Applied Networking Research Workshop 2020

Debugging QUIC and HTTP/3 with [qlog] and <qvis>

Robin Marx, Maxime Piraux, Wim Lamotte and Peter Quax

Hasselt University and UCLouvain

Robin Marx

@programmingart

Last year PhD Student

Hasselt University, Belgium

Big J.R.R. Tolkien Fan

QUIC and HTTP/3 are quite extensive

6 “Core” specifications:

- QUIC invariants : 10 pages
- Core Transport : 187 pages
- TLS mapping : 59 pages
- Recovery (loss and congestion) : 46 pages
- HTTP/3 : 72 pages
- QPACK header compression : 44 pages

418 pages total

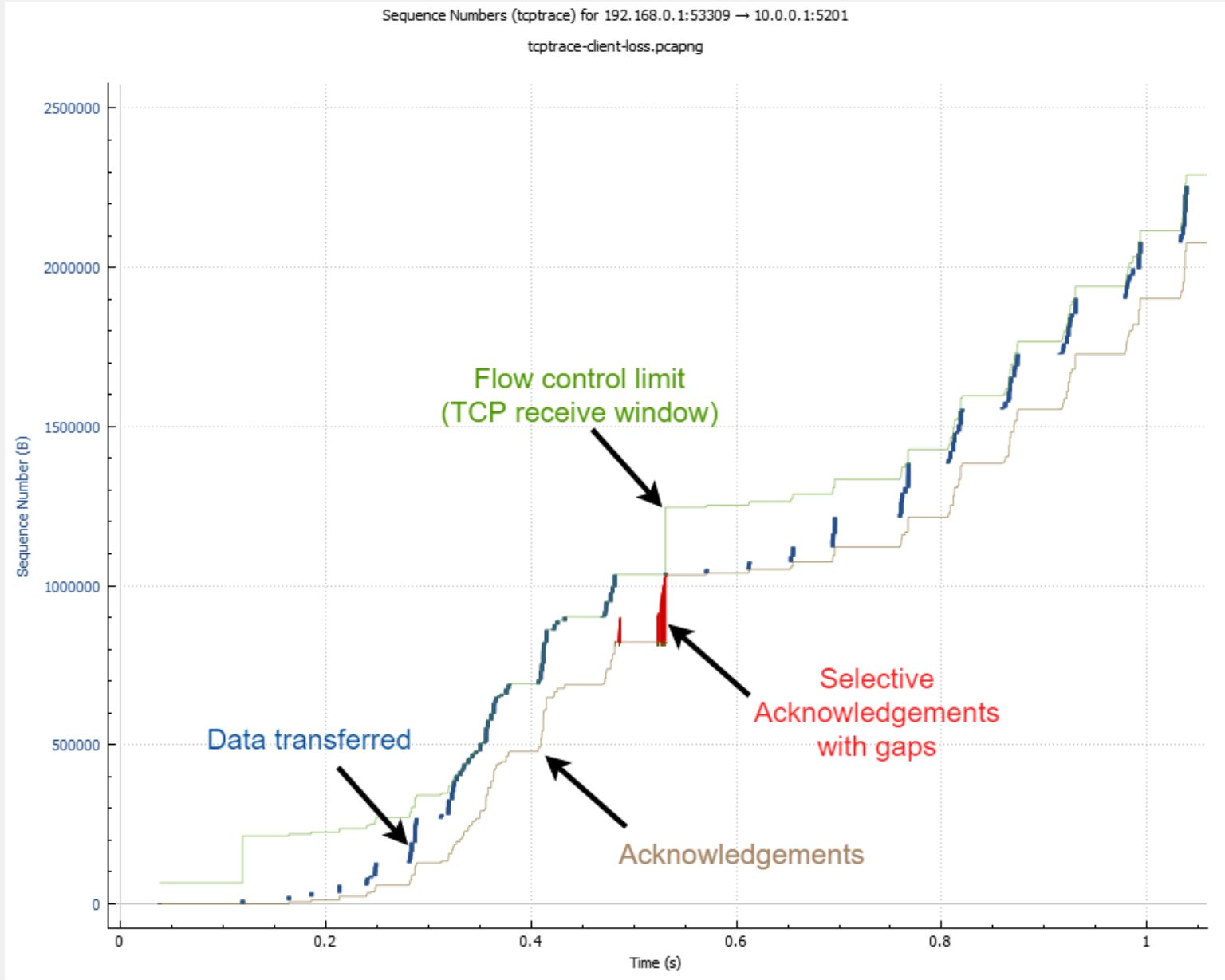
=

108 more than The Hobbit

Many other drafts/extensions:

