Towards QUIC debuggability

Logging with qlog
Visualizing with QUICvis

Robin Marx – Jonas Reynders – Kevin Pittevils – Peter Quax – Wim Lamotte
Internet

What is HTTP/2 and is it going to speed up the web?

Biggest change to how the web works since 1999 should make browsing on desktop and mobile faster

Samuel Gibbs
Wed 18 Feb 2015 15.10 GMT

The web is about to get faster thanks to a new version of HTTP - the biggest change since 1999 to the protocol that underpins the world wide web as we know it today.

https://www.theguardian.com/technology/2015/feb/18/http2-speed-up-web-browsing-desktop-mobile
HTTP/2 Prioritization: difficult to debug

[Image of Wireshark analysis]

HTTP/2 Prioritization: is clearer when visualized

Round Robin
HTTP/2 Prioritization: can be quite complex

https://speeder.edm.uhasselt.be/www18/
HTTP/2 Prioritization in WebPageTest: uncovers quite some bugs

![Diagram of HTTP/2 Prioritization]

Firefox: 0.0
andydavies.me/h2priorities/: 0.0

https://twitter.com/AndyDavies/status/1065916677408346112
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

https://github.com/rmarx/quicker
QUIC timeline

Note: this looks like Round-Robin again, but it's not (it's HTTP/0.9)! See later
QUIC sequence diagram

- Client + Server logs
  - Exact latency
  - Flight + processing!
QUIC sequence diagram

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

RTT estimates

Re-transmits

Loss

Re-ordering
QUIC logging: The Wild Wild West

Quicker: logs TLS secrets
QUIC logging: The Wild Wild West

ngtcp2: doesn’t log transmitted packets, only frames
QUIC logging: The Wild Wild West

Quant: HTML…
QUIC logging

- Very disparate formats, but most info is there
  - Just have to write **and update** a lot of custom parsers 😞

- However:
  - Not all log fine-grained congestion + flow control logic
  - Unclear which frames from a lost packet are resent and why
    - E.g., overlapping stream frames in winquic
    - Some only log the bare minimum

- Expect this to be even worse for HTTP/3 layer
QUIC logging: standardized

- How about instead: single, standardized format?
- Easy to parse, easy to customize
QUIC logging: qlog : simple to filter

```
{
  "connectionid": "0x763f8eaf61aa3ffe84270c0644bdbeb0d",
  "starttime": 1543917600,
  "fields": [{
    "time": "category", "type", "trigger", "data"
  },
  [50, "TLS", "RRRT_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_QUEUEED", "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": ...}]
]```
QUIC logging: qlog : clear cause and effect

```
["connectionid": "0x763f8eaf61a3ffe84270c0644b2bd2b0d", "starttime": 1543917600,
 "fields":
 ["time", "category", "type", "trigger", "data"],
 "events": [
 [50, "TLC", "ORIG_KEY", "PACKET_RX",{"key": ...}],
 [51, "HTTP", "STREAM_OPEN", "PUSH",{"id": 0, "headers": ...}],
 [200, "TRANSPORT", "STREAM_OPEN", "STREAM",{"nr": 50, "contents": "GET (ping.html)",
 [201, "HTTP", "STREAM_OPEN", "GET",{"id": 16, "headers": ...}],
 [201, "TRANSPORT", "STREAM_FRAME_NEW", "PACKET_RX",{"id": 16, "contents": "pong", ...}],
 [201, "TRANSPORT", "PACKET_NEW", "PACKET_RX",{"nr": 67, "frames": [16, ...], ...}],
 [203, "RECOVERY", "PACKET_QUEUEED", "CWND_EXCEEDED",{"nr": 67, "cwnd": 14600, ...}],
 [250, "TRANSPORT", "ACK_NEW", "PACKET_RX",{"nr": 51, "ackeld": 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": ...}]
]```
QUIC logging: qlog: simple to follow

```
{
  "connectionid": "0x763f8eaf61aa3ffe84270c0644b3d2b0d", 
  "starttime": 1543917600,
  "fields":
    ["time", "category", "type", "trigger", "data"],
  "events": [
    [50, "TLS", "0RTT_KEY", "PACKET_RX", {"key": ...}],
    [51, "HTTP", "STREAM_OPEN", "PUSH", {"id": 0, "headers": ...}]
  ]
}
```
QUIC logging: standardized : various options

- Parallel work: Google quic-trace
  - Protocol buffers (vs JSON)
  - Focus on congestion control
  - Also list reasons and types

```
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;
};
```

https://github.com/google/quic-trace
QUIC logging: standardized

- Easy to access
  - https://example.com/.well-known/h2/state
QUIC logging: standardized

- Easy to access
  - 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
QUIC logging: standardized

- Easy to access
  - 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
QUIC logging: standardized

- Easy to access, 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

- Make it non-trivial to enable (by accident)
  - Sensible defaults
QUIC debugging: logging + visualizations

(Full) Factorial tests

Store
Process
Aggregate

Visualize
Analyse
Share
QUICvis examples: connectivity lost

[Diagram showing network traffic over time]
QUICvis examples:
Duplicate packet nr
QUICvis: Flow and congestion control logic

Stream: 4 8 12 16 20

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

no packet loss

MAX_DATA (connection level)
Congestion window (CWND)
Data allowance (CWND - bytes in flight)
QUICvis: Flow and congestion control logic

QUIC Connection 1

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

Transmitted packets (TX)
Received packets (RX)
Frames per stream (TX + RX)
QUICvis: Flow and congestion control logic

10% packet loss

Stream: 4 8 12 16 20

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

data (bytes)

0 4K 8K 12K 16K 20K

0 135 270 405 540 675 810 945 1080 1215 1350

time (ms)

data (bytes)

0 18K 36K 54K 72K 90K

UHASSELT EDM
Conclusion

- Combining logs from multiple viewpoints is very powerful
- Let’s think about QUIC debuggability before v1 hits the shelves
  - Instead of 3 years from now

Feedback welcome!

- QUICvis:
  - How to handle overlapping data?
  - Which use cases would be of most interest to you?
  - Which things will be hardest to debug?

- qlog
  - Interest in adding qlog to existing codebases?
  - Better options / ideas?
  - What is needed to get a (semi-) (de-facto) standardized format?
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
QUIC and HTTP/3

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

- 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