Logging, Tooling and Debugging for Modern Network Protocols

Robin Marx - @programmingart
(Hasselt University – Belgium)

UHASSELT EDM

IETF 104 – Prague – March 2019

Disclaimer

- 3rd year PhD Student
- HTTP/2 and QUIC
- Newcomer: not much experience with wider (IETF) stuff
- Please tell me why I'm absolutely and terribly wrong

Let me tell you a story

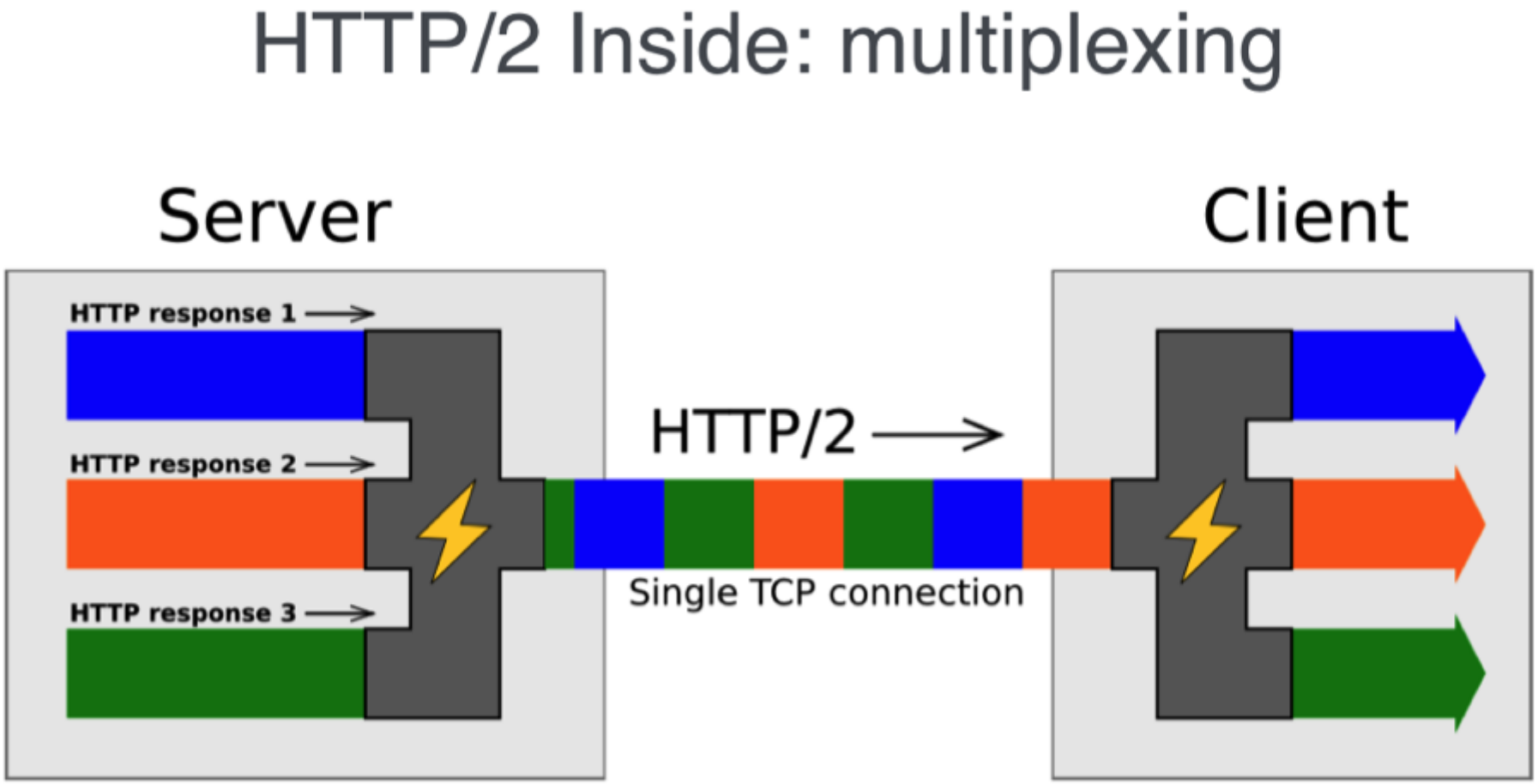


Image: https://www.nginx.com/blog/http2-module-nginx/

Let me tell you a story

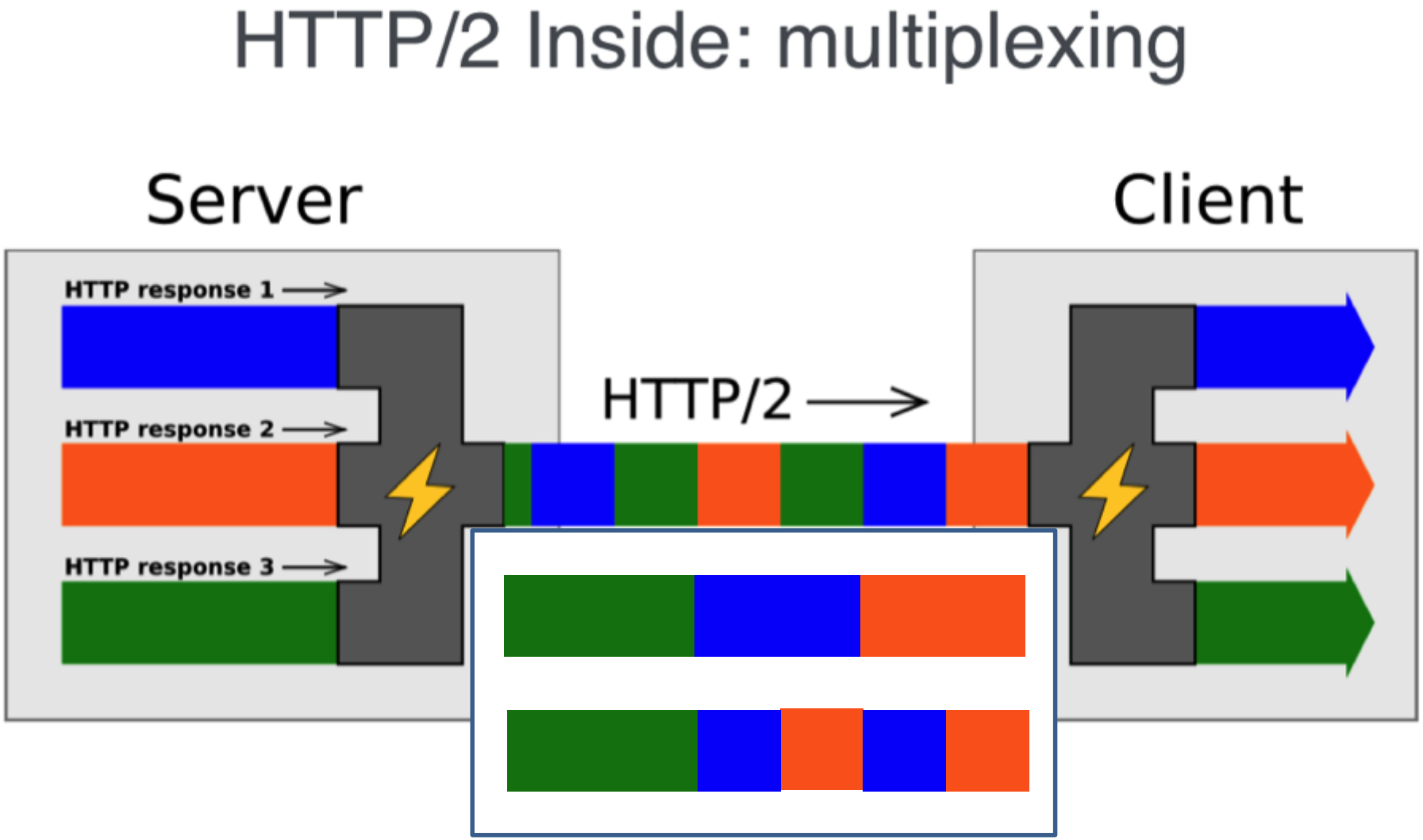
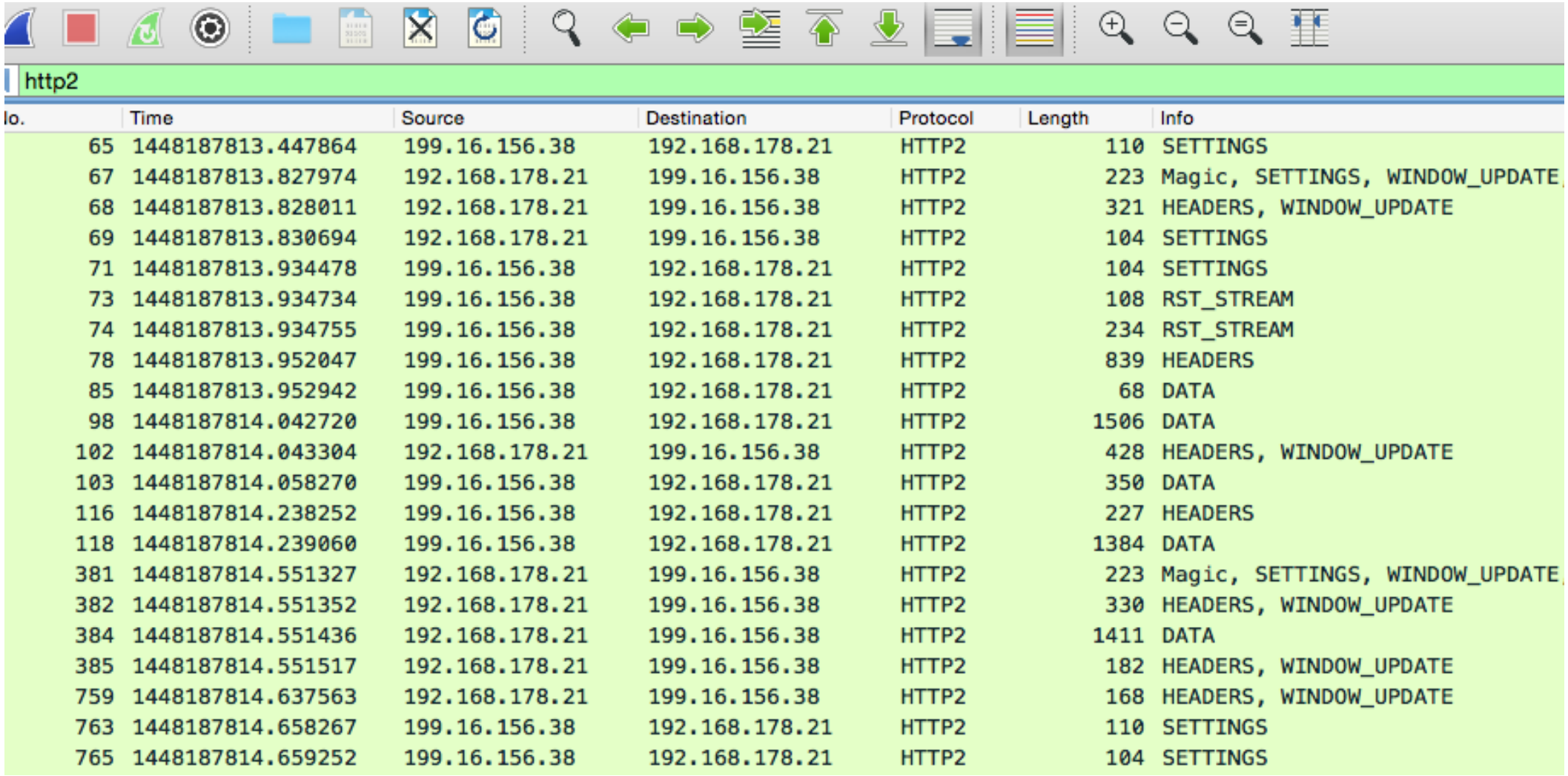