- Applicability, manageability, DATAGRAM, load balancing, H3 priorities, …
- Multipath, ACK frequency, loss bits, …

https://github.com/quicwg/base-drafts

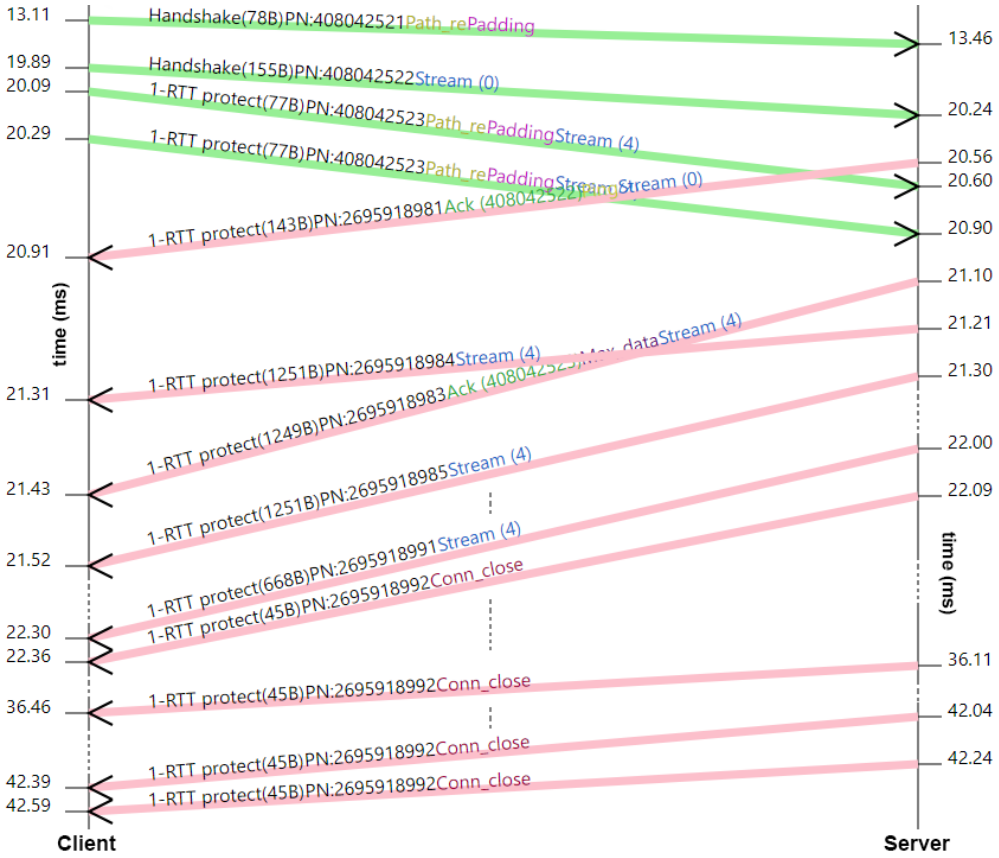


