Understanding QUIC and HTTP/3

with qlog and quicvis

Robin Marx - @programmingart
(Hasselt University – Belgium)

UHASSELT EDM

Performance Calendar

The speed geek's favorite time of the year

4th Dec 2018

QUIC and HTTP/3 : Too big to fail?!

by Robin Marx

QUIC and HTTP/3 : Too big to fail?!

The new QUIC and HTTP/3 protocols are coming and they are the bee's knees! Combining lessons and best practices from over 30 years of networking, the new protocol stack offers major improvements to performance, privacy, security and flexibility.

Much has been said about the potential benefits of QUIC, most of it based on Google's experience with an early version of the protocol. However, its potential shortcomings are rarely talked about and little is yet known about the properties of the upcoming, standardized versions (as they are still under active development). This post takes a (nuanced) "devil's advocate" viewpoint and looks at how QUIC and HTTP/3 might still fail in practice, despite the large amount of current enthusiasm. In all fairness, I will also mention counter arguments to each point and let the reader make up their own mind, hopefully after plenty of additional discussion.

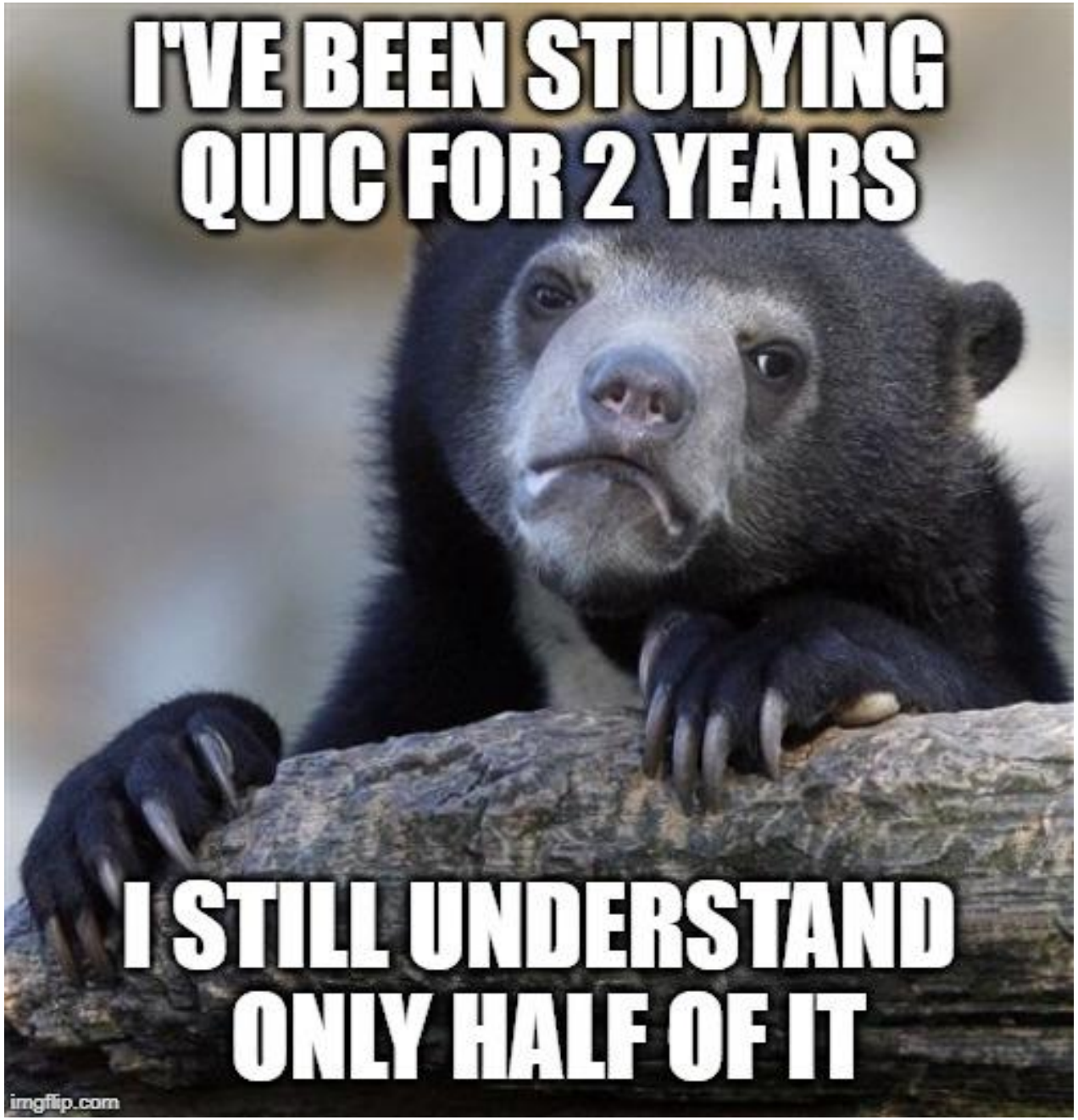
> Note: if you're not really sure what QUIC and HTTP/3 are in the first place, it's best to get up to speed a bit before reading this post, which assumes some familiarity with the topic. Some resources that might help you with that:
>
> - Mattias Geniar's blog post
> - Cloudflare's write-up
> - Robert Graham's comments
> - Daniel Stenberg (@bagder)'s HTTP/3 explained
> - Mailing list explanation and blog post by Patrick McManus
> - And my own talk from DeltaVConf this year

ABOUT THE AUTHOR

Robin Marx is a Web Performance PhD candidate at Hasselt University, Belgium. He is mainly looking into HTTP/2 and QUIC performance, and maintains the TypeScript QUIC implementation Quicker. In a previous life he was a multiplayer game programmer and co-founder of LuGus Studios. YouTube videos of Robin are either humoristic technical talks or him hitting other people with longswords.

QUIC and HTTP/3 are quite complex…

- V1
  - Congestion control + loss detection
    - Fairness issues
  - Flow control
  - Encryption and integrity protection
  - Connection migration
  - 0-RTT support
  - Independent streams
    - Partly re-introduces HOL-blocking in HTTP/3
  - Low overhead, binary encoding
  - DoS prevention
  - Stateless Retry
  - Retransmission logic
    - Brings up interesting prioritization questions
  - …
- Postponed to V2
  - Multipath
  - Forward error correction
  - Unreliable data transfer
  - Support for other crypto besides TLS 1.3
  - Most (all?) non-HTTP/3 applications
    - IoT, realtime media, …
  - How to expose all of this to the developer?
    - E.g., TAPS, WebTransport, …


I'VE BEEN STUDYING
QUIC FOR 2 YEARS
I STILL UNDERSTAND
ONLY HALF OF IT
imgflip.com

If QUIC is to become the dominant transport protocol…

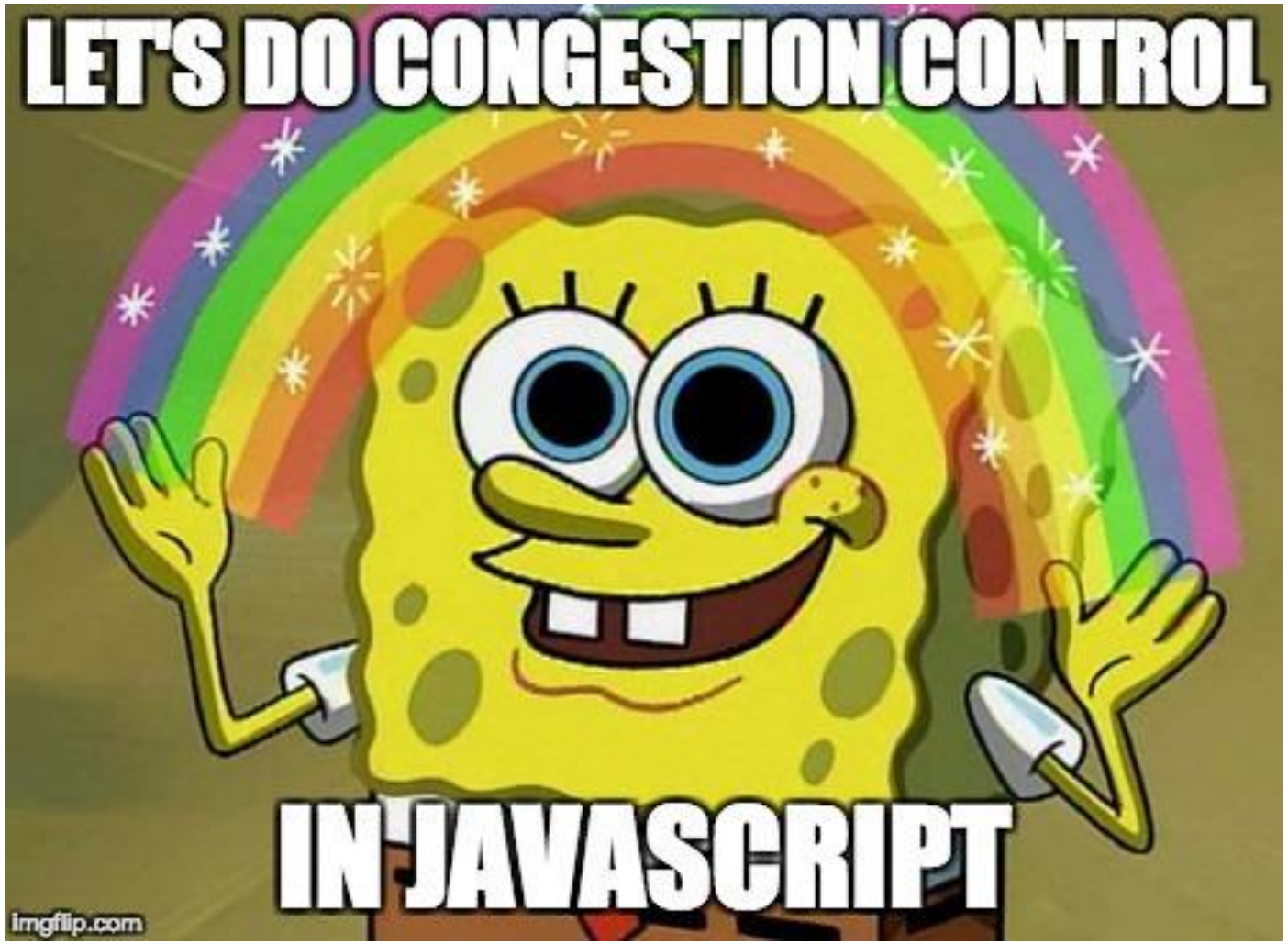
- Students should learn QUIC basics no later than 3rd bachelor
- Researchers should dig deep into all aspects, in specific settings
- Application developers should be able to debug behaviour in complex setups