Image: https://www.nginx.com/blog/http2-module-nginx/

HTTP/2 Prioritization: difficult to debug

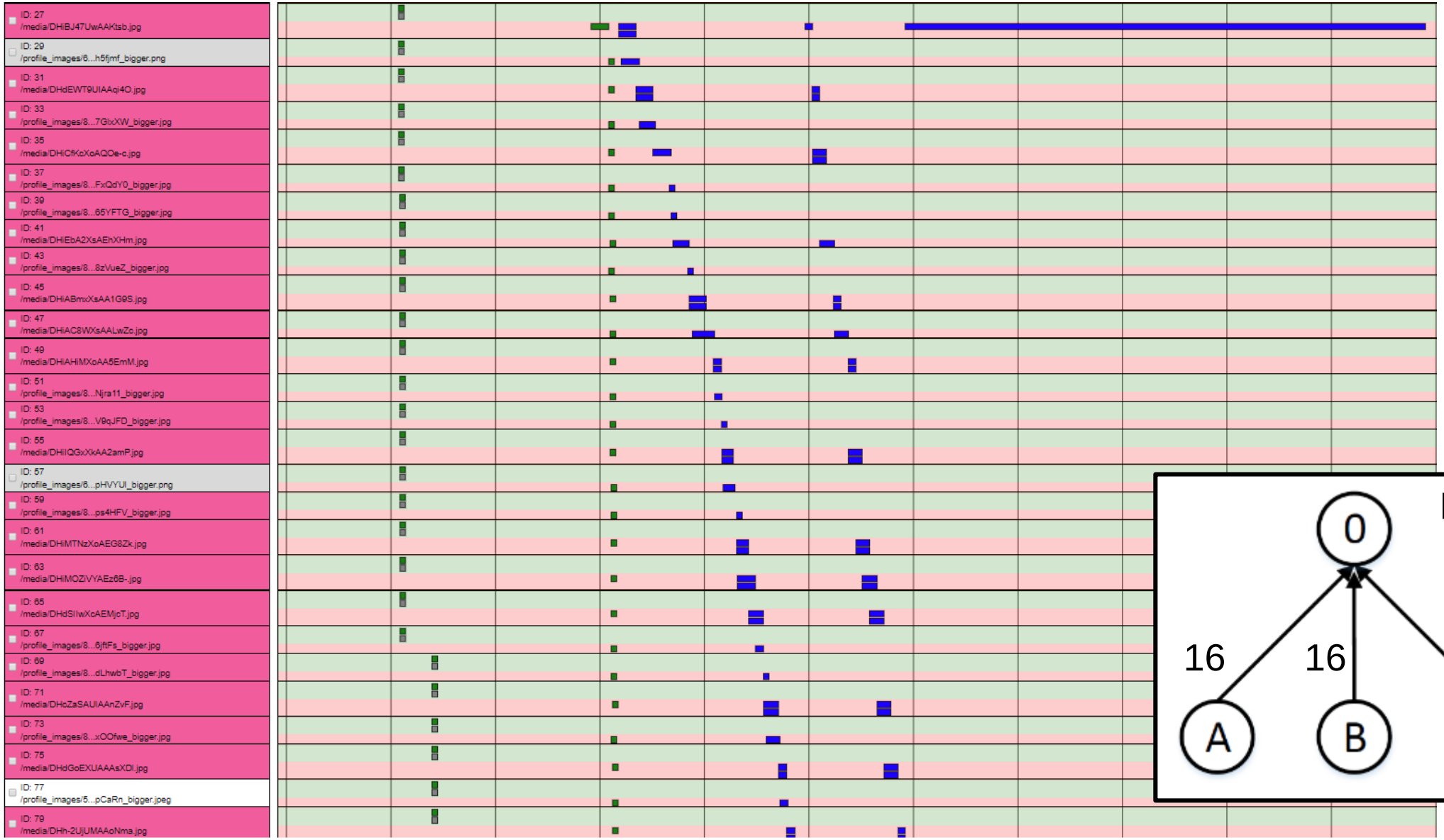


http2

lo.	Time	Source	Destination	Protocol	Length	Info
65	1448187813.447864	199.16.156.38	192.168.178.21	HTTP2	110	SETTINGS
67	1448187813.827974	192.168.178.21	199.16.156.38	HTTP2	223	Magic, SETTINGS, WINDOW_UPDATE
68	1448187813.828011	192.168.178.21	199.16.156.38	HTTP2	321	HEADERS, WINDOW_UPDATE
69	1448187813.830694	192.168.178.21	199.16.156.38	HTTP2	104	SETTINGS
71	1448187813.934478	199.16.156.38	192.168.178.21	HTTP2	104	SETTINGS
73	1448187813.934734	199.16.156.38	192.168.178.21	HTTP2	108	RST_STREAM
74	1448187813.934755	199.16.156.38	192.168.178.21	HTTP2	234	RST_STREAM
78	1448187813.952047	199.16.156.38	192.168.178.21	HTTP2	839	HEADERS
85	1448187813.952942	199.16.156.38	192.168.178.21	HTTP2	68	DATA
98	1448187814.042720	199.16.156.38	192.168.178.21	HTTP2	1506	DATA
102	1448187814.043304	192.168.178.21	199.16.156.38	HTTP2	428	HEADERS, WINDOW_UPDATE
103	1448187814.058270	199.16.156.38	192.168.178.21	HTTP2	350	DATA
116	1448187814.238252	199.16.156.38	192.168.178.21	HTTP2	227	HEADERS
118	1448187814.239060	199.16.156.38	192.168.178.21	HTTP2	1384	DATA
381	1448187814.551327	192.168.178.21	199.16.156.38	HTTP2	223	Magic, SETTINGS, WINDOW_UPDATE
382	1448187814.551352	192.168.178.21	199.16.156.38	HTTP2	330	HEADERS, WINDOW_UPDATE
384	1448187814.551436	192.168.178.21	199.16.156.38	HTTP2	1411	DATA
385	1448187814.551517	192.168.178.21	199.16.156.38	HTTP2	182	HEADERS, WINDOW_UPDATE
759	1448187814.637563	192.168.178.21	199.16.156.38	HTTP2	168	HEADERS, WINDOW_UPDATE
763	1448187814.658267	199.16.156.38	192.168.178.21	HTTP2	110	SETTINGS
765	1448187814.659252	199.16.156.38	192.168.178.21	HTTP2	104	SETTINGS

chrome://net-internals/#http2
https://vanwilgenburg.wordpress.com/2015/11/22/how-to-capture-and-decode-http2-traffic-with-wireshark/

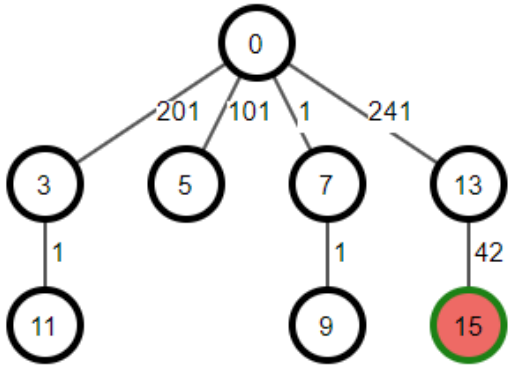
HTTP/2 Prioritization: is clearer when visualized



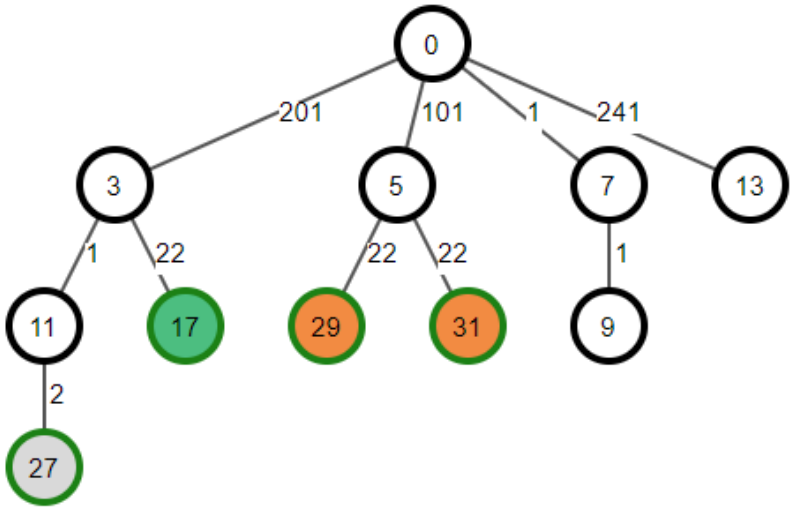
https://github.com/rmarx/h2vis

HTTP/2 Prioritization: can be quite complex

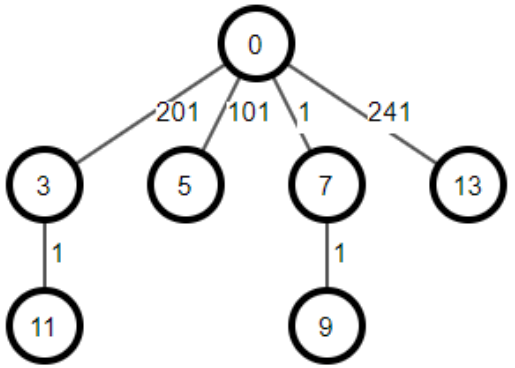
ID: 3	T: PRIORITY	S: CLIENT	P: 0	E: F	W: 201
ID: 5	T: PRIORITY	S: CLIENT	P: 0	E: F	W: 101
ID: 7	T: PRIORITY	S: CLIENT	P: 0	E: F	W: 1
ID: 9	T: PRIORITY	S: CLIENT	P: 7	E: F	W: 1
ID: 11	T: PRIORITY	S: CLIENT	P: 3	E: F	W: 1
ID: 13	T: PRIORITY	S: CLIENT	P: 0	E: F	W: 241
ID: 15	T: HEADERS	S: CLIENT	P: 13	W: 42	



ID: 15	T: DATA	S: SERVER	ES: T	
ID: 17	T: HEADERS	S: CLIENT	P: 3	W: 22
ID: 19	T: HEADERS	S: CLIENT	P: 11	W: 2
ID: 19	T: RST_STREAM	S: CLIENT		
ID: 21	T: HEADERS	S: CLIENT	P: 11	W: 2
ID: 21	T: RST_STREAM	S: CLIENT		
ID: 23	T: HEADERS	S: CLIENT	P: 11	W: 2
ID: 23	T: RST_STREAM	S: CLIENT		
ID: 25	T: HEADERS	S: CLIENT	P: 11	W: 2
ID: 25	T: RST_STREAM	S: CLIENT		
ID: 27	T: HEADERS	S: CLIENT	P: 11	W: 2
ID: 29	T: HEADERS	S: CLIENT	P: 5	W: 22
ID: 31	T: HEADERS	S: CLIENT	P: 5	W: 22



ID: 17	T: DATA	S: SERVER	ES: T
ID: 19	T: DATA	S: SERVER	ES: T
ID: 21	T: DATA	S: SERVER	ES: T
ID: 23	T: DATA	S: SERVER	ES: T
ID: 25	T: DATA	S: SERVER	ES: T
ID: 27	T: DATA	S: SERVER	ES: T
ID: 29	T: DATA	S: SERVER	ES: T
ID: 31	T: DATA	S: SERVER	ES: T



time

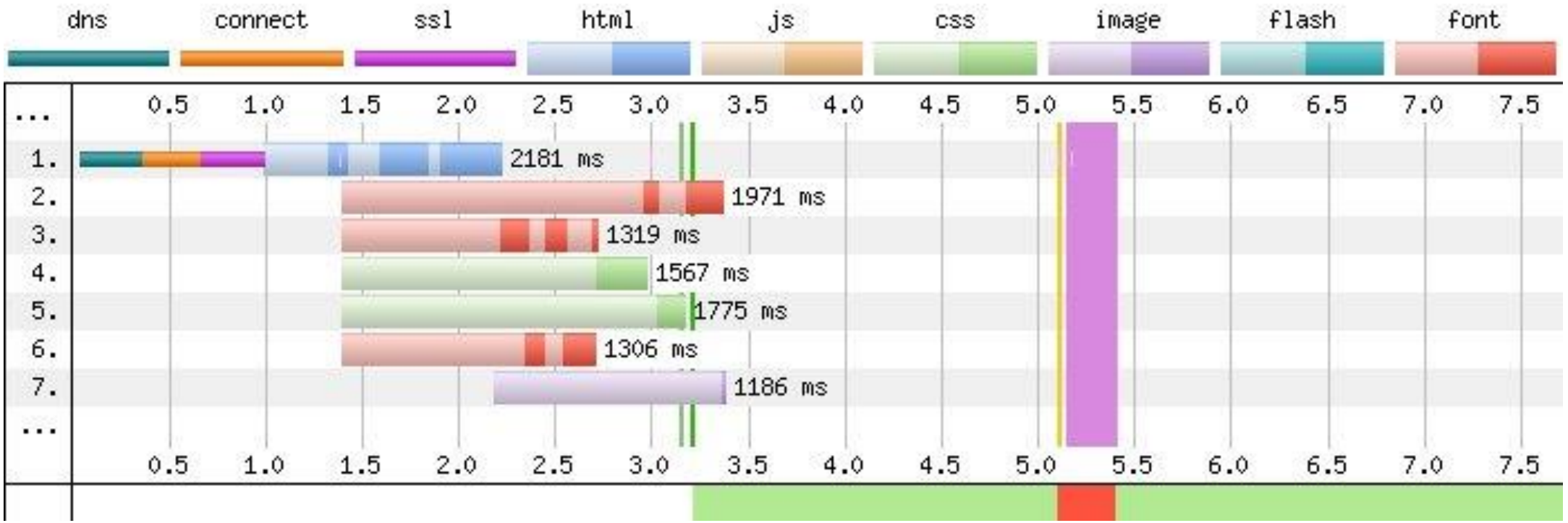
HTTP/2 Prioritization: Not everything is visible on the wire

