Logging, Tooling and Debugging for Modern Network Protocols

Robin Marx - @programmingart
(Hasselt University – Belgium)
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

https://quic.edm.uhasselt.be/
Let me tell you a story

HTTP/2 Inside: multiplexing

Server

HTTP response 1
HTTP response 2
HTTP response 3

Client

HTTP/2
Single TCP connection

Image: https://www.nginx.com/blog/http2-module-nginx/
Let me tell you a story

HTTP/2 Inside: multiplexing

Image: https://www.nginx.com/blog/http2-module-nginx/
HTTP/2 Prioritization: difficult to debug

```
<table>
<thead>
<tr>
<th>Lo.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
<td>1448187813.447864</td>
<td>199.16.156.38</td>
<td>192.168.178.21</td>
<td>HTTP2</td>
<td>110</td>
<td>SETTINGS</td>
</tr>
<tr>
<td>67</td>
<td>1448187813.827974</td>
<td>192.168.178.21</td>
<td>199.16.156.38</td>
<td>HTTP2</td>
<td>223</td>
<td>Magic, SETTINGS, WINDOW_UPDATE</td>
</tr>
<tr>
<td>68</td>
<td>1448187813.828311</td>
<td>192.168.178.21</td>
<td>199.16.156.38</td>
<td>HTTP2</td>
<td>321</td>
<td>HEADERS, WINDOW_UPDATE</td>
</tr>
<tr>
<td>69</td>
<td>1448187813.830694</td>
<td>192.168.178.21</td>
<td>199.16.156.38</td>
<td>HTTP2</td>
<td>104</td>
<td>SETTINGS</td>
</tr>
<tr>
<td>71</td>
<td>1448187813.934478</td>
<td>199.16.156.38</td>
<td>192.168.178.21</td>
<td>HTTP2</td>
<td>104</td>
<td>SETTINGS</td>
</tr>
<tr>
<td>73</td>
<td>1448187813.934734</td>
<td>199.16.156.38</td>
<td>192.168.178.21</td>
<td>HTTP2</td>
<td>108</td>
<td>RST_STREAM</td>
</tr>
<tr>
<td>74</td>
<td>1448187813.934755</td>
<td>199.16.156.38</td>
<td>192.168.178.21</td>
<td>HTTP2</td>
<td>234</td>
<td>RST_STREAM</td>
</tr>
<tr>
<td>78</td>
<td>1448187813.952647</td>
<td>199.16.156.38</td>
<td>192.168.178.21</td>
<td>HTTP2</td>
<td>839</td>
<td>HEADERS</td>
</tr>
<tr>
<td>85</td>
<td>1448187813.952942</td>
<td>199.16.156.38</td>
<td>192.168.178.21</td>
<td>HTTP2</td>
<td>68</td>
<td>DATA</td>
</tr>
<tr>
<td>98</td>
<td>1448187814.042720</td>
<td>199.16.156.38</td>
<td>192.168.178.21</td>
<td>HTTP2</td>
<td>1506</td>
<td>DATA</td>
</tr>
<tr>
<td>102</td>
<td>1448187814.043304</td>
<td>192.168.178.21</td>
<td>199.16.156.38</td>
<td>HTTP2</td>
<td>428</td>
<td>HEADERS, WINDOW_UPDATE</td>
</tr>
<tr>
<td>103</td>
<td>1448187814.058270</td>
<td>199.16.156.38</td>
<td>192.168.178.21</td>
<td>HTTP2</td>
<td>350</td>
<td>DATA</td>
</tr>
<tr>
<td>116</td>
<td>1448187814.238252</td>
<td>199.16.156.38</td>
<td>192.168.178.21</td>
<td>HTTP2</td>
<td>227</td>
<td>HEADERS</td>
</tr>
<tr>
<td>118</td>
<td>1448187814.239680</td>
<td>199.16.156.38</td>
<td>192.168.178.21</td>
<td>HTTP2</td>
<td>1384</td>
<td>DATA</td>
</tr>
<tr>
<td>381</td>
<td>1448187814.551327</td>
<td>192.168.178.21</td>
<td>199.16.156.38</td>
<td>HTTP2</td>
<td>223</td>
<td>Magic, SETTINGS, WINDOW_UPDATE</td>
</tr>
<tr>
<td>382</td>
<td>1448187814.551352</td>
<td>192.168.178.21</td>
<td>199.16.156.38</td>
<td>HTTP2</td>
<td>330</td>
<td>HEADERS, WINDOW_UPDATE</td>
</tr>
<tr>
<td>384</td>
<td>1448187814.551436</td>
<td>192.168.178.21</td>
<td>199.16.156.38</td>
<td>HTTP2</td>
<td>1411</td>
<td>DATA</td>
</tr>
<tr>
<td>385</td>
<td>1448187814.551517</td>
<td>192.168.178.21</td>
<td>199.16.156.38</td>
<td>HTTP2</td>
<td>182</td>
<td>HEADERS, WINDOW_UPDATE</td>
</tr>
<tr>
<td>759</td>
<td>1448187814.637563</td>
<td>192.168.178.21</td>
<td>199.16.156.38</td>
<td>HTTP2</td>
<td>168</td>
<td>HEADERS, WINDOW_UPDATE</td>
</tr>
<tr>
<td>763</td>
<td>1448187814.658267</td>
<td>199.16.156.38</td>
<td>192.168.178.21</td>
<td>HTTP2</td>
<td>110</td>
<td>SETTINGS</td>
</tr>
<tr>
<td>765</td>
<td>1448187814.659252</td>
<td>199.16.156.38</td>
<td>192.168.178.21</td>
<td>HTTP2</td>
<td>104</td>
<td>SETTINGS</td>
</tr>
</tbody>
</table>
```
HTTP/2 Prioritization: is clearer when visualized