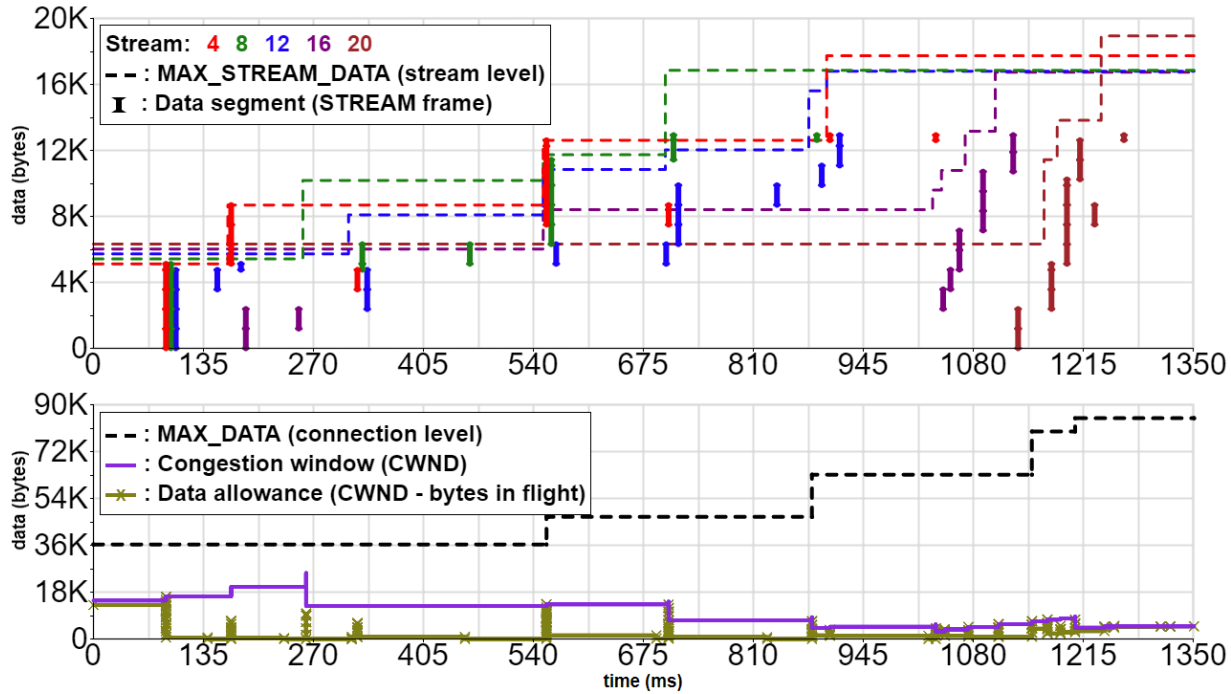
Click to select a portion of the graph. → 3235 pkts, 4718kB ← 1765 pkts, 0 bytes

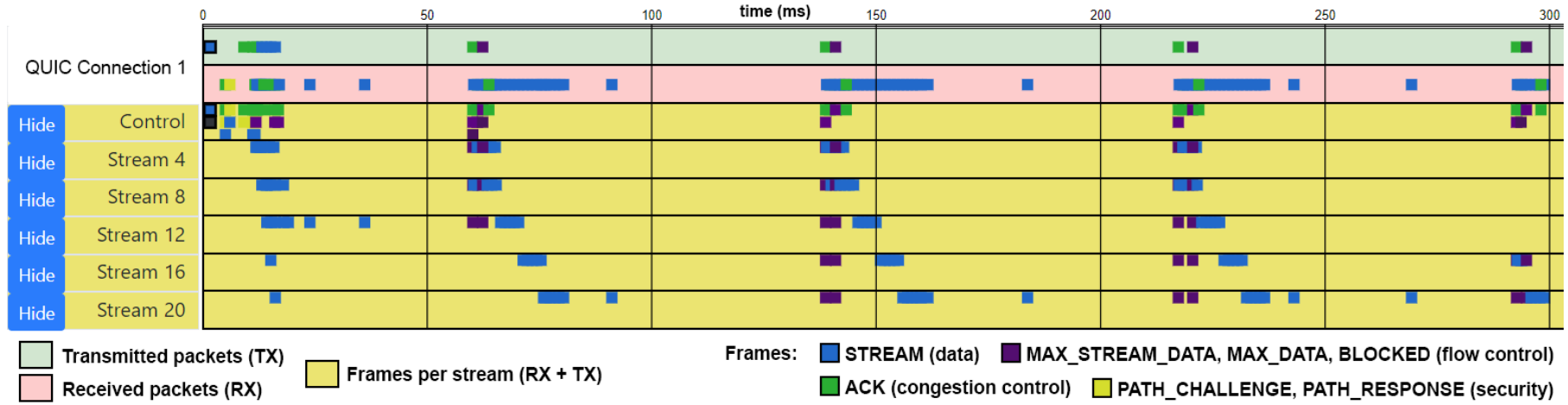
Type Time / Sequence (tcptrace) ☐ Select SACKs Stream 0 Switch Direction