- No explicit dependency tree sync from server to client
  - Unclear when/if nodes are removed

- Moreover: spec allows server to ignore client!
  - Rely fully on multiplexing observations to try and deduce actual prioritization behavior
  - Or, yes, of course, we could start reading the source code…

HTTP/2 Prioritization issues by example

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



webpagetest.org

HTTP/2 Prioritization issues by example

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

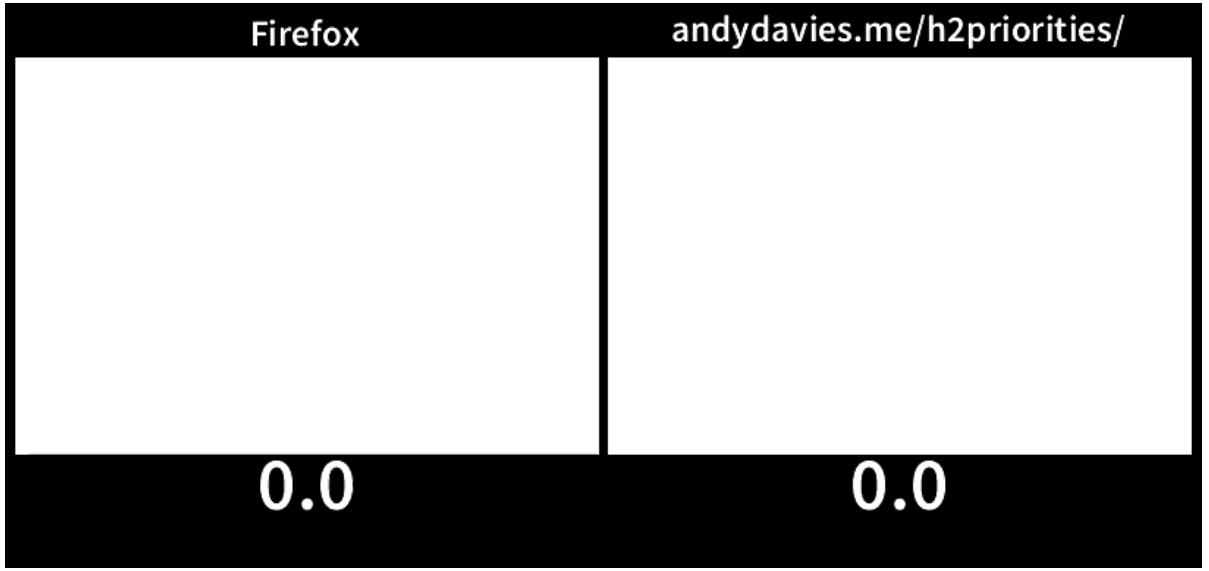
CDN / Hosting	Status	Test Result
[redacted]	Pass ✓	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗*	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	Pass ✓	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	Pass ✓	Dec 22, 2018
[redacted]	Pass ✓	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	Pass ✓	Dec 22, 2018
[redacted]	Pass ✓	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018
[redacted]	FAIL ✗	Dec 22, 2018

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

9 / 34 deployments pass

Unnamed CDN

H2O server



Observations

- Wire image does not contain all needed information
  - Internal endpoint state and decisions are important for debugging
- Visual tooling can help
  - Possibly not @ scale, but certainly for debugging individual issues
- Note: H2 prioritization is just one example
  - Will not even attempt to open the can of worms that is Push

The story continues…

"HTTP/2 was too easy, I need a real challenge"

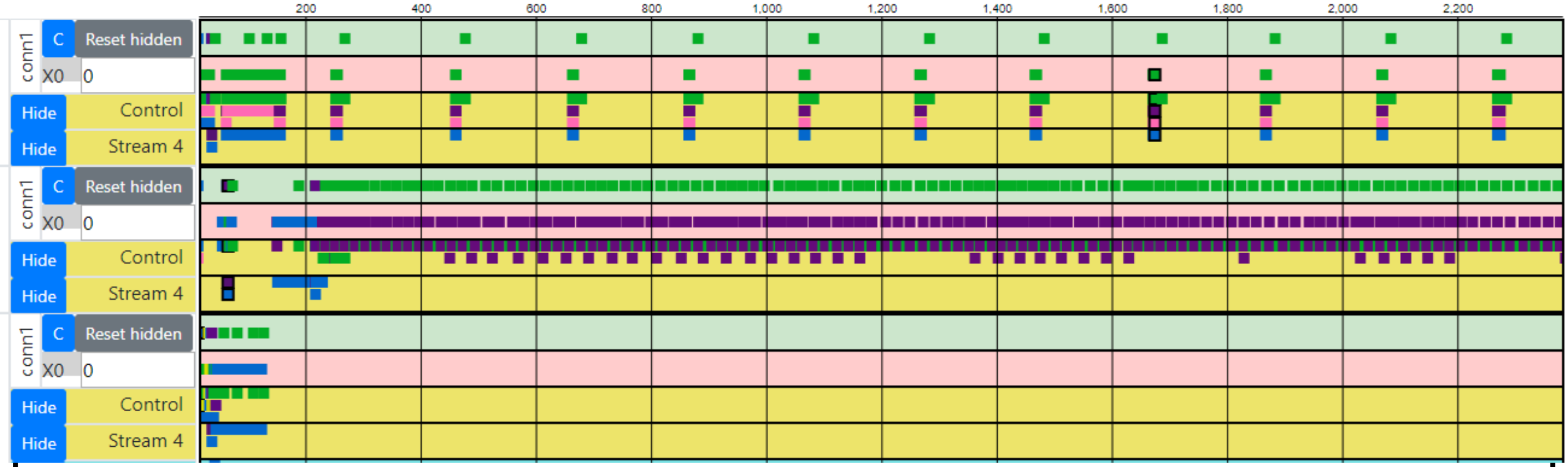
- every member of quicwg

QUIC and HTTP/3

- **Much** more complex than HTTP/2
  - Congestion control, flow control, handshake, 0-RTT, migration, …
  - Coming up: multipath, FEC, unreliability, …

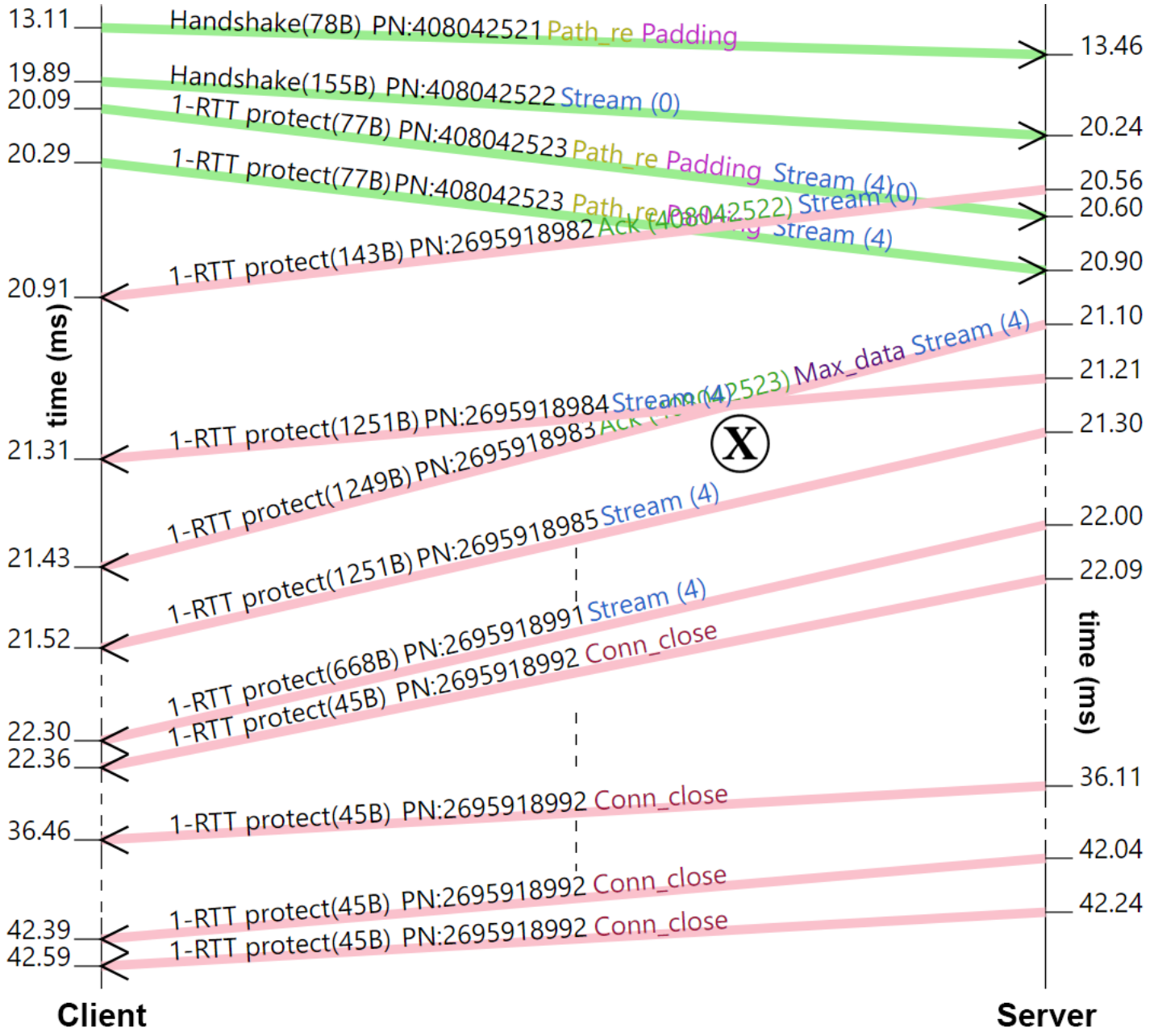
- Everything is re-implemented from scratch, so
- There will be:
  - Bugs
  - Suboptimal performance
  - Incomplete implementations
  - Consciously differing implementation choices and trade-offs