https://github.com/rmarx/h2vis
HTTP/2 Prioritization: can be quite complex

https://speeder.edm.uhasselt.be/www18/
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

webpagetest.org
HTTP/2 Prioritization issues by example

<table>
<thead>
<tr>
<th>CDN / Hosting</th>
<th>Status</th>
<th>Test Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pass</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>Pass</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>Pass</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td>Dec 22, 2018</td>
</tr>
<tr>
<td></td>
<td>PASS</td>
<td>Jan 1, 2019</td>
</tr>
</tbody>
</table>

9 / 34 deployments pass

Unnamed CDN

H2O server

Firefox

andydavies.me/h2priorities/

0.0

0.0
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
QUIC sequence diagram

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

- **RTT estimates**

- **Loss**

- **Retransmits**

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

? What data is resent when and why?
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
The plot thickens!

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

<table>
<thead>
<tr>
<th>Format type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSV</td>
<td>Syslog, SIFTR, Common Log Format, QUIC logs</td>
</tr>
<tr>
<td>JSON(-schema)</td>
<td>REST, GraphQL, NetLog</td>
</tr>
<tr>
<td>(semantic) XML</td>
<td>SOAP, WSDM, HTML</td>
</tr>
<tr>
<td>Binary</td>
<td>Protocol buffers, Apache thrift, pcap</td>
</tr>
</tbody>
</table>

“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,
  }
},
```

- **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
```

- HTTP/2 de-facto standard "debug state" format
- Chromium NetLog
- Quic-trace

chrome://net-export
https://netlog-viewer.appspot.com
https://github.com/google/quic-trace
Our proposal: qlog

```json
{
"connectionid": "0x763f8eaf61aa3ffe84270c0644b4dbd2b0d", "starttime": 1543917600,
"fields":
  "time", "category", "type", "trigger", "data",
"events": [
  [50, "TLS", "RTT_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", "ACK", "PACKET_RX", {"nr": 67, "cwnd": 14600, ...}],
  [250, "TRANSPORT", "ACK", "PACKET_TX", {"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)