Right now, in-depth QUIC knowledge resides with 50 – 100 people worldwide

- We need examples
- We need documentation
- We need tools

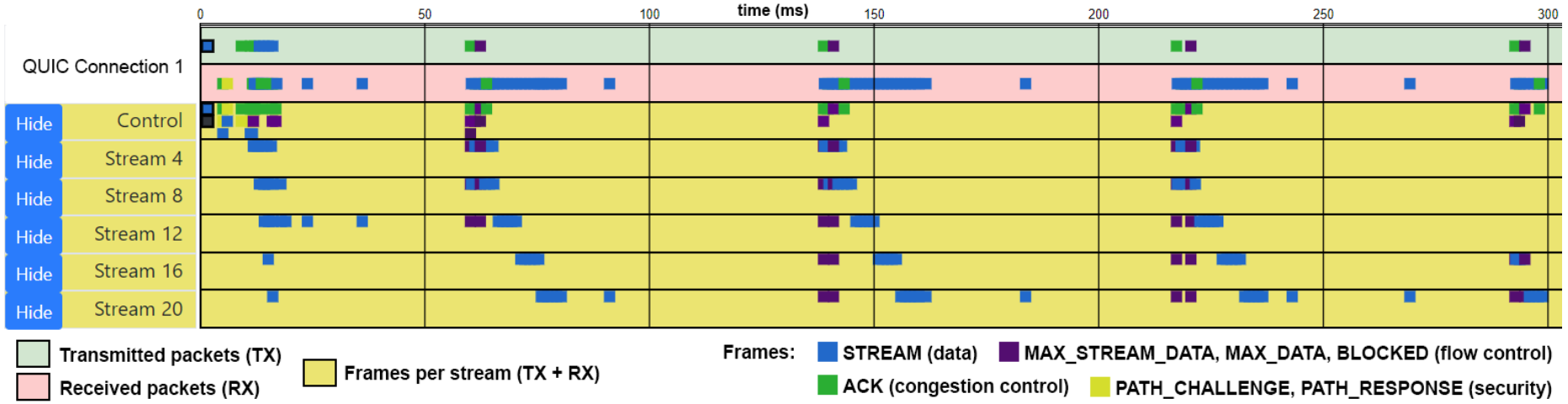
https://github.com/rmarx/quicker

Quicker : TypeScript implementation

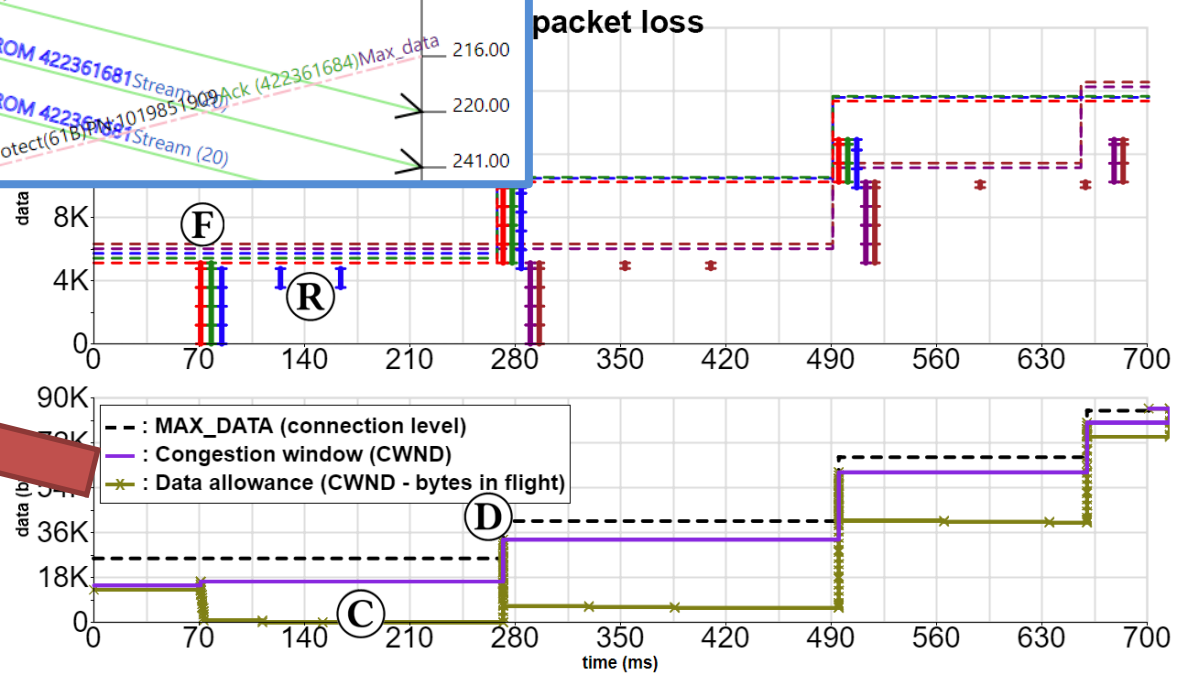


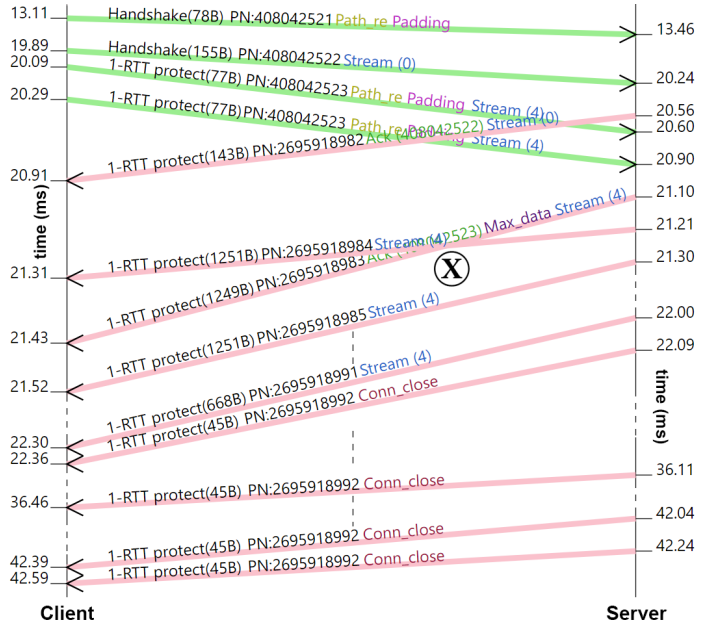
https://github.com/rmarx/quicker/tree/congestionControl
https://github.com/rmarx/quicker/tree/http3-19
https://github.com/DaanDeMeyer/h3c
https://github.com/DaanDeMeyer/chromium