QUIC timeline: easy to compare different implementations

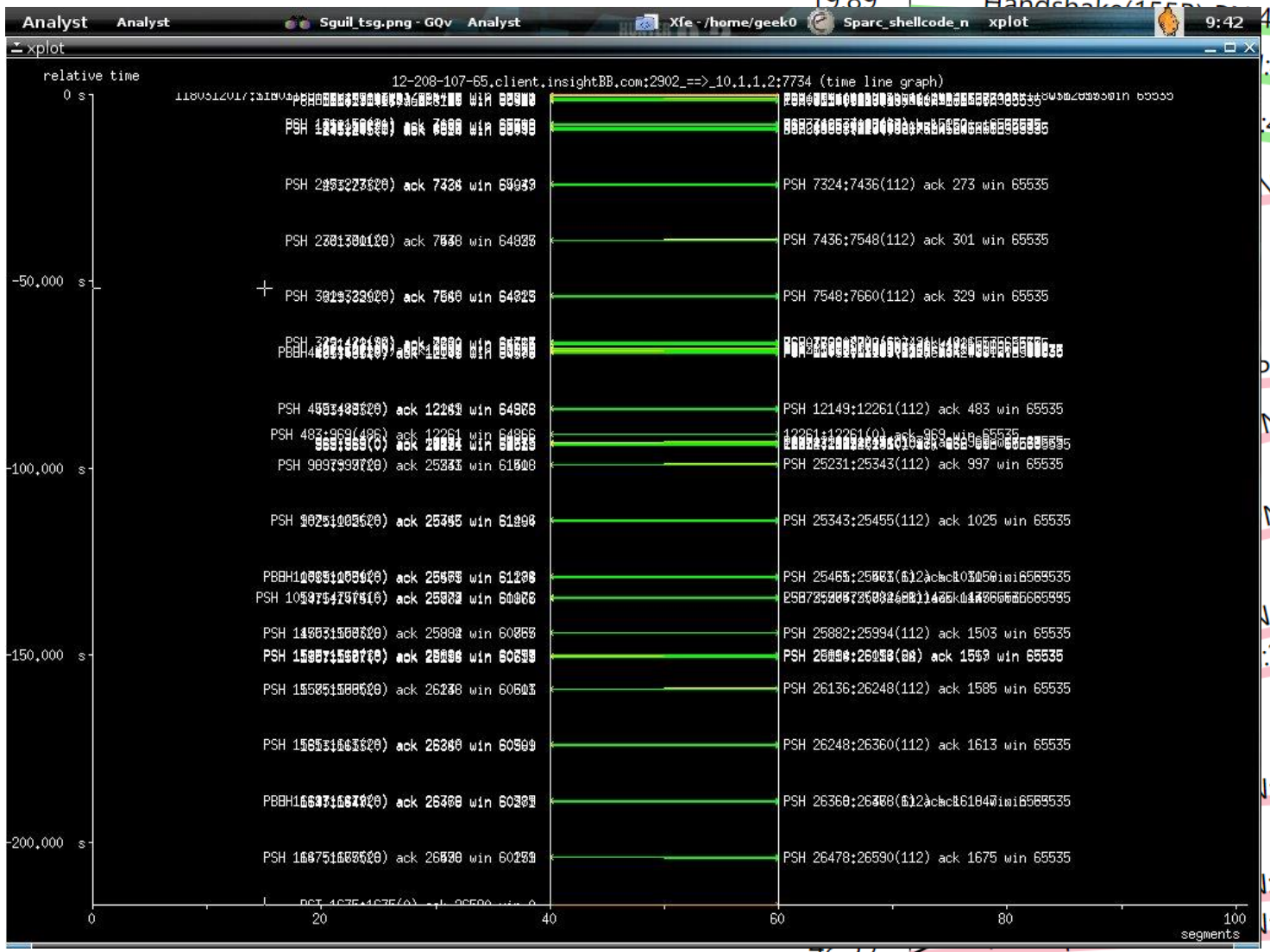


Believe it or not, this is actually the exact same test case
(blocked by flow control)

QUIC sequence diagram

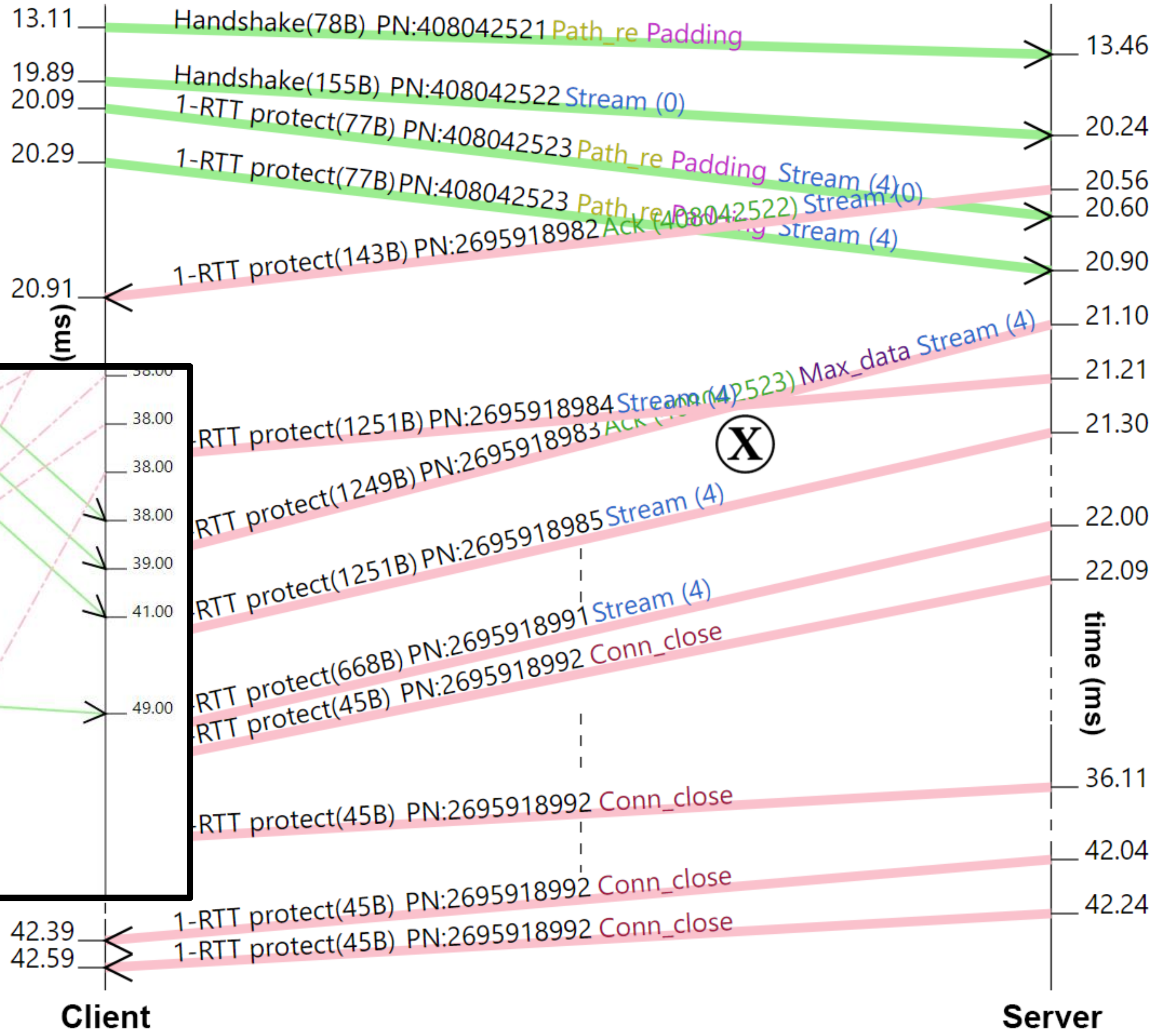


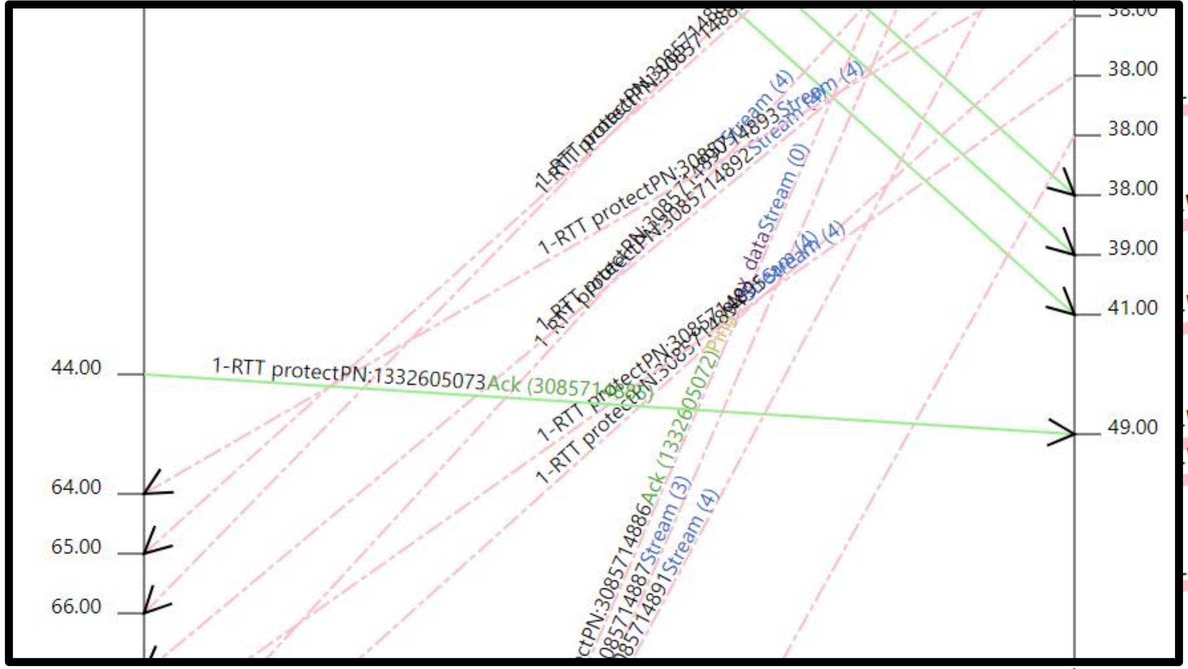
QUIC sequence diagram



QUIC sequence diagram

- Client + Server logs
  - Exact latency
    - Flight + processing!





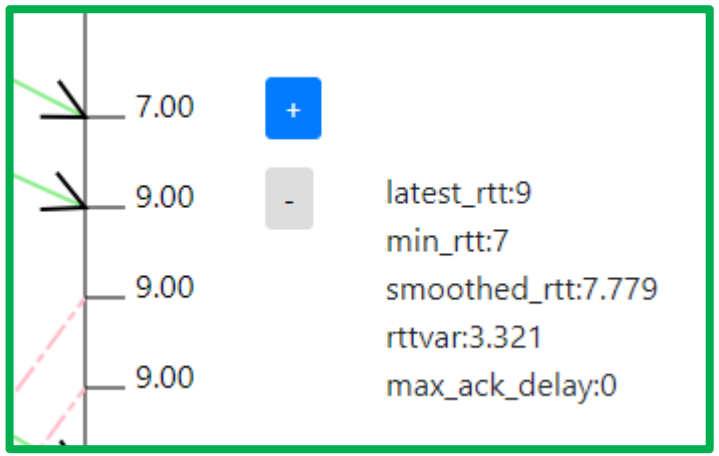
Re-ordering

UHASSELT EDM

QUIC sequence diagram

- Client + Server logs
  - Exact latency
    - Flight + processing!
  - Many extra goodies

Loss



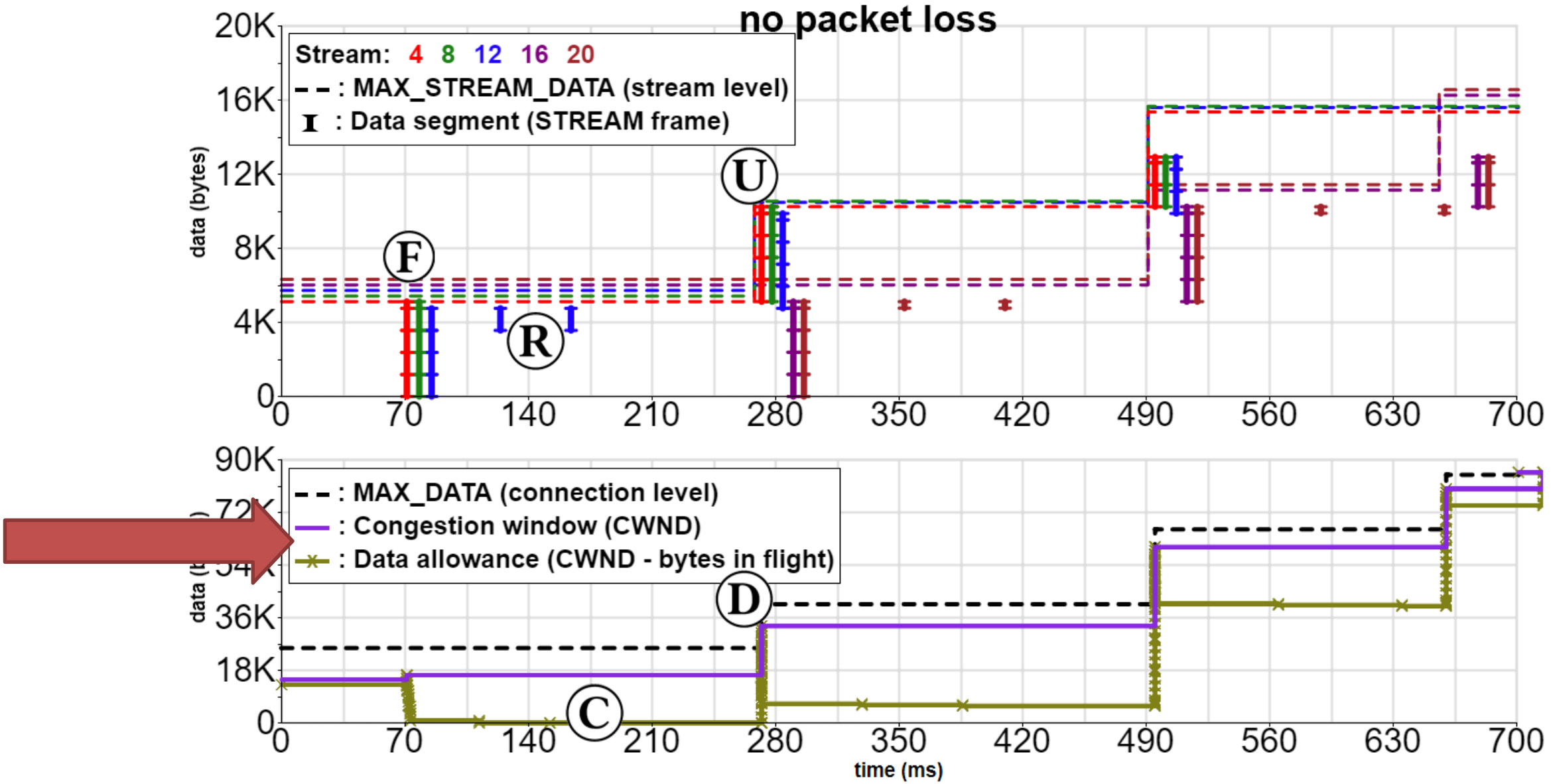
RTT estimates

Retransmits

- QUIC packets aren't just retransmitted
  - At least Packet Number change
  - At worst: complete re-shuffle

? What data is resent when and why?