Mouse ○ drags ◉ zooms Reset

Save As... Close Help







https://quic.edm.uhasselt.be/

Common tool input format?

packet captures

- Do not contain internal state

https://youtu.be/nErrFHPatq0?t=4339
https://youtu.be/LiNLz1QuT0s?t=3233

Common tool input format?

packet captures

- Do not contain internal state

ad-hoc endpoint logs

- Are different across implementations
- Are unstructured

https://youtu.be/nErrFHPatq0?t=4339
https://youtu.be/LiNLz1QuT0s?t=3233

```
33009109 pkt tx pkt 0 dcid=0x108c2996a1d18a8bb1f7611937eb5f30 scid=0xb5080d83e09acbce1e6e4b9076330091
33009109 frm tx 0 Short(0x00) STREAM(0x13) id=0x0 fin=1 offset=0 len=
33009109 rcv loss_detection_timer=15415150049329323 52 last_hs_tx_pkt_
33009109 con recv packet len=63
33009109 pkt rx pkt 2 dcid=0xb5080d83e09acbce1e6e4b907633009109 scid=
33009109 frm rx 2 Handshake(0x7d) ACK(0x1a) largest_ack=0 ack_delay=6
33009109 frm rx 2 Handshake(0x7d) ACK(0x1a) block=[0..0] block_count=
33009109 rcv latest_rtt=47 min_rtt=32 smoothed_rtt=34.076 rttvar=15.9
33009109 rcv packet 0 acked, slow start cwnd=13370
33009109 pkt read packet 63 left 0
33009109 rcv loss detection timer fired
33009109 rcv handshake_count=0 tlp_count=1 rto_count=0
33009109 con transmit probe pkt left=1
33009109 pkt tx pkt 1 dcid=0x108c2996a1d18a8bb1f7611937eb5f30 scid=0x
33009109 frm tx 1 Short(0x00) PING(0x07)
33009109 con probe pkt size=35
33009109 con recv packet len=169
33009109 pkt rx pkt 0 dcid=0xb5080d83e09acbce1e6e4b907633009109 scid=0x type=Short(0x00) len=0
33009109 frm rx 0 Short(0x00) CRYPTO(0x18) offset=0 len=130

3d 0e 65 08 00 00 00  |...=... .=.e....|
9b 6d 9d d0 6b 98 4f  |..... .A.m..k.O|
11 ea fd e4 cd 1b d5  |..WWz.t>........|
00 2a 00 04 ff ff ff  |[.u.Q.....*.....|
06 2e 42 d3 08 00 00  |....=... ..B....|
05 93 85 08 6b e5 0f  |....... %....k..|
63 9d 27 1f 16 67 68  |Cc..[....c.'..gh|
08 00 2a 00 04 ff ff  |x.B?..w....*....|
                      |..|
```

```
con recv
pkt rx pk
frm rx 2
frm rx 2
rcv lates
rcv packe
```

https://youtu.be/nErrFHPatq0?t=4339
https://youtu.be/LiNLz1QuT0s?t=3233

Common tool input format?

packet captures

- Do not contain internal state

[qlog]