QUIC tools

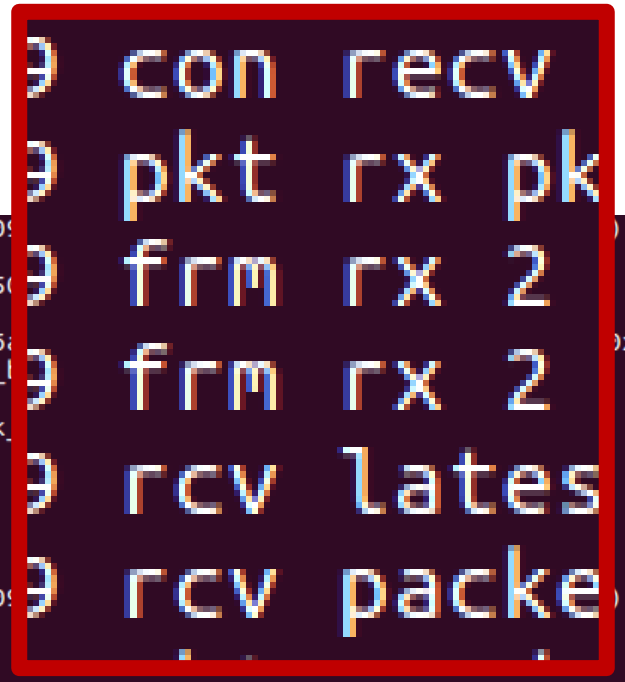


not in .pcap



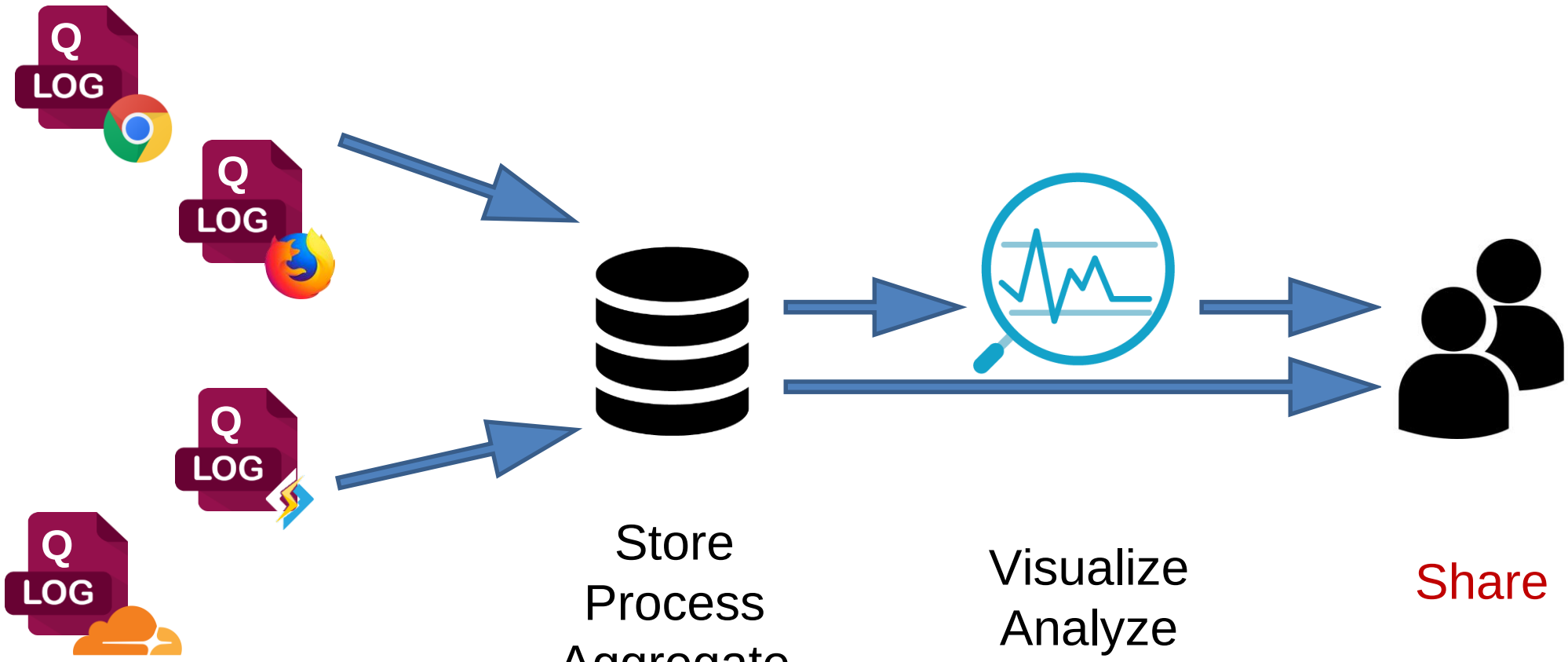


QUIC logging: The Wild Wild West




```
I00000036 0xb5080d83e09acbce1e6e4b907633009109 pkt tx pkt 0 dcid=0x108c2996a1d18a8bb1f7611937eb5f30 scid=0xb5080d83e09 ) len=0
I00000036 0xb5080d83e09acbce1e6e4b907633009109 frm tx 0 Short(0x00) STREAM(0x13) id=0x0 fin=1 offset=0 len=16 uni=0
I00000036 0xb5080d83e09acbce1e6e4b907633009109 rcv loss_detection_timer=1541515004932932352 last_hs_tx_pkt_ts=1541515(
I00000090 0xb5080d83e09acbce1e6e4b907633009109 con recv packet len=63
I00000090 0xb5080d83e09acbce1e6e4b907633009109 pkt rx pkt 2 dcid=0xb5080d83e09acbce1e6e4b907633009109 scid=0x108c2996a 0x7d) len=23
I00000090 0xb5080d83e09acbce1e6e4b907633009109 frm rx 2 Handshake(0x7d) ACK(0x1a) largest_ack=0 ack_delay=6(863) ack_
I00000090 0xb5080d83e09acbce1e6e4b907633009109 frm rx 2 Handshake(0x7d) ACK(0x1a) block=[0..0] block_count=0
I00000090 0xb5080d83e09acbce1e6e4b907633009109 rcv latest_rtt=47 min_rtt=32 smoothed_rtt=34.076 rttvar=15.920 max_ack_
I00000090 0xb5080d83e09acbce1e6e4b907633009109 rcv packet 0 acked, slow start cwnd=13370
I00000090 0xb5080d83e09acbce1e6e4b907633009109 pkt read packet 63 left 0
I00000092 0xb5080d83e09acbce1e6e4b907633009109 rcv loss detection timer fired
I00000092 0xb5080d83e09acbce1e6e4b907633009109 rcv handshake_count=0 tlp_count=1 rto_count=0
I00000092 0xb5080d83e09acbce1e6e4b907633009109 con transmit probe pkt left=1
I00000092 0xb5080d83e09acbce1e6e4b907633009109 pkt tx pkt 1 dcid=0x108c2996a1d18a8bb1f7611937eb5f30 scid=0xb5080d83e09 ) len=0
I00000092 0xb5080d83e09acbce1e6e4b907633009109 frm tx 1 Short(0x00) PING(0x07)
I00000092 0xb5080d83e09acbce1e6e4b907633009109 con probe pkt size=35
I00000103 0xb5080d83e09acbce1e6e4b907633009109 con recv packet len=169
I00000103 0xb5080d83e09acbce1e6e4b907633009109 pkt rx pkt 0 dcid=0xb5080d83e09acbce1e6e4b907633009109 scid=0x type=Short(0x00) len=0
I00000103 0xb5080d83e09acbce1e6e4b907633009109 frm rx 0 Short(0x00) CRYPTO(0x18) offset=0 len=130
Ordered CRYPTO data
00000000  04 00 00 3d 00 00 1c 20  db 3d 0e 65 08 00 00 00  |...=... .=.e....|
00000010  00 00 00 00 00 00 20 da  41 9b 6d 9d d0 6b 98 4f  |...... .A.m..k.O|
00000020  bc bc 57 57 7a eb 74 3e  a2 11 ea fd e4 cd 1b d5  |..WWz.t>........|
00000030  5b 1b 75 f3 51 1a 09 00  08 00 2a 00 04 ff ff ff  |[.u.Q.....*.....|
00000040  ff 04 00 00 3d 00 00 1c  20 06 2e 42 d3 08 00 00  |....=... ..B....|
00000050  00 00 00 00 00 01 00 20  25 05 93 85 08 6b e5 0f  |....... %....k..|
00000060  43 63 a9 b7 5b c4 e9 d4  9b 63 9d 27 1f 16 67 68  |Cc..[....c.'..gh|
00000070  78 a0 42 3f cb b2 77 f8  00 08 00 2a 00 04 ff ff  |x.B?..w....*....|
00000080  ff ff                                            |..|
00000082
```

Standardized QUIC endpoint logging format



Our proposal: qlog

```
{"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": ...}]
 ]}
```

JSON:

- Easy to use in web-based tools (and most programming languages)
- Human-readable
- Minimally verbose while keeping flexibility (vs csv)

qlog : simple to filter (both when reading and writing)

```
{"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": ...}]
 ]}
```

“HTTP_STREAM_OPEN” VS “HTTP”, “STREAM_OPEN”

Also saves on verbosity: akin to csv column names

qlog : clear cause and effect

```
{"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": ...}]
 ]}
```

Mainly when cause of event isn't clear from context
However, also easier for tooling: focus on certain triggers
→ Explicit vs implicit/heuristic logging

qlog : structured metadata

```
{"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": ...}]
 ]}
```

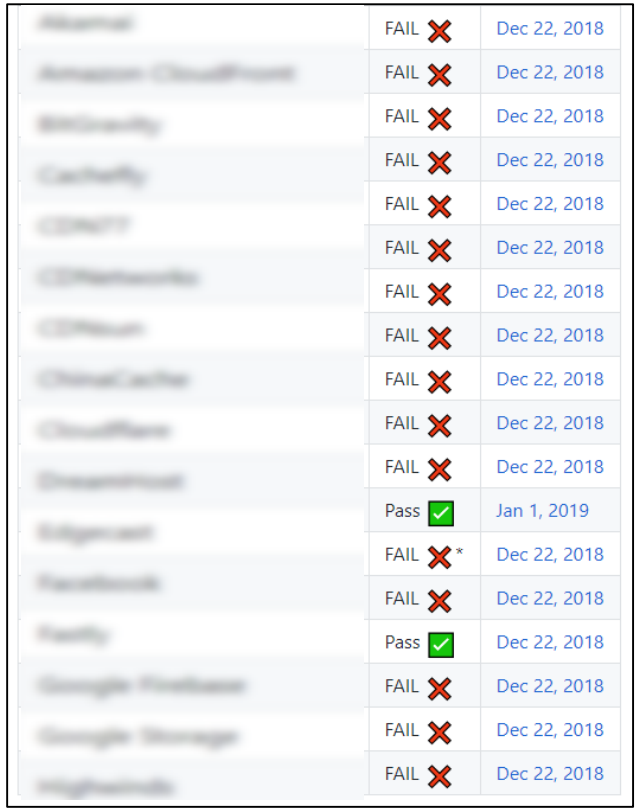
INITIAL 15 1523 VS type="initial", nr=15, size=1523

Standardized general purpose endpoint logging format?