QUIC Flow and congestion control logic diagram



https://uclouvain.be/en/research-institutes/icteam/ingi/news/the-superpowers-of-ebpf-in-the-networking-stack.html

Observations

- Combining multiple vantage points has major benefits
  - Not just client + server, but also in-network
  - Comparing similar logs is also interesting

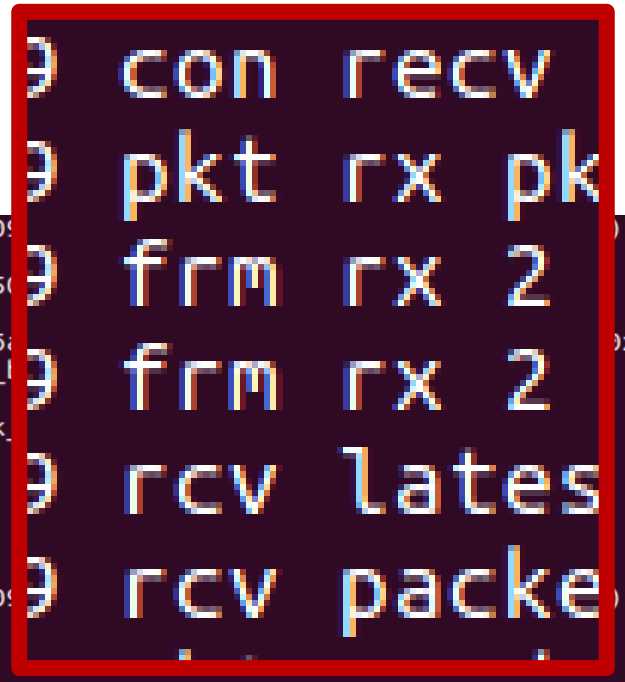
- Endpoint logs contain much additional information
  - But: none of them look the same…

Observations

"For an organization dedicated to standardization, their output logs are weirdly inconsistent"

- My poor bachelor student

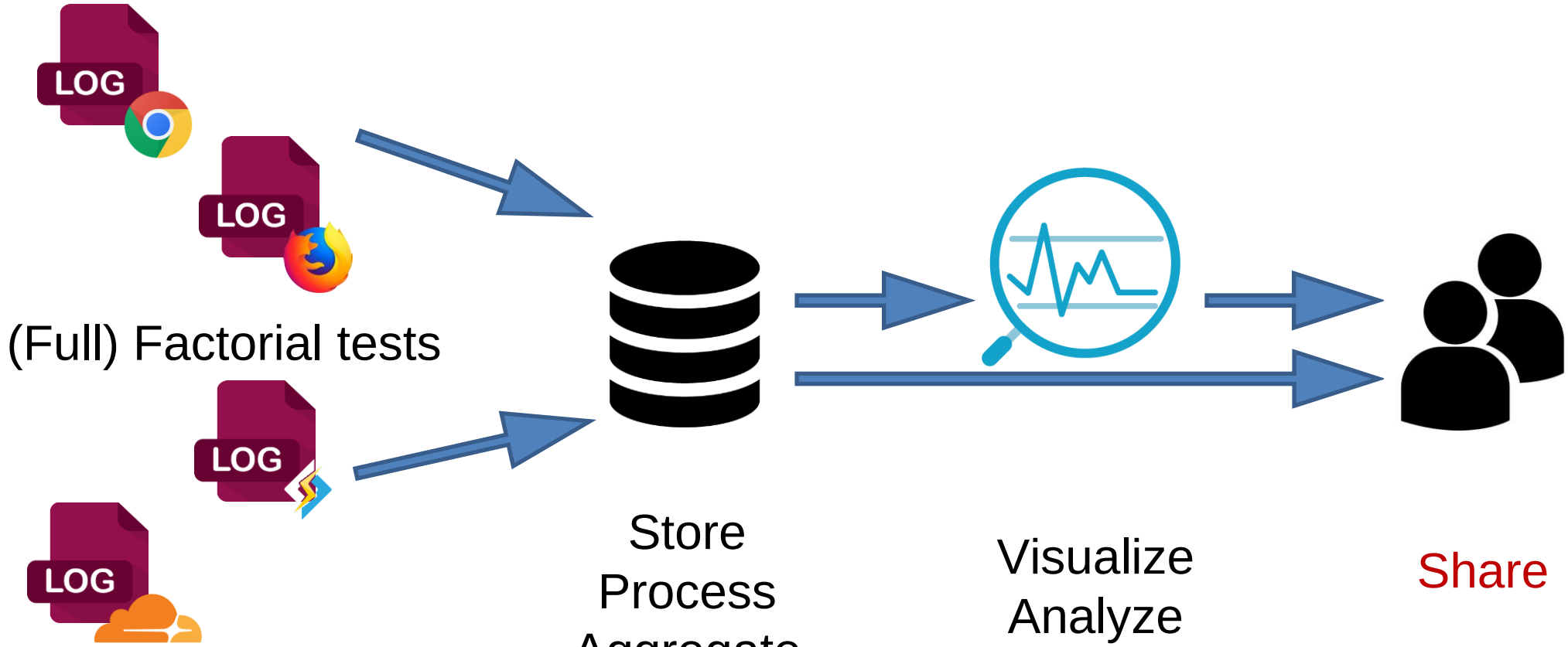
QUIC logging: The Wild Wild West




```
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
```

The plot thickens!

- How about instead: single, standardized schema?
- Both format and content



Format

Format type	Example
CSV	Syslog, SIFTR, Common Log Format, QUIC logs
JSON(-schema)	REST, GraphQL, NetLog
(semantic) XML	SOAP, WSDM, HTML
Binary	Protocol buffers, Apache thrift, pcap

"It doesn't really matter"

- Last person in the room

Two main types of logging

- **Summary**
- Poll-based

```
"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,
  }
},
```

- HTTP/2 de-facto standard "debug state" format

- **Events / Packets**
- Stream-based

```
t=181562 HTTP2_SESSION_STREAM_STATE_CHANGE
        --> stream_id = 3
        --> state = OPEN
t=181562 HTTP2_SESSION_UPDATE_RECV_WINDOW
        --> delta = 1
        --> window_size = 15728640
t=181562 HTTP2_SESSION_RECV_DATA
        --> fin = false
        --> size = 1398
        --> stream_id = 3
t=181585 HTTP2_SESSION_UPDATE_RECV_WINDOW
        --> delta = -1398
        --> window_size = 15727242
t=181585 HTTP2_SESSION_RECV_DATA
        --> fin = true
        --> size = 5981
        --> stream_id = 3
t=181585 HTTP2_SESSION_STREAM_STATE_CHANGE
        --> stream_id = 3
        --> state = CLOSE
```

- Chromium NetLog
- Quic-trace

chrome://net-export
https://netlog-viewer.appspot.com
https://github.com/google/quic-trace
https://tools.ietf.org/html/draft-benfield-http2-debug-state-01

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

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”

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

Trigger, reason, cause, … what's in a name?

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

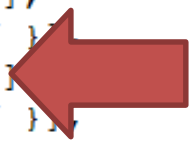
qlog : event-based to the extreme

```
[54, TRANSPORT, PACKET_TX,          LINE, {type:Initial,   packet_number:0, connID: 091b12835fe69fbe68 }],
[64, TRANSPORT, PACKET_RX,          LINE, {type:Initial,   packet_number:0, connID: 0a0fffb31d57473197ff }],
[74, TRANSPORT, PACKET_TX,          LINE, {type:Initial,   packet_number:1, connID: 091b12835fe69fbe68 }],
[84, TRANSPORT, PACKET_RX,          LINE, {type:Handshake, packet_number:0, connID: 0a0fffb31d57473197ff }],
[94, TRANSPORT, PACKET_TX,          LINE, {type:Handshake, packet_number:0, connID: 0a0fffb31d57473197ff }],
```

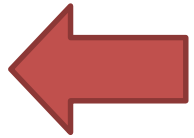
Something is wrong here, can you spot it?

qlog : event-based to the extreme

```
[54, TRANSPORT, PACKET_TX,              LINE, {type:Initial,   packet_number:0, connID: 091b12835fe69fbe68 }],
[64, TRANSPORT, PACKET_RX,              LINE, {type:Initial,   packet_number:0, connID: 0a0fffb31d57473197ff }],
[74, TRANSPORT, PACKET_TX,              LINE, {type:Initial,   packet_number:1, connID: 091b12835fe69fbe68 }],
[84, TRANSPORT, PACKET_RX,              LINE, {type:Handshake, packet_number:0, connID: 0a0fffb31d57473197ff }],
[94, TRANSPORT, PACKET_TX,              LINE, {type:Handshake, packet_number:0, connID: 0a0fffb31d57473197ff }],
```



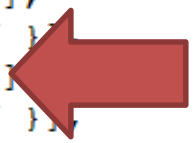
```
[54, TRANSPORT, PACKET_TX,              LINE,      {type:Initial,   packet_number:0 }],
[64, TRANSPORT, PACKET_RX,              LINE,      {type:Initial,   packet_number:0 }],
[74, TRANSPORT, PACKET_TX,              LINE,      {type:Initial,   packet_number:1 }],
[84, TRANSPORT, PACKET_RX,              LINE,      {type:Handshake, packet_number:0 }],
[84, TRANSPORT, CONNECTION_ID_UPDATE, PACKET_RX, {old:091b12835fe69fbe68, new:0a0fffb31d57473197ff}],
[94, TRANSPORT, PACKET_TX,              LINE,      {type:Handshake, packet_number:0 }],
```



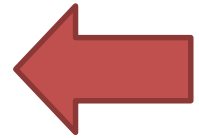
TX should use the updated connID immediately not wait until Handshake

qlog : event-based to the extreme

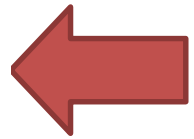
```
[54, TRANSPORT, PACKET_TX,              LINE, {type:Initial,   packet_number:0, connID: 091b12835fe69fbe68 }],
[64, TRANSPORT, PACKET_RX,              LINE, {type:Initial,   packet_number:0, connID: 0a0fffb31d57473197ff }],
[74, TRANSPORT, PACKET_TX,              LINE, {type:Initial,   packet_number:1, connID: 091b12835fe69fbe68 }],
[84, TRANSPORT, PACKET_RX,              LINE, {type:Handshake, packet_number:0, connID: 0a0fffb31d57473197ff }],
[94, TRANSPORT, PACKET_TX,              LINE, {type:Handshake, packet_number:0, connID: 0a0fffb31d57473197ff }],
```



```
[54, TRANSPORT, PACKET_TX,              LINE,      {type:Initial,   packet_number:0 }],
[64, TRANSPORT, PACKET_RX,              LINE,      {type:Initial,   packet_number:0 }],
[74, TRANSPORT, PACKET_TX,              LINE,      {type:Initial,   packet_number:1 }],
[84, TRANSPORT, PACKET_RX,              LINE,      {type:Handshake, packet_number:0 }],
[84, TRANSPORT, CONNECTION_ID_UPDATE, PACKET_RX, {old:091b12835fe69fbe68, new:0a0fffb31d57473197ff}],
[94, TRANSPORT, PACKET_TX,              LINE,      {type:Handshake, packet_number:0 }],
```



```
[54, TRANSPORT, PACKET_TX,              LINE,      {type:Initial,   packet_number:0 }],
[64, TRANSPORT, PACKET_RX,              LINE,      {type:Initial,   packet_number:0 }],
[64, TRANSPORT, CONNECTION_ID_UPDATE, PACKET_RX, {old:091b12835fe69fbe68, new: 0a0fffb31d57473197ff}],
[74, TRANSPORT, PACKET_TX,              LINE,      {type:Initial,   packet_number:1 }],
[84, TRANSPORT, PACKET_RX,              LINE,      {type:Handshake, packet_number:0 }],
[94, TRANSPORT, PACKET_TX,              LINE,      {type:Handshake, packet_number:0 }],
```