structured endpoint logs

You can log what you want, just not how you want it

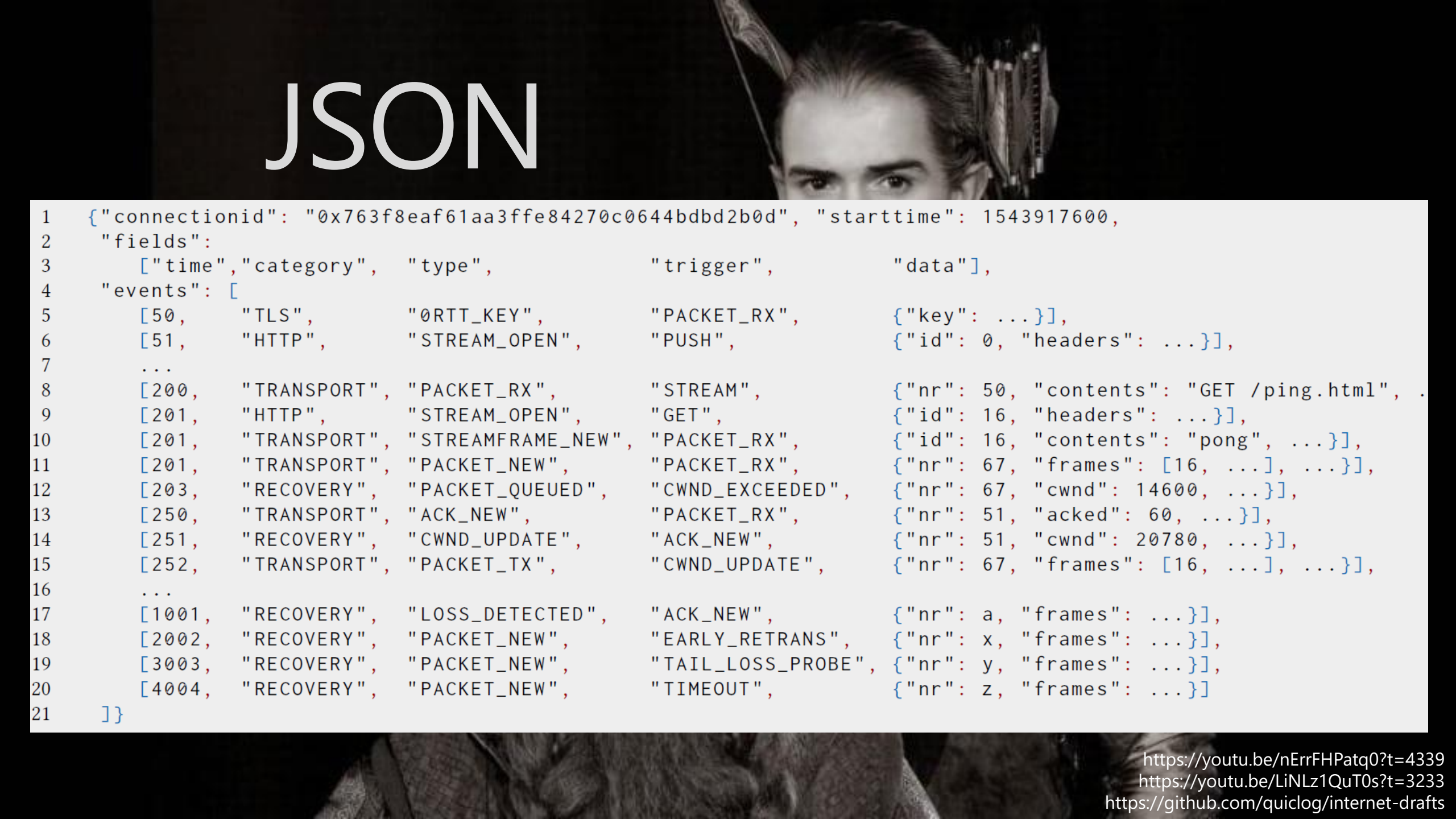
ad-hoc endpoint logs

- Are different across implementations
- Are unstructured

https://youtu.be/nErrFHPatq0?t=4339
https://youtu.be/LiNLz1QuT0s?t=3233
https://github.com/quiclog/internet-drafts

https://youtu.be/nErrFHPatq0?t=4339
https://youtu.be/LiNLz1QuT0s?t=3233
https://github.com/quiclog/internet-drafts