- Why just for QUIC and HTTP/3?
  - TCP endpoint states
  - RTP / WebRTC / DTLS
  - Anything really…
- Wait… doesn't this exist yet?
  - Turns out: no…

Current state of tooling and logging

- Public
  - .pcap-based (e.g., tcptrace, wireshark)
  - In-browser devtools (but very high-level)
- Private
  - Many more proprietary/internal tools
  - Focused on individual implementation/logging
- In many (academic) cases : none
  - Wrong interpretations of results
  - Important bugs can remain undetected for a long time



[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	Pass ✓	Jan 1, 2019
[redacted]	FAIL ✗*	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	Pass ✓	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018

We seem to rely on a *limited amount* of (not thoroughly tested) implementations, which are understood only through their (high-level), *end-to-end behaviour*

https://github.com/andydavies/http2-prioritization-issues
https://twitter.com/AndyDavies/status/1065916677408346112
https://blog.cloudflare.com/http-2-prioritization-with-nginx/

Standardized general purpose endpoint logging format

- Discussed at IETF 104
  - Too early for its own working group
  - Use QUIC as incubator / concrete use case

- 1. High-level schema
  - Semi- protocol agnostic
  - Take into account a variety of use cases

- 2. QUIC + H3 event definitions
  - Names + metadata semantics for each type of event
  - Later also:
    - Method of access
    - Security and privacy considerations

https://quiclog.github.io/internet-drafts/draft-marx-quic-logging-main-schema.html
https://quiclog.github.io/internet-drafts/draft-marx-quic-logging-event-definitions.html

High-level logging schema

- Main tenets
  - Flexibility in the format, complexity in the tooling
  - Extensible but pragmatic (e.g., no complex fixed schema with extension points)
  - Streamable, event-based
  - Aggregation and transformation friendly
  - Explicit and human-readable

1. Flexibility : included fields depend on use case

```
"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": ...}],
```

Fields are now more dynamic

- Depend on use case
- Split in per-event and shared fields

common_fields

- Value listed once in header

event_fields

- Value listed separately for each event instance

```
"common_fields": {
    "group_id": "127ecc830d98f9d54a42c4f0842aa87e181a",
    "protocol_type":  "QUIC_HTTP3",
    "reference_time": "1553986553572"
},
"event_fields": [
    "relative_time",
    "CATEGORY",
    "EVENT_TYPE",
    "TRIGGER",
    "DATA"
],
```

1. Flexibility : mingle different types of log together

```
"common_fields": {
},
"event_fields": [
    "time",
    "protocol_type",
    "group_id",
    "CATEGORY",
    "EVENT_TYPE",
    "TRIGGER",
    "DATA"
],
"events": [[
        1553986553579,
        "QUIC_HTTP3",
        { "ip1": "2001:67c:1232:144:9498:6df6:f450:110b", "ip2": "2001:67c:2b0:1c1::198", "port1": 59105, "port2": 80 }
        "TRANSPORT",
        "PACKET_RX",
        "LINE",
        {...}
    ],[
        1553986553588,
        "TCP",
        { "ip1": "10.0.6.137", "ip2": "52.58.13.57", "port1": 56522, "port2": 443 }
        "TRANSPORT",
        "RETRANSMIT",
        "TIMEOUT",
        {...}
    ],[
```

1. Flexibility : file size optimization

```
{
  "common_fields": {
      "protocol_type":  ["QUIC_HTTP3", "TCP"],
      "group_ids": [
          { "ip1": "2001:67c:1232:144:9498:6df6:f450:110b", "ip2": "2001:67c:2b0:1c1::198", "port1": 59105, "port2": 80 },
          { "ip1": "10.0.6.137", "ip2": "52.58.13.57", "port1": 56522, "port2": 443 }
      ]
  }
  "event_fields": [
      "time",
      "protocol_type",
      "group_id",
      "CATEGORY",
      "EVENT_TYPE",
      "TRIGGER",
      "DATA"
  ],
  "events": [[
          1553986553579,
          0,
          0
          "TRANSPORT",
          "PACKET_RX",
          "LINE",
          [...]
      ],[
          1553986553588,
          1,
          1
          "APPLICATION",
          "DATA_FRAME_NEW",
          "GET",
          [...]
      ],[
```

Reference-by-index

- Smaller files, less readability
- Could be done for any field
- Done in Chromium's internal format Netlog

https://github.com/quiclog/internet-drafts/issues/8

1. Flexibility : file size optimization to the extreme

```
[ 57, "TRANSPORT", "FRAME_CREATED", "TRIGGER", ["frame_type": "STREAM", "packet_number": 15, "contents": [...]]],
[ 58, "TRANSPORT", "FRAME_CREATED", "TRIGGER", ["frame_type": "STREAM", "packet_number": 16, "contents": [...]]]
```


https://github.com/quiclog/internet-drafts/issues/8

1. Flexibility : file size optimization to the extreme

```
[ 57, "TRANSPORT", "FRAME_CREATED", "TRIGGER", ["frame_type": "STREAM", "packet_number": 15, "contents": [...]]],
[ 58, "TRANSPORT", "FRAME_CREATED", "TRIGGER", ["frame_type": "STREAM", "packet_number": 16, "contents": [...]]]
```

```
"data_fields" : {
      "TRANSPORT+FRAME_CREATED" : [
        "frame_type",
        "packet_number",
        "contents"
      ]
}
...
[ 57, "TRANSPORT", "FRAME_CREATED", "TRIGGER", ["STREAM", 15, [...]]],
[ 58, "TRANSPORT", "FRAME_CREATED", "TRIGGER", ["STREAM", 16, [...]]]
```

“Fully self-describing format”

→ Extensible,
but pragmatic

https://github.com/quiclog/internet-drafts/issues/8

2. Streamability

- "Live debugging" : tool updates as events come in
- JSON is not a streamable format per se

```
{"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": ...}],
    ...
    [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": ...}]
 ]}
```

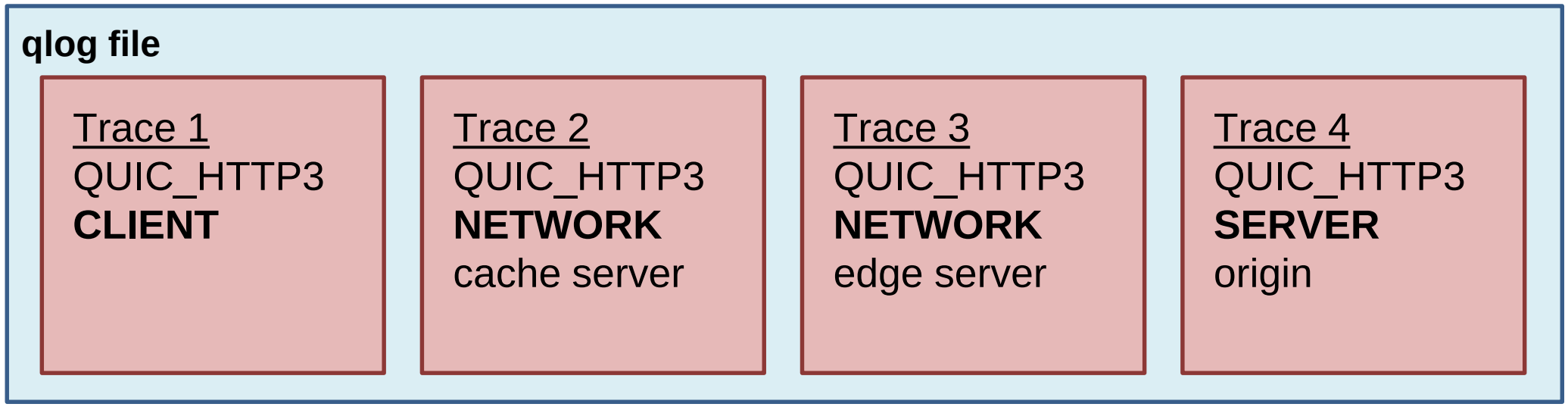
Easy enough to stream individual events

These two characters are apparently pretty important

- "Solution": streaming JSON parser

https://github.com/quiclog/internet-drafts/issues/2

3. Aggregation friendly : multiple traces in 1 file



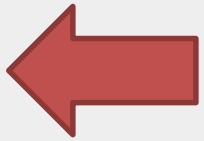
- `vantage_point`
  - Combine traces from several locations for end-to-end overview
  - Also possible: split out per protocol, per flow, type of event, …
    - e.g., trace 1 = HTTP/2 from server, trace 2 = TCP from eBPF

3. Aggregation friendly : Tooling support

Quickly sift through hundreds of logs (put on top + streaming parser)

```
"summary": {
    "trace_count":number, // amount of traces in this file
    "max_duration":string, // time duration of the longest trace
    "max_outgoing_loss_rate":number, // highest loss rate for outgoing packets over all traces
    "total_event_count":number // total number of events across all traces
}
```