Separate “on change” events are much easier to spot and reason about

qlog : event-based pitfalls

```
[54, TRANSPORT, STREAM_FRAME_NEW,           {stream_id: 2, offset: 15, size: 1234, ... }],
[54, TRANSPORT, STREAM_FRAME_NEW,           {stream_id: 6, offset: 200, size: 1001, ... }],
[54, TRANSPORT, ACK_FRAME_NEW,              {ranges: [[0,5],[6,10]], ... }],
[54, TRANSPORT, MAX_STREAM_DATA_FRAME_NEW, {stream_id: 7, max: 16384, ... }],

...

[84, TRANSPORT, PACKET_TX,                  {type: 1RTT, size: 1300 }],
[84, TRANSPORT, PACKET_TX,                  {type: 1RTT, size: 1056 }],
```

Which frames are in which packet?

qlog : flexibility

- Trivial to add new event types per-implementation

```
[94, INTERNAL,  BULK_MALLOC, STREAM_OPENED, {start_address: 0xdeadbeef, size: 123456}],
[94, INTERNAL,  FIXME,       PACKET_RX,     {message: "We should check for duplicate packet numbers"}],
```

- Trivial to leave out information
  - On generation, on read, on transform, …
- Easy to combine information
  - E.g., aggregate: TCP from eBPF, HTTP from app-space

- Tools should be built with flexibility in mind
  - Clearly indicate which data they expect + validate on load
  - Possibly provide heuristic-based fallbacks if wanted
  - Combine full logs with partial logs (e.g., endpoint qlog + pcap)

qlog : where to get the logs?

- Easy to access + aggregate vantage points
  - https://example.com/.well-known/h2/state
  - https://example.com/.well-known/h3/state (this connection)
  - https://example.com/.well-known/h3/state/{connID} (other connection)
  - https://example.com/.well-known/h3/state/list (list of all connections)
  - chrome://net-internals/h3/state/{connID}
  - about:networking/h3/state/list
  - WebPageTest
    - Simply fetch server-log after test is done (vs needing to let browser do it)
    - Get browser log via devtools integration

qlog : where to get the logs?

- Log files or event streams?
  - Event-based logging is easy to do as on-demand, live stream
    - e.g., interactive debugging
  - Perfect for integration as a web-resource!
    - Client can POST their logs, GET server logs, incrementally
      - E.g., use reporting URL
    - We get 1, all-encompassing log file as output

qlog : easy access means need for security

- Secure to access
  - /h3/state/{connID}?token=53CR3T
    - Server config file
    - Passed as QUIC transport parameter?
  - Disable logging of sensitive info
    - Only congestion info, no packet contents, keys, …
    - Interesting for live deployments
  - Encrypt logs themselves
    - If attacker obtains logs, cannot access

Summary: Robin's logging best practices

- Event-based
  - VS summary-based

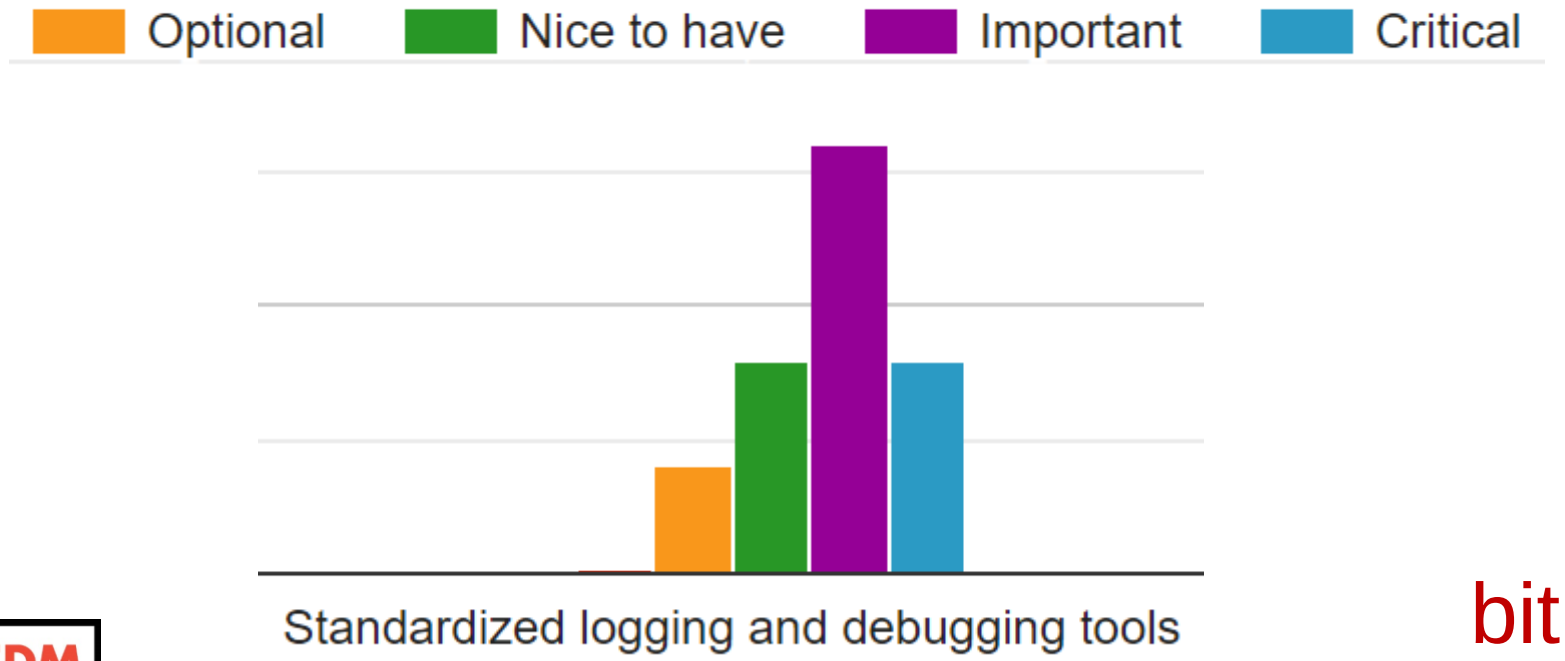
Usually, we stop here

  - Triggers / reasons
- Multiple vantage points
  - Combine-able logs
  - No need for same initial format: transformers can help
- Flexible
  - Easy to combine, filter, extend
  - Tools have to be built with this in mind
- Accessible
  - Easy to fetch automatically, preferably as a stream
  - Secure by default

My opinions

- QUIC + HTTP/3 implementers are in for a world of pain
  - Most just don’t realize it yet

- Will take longer than needed to solve initial deployment issues
  - Focus on tooling and logging from the beginning could have helped prevent this



bit.ly/quicsurvey

My opinions

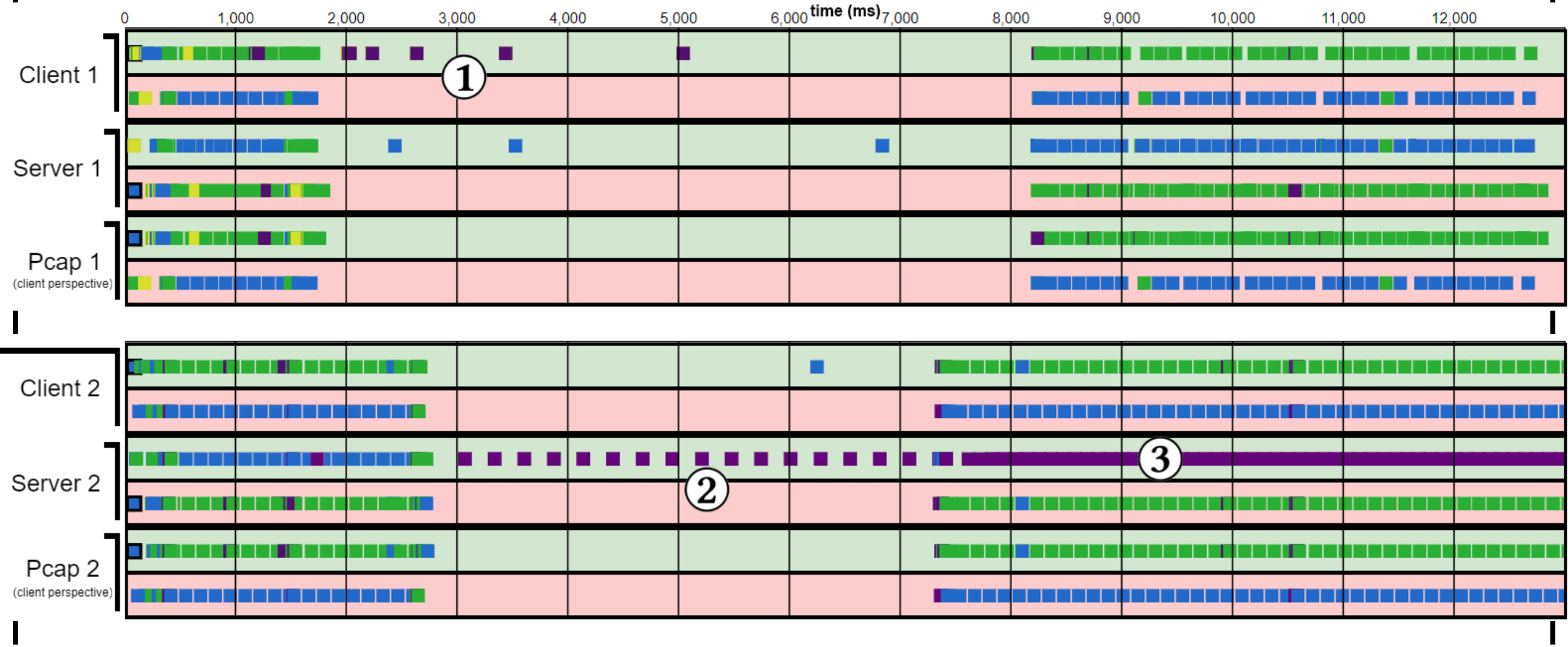
- IETF can/SHOULD make recommendations for:
  - Methods of access and their security
  - Basic concepts and applicability of logging aspects
  - Default (high-level) schema (e.g., basis of qlog)
- Skeptical about defining full schema in 1 go
  - Will need per-protocol changes anyway, don't want to go ~IANA route
- More workable: "new proposals SHOULD include logging early on"
  - Define logging approach based on IETF recommendations
  - Potentially requiring default (high-level) schema or at least transformations
  - Implementations SHOULD follow this schema (or write transformations)

What do **YOU** think?

