RTT

<table>
<thead>
<tr>
<th>victim IP: 1.2.3.4</th>
<th>attacker IP: 6.7.8.9</th>
<th>server IP: 5.5.5.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source IP</td>
<td>Destination IP</td>
<td>1.2.3.4</td>
</tr>
<tr>
<td>1.2.3.4</td>
<td>5.5.5.5</td>
<td></td>
</tr>
</tbody>
</table>

1. the attacker spoofs the source IP on the 0-RTT request

2. the server sends a lot of response data to the victim
Amplification prevention in QUIC

1. The attacker spoofs the source IP on the 0-RTT request.
2. The server has an amplification limit of 3x the data received from the client.
Not just 0-RTT, also “normal” handshake

1. The attacker spoofs the source IP on the handshake

2. The server has an amplification limit of 3x the data received from the client

TLS certificate can push handshake size > 3x limit
curl-http3 docker image

Supports both pcaps (with TLS keys) and qlog output!

docker run -it --rm
  --volume $(pwd)/pcaps_on_host:/srv
  --env QLOGDIR=/srv
  --env SSLKEYLOGFILE=/srv/tls_keys.txt

rmarx/curl-http3

bash -c "tcpdump -w /srv/packets.pcap -i eth0 & sleep 1;
curl -IL https://www.sre.com --http3;
sleep 2; pkill tcpdump; sleep 2"

https://github.com/rmarx/curl-http3
<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0.826686</td>
<td>172.17.0.2</td>
<td>192.168.65.5</td>
<td>DNS</td>
<td>78</td>
<td>Standard query 0x451d A <a href="http://www.socal.com">www.socal.com</a></td>
</tr>
<tr>
<td>8</td>
<td>0.826840</td>
<td>172.17.0.2</td>
<td>192.168.65.5</td>
<td>DNS</td>
<td>78</td>
<td>Standard query 0x4f1d AAA <a href="http://www.socal.com">www.socal.com</a></td>
</tr>
<tr>
<td>9</td>
<td>0.889106</td>
<td>192.168.65.5</td>
<td>172.17.0.2</td>
<td>DNS</td>
<td>220</td>
<td>Standard query response 0x451d</td>
</tr>
<tr>
<td>...</td>
<td>0.889651</td>
<td>192.168.65.5</td>
<td>192.168.65.5</td>
<td>DNS</td>
<td>244</td>
<td>Standard query response 0x4f1d</td>
</tr>
<tr>
<td></td>
<td>0.908597</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam...</td>
<td>QUIC</td>
<td>1242</td>
<td>Initial, DCID=826be2c18846d160</td>
</tr>
<tr>
<td></td>
<td>0.953532</td>
<td>e15712.dscx.akam...</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Initial, DCID=fd8ce54430774240</td>
</tr>
<tr>
<td></td>
<td>0.953548</td>
<td>e15712.dscx.akam...</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=fd8ce54430774240</td>
</tr>
<tr>
<td></td>
<td>0.953549</td>
<td>e15712.dscx.akam...</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=fd8ce54430774240</td>
</tr>
<tr>
<td></td>
<td>0.956409</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam...</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=fd8ce54430774240</td>
</tr>
<tr>
<td></td>
<td>0.995079</td>
<td>e15712.dscx.akam...</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=fd8ce54430774240</td>
</tr>
<tr>
<td></td>
<td>0.995081</td>
<td>e15712.dscx.akam...</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1131</td>
<td>Handshake, DCID=fd8ce54430774240</td>
</tr>
<tr>
<td></td>
<td>1.000640</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam...</td>
<td>QUIC</td>
<td>142</td>
<td>Handshake, DCID=0584275416010</td>
</tr>
<tr>
<td></td>
<td>1.002569</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam...</td>
<td>HTTP3</td>
<td>92</td>
<td>Protected Payload (KP0), DCID=0584275416010</td>
</tr>
</tbody>
</table>