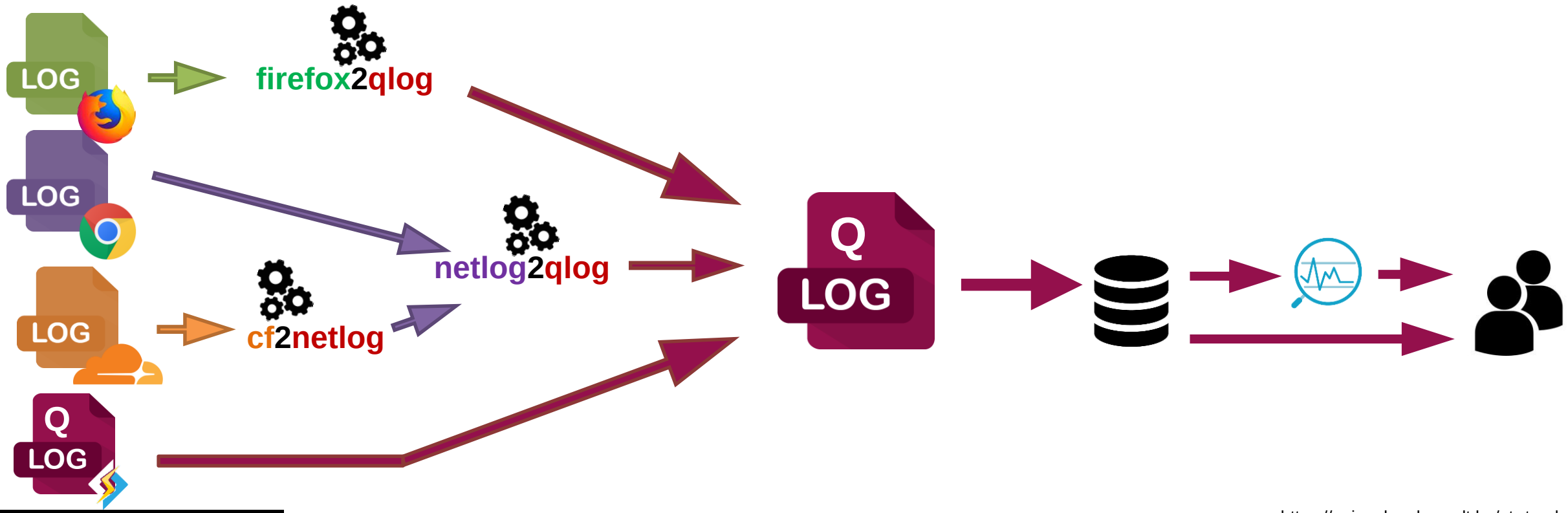
```
"configuration": {
        "time_units": "ms",
        "time_offset": 100,

        "quicvis.timeline.settings": {
                "xmin": 1000,
                "xmax": 2000,
                "streams.enabled": [1,5,9],
                "color.scheme": "HIGHLIGHT_LOSS"
        }
}
```



Immediately clear what other person should be looking at

4. Transformation friendly

- "No one is going to output qlog directly"
  - Liar! But even then: they don't need to
  - "common logging format" → "common tool input format"



https://quic.edm.uhasselt.be/qtr-to-qlog/
https://github.com/quiclog/pcap2qlog

Many more open questions

- Textual vs binary (readability vs file size savings/logging perf)
- Are separate triggers useful?
- Preventing proliferation of someting2qlog converters?
- Fine-grainedness of events
- Privacy aspects

- Single format for many use cases?
  - Even within QUIC: CDN vs Facebook app vs browsers vs IoT vs …
  - Why doesn't this exist yet?

Many more open questions

- Textual vs binary (readability vs file size savings/logging perf)
- Are separate triggers useful?
- Preventing proliferation of someting2qlog converters?
- Fine-grainedness of events
- Privacy aspects

- Single format for many use cases?
  - Even within QUIC: CDN vs Facebook app vs browsers vs IoT vs …
  - Why doesn't this exist yet?

- How the hell do you publish papers on this topic?

Let’s talk tools!

DEMO TIME

Also plenty of open questions about tools

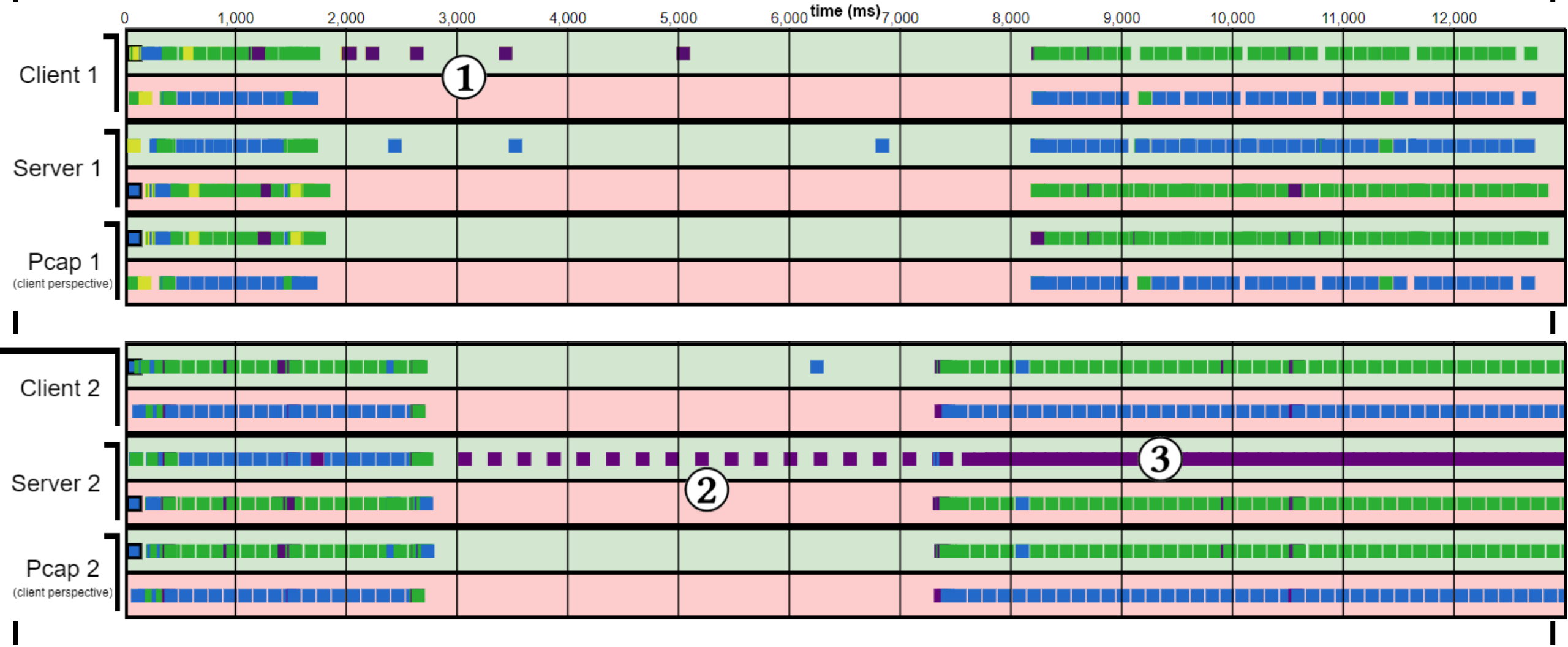
- How to handle overlapping data?
- Many small tools vs a few mega-tools?
- Truly re-usable and integrate-able tools
  - More of a software engineering challenge…
- Need to know what you are looking for up-front…
  - Tools that automatically identify problematic areas in a trace?
- Tools need to indicate which events they rely on
- Chicken or the egg: tools or qlog support?

- Which tools would you use? (which do you use today?)

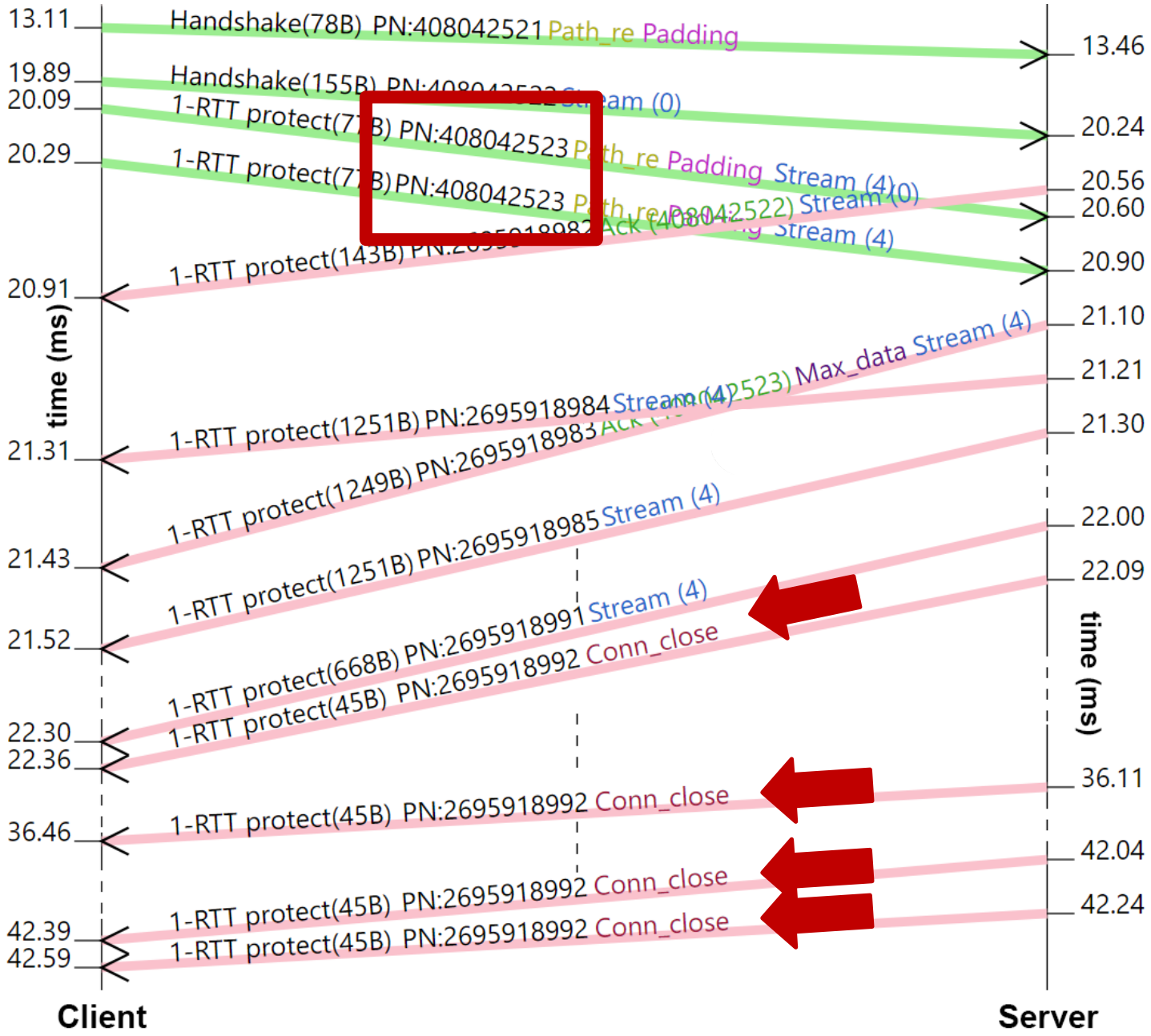
QUIC visualization: bug/behaviour examples