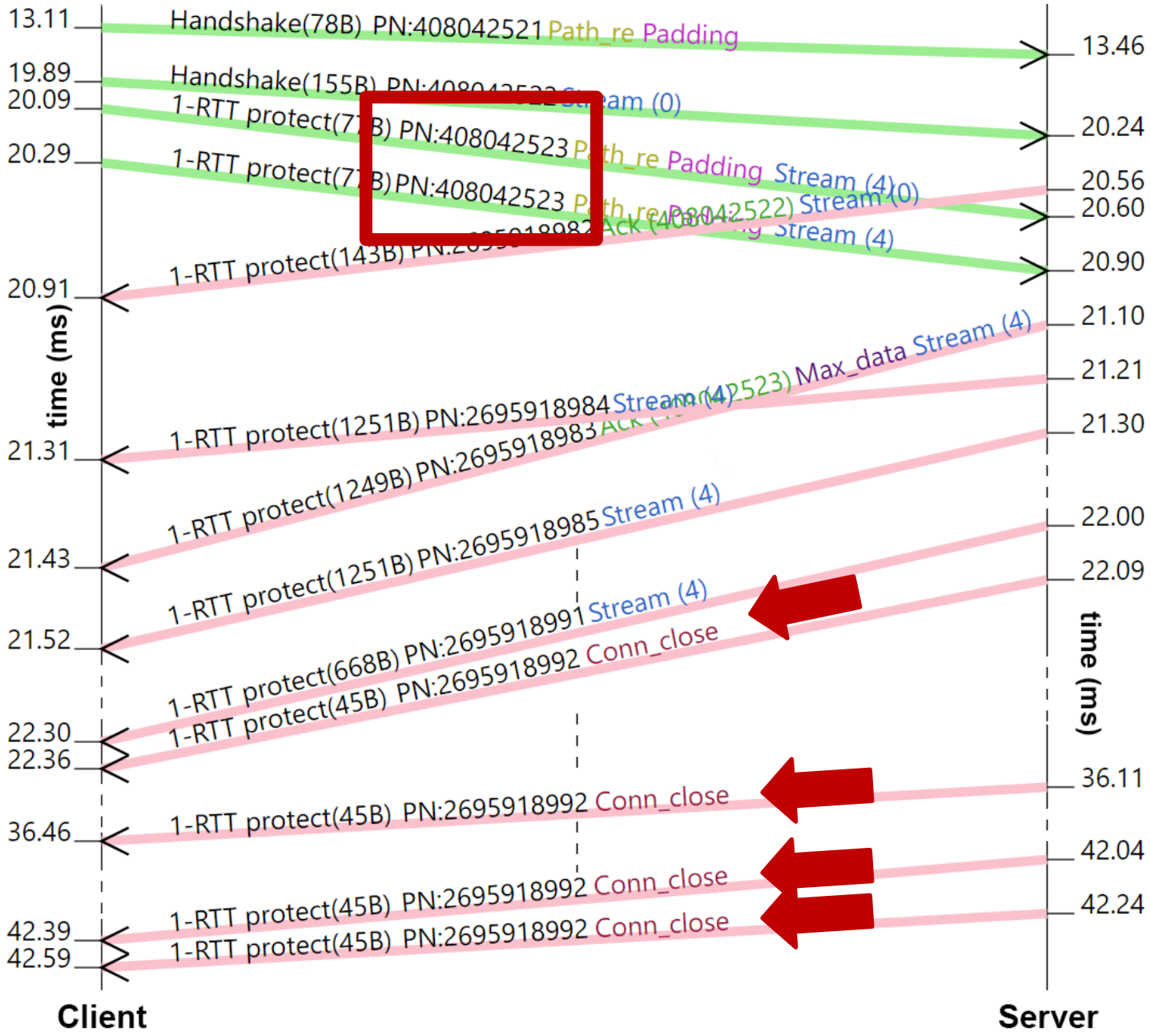
QUIC visualization: bug/behaviour examples

Extra slides / potential question support

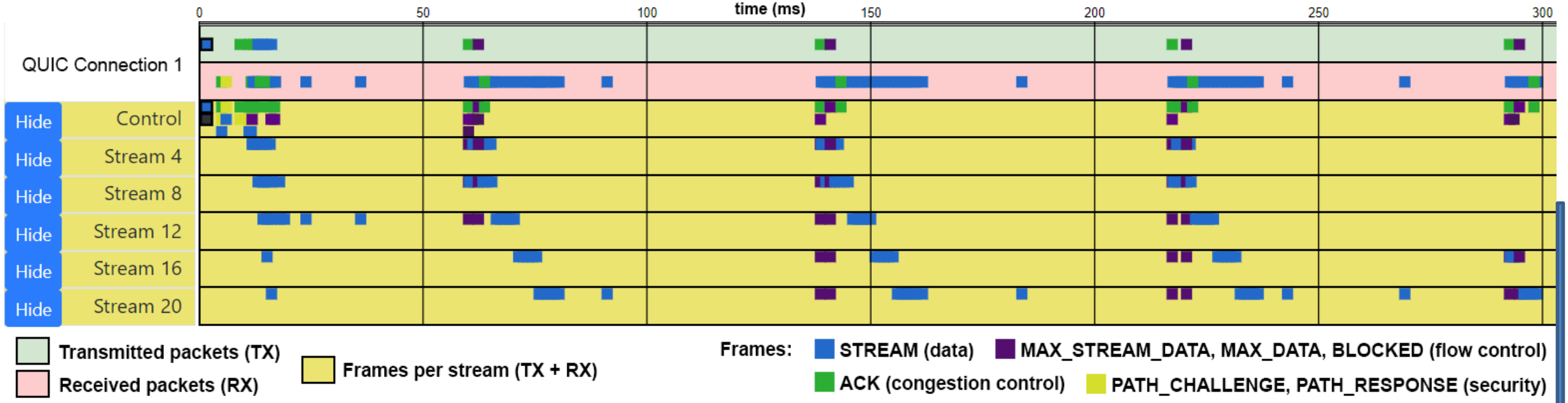
QUICvis examples : connectivity lost

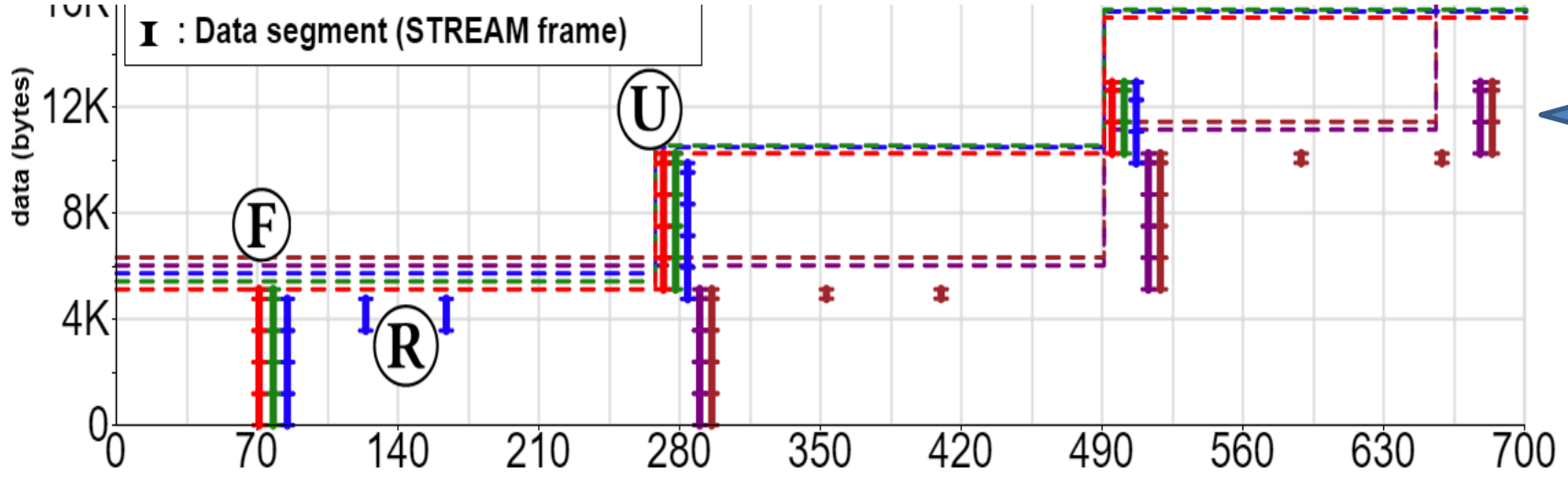


QUICvis examples : Duplicate packet nr

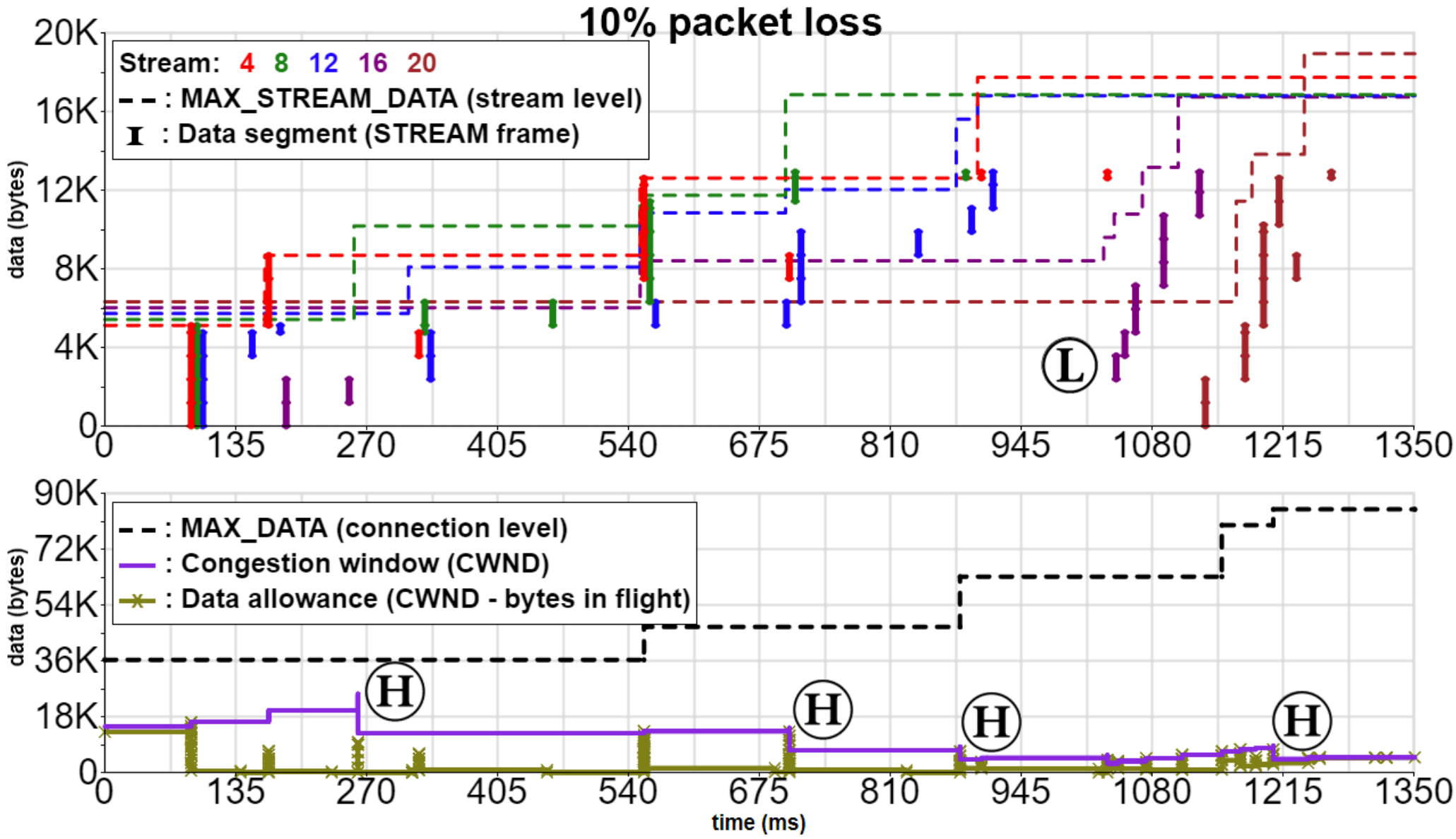


QUICvis : Flow and congestion control logic





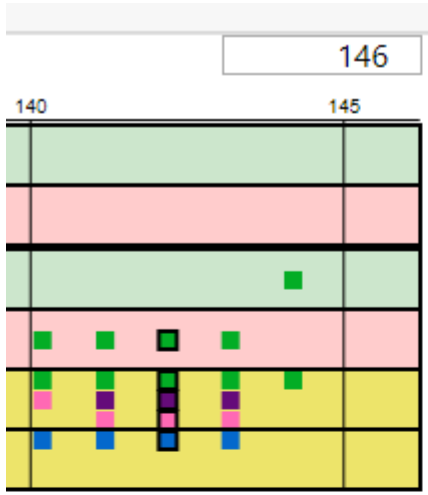
QUICvis : Flow and congestion control logic