JSON



```
{"connectionid": "0x763f8eaf61aa3ffe84270c0644bdbd2b0d", "starttime": 1543917600,
 "fields":
    ["time","category",  "type",           "trigger",         "data"],
 "events": [
    [50,    "TLS",       "0RTT_KEY",       "PACKET_RX",       {"key": ...}],
    [51,    "HTTP",      "STREAM_OPEN",    "PUSH",            {"id": 0, "headers": ...}],
    ...
    [200,   "TRANSPORT", "PACKET_RX",      "STREAM",          {"nr": 50, "contents": "GET /ping.html", .
    [201,   "HTTP",      "STREAM_OPEN",    "GET",             {"id": 16, "headers": ...}],
    [201,   "TRANSPORT", "STREAMFRAME_NEW","PACKET_RX",       {"id": 16, "contents": "pong", ...}],
    [201,   "TRANSPORT", "PACKET_NEW",     "PACKET_RX",       {"nr": 67, "frames": [16, ...], ...}],
    [203,   "RECOVERY",  "PACKET_QUEUED",  "CWND_EXCEEDED",   {"nr": 67, "cwnd": 14600, ...}],
    [250,   "TRANSPORT", "ACK_NEW",        "PACKET_RX",       {"nr": 51, "acked": 60, ...}],
    [251,   "RECOVERY",  "CWND_UPDATE",    "ACK_NEW",         {"nr": 51, "cwnd": 20780, ...}],
    [252,   "TRANSPORT", "PACKET_TX",      "CWND_UPDATE",     {"nr": 67, "frames": [16, ...], ...}],
    ...
    [1001,  "RECOVERY",  "LOSS_DETECTED",  "ACK_NEW",         {"nr": a, "frames": ...}],
    [2002,  "RECOVERY",  "PACKET_NEW",     "EARLY_RETRANS",   {"nr": x, "frames": ...}],
    [3003,  "RECOVERY",  "PACKET_NEW",     "TAIL_LOSS_PROBE", {"nr": y, "frames": ...}],
    [4004,  "RECOVERY",  "PACKET_NEW",     "TIMEOUT",         {"nr": z, "frames": ...}]
 ]}
```

https://youtu.be/nErrFHPatq0?t=4339
https://youtu.be/LiNLz1QuT0s?t=3233
https://github.com/quiclog/internet-drafts

2 years later...

12/18 QUIC implementations support qlog

- Facebook, Cloudflare, Mozilla, NodeJS (ngtcp2), ...
- 2 more with plans to add qlog in the future
- 2 others use a (different) structured format

Facebook has deployed it in production

- Store over 30 billion qlog events daily

https://crates.io/crates/qlog
https://github.com/quicwg/base-drafts/wiki/Implementations
https://blog.cloudflare.com/cubic-and-hystart-support-in-quiche

But... why?

Expert survey

- Recruited via QUIC mailing list (and gentle prodding)
- 28 participants
- at least 1 participant from all but 2 of the 18 implementations
- All QUIC developers (22) and researchers (6)

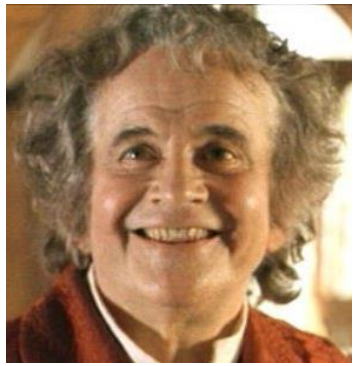
+ in-depth interview with Facebook

Debugging and analysis for QUIC in general

- Which types of logging and why?
- Which tools and why?
- Which (future) use cases?

They like qlog because:

1. They want to use 3rd party tools (like <qvis>)
2. It makes it easy to create custom tools



3. qlog is flexible

They *don't* like qlog because:

4. JSON is verbose and slow

The toolsuite can be found online at:

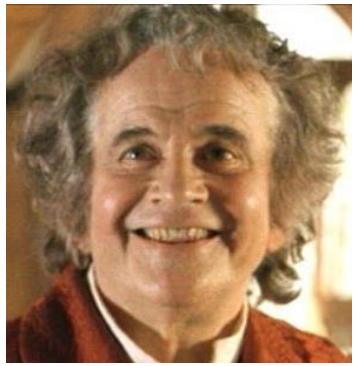
- https://qvis.edm.uhasselt.be

Example traces can be found at:

- https://qlog.edm.uhasselt.be/anrw
- https://qlog.edm.uhasselt.be/sigcomm

They like qlog because:

1. They want to use 3rd party tools (like <qvis>)
2. It makes it easy to create custom tools



3. qlog is flexible and schemaless

They *don't* like qlog because:

4. JSON is verbose and slow

qlog is flexible : 1/3

qlog defines events and fields

- But most are optional
- And other events are explicitly allowed

```
packet_dropped
{
    packet_type?:PacketType,
    packet_size?:uint64,
    raw?:bytes,

    trigger?: string
}
```