Extra slides / potential question support

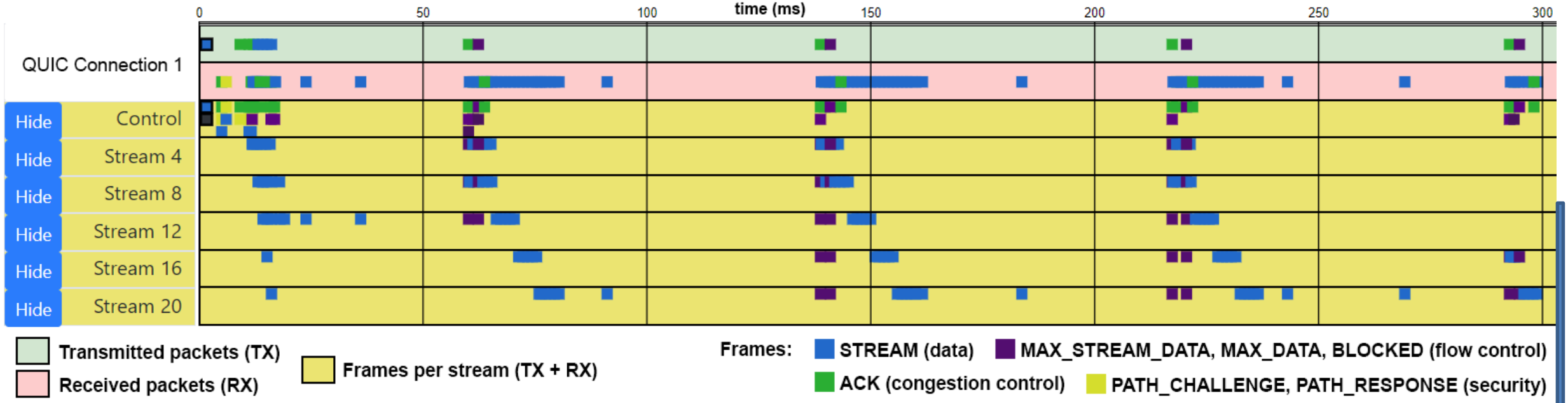
QUICvis examples : connectivity lost

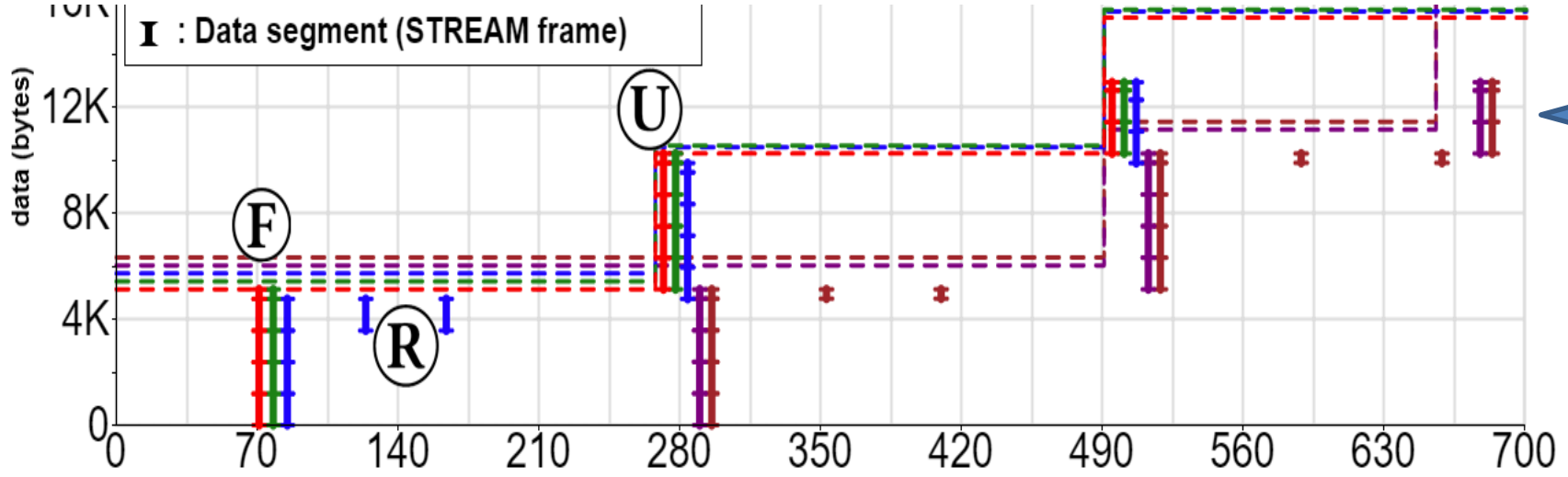


QUICvis examples : Duplicate packet nr

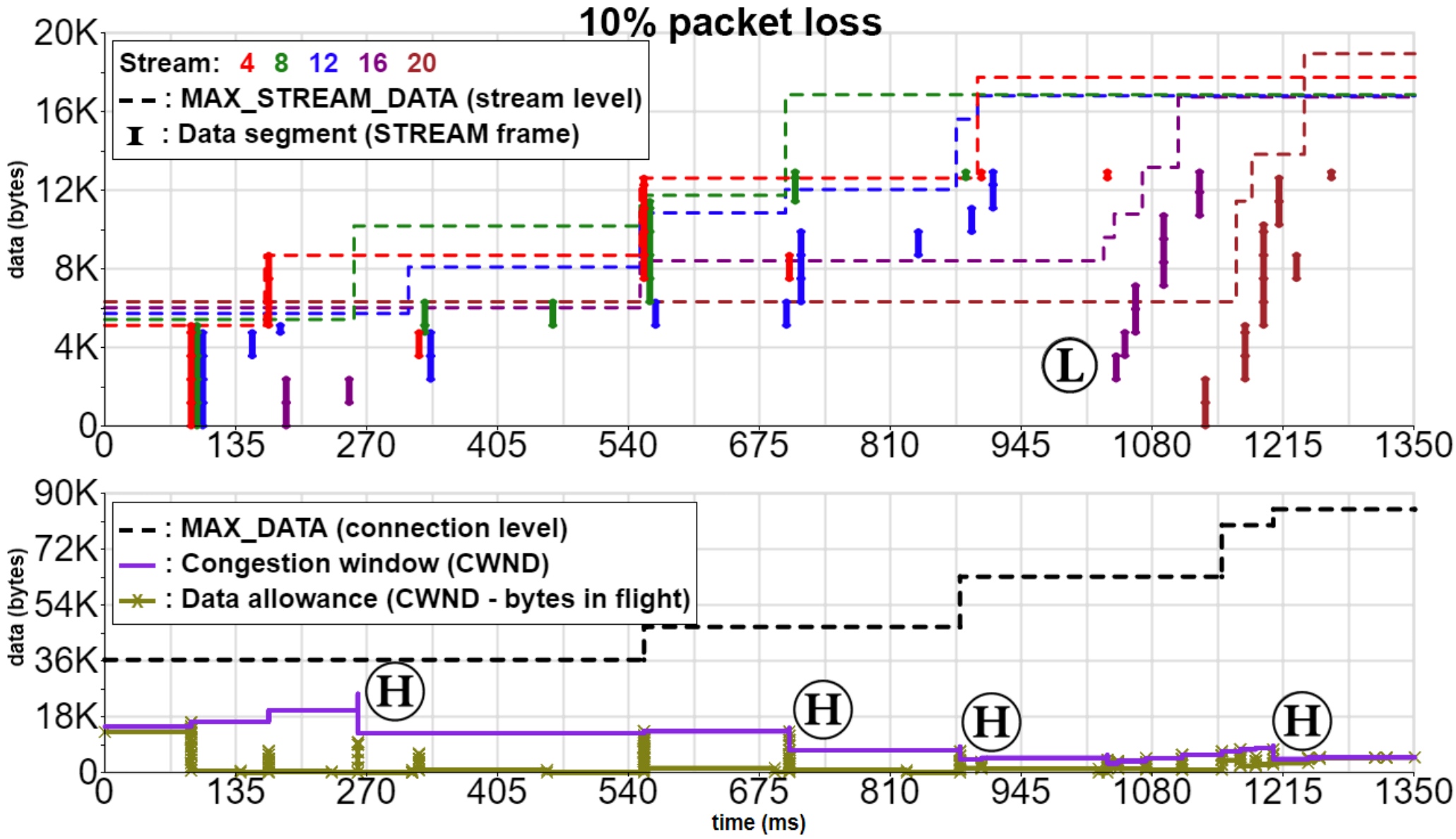


QUICvis : Flow and congestion control logic





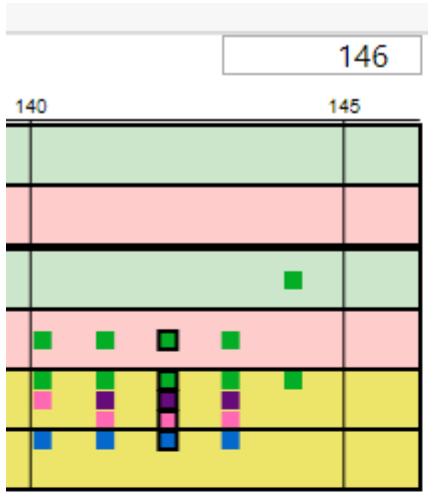
QUICvis : Flow and congestion control logic



Sending data along with BLOCKED, going over the limit

- payloadinfo
  - Max_data
    - frametype : 4
    - maximum_data : 102400
  - Max_str_data
    - frametype : 5
    - stream_id : 4
    - maximum_data : 204800

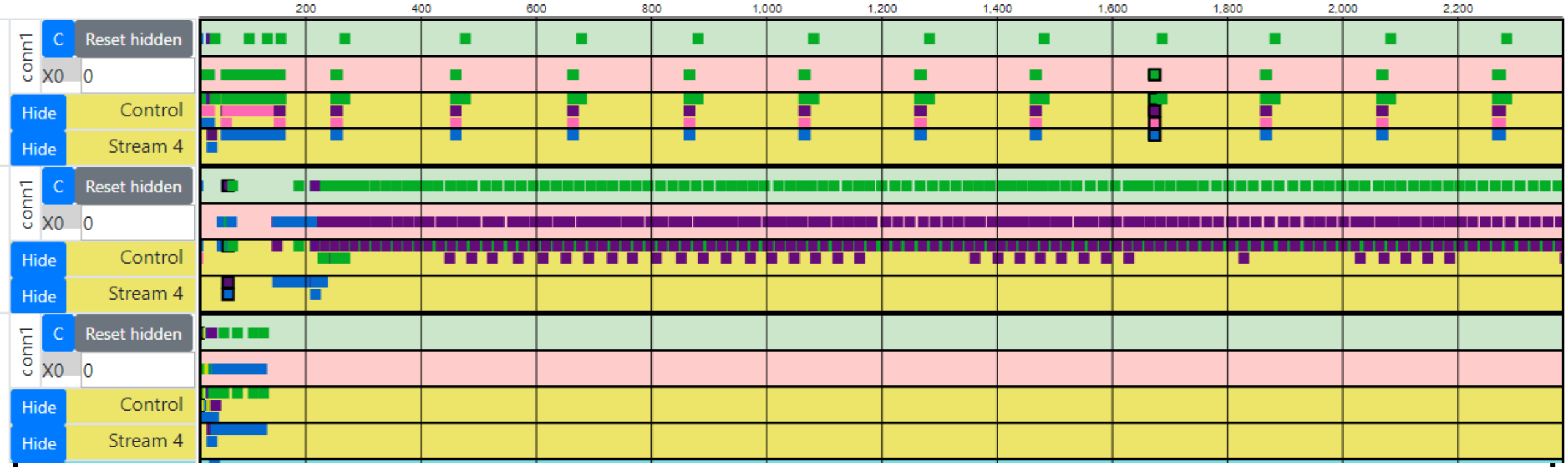