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

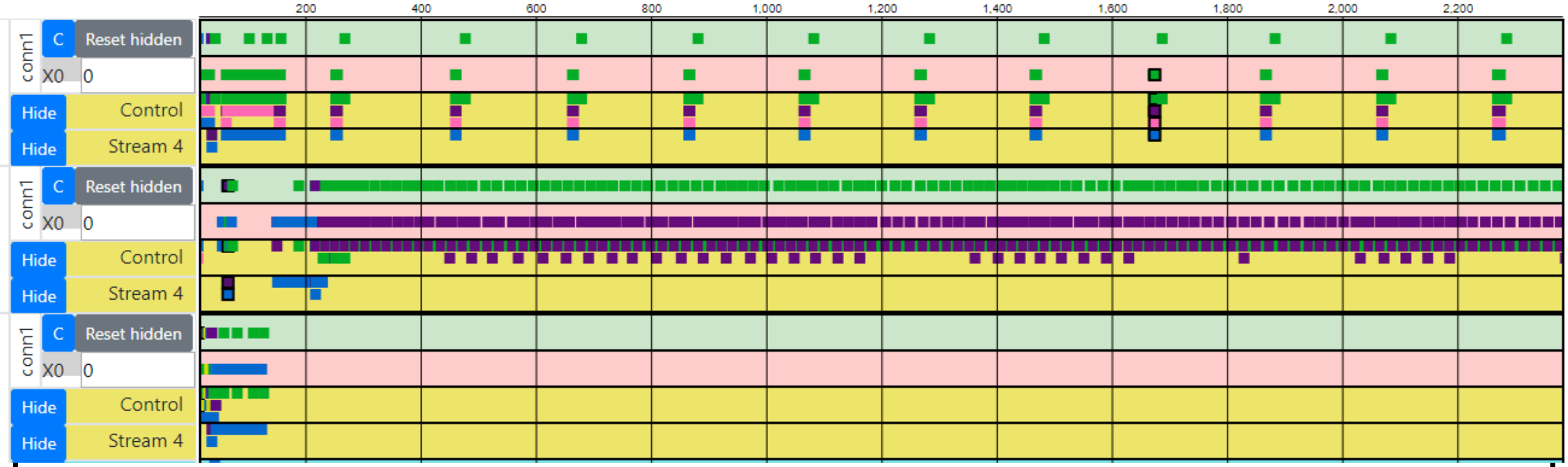
Client sends erroneous flow control allowances



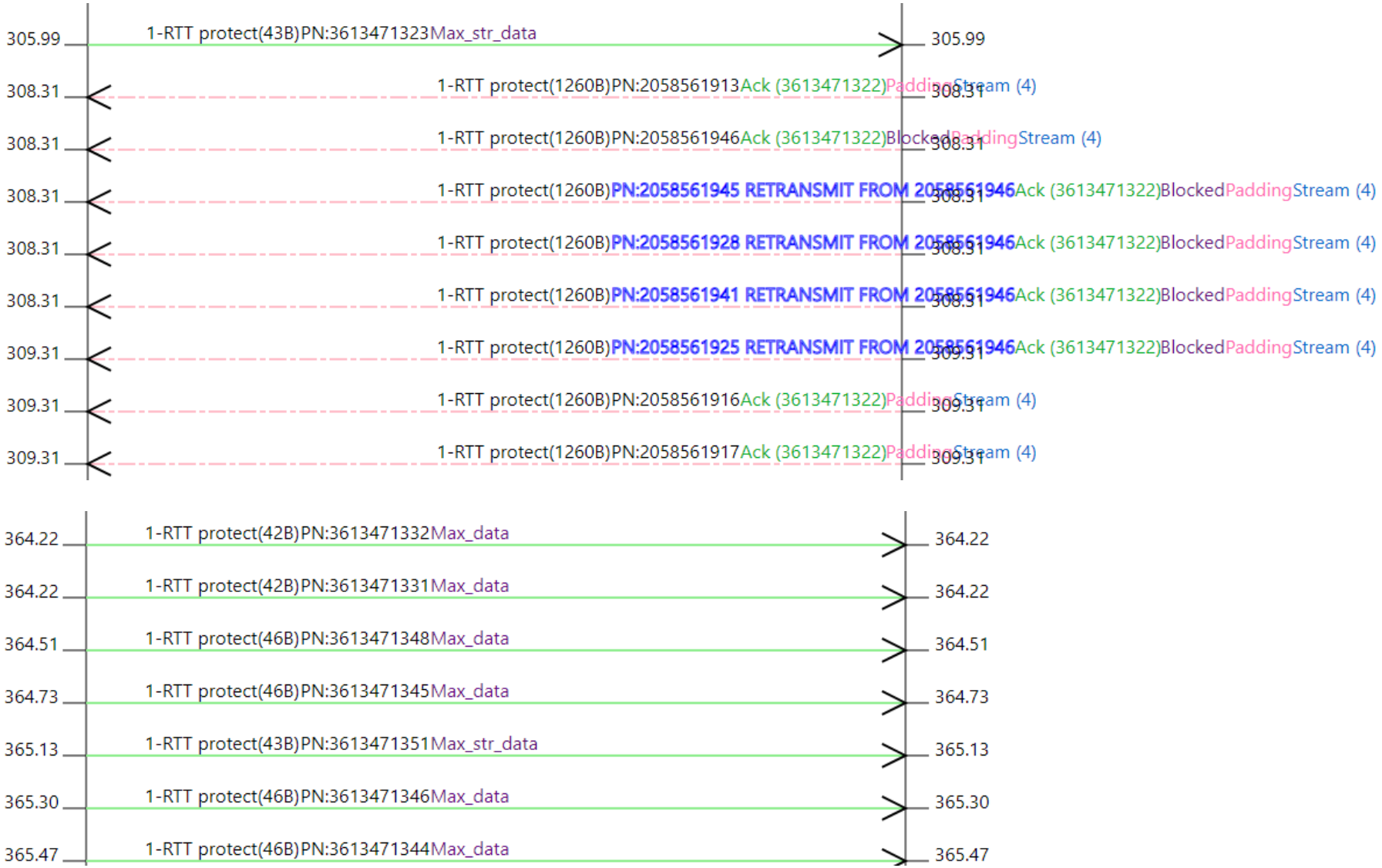
Server sends BLOCKED, accompanied by STREAM, going over the max_data

- 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

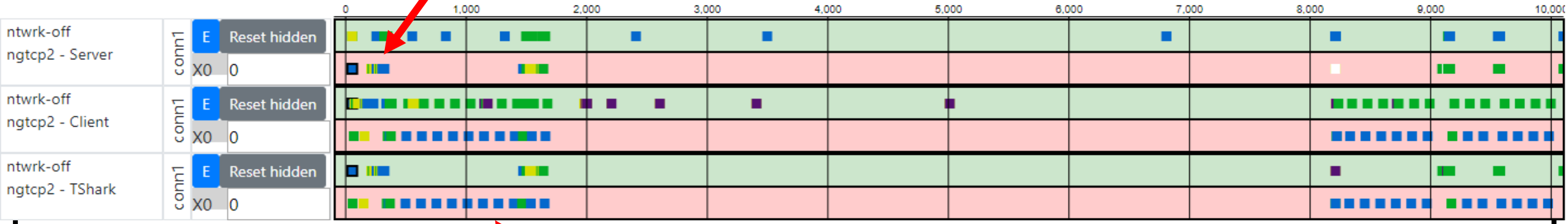


Server retransmits too much, client answers to each blocked



Pacing (network, not server)

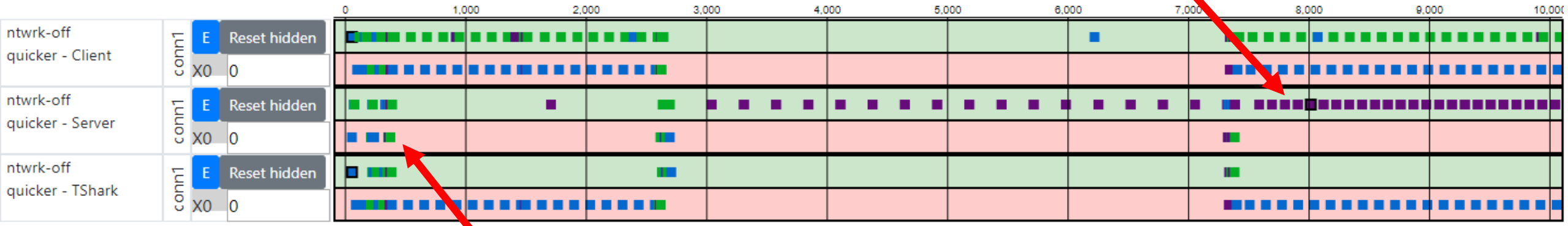
Server sends all at once



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

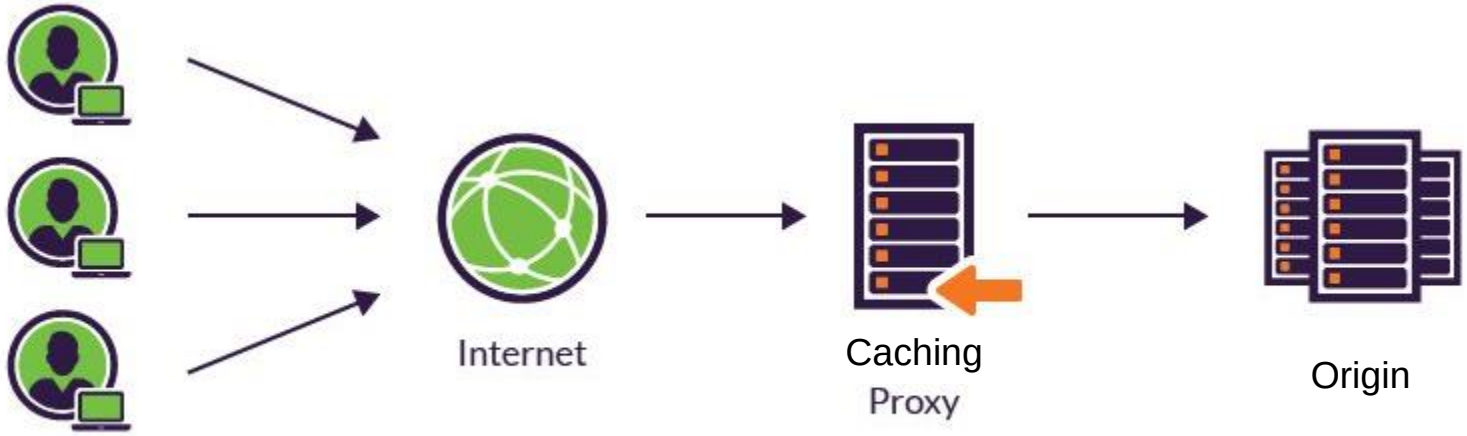
- Many people will be looking into the behavior
  - Initial implementations + conformance testing (current stage)
  - Early and at-scale deployments
  - Academic research (and teaching!)

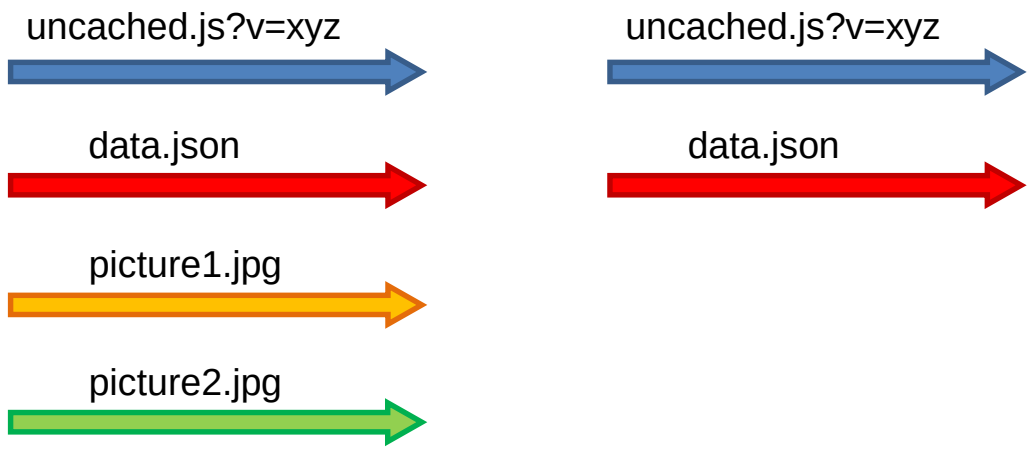
  - Cycle starts over with new features in v2

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

Broader view

- Individual tools seem to focus on 1 part of the protocol stack
- Things like H2 and especially QUIC span multiple layers
  - Cross-layer interactions can lead to difficult to debug issues
  - Looking at separate logs for the same events can be difficult
  - Really want everything in 1 go, preferably on the two endpoints (and potentially network!) at the same time

Let’s hear it

- Not even sure this should be an IETF standard for QUIC
- Let alone for something wider

- However
  - If we continue in this vein, will become more and more important over time
  - Maybe something like requiring debugging/logging to be part of any new standard? Possibly in a less-strict way though?

- What do people think?