```
1 {"connectionid": "0x763f8eaf61aa3ffe84270c0644bd0d2b0d", "starttime": 1543917600,
2   "fields": [
3     ["time", "category", "type", "trigger", "data"],
4   ]
5   "events": [
6     [50, "TLS", "0RTT_KEY", "PACKET_RX", {"key": ...}],
7     [51, "HTTP", "STREAM_OPEN", "PUSH", {"id": 0, "headers": ...}],
8   ...]
9     [200, "TRANSPORT", "PACKET_RX", "STREAM", {"nr": 50, "contents": "GET /ping.html"},
10    [201, "HTTP", "STREAM_OPEN", "GET", {"id": 16, "headers": ...}],
11    [201, "TRANSPORT", "PACKET_RX", "GET", {"id": 16, "contents": "pong"}],
12    [201, "TRANSPORT", "PACKET_RX", "GET", {"id": 16, "headers": ...}],
13    [203, "RECOVERY", "PACKET_RX", "ACK_NEW", {"nr": 67, "frames": [16, ...]},
14    [204, "ACK_NEW", "PACKET_RX", "ACK_NEW", {"nr": 51, "acked": 60}],
15    [252, "TRANSPORT", "PACKET_TX", "CWND_UPDATE", {"nr": 67, "frames": [16, ...]},
16    [1001, "RECOVERY", "LOSS_DETECTED", "ACK_NEW", {"nr": a, "frames": ...}],
17    [2002, "RECOVERY", "PACKET_RX", "EALRY_RETRANS", {"nr": x, "frames": ...}],
18    [3003, "RECOVERY", "PACKET_RX", "TAIL_LOSS_PROBE", {"nr": y, "frames": ...}],
19    [4004, "RECOVERY", "PACKET_RX", "TIMEOUT", {"nr": z, "frames": ...}]
20   ]
21 ]
```

“HTTP_STREAM_OPEN” VS “HTTP”, “STREAM_OPEN”
qlog: clear cause and effect

Trigger, reason, cause, ... what's in a name?
qlog : structured metadata

