Visualizing QUIC and HTTP/3 with qlog and qvis

Robin Marx
Hasselt University – tUL – EDM
Diepenbeek, Belgium
robin.marx@uhasselt.be

Wim Lamotte
Hasselt University – tUL – EDM
Diepenbeek, Belgium
wim.lamotte@uhasselt.be

Peter Quax
Hasselt University – tUL – Flanders Make - EDM
Diepenbeek, Belgium
peter.quax@uhasselt.be

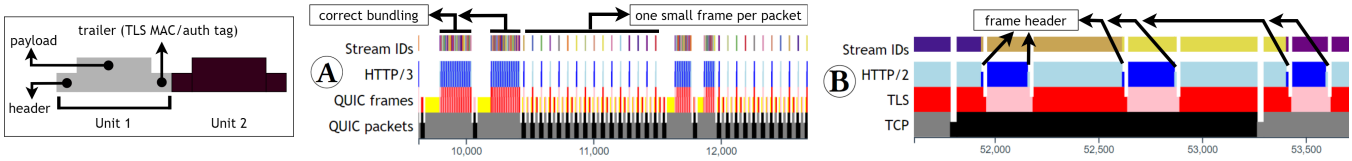

Figure 1: A QUIC+HTTP/3 and TCP+TLS+HTTP/2 trace visualized in the Packetization Diagram. The x-axis is in bytes received. Alternating colors on each row indicate the switch to a new TCP/QUIC packet, TLS record, QUIC/HTTP frame or HTTP stream. Elements that align vertically are packed into the lower layer's payload.

1 INTRODUCTION & MOTIVATION

The new QUIC and HTTP/3 (H3) protocols being finalized by the IETF are powerful but also highly complex. They combine advanced approaches from predecessors such as TCP (e.g., congestion and flow control, reliability) and HTTP/2 (H2) (e.g., stream multiplexing, prioritization), with cutting-edge features (e.g., 0-RTT data, connection migration). As QUIC runs on top of UDP, these intricate systems have to be re-implemented from scratch, often in userspace, which has turned out to be error-prone. QUIC also fully integrates TLS 1.3 and is end-to-end encrypted at the transport layer. This means that, unlike with TCP, elements like packet and acknowledgement numbers are indiscernible to passive observers in encrypted packet traces (e.g., *.pcap* files analyzed with tools like Wireshark). As such, QUIC requires (ephemeral) TLS keys for even high-level analysis, leading to scalability, privacy and security issues. It is clear that QUIC+H3's overall complexity and heavy security focus make them difficult to implement, debug, observe, analyse, use and teach in practice.

Since 2018 [1] we have been working on two intertwined projects to aid in conquering this challenge. Firstly, we proposed a paradigm shift from using packet traces to logging information directly at the endpoints instead (e.g., client and server, but also CDN nodes and load balancers). This is accomplished with *qlog*, a standard structured logging format[1]. The *qlog* specification defines how common events (e.g., packet_sent) and their metadata (e.g., header fields, packet size) can be logged using JSON, making them readable for both humans and machines. To get around the privacy and security issues, endpoints can themselves decide which event data to log and expose (e.g., being very succinct in a live deployment but more verbose during debugging), independent from what they send on the wire. Additionally, this allows endpoints to log detailed internal state typically not transferred to the peer directly (e.g., a congestion_metric_updated event logging the current congestion window and RTT estimates), which in turn helps with deep root-cause analysis. Our *qlog* approach has since found broad adoption in the QUIC community, with 12 out of 18 active QUIC implementations[2] supporting the format (including Cloudflare, Mozilla, NodeJS and especially Facebook, who log 30 billion *qlog* events in production daily).

Secondly, we have implemented *qvis*, an open-source suite of interactive web-based tools that visualize the protocols' behaviours[3]. The tools can ingest *qlog* files directly, as well as decrypted *.pcaps*, which are automatically transformed into (partial) *qlogs* (lacking the internal endpoint state). General purpose tools (like the Sequence Diagram or Statistics overview) allow users to get an initial idea of which problems can be present in a connection trace. More specialized visualizations (like the Congestion, Multiplexing and Packetization diagrams) then allow focused, in-depth analysis. The *qvis* tools have since been used by us and many others (e.g., Cloudflare[4]) to find and fix bugs and inefficiencies in the protocol implementations. Our demo will use real-world examples of high-impact bugs to demonstrate the effectiveness of our *qvis* tools and method (see §2).

It should be noted that some of the demo examples are also part of the dataset of our accepted submission to the SIGCOMM'20 EPIQ workshop[5]. However, we want to bring our approach to the attention of the wider academic SIGCOMM audience for two main reasons. Firstly, because we have recently started extending our approach to other protocols as

[1]https://github.com/quiclog/internet-drafts

[2]https://github.com/quicwg/base-drafts/wiki/Implementations
[3]https://github.com/quiclog/qvis
[4]https://blog.cloudflare.com/cubic-and-hystart-support-in-quiche
[5]https://qlog.edm.uhasselt.be/epiq

