Demo: Debugging QUIC with qlog and QUICvis

Robin Marx, Peter Quax, Wim Lamotte
{firstname.lastname}@uhasselt.be

Jonas Reynders, Kevin Pittevils
{firstname.lastname}@student.uhasselt.be

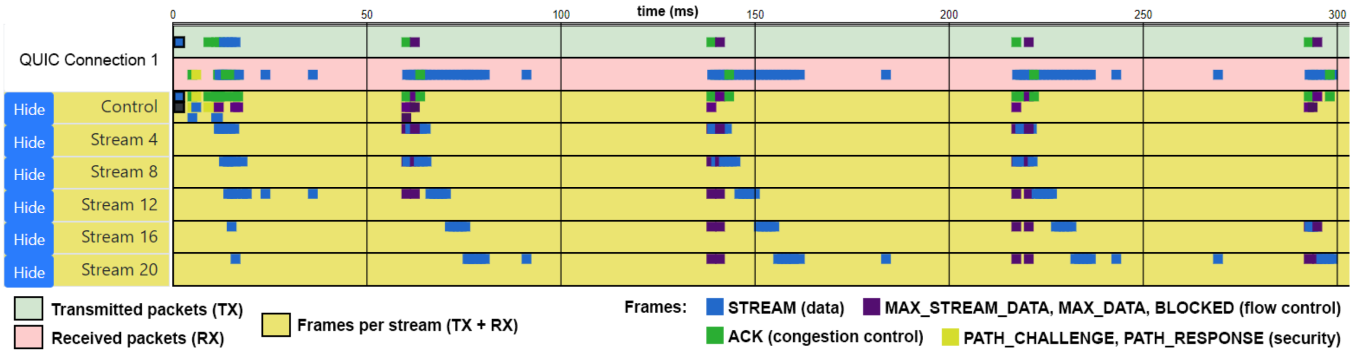

Figure 1: The QUICvis timeline-view, showing the parallel transfer of five resources.

1 INTRODUCTION AND MOTIVATION

QUIC has been called the mother of all web protocols, as it deeply integrates aspects of TCP (flow control, congestion control, loss recovery), TLS (handshake, encryption keys) and HTTP/2 (streams, prioritization) together into one cross-layer implementation, creating a new reliable transport mechanism over UDP. QUIC is highly optimized, almost entirely end-to-end encrypted and, in order to allow the protocol to evolve quickly, implementations are currently done in user-space. This, combined with QUIC's high ambitions, precludes the re-use of existing codebases, leading to new code for complex mechanisms (e.g., reliablity, congestion control, encryption logic, compression, HTTP mapping). Indeed, over 15 of such new implementations for the IETF's soon to be standardized version of QUIC have been announced (https://github.com/quicwg/base-drafts/wiki/Implementations).

With such a complex undertaking, it is highly likely that there will be various bugs and unexpected behaviours in the young QUIC implementations for quite some time to come, especially since advanced features like Multipath and Forward Error Correction are not planned until the next version. But even with flawlessly implemented codebases, much research and (live) testing will be needed to fine-tune parameters, and to determine best practices and performance characteristics.

However, QUIC's complexity precludes straightforward debugging, testing and evaluation. Tools like Wireshark are a must for decrypting and deserializing packet traces. Alternatively, one could interpret the (plain text) logging output of different QUIC implementations, but in our experience they often do not contain all necessary state information and logging formats are wildly heterogeneous across codebases. All this makes it more difficult for users to debug and understand QUIC's behaviour, and causes significant effort to be required to make more advanced interactive tools that visualize the various aspects of the protocol. This is bad, as our experience researching various HTTP/2 stacks and their behaviours, has shown that such tools can dramatically reduce debugging time and greatly aid root cause analysis of bugs and issues.

This work is our attempt to promote the topic of long-term debuggability and testability of QUIC to the larger community and start a discussion. We do this by proposing two items: an easily deployable common endpoint logging format called qlog, and a toolset of interactive visualizations called QUICvis.

2 QLOG

As existing logs and packet captures are mostly non-uniformly formatted and often miss crucial internal state information, we propose a common endpoint logging format. The main idea is to log details on almost all events and also tag them with high-level metadata such as category, event-type and trigger (i.e., the direct cause of this event). This makes it easy to perform high-level filtering and to trace event chains. qlog is also flexible: depending on the use case, security considerations or overhead allowances, runs could only log a subset of supported categories, types and data (e.g., when debugging congestion control, detailed HTTP-related information is probably superfluous). As is the case with the proposed HTTP/2 debugging format from Benfield et al., the logs could be available at fixed URLs (e.g., http://example.com/.well-known/hq/state/connid=XYZ and chrome://net-internals/#hq/id=XYZ).

Listing 1 gives a very rough idea of what such a format could look like. Note that we fully expect the exact format and types of categories, events etc. to change as our proposal is discussed and the QUIC specification evolves. We have implemented a first basic version of qlog and find it achieves its goals but also adds non-trivial implementation overhead.