**TLSv1.3 Record Layer: Handshake Protocol: Client Hello**

- **Handshake Protocol: Client Hello**
  - **Handshake Type: Client Hello (1)**
  - **Length:** 272
  - **Version:** TLS 1.2 (0x0303)
  - **Random:** 6020f04ed62b759852c01b9f54c5599f9e39c499a2e053009614c899492af8d2
  - **Session ID Length:** 0
  - **Cipher Suites Length:** 6
  - **Cipher Suites (3 suites):**
    - Compression Methods Length: 1
    - Compression Methods (1 method)
    - Extensions Length: 225
<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0.826686</td>
<td>172.17.0.2</td>
<td>192.168.65.5</td>
<td>DNS</td>
<td>78</td>
<td>Standard query 0x451d A <a href="http://www.something.com">www.something.com</a></td>
</tr>
<tr>
<td>8</td>
<td>0.823840</td>
<td>172.17.0.2</td>
<td>192.168.65.5</td>
<td>DNS</td>
<td>78</td>
<td>Standard query 0x4f1d AAAA <a href="http://www.something.com">www.something.com</a></td>
</tr>
<tr>
<td>9</td>
<td>0.889106</td>
<td>192.168.65.5</td>
<td>172.17.0.2</td>
<td>DNS</td>
<td>220</td>
<td>Standard query response 0x451d A</td>
</tr>
<tr>
<td>10</td>
<td>0.889651</td>
<td>192.168.65.5</td>
<td>172.17.0.2</td>
<td>DNS</td>
<td>244</td>
<td>Standard query response 0x4f1d AAAA</td>
</tr>
<tr>
<td>11</td>
<td>0.908597</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam...</td>
<td>QUIC</td>
<td>1242</td>
<td>Initial, DCID=826be2c18846d16c</td>
</tr>
<tr>
<td>12</td>
<td>0.953532</td>
<td>e15712.dscx.akam...</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Initial, DCID=fd8ce5443077424</td>
</tr>
<tr>
<td>13</td>
<td>0.953548</td>
<td>e15712.dscx.akam...</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=fd8ce5443077424</td>
</tr>
<tr>
<td>14</td>
<td>0.953549</td>
<td>e15712.dscx.akam...</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=fd8ce5443077424</td>
</tr>
<tr>
<td>15</td>
<td>0.956409</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam...</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=05842754160010</td>
</tr>
<tr>
<td>16</td>
<td>0.995079</td>
<td>e15712.dscx.akam...</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=fd8ce5443077424</td>
</tr>
<tr>
<td>17</td>
<td>0.995081</td>
<td>e15712.dscx.akam...</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1131</td>
<td>Handshake, DCID=fd8ce5443077424</td>
</tr>
<tr>
<td>18</td>
<td>1.000640</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam...</td>
<td>QUIC</td>
<td>142</td>
<td>Handshake, DCID=05842754160010</td>
</tr>
<tr>
<td>19</td>
<td>1.002569</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam...</td>
<td>HTTP3</td>
<td>92</td>
<td>Protected Payload (KP0), DCID=05842754160010</td>
</tr>
</tbody>
</table>

Client sent 1x 1242 bytes
Server is limited to 3x 1242 bytes
<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0.826686</td>
<td>172.17.0.2</td>
<td>192.168.65.5</td>
<td>DNS</td>
<td>78</td>
<td>Standard query 0x451d A <a href="http://www.s">www.s</a>..</td>
</tr>
<tr>
<td>8</td>
<td>0.826840</td>
<td>172.17.0.2</td>
<td>192.168.65.5</td>
<td>DNS</td>
<td>78</td>
<td>Standard query 0x4f1d AAAA <a href="http://www.s">www.s</a>..</td>
</tr>
<tr>
<td>9</td>
<td>0.889106</td>
<td>192.168.65.5</td>
<td>172.17.0.2</td>
<td>DNS</td>
<td>220</td>
<td>Standard query response 0x451d A <a href="http://www.s">www.s</a>..</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>0.908597</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam...</td>
<td>QUIC</td>
<td>1242</td>
<td>Initial, DCID=826be2c18846d160</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>0.953532</td>
<td>e15712.dscx.akam...</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=fd8ce5443077424</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>12</td>
<td>0.953548</td>
<td>e15712.dscx.akam...</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=fd8ce5443077424</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>13</td>
<td>0.953549</td>
<td>e15712.dscx.akam...</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=fd8ce5443077424</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>14</td>
<td>0.956409</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam...</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=05842754160010</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>0.995079</td>
<td>e15712.dscx.akam...</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=fd8ce5443077424</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>16</td>
<td>0.995081</td>
<td>e15712.dscx.akam...</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=fd8ce5443077424</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>17</td>
<td>1.000640</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam...</td>
<td>QUIC</td>
<td>142</td>
<td>Handshake, DCID=05842754160010</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>18</td>
<td>1.002569</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam...</td>
<td>HTTP3</td>
<td>92</td>
<td>Protected Payload (KP0), DCID=05842754160010</td>
</tr>
</tbody>
</table>

Source Connection ID: fd8ce5443077424cc0355f54e16838e9fa57030
Length: 22
Packet Number: 0
Payload: 77cb42358bfeb9ce6ea6b55488b9ea90c5b22b5ca4

**ACK**
- Frame Type: ACK (0x0000000000000002)
- Largest Acknowledged: 3
- ACK Delay: 10
- ACK Range Count: 0
- First ACK Range: 1