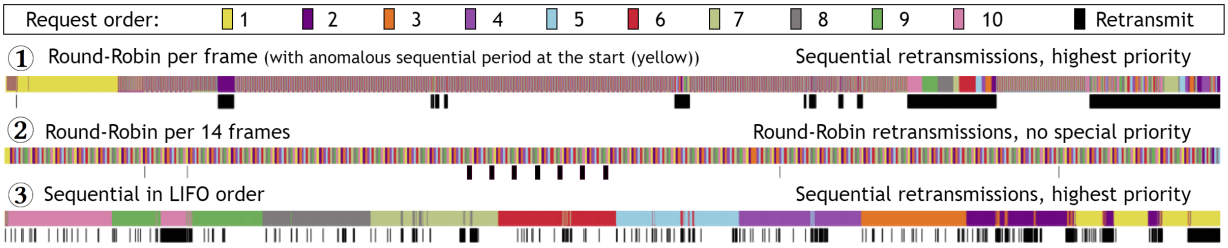

Figure 2: Partial Multiplexing Diagrams for three different QUIC stacks sending 10 1MB files in parallel. Each small colored rectangle is one packet belonging to a file. Long colored areas indicate sequential scheduling. Black areas indicate which frames above them contain retransmitted data. Data is sent from left to right.

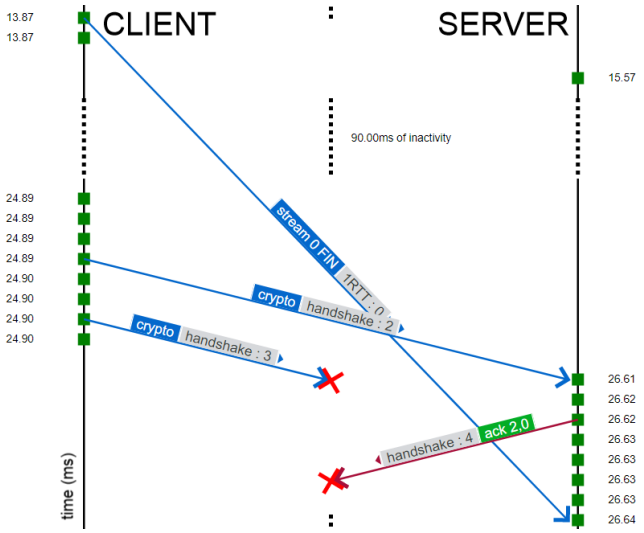

Figure 3: The Sequence Diagram accurately shows packet loss (red X) and re-ordering (crossing lines).

well. For example, for TCP+TLS+H2, we combine (decrypted) *.pcaps* with internal state from eBPF kernel probes and H2 browser logs so we can use the *qvis* tools to analyse these protocols in-depth as well (this will also be part of the demo, see Figure 1 Ⓑ). Secondly, because we feel our general endpoint-logging approach and tooling can be the basis of a powerful method for both teaching and researching protocols going forward. For both aspects, we hope to create broader awareness in the networking community and to gain additional insights and feedback to help better steer our further efforts.

2 DEMO CONTENTS

The demo will first shortly introduce the *qlog* format itself and the benefits it provides. Then, we explain *qvis* features using traces containing bugs we and others encountered in 15 different QUIC codebases (we have dozens of traces available, and will select depending on the length of the demo and interest of the audience). During the demo, attendees will not only learn about our visualizations and how to use them, but also about advanced QUIC and H3 protocol features and implementation details. We will focus on four main visualizations:

Firstly, the **Sequence Diagram** plots different viewpoints of the same connection (e.g., client and server in Figure 3) on a vertical timeline. By correlating events across traces, the tool accurately shows RTTs, packet loss, and re-ordering/jitter. This helps find issues in QUIC's complex connection handshake (e.g., deadlocks during high loss) and 0-RTT handling (e.g., one stack did not use congestion control for early data).

Secondly, similar to tcptrace's Time-Sequence diagram, the **Congestion Graph** (not shown here) plots data sent and acknowledgements received, as well as current congestion window, bytes in flight, RTT measurements and flow control limits. This helps to verify congestion controller implementations and advanced features (e.g., pacing and hystart), as well as to identify when a sender is limited by flow control.

Thirdly, the **Multiplexing Diagram** color-codes streams to show how their data chunks are interleaved by the sender. This allows quick identification of weird anomalies in expected patterns (e.g., the sequential period in a normally Round-Robin scheduler in Figure 2 ①, or ③ sending data in Last-In First-Out order (instead of FIFO)). Additionally, the tool highlights how QUIC data retransmissions are scheduled (black areas), which can diverge greatly from TCP's approach (e.g., ② interleaves retransmissions with new data). It also visualizes the impact of Head-of-Line blocking (not in Fig. 2).

Finally, the **Packetization Diagram** shows how protocol units (frames, records and packets) are sized, combined and packed. This highlights wire-format efficiency (e.g., in Figure 1 Ⓐ, the QUIC stack sometimes did not bundle small H3 frames together correctly, sending a tiny packet for each instead) and cross-layer interactions (e.g., in Ⓑ, all application data was unnecessarily flushed to a new TLS record whenever a 9-byte H2 frame header was written).

REFERENCES

[1] Robin Marx, Wim Lamotte, Jonas Reynders, Kevin Pittevils, and Peter Quax. 2018. Towards QUIC Debuggability. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC (EPIQ'18)*. ACM, 1–7. https://doi.org/10.1145/3284850.3284851

TECHNICAL REQUIREMENTS

There should be no special technical requirements for this demo. The tools and all example traces are available online (see https://qvis.edm.uhasselt.be and https://qlog.edm.uhasselt.be/sigcomm), so attendees can follow along at home and even explore the examples on their own terms.

A short video demonstrating qvis for this proposal can be found at https://youtu.be/vg2ss3NIVIU.