Client sends erroneous flow control allowances



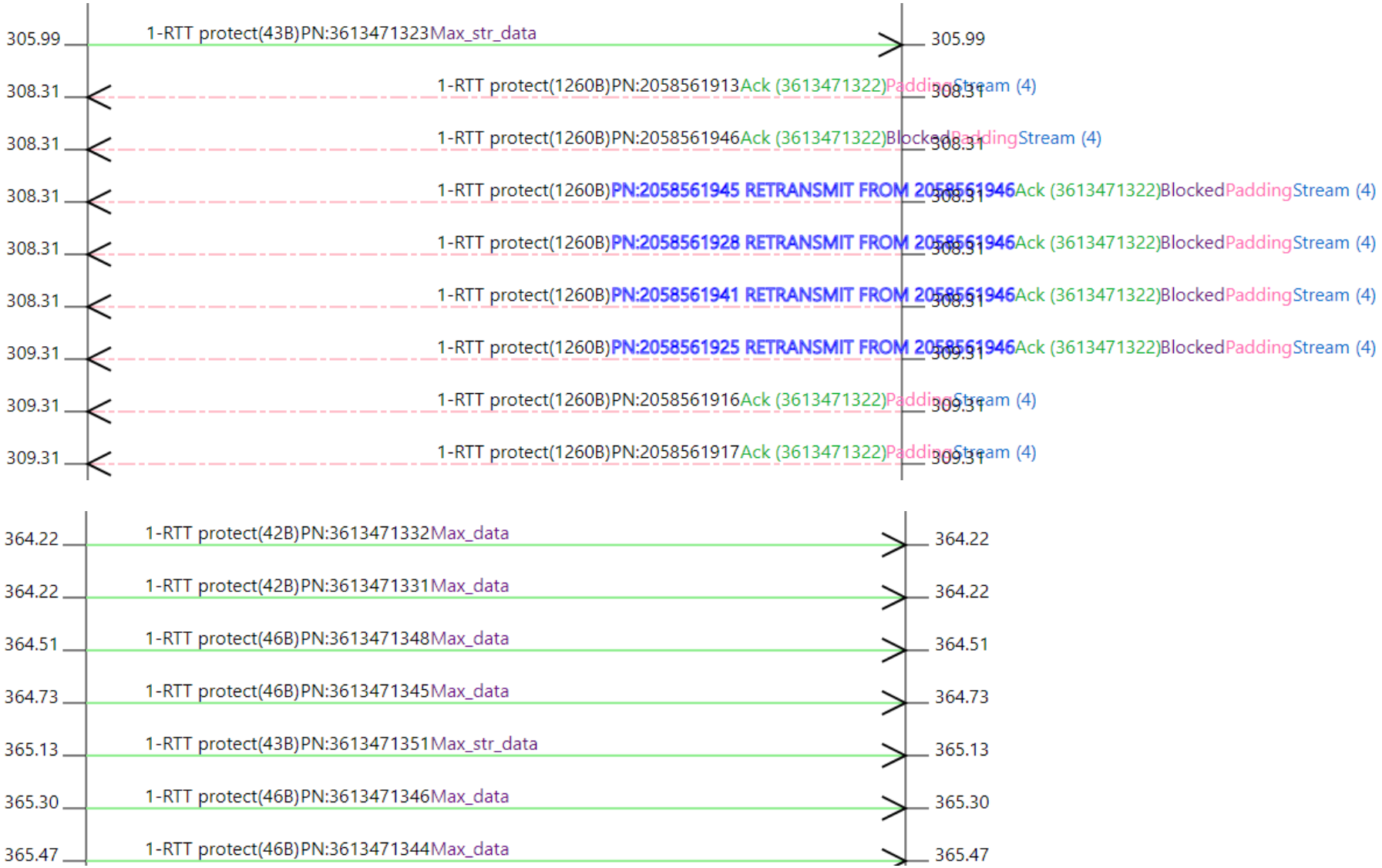
Server sends BLOCKED, accompanied by STREAM, going over the max_data

- Ack
  - frametype : 13
  - largest_ack : 718238325
  - ack_delay : 760
  - ack_block_count : 0
  - ack_blocks : []
- Blocked
  - frametype : 8
  - offset : 102400
- Padding
  - frametype : 0
  - length : 51
- Stream
  - frametype : 22
  - type_flags : { "off_flag": true, "len_flag": true, "fin_flag": false }
  - stream_id : 4
  - offset : 101460
  - length : 1140
  - stream_data : 626f726973206e69736920757420616c'borisnisiu
- serverinfo