**QUIC IETF**
[Expert Info (Note/Protocol): (Random) padding data appended to the datagram]
<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0.826686</td>
<td>172.17.0.2</td>
<td>192.168.65.5</td>
<td>DNS</td>
<td>78</td>
<td>Standard query 0x451d A <a href="http://www.something.com">www.something.com</a></td>
</tr>
<tr>
<td>8</td>
<td>0.826840</td>
<td>172.17.0.2</td>
<td>192.168.65.5</td>
<td>DNS</td>
<td>78</td>
<td>Standard query 0x4f1d AAAA <a href="http://www.something.com">www.something.com</a></td>
</tr>
<tr>
<td>9</td>
<td>0.889106</td>
<td>192.168.65.5</td>
<td>172.17.0.2</td>
<td>DNS</td>
<td>220</td>
<td>Standard query response 0x451d</td>
</tr>
<tr>
<td>...</td>
<td>0.889651</td>
<td>192.168.65.5</td>
<td>192.168.65.5</td>
<td>DNS</td>
<td>244</td>
<td>Standard query response 0x4f1d</td>
</tr>
<tr>
<td>...</td>
<td>0.908597</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam.com</td>
<td>QUIC</td>
<td>1242</td>
<td>Initial, DCID=826be2c18846d1600</td>
</tr>
<tr>
<td>...</td>
<td>0.953532</td>
<td>e15712.dscx.akam.com</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Initial, DCID=fd8ce54430774240</td>
</tr>
<tr>
<td>...</td>
<td>0.953548</td>
<td>e15712.dscx.akam.com</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=fd8ce54430774240</td>
</tr>
<tr>
<td>...</td>
<td>0.953549</td>
<td>e15712.dscx.akam.com</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=fd8ce54430774240</td>
</tr>
<tr>
<td>...</td>
<td>0.956409</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam.com</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=58427541600100</td>
</tr>
<tr>
<td>...</td>
<td>0.959709</td>
<td>e15712.dscx.akam.com</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1242</td>
<td>Handshake, DCID=58427541600100</td>
</tr>
<tr>
<td>...</td>
<td>0.995081</td>
<td>e15712.dscx.akam.com</td>
<td>172.17.0.2</td>
<td>QUIC</td>
<td>1131</td>
<td>Handshake, DCID=fd8ce54430774240</td>
</tr>
<tr>
<td>...</td>
<td>1.000640</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam.com</td>
<td>QUIC</td>
<td>142</td>
<td>Handshake, DCID=58427541600100</td>
</tr>
<tr>
<td>...</td>
<td>1.002569</td>
<td>172.17.0.2</td>
<td>e15712.dscx.akam.com</td>
<td>HTTP3</td>
<td>92</td>
<td>Protected Payload (KP0), DCID=fd8ce54430774240</td>
</tr>
</tbody>
</table>

**TLSv1.3 Record Layer: Handshake Protocol: Multiple Handshake Messages**

- Handshake Protocol: Certificate (last fragment)
- [4 Reassembled Handshake Fragments (3978 bytes): #13(966), #14(1141), #16(1141), #17(730)]
- Handshake Protocol: Certificate
  - Handshake Type: Certificate (11)
    - Length: 3974
    - Certificate Request Context Length: 0
    - Certificates Length: 3970
    - Certificates (3970 bytes)
  - Handshake Protocol: Certificate Verify
    - Handshake Type: Certificate Verify (15)
    - Length: 260

**3974 > 3726 (=3 x 1242)**
Lots of magic numbers here

Initial packet sizes

Many deployments ignore 3X and go to 4, 5 or 6X just to get handshake done in 1 RTT
I THOUGHT THINGS WERE BAD UNDER MUFASA.
Deploy your own HTTP/3

Reminder:
- Not all code is usually open source
- Open source configuration != actual deployment

(Almost) full support:
- Caddy
- H2O
- Hypercorn

Most mature, most complicated:
- MVfst
- tis
- QUICHE
- MsQuic

In beta/testing:
- NGINX
- HAProxy

Months/years out:
- Apache
- Node

https://github.com/quicwg/base-drafts/wiki/Implementations
Deploy your own HTTP/3

Reminder:
Not all code is usually open source
Open source configuration != actual deployment

OpenSSL refuses to provide sensible APIs for QUIC integration…

Use QuicTLS, BoringSSL, LibreSSL, or WolfSSL

But wait, there’s more...