Listing 1: Simplified example of the qlog format in JSON, showing a packet being queued due to congestion control

```
{"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",  "PACKET_NEW",       "EARLY_RETRANS",   {"nr": x, "frames": ...}],
    [2002,  "RECOVERY",  "PACKET_NEW",       "TAIL_LOSS_PROBE", {"nr": y, "frames": ...}],
    [3003,  "RECOVERY",  "PACKET_NEW",       "TIMEOUT",         {"nr": z, "frames": ...}]
 ]}
```

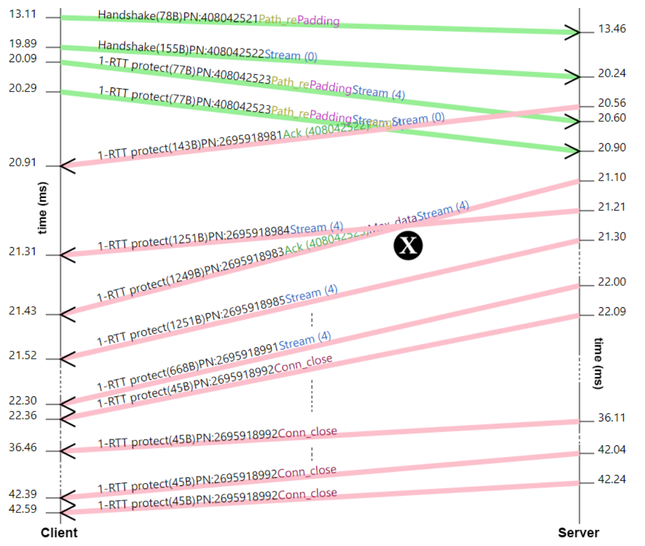
13.11
Handshake(78B)PN:408042521Path_rePadding
13.46
19.89
Handshake(155B)PN:408042522Stream (0)
20.09
1-RTT protect(77B)PN:408042523Path_rePaddingStream (4)
20.24
20.29
1-RTT protect(77B)PN:408042523Path_rePaddingStream (4)
20.56
20.60
1-RTT protect(143B)PN:2695918981Ack (408042522)Stream (0)
20.90
20.91
21.10
21.21
1-RTT protect(1251B)PN:2695918984Stream (4)
Max_dataStream (4)
21.30
21.31
1-RTT protect(1249B)PN:2695918983Ack (408042523)
X
22.00
21.43
1-RTT protect(1251B)PN:2695918985Stream (4)
22.09
21.52
1-RTT protect(668B)PN:2695918991Stream (4)
1-RTT protect(45B)PN:2695918992Conn_close
22.30
22.36
36.11
36.46
1-RTT protect(45B)PN:2695918992Conn_close
42.04
42.24
1-RTT protect(45B)PN:2695918992Conn_close
1-RTT protect(45B)PN:2695918992Conn_close
42.39
42.59
time (ms)
time (ms)
Client
Server

Figure 2: The QUICvis sequence diagram. ACK-only packets not shown for clarity.

As such, we contemplate starting work on an open source qlog library with bindings for various programming languages.

3 QUICVIS

Working directly from textual formats or packet listings can be inefficient and error-prone, especially when comparing various (long) traces. Interactive visual tools can help in debugging, as they can hide superfluous detail, allowing the user to focus on the context at hand. For instance, one tool can allow fast discovery of a problematic area in a trace, after which another tool can help in root cause analysis of that problem. In QUICvis, we have implemented three such visualizations:

- **Timeline** (Figure 1): Allows for a bird's-eye overview of the trace and also supports zooming, panning and highlighting to interactively reveal more details. Different traces can easily be compared by stacking them vertically.
- **Sequence diagram** (Figure 2): Visualizes data flow between two endpoints. When combining two logs (client and server side), we can show RTT, loss, re-transmits and reordering (crossing lines, Figure 2 Ⓧ) very accurately.
- **Congestion control/flow control graph** (Omitted due to space considerations, visible on our website below): Inspired by tcptrace's time-sequence diagram, this graph allows tracking of congestion parameters and (per-stream) flow control and data flow. Using endpoint logs, we plot internal state not gleaned from packet captures.

We have evaluated these visualizations by running conformance tests on three WIP QUIC implementations (i.e., our own Quicker codebase, Quant and Ngtcp2). Processing the results was fast and we rapidly identified various bugs (e.g., in Figure 2, a duplicate packet number occurence is erroneously answered with a CONNECTION_CLOSE, which is itself repeated in four packets having the exact same packet number). Complex behaviour such as flow control was also clearly identifiable (e.g., in Figure 1, blue data frames per stream are halted until reception of a purple MAX_STREAM_DATA update).

4 DEMO CONTENTS

Our demo will mainly showcase live versions of the QUICvis tools, primarily on previously captured traces of at least three IETF QUIC implementations containing interesting bugs or behaviour, and possibly on traces live-captured at the event. We hope to have meaningful discussions on which visualizations are most useful and also on the proper shape and usage of the qlog format. Note that, as we plan to continue development on QUICvis and qlog, there is a high chance that we will have additional visualizations implemented and demonstrable by December. Our current and future work is open source and can be assessed as web-based demos at https://quic.edm.uhasselt.be.

Besides table space, a power outlet and WLAN connectivity, an additional (HDMI) screen or projector would be appreciated for the demo.