```
{ "connectionid": "0x763f8eaf61aa3ffe84270c0644bd2b0d", "starttime": 1543917600, 
  "fields": 
    [ 
      { "time": "TLS", "0RTT_KEY": "PACKET_RX", "PUSH", 
      { "id": 50, "contents": "GET /ping.html", "headers": ... }, 
      { "id": 16, "contents": "pong", "headers": ... }, 
      { "id": 67, "frames": [16, ...], "cwnd": 14600, ... }, 
      { "id": 67, "frames": [16, ...], "cwnd": 20780, ... }, 
      { "id": a, "frames": ... }, 
      { "id": x, "frames": ... }, 
      { "id": y, "frames": ... }, 
      { "id": z, "frames": ... ] 
      [51, "HTTP", "STREAM_OPEN", "PASS", 
      { "id": 50, "contents": "GET /ping.html", "headers": ... }, 
      { "id": 16, "contents": "pong", "headers": ... }, 
      { "id": 67, "frames": [16, ...], "cwnd": 14600, ... }, 
      { "id": 67, "frames": [16, ...], "cwnd": 20780, ... }, 
      { "id": a, "frames": ... }, 
      { "id": x, "frames": ... }, 
      { "id": y, "frames": ... }, 
      { "id": z, "frames": ... ] 
      [51, "HTTP", "STREAM_OPEN", "PASS", 
      { "id": 50, "contents": "GET /ping.html", "headers": ... }, 
      { "id": 16, "contents": "pong", "headers": ... }, 
      { "id": 67, "frames": [16, ...], "cwnd": 14600, ... }, 
      { "id": 67, "frames": [16, ...], "cwnd": 20780, ... }, 
      { "id": a, "frames": ... }, 
      { "id": x, "frames": ... }, 
      { "id": y, "frames": ... }, 
      { "id": z, "frames": ... ] 
      [51, "HTTP", "STREAM_OPEN", "PASS", 
      { "id": 50, "contents": "GET /ping.html", "headers": ... }, 
      { "id": 16, "contents": "pong", "headers": ... }, 
      { "id": 67, "frames": [16, ...], "cwnd": 14600, ... }, 
      { "id": 67, "frames": [16, ...], "cwnd": 20780, ... }, 
      { "id": a, "frames": ... }, 
      { "id": x, "frames": ... }, 
      { "id": y, "frames": ... }, 
      { "id": z, "frames": ... ] 
      [51, "HTTP", "STREAM_OPEN", "PASS", 
      { "id": 50, "contents": "GET /ping.html", "headers": ... }, 
      { "id": 16, "contents": "pong", "headers": ... }, 
      { "id": 67, "frames": [16, ...], "cwnd": 14600, ... }, 
      { "id": 67, "frames": [16, ...], "cwnd": 20780, ... }, 
      { "id": a, "frames": ... }, 
      { "id": x, "frames": ... }, 
      { "id": y, "frames": ... }, 
      { "id": z, "frames": ... ] 
      [51, "HTTP", "STREAM_OPEN", "PASS", 
      { "id": 50, "contents": "GET /ping.html", "headers": ... }, 
      { "id": 16, "contents": "pong", "headers": ... }, 
      { "id": 67, "frames": [16, ...], "cwnd": 14600, ... }, 
      { "id": 67, "frames": [16, ...], "cwnd": 20780, ... }, 
      { "id": a, "frames": ... }, 
      { "id": x, "frames": ... }, 
      { "id": y, "frames": ... }, 
      { "id": z, "frames": ... ] 
      [51, "HTTP", "STREAM_OPEN", "PASS", 
      { "id": 50, "contents": "GET /ping.html", "headers": ... }, 
      { "id": 16, "contents": "pong", "headers": ... }, 
      { "id": 67, "frames": [16, ...], "cwnd": 14600, ... }, 
      { "id": 67, "frames": [16, ...], "cwnd": 20780, ... }, 
      { "id": a, "frames": ... }, 
      { "id": x, "frames": ... }, 
      { "id": y, "frames": ... }, 
      { "id": 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: 091b12835fe69f6e83 }],
[64, TRANSPORT, PACKET_RX, LINE, {type:Initial, packet_number:0, connID: 0a0fffb31d57473197ff }],
[74, TRANSPORT, PACKET_TX, LINE, {type:Initial, packet_number:1, connID: 091b12835fe69f6e83 }],
[64, 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?
TX should use the updated connID immediately not wait until Handshake
qlog: event-based to the extreme

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

[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}].
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
- 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

Usually, we stop here
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

[Chart showing importance of standardized logging and debugging tools]

[Link: 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

Transmitted packets (TX)
Received packets (RX)
Frames per stream (TX + RX)

Frames:
- STREAM (data)
- MAX_STREAM_DATA, MAX_DATA, BLOCKED (flow control)
- ACK (congestion control)
- PATH_CHALLENGE, PATH_RESPONSE (security)

I: Data segment (STREAM frame)
F
R
U
QUICvis: Flow and congestion control logic

10% packet loss

Stream: 4 8 12 16 20

- MAX_STREAM_DATA (stream level)
- Data segment (STREAM frame)

MAX_DATA (connection level)
- Congestion window (CWND)
- Data allowance (CWND - bytes in flight)

UHASSELT EDM
Sending data along with BLOCKED, going over the limit

Client sends erroneous flow control allowances

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

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

```json
"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
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

```
QUIC_SESSION_STREAM_FRAME_RECEIVED
  --> fin = true
  --> length = 0
  --> offset = "0"
  --> stream_id = 5
```

```
QUIC_CHROMIUM_CLIENT_STREAM_READ_RESPONSE_HEADERS
  --> :status: 304
  age: 187
  alt-svc: quic=":443"; ma=2592000; v="46,44,43,39"
  etag: "1552399307"
```

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

+ Finer grained

- High overhead
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
  - Reasons logged explicitly

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

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

Finer grained

High overhead

https://github.com/google/quic-trade
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?