Keep sending data VS flood of BLOCKED

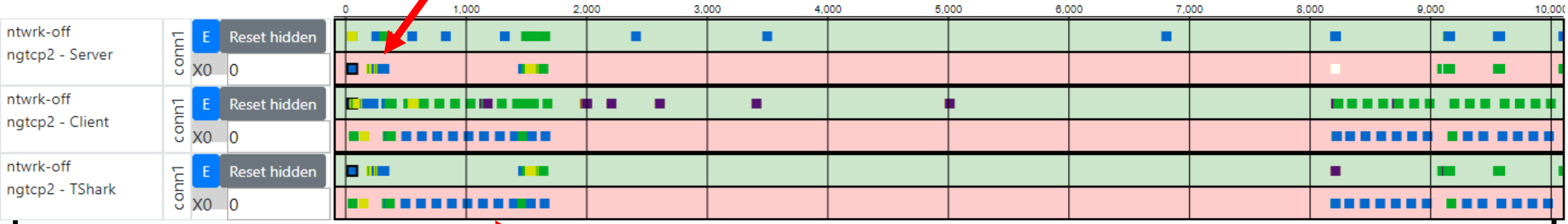


Server retransmits too much, client answers to each blocked



Pacing (network, not server)

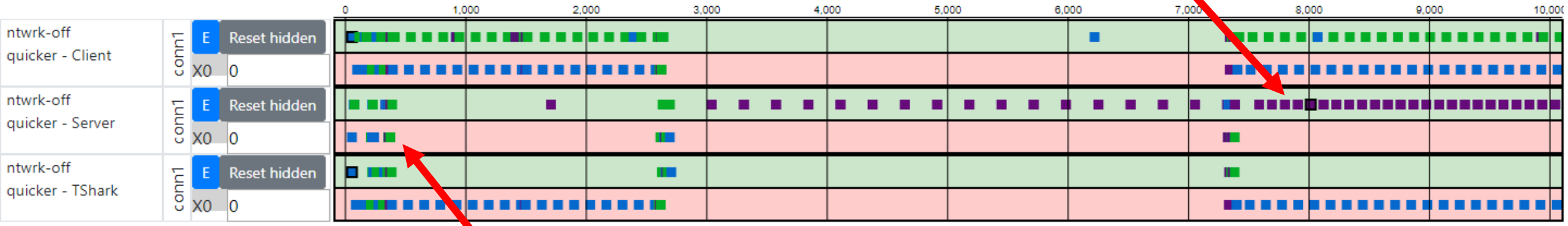
Server sends all at once



Client and network see very spaced-out

Pacing (server, not network)

Server sends interleaved itself



Server sends all at once at first

Extra slides

QUIC and HTTP/3

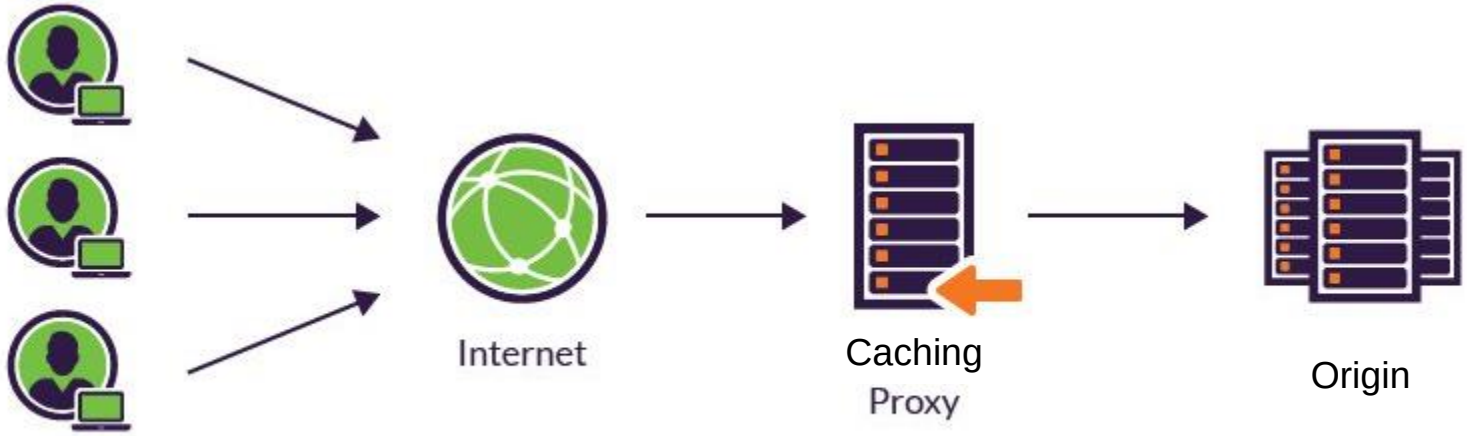
- Many people will be looking into the behavior
  - Initial implementations + conformance testing (current stage)
  - Early and at-scale deployments
  - Academic research (and teaching!)

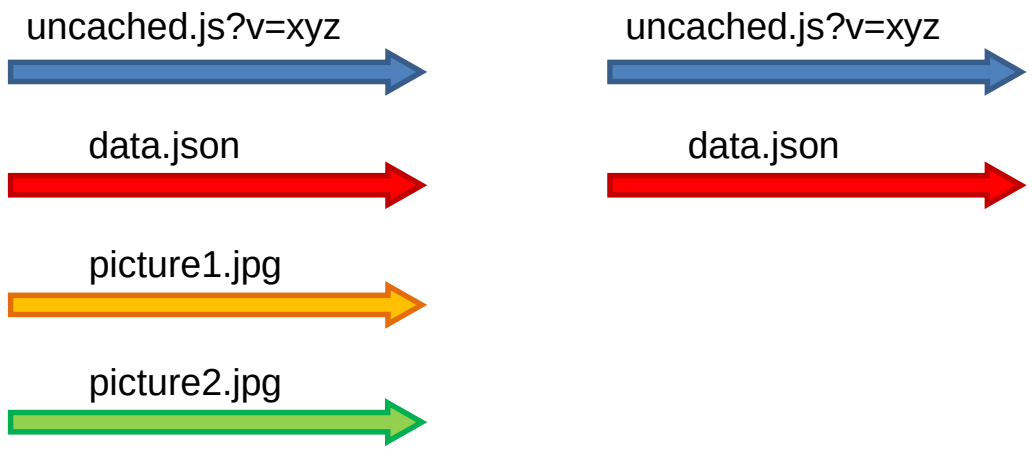
  - Cycle starts over with new features in v2

Many use cases

- Debugging
- Live deployment
- Education
- New feature development
- Large scale verification

In the wild, things start getting hairy real quick: bufferbloat





Expected:

Actual:

Image: https://www.incapsula.com/cdn-guide/glossary/reverse-proxy.html
https://blog.cloudflare.com/http-2-prioritization-with-nginx/
https://github.com/andydavies/http2-prioritization-issues

Standard logging: existing alternatives

- HTTP/2 debug state
  - .json response for .well-known/h2/state
  - High-level summary of internal h2 state
  - Poll-based, manually diff changes between states


```
"streams": {
  "5": {
    "state": "HALF_CLOSED_REMOTE",
    "flowIn": 65535,
    "flowOut": 6291456,
    "dataIn": 0,
    "dataOut": 0,
    "paddingIn": 0,
    "paddingOut": 0,
    "created": 1470835059.619137
  },
  "7": {
    "state": "OPEN",
    "flowIn": 65535,
    "flowOut": 6291456,
    "queuedData": 59093,
  }
},
```


Low overhead

Coarse grained

https://tools.ietf.org/html/draft-benfield-http2-debug-state-01

Standard logging: existing alternatives

- NetLog (Chromium)
  - .json log of full browser window
  - Medium-level (no congestion stuff, prioritization, loss, …)
  - Event-based, one entry for every state change

```
t=186143 [st=  40]    QUIC_SESSION_STREAM_FRAME_RECEIVED
                      --> fin = true
                      --> length = 0
                      --> offset = "0"
                      --> stream_id = 5
t=186143 [st=  40]    QUIC_CHROMIUM_CLIENT_STREAM_READ_RESPONSE_HEADERS
                      --> :status: 304
                          age: 187
                          alt-svc: quic=":443"; ma=2592000; v="46,44,43,39"
                          date: Wed, 13 Mar 2019 13:14:13 GMT
                          etag: "1552399307"
                          expires: Wed, 13 Mar 2019 13:19:13 GMT
t=186158 [st=  55]    QUIC_CHROMIUM_CLIENT_STREAM_SEND_REQUEST_HEADERS
                      --> :authority: i.ytimg.com
                          :method: GET
                          :path: /vi/v8QiKkd4-ms/hqdefault.jpg?sqp=-oaymwEY(
                          :scheme: https
                          accept: image/webp,image/apng,image/*,*/*;q=0.8
                          accept-encoding: gzip, deflate, br
                          accept-language: en-US,en;q=0.9,nl;q=0.8
                          referer: https://www.youtube.com/
                          user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; :
                      --> quic_priority = 3
                      --> quic_stream_id = 7
```

- Event correlation to "sources"
- Event phase: start, end, none

Finer grained

High overhead

chrome://net-export
https://netlog-viewer.appspot.com

Standard logging: existing alternatives

- quic-trace
  - .json response (from protocolbuffer)
  - Low-level (focus on congestion control and loss)
  - Event-based, one entry for every state change

```
enum EventType {
  UNKNOWN_EVENT = 0;
  PACKET_SENT = 1;
  PACKET_RECEIVED = 2;
  PACKET_LOST = 3;
  APPLICATION_LIMITED = 4;
  EXTERNAL_PARAMETERS = 5;
};

enum TransmissionReason {
  NORMAL_TRANSMISSION = 0;
  TAIL_LOSS_PROBE = 1;
  RTO_TRANSMISSION = 2;
  PROBING_TRANSMISSION = 3;
};
```

- Reasons logged explicitly

Finer grained

High overhead

https://github.com/google/quic-trace