Content servers are 1 thing, what about:

- Load balancers?
- Terminating/reverse proxies?
- Firewalls?
- Observability?
- TLS key/certificate management?
- DNS????
- ...
TCP “Connection”

All client TCP connections need to be re-established after 4-tuple changes

- Active migration (wifi to 4G)
- NAT rebinding after timeout

⇒ Connection handshake delay
⇒ Loss of HTTP/TLS context

Gets load balanced to different server/cluster?
**QUIC’s “Connection ID”**

1. **TCP handshake** from WiFi IP starts connection

2. **TCP data packet** from unknown IP... no idea what to do with this

Let’s use this **QUIC Connection ID (CID)** for now:

The network is different, but the CID is the same, so it’s the same connection
CID included in each QUIC packet!

<table>
<thead>
<tr>
<th>Flags</th>
<th>Connection ID</th>
<th>Packet number</th>
<th>ACK frame</th>
<th>Flow control frame</th>
<th>Stream frame header</th>
<th>Stream frame payload</th>
<th>HTTP/3 data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Packet Header | Payload
Stateless Load Balancing changes

4-tuple hash:
- Source IP
- Source Port
- Destination IP
- Destination Port
Stateless Load Balancing changes

4-tuple hash:
- Source IP
- Source Port
- Destination IP
- Destination Port

Connection ID
**Linkability** problem:

CID can be used to track users across different networks!
Solution: Change CIDs when network changes!

I switched networks, so let's use the next CID:

Let's use this Connection ID (CID) for now:

In the future, you can also use these CIDs:
(this is sent encrypted, so observers don't know about the new CIDs)

I know is actually
(but observers don't!)
Problem: Breaks stateless load balancing...

4-tuple hash:
- Source IP
- Source Port
- Destination IP
- Destination Port

Connection ID
Solution: encode LB information in CID

<table>
<thead>
<tr>
<th>Random Connection ID</th>
<th>4-20 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Load-Balanced Connection ID</th>
<th>4-20 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>214</td>
<td>?</td>
</tr>
</tbody>
</table>

Backend Server Nr
Solution: encode LB information in CID

- Random Connection ID
  - 4-20 bytes
  - ?

- Load-Balanced Connection ID
  - 214
  - ?
  - 4-20 bytes

- Zero-downtime CID
  - 214
  - ?
  - 2
  - 4-20 bytes

Backend Server Nr

Process Nr

EPIQ 18 Facebook Keynote: https://www.youtube.com/watch?v=8lYHNzoPS2o
Problem: Who chooses the CID?

**Client** chooses CID

But client doesn’t have Load Balancing info…
Solution: Separate CIDs for Client and Server

Different directions use different CIDs

- My CID is: 1
- Future CIDs are: 2

I switched networks, so use the next server CID

- Client has migrated, I need to switch to their next CID

- My CID is: [ ]
- Future CIDs are: [ ]
Solution: Separate CIDs for Client and Server

Different directions use different CIDs

Handshake Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2) = 2,
  Reserved Bits (2),
  Packet Number Length (2),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..160),
  Source Connection ID Length (8),
  Source Connection ID (0..160),
  Length (i),
  Packet Number (8..32),
  Packet Payload (8..),
}
Problem: Leaking internal deployment info in CID

- Figure out #servers / LB logic
- DDoS 1 specific server

Server Nr

Facebook network and load balancing internals from CID observations: https://arxiv.org/abs/2209.00965
Solution: Encrypt CID

Encrypt CID at the server, decrypt at Load Balancer
- Can use **single** shared private key across deployment/cluster

CID encryption
- Single key
- Per cluster

Tip of the Iceberg

- server migration / preferred address
- retiring connection IDs
- CID and key updates
- Transport parameter negotiation
- Long vs Short packet headers
- zero-length CIDs
- MULTIPATH?!?!
- Packet coalescing and chaos protection/GREASE
- RETRY and STATELESS RESET
- Happy eyeballs
- 0-RTT ticket re-use
- Replay attacks
- Session resumption, address validation and STEKs
- Encrypted Client Hello (ECH)
- DNS HTTPS records

Key challenges:
- Load balancing
- Firewalling
- 0-RTT + session resumption
- TLS key/certificate management
Akamai also does Cloud now

Akamai + Linode

The world's most distributed compute platform, from cloud to edge
End-to-end qlog extraction

HTTP/3 test page - all options
Conclusion
Conclusion

HTTP/3

Complex, yes, but mostly just different

Ease yourself into it:

https://www.youtube.com/playlist?list=PL3tsOU35YefabIEzJa_cq6vjkojNr0UZh
https://www.smashingmagazine.com/2021/08/http3-core-concepts-part1/
https://calendar.perfplanet.com/2022/http-3-prioritization-demystified/
Would you like to know more?

Robin Marx

@programmingart
linkedin.com/in/rmarx
rmarx@akamai.com