Used extensively in practice

- Implementation-specific state (e.g., BBR parameters)
- New QUIC extensions (Mulitpath, DATAGRAM, Ack Frequency, loss bits, …)
- 1 implementation completely switched from ad-hoc to qlog

→ **No need to wait for a qlog or qvis update to visualize new things**

qlog is flexible : 2/3

Easy to use and parse

- Facebook streams individual events to a database
  - Later uses queries to find interesting traces (e.g., % of packet_lost events)
- Log-based unit testing
  - “Was the spin-bit spinning?” → are there qlog spin_bit_updated events?

```
std::vector<int> indices = getQLogEventIndices(QLogEventType::AppLimitedUpdate, qLogger);
EXPECT_EQ(indices.size(), 2);

auto event = qLogger->logs[indices[0]];
EXPECT_EQ(event->limited, true);

auto event2 = qLogger->logs[indices[1]];
EXPECT_EQ(event2->limited, false);
```

https://github.com/aiortc/aioquic
https://github.com/facebookincubator/mvfst

qlog is flexible : 3/3

Easy to transform from/to other formats

- pcap2qlog, netlog2qlog, quictrace2qlog, etc.

Easy to extend to other protocols

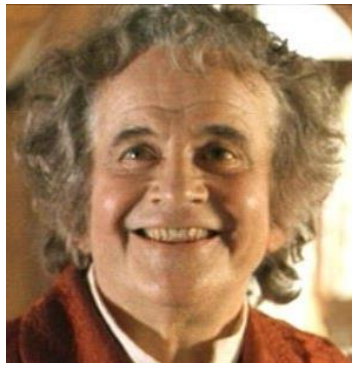
- DNS over QUIC, DNS over HTTP/3
- **TCP + TLS + HTTP/2**

→ combine pcaps with eBPF kernel probes and H2 browser logs

https://github.com/quiclog/pcap2qlog
https://github.com/quiclog/quictrace2qlog
https://github.com/moonfalir/quicSim-docker/tree/master/tcpebpf
https://github.com/triplewy/qvis/tree/master/visualizations/src/components/filemanager/netlogconverter
https://github.com/triplewy/qvis/blob/master/visualizations/src/components/filemanager/pcapconverter/tcptoqlog.ts

They like qlog because:

1. They want to use 3rd party tools (like <qvis>)
2. It makes it easy to create custom tools

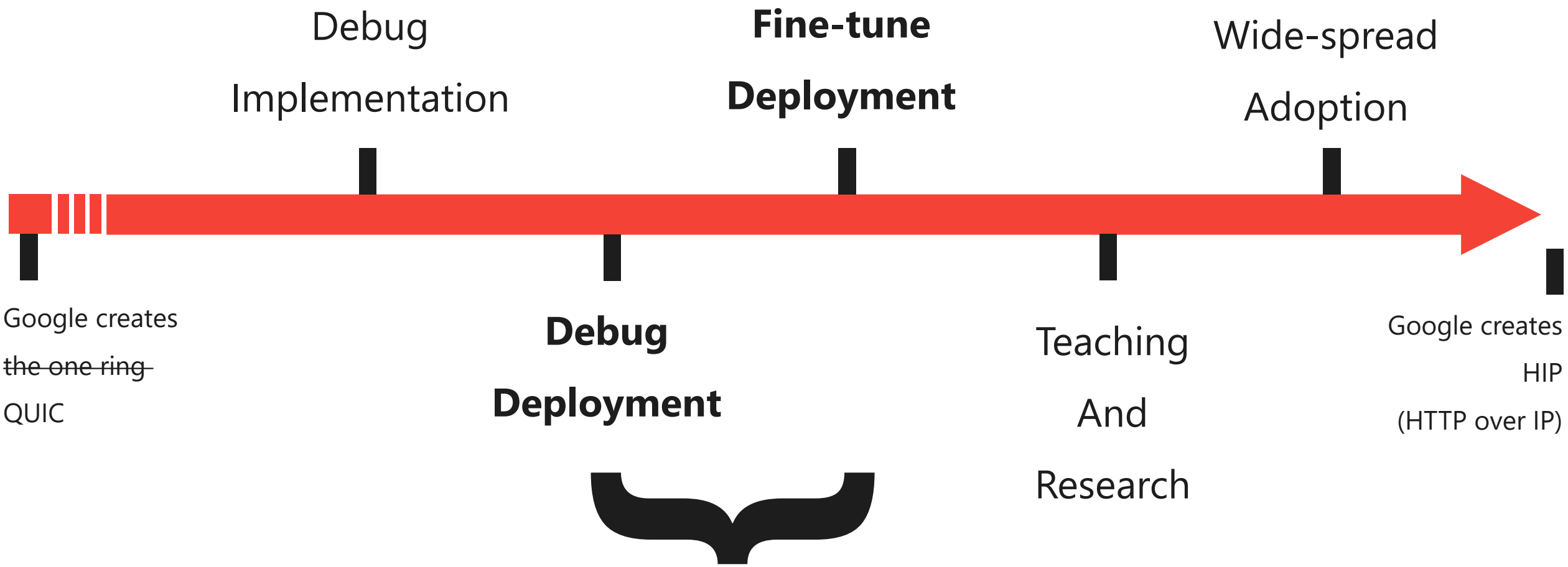


3. qlog is flexible and schemaless

They *don't* like qlog because:

4. JSON is verbose and slow

The IETF QUIC Timeline



Logging needs to run at scale

Connection tracing at scale?

packet captures

- Are large because QUIC is encrypted
- Privacy and security concerns

Connection tracing at scale?

packet captures

- Are large because QUIC is encrypted
- Privacy and security concerns

spin and loss bits

- The nays have it?
- Would still be fairly limited

Connection tracing at scale?

packet captures

- Are large because QUIC is encrypted
- Privacy and security concerns

[qlog] structured endpoint logs

Log only what you need

spin and loss bits

- The nays have it?
- Would still be fairly limited

JSON does not scale

Binary format would be better

- Counter-argument: much less flexible!
- (Semi) Counter-argument: Facebook uses qlog in production
- Counter-argument: JSON compresses well

https://github.com/quiclog/internet-drafts/issues/30

JSON does not scale

Binary format would be better

- Counter-argument: much less flexible!
- (Semi) Counter-argument: Facebook uses qlog in production
- Counter-argument: JSON compresses well

500 MB

file download

format	raw
pcap	561.57
JSON	276.02
CBOR	215.53
protobuf	66.15

resulting log file sizes in MB

https://github.com/quiclog/internet-drafts/issues/30

JSON does not scale

Binary format would be better

- Counter-argument: much less flexible!
- (Semi) Counter-argument: Facebook uses qlog in production
- Counter-argument: JSON compresses well

500 MB file download

format	raw	gzip6	brotli4
pcap	561.57	529.01	528.85
JSON	276.02	19.15	19.40
CBOR	215.53	17.78	18.90
protobuf	66.15	14.56	10.71

resulting log file sizes in MB

https://github.com/quiclog/internet-drafts/issues/30

Solution: Pick your poison

qlog is a loose schema, implementers choose the format

- JSON is the default
- But updated definitions to make it easier to define a binary setup
  - Binary to JSON (e.g., for tooling) should be easy

Will be in qlog draft-02 (this week or next)

- Will need additional evaluation over time

https://github.com/quiclog/internet-drafts/issues/30

In conclusion

Tooling has really helped in debugging QUIC

(we even got people to output raw JSON…)

Structured logging can be the way to go for wider deployment

(but more work needed to determine scaling requirements)

Future work + why IETF?

Can qlog solve the spinbit use case for network operators?

- Endpoint owners sharing logs? How to scale and automate that?
- *Similar concepts discussed in IPPM right after this!*

How do we define privacy and security guidelines?

- Which fields should we strip? Anonymize?

Should this be bigger than just QUIC and HTTP/3?

robin.marx@uhasselt.be

https://tools.ietf.org/html/draft-cfb-ippm-spinbit-measurements-02
https://huitema.wordpress.com/2020/07/21/scrubbing-quic-logs-for-privacy/

Image sources

- https://img.icons8.com/cotton/2x/survey.png
- https://www.vecteezy.com/vector-art/633173-clock-icon-symbol-sign
- https://cdn4.vectorstock.com/i/1000x1000/20/13/thumb-up-and-down-icon-vector-20072013.jpg
-