Debugging QUIC and HTTP/3 with qlog and qvis

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

Maxime Piraux
maxime.piraux@uclouvain.be
UCLouvain
Louvain-la-Neuve - Belgium

Peter Quax*
Wim Lamotte
Hasselt University – tUL
EDM - *Flanders Make
Diepenbeek - Belgium

ABSTRACT

The QUIC and HTTP/3 protocols are powerful but complex and difficult to debug and analyse. Our previous work proposed the *qlog* format for structured endpoint logging to aid in taming this complexity. This follow-up study evaluates the real-world implementations, uses and deployments of *qlog* and our associated *qvis* tooling in academia and industry. Our survey among 28 QUIC experts shows high community involvement, while Facebook confirms *qlog* can handle Internet scale. Lessons learned from researching 16 QUIC+HTTP/3 and five TCP+TLS+HTTP/2 implementations demonstrate that *qlog* and *qvis* are essential tools for performing root-cause analysis when debugging modern Web protocols.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Protocol testing and verification**; **Network protocol design**.

KEYWORDS

QUIC; HTTP/3; Transport Protocol; Logging; Visualization

ACM Reference Format:
Robin Marx, Maxime Piraux, Peter Quax, and Wim Lamotte. 2020. Debugging QUIC and HTTP/3 with qlog and qvis. In *Applied Networking Research Workshop (ANRW '20), July 2020, Online, Spain.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3404868.3406663

1 INTRODUCTION & MOTIVATION

The new QUIC and HTTP/3 (H3) protocols [16, 43] provide a large number of exciting features (such as zero Round-Trip-Time (RTT) handshakes and connection migration), but with great power comes great complexity. Spread over 440 pages in ten documents [86], the protocols are challenging to understand and implement. In 2018, by working on our own implementations [28, 57, 67], it became evident that the QUIC community would need extensive tooling to help debug and validate their systems. These tools would ideally be re-usable across codebases, in turn requiring a common input data format. Looking at tools for established protocols like the TCP+TLS+HTTP/2 (H2) stack (e.g., Wireshark, tcptrace, and captcp [10, 62, 65]), we see they ingest the protocols' wire image directly (e.g., via (decrypted) packet capture (pcap) files). While this can work for QUIC, it is suboptimal, as the wire image lacks pieces of internal state crucial to debugging complex components (e.g., congestion control variables, why a packet was marked lost). This state was at that time available only at the endpoints in ad-hoc command line logs, which are implementation-specific and difficult to interpret and parse.

To combat this issue, we proposed *qlog*, a structured, JSON-based logging format for QUIC+H3 endpoints [55]. It standardizes how typical protocol events and internal implementation state should be logged. This is done in an extensible manner: each *qlog* event is defined by a timestamp, a category (e.g., "transport"), an event type (e.g., "packet_sent") and some type-specific data (e.g., the size of the sent packet and its header fields). Events are listed in per-connection traces, and traces from multiple endpoints (e.g., from the client, load balancer, edge node, origin server) can be grouped into a single *qlog* file. Further information on *qlog* can be found online [58], but is not necessary for the rest of this discussion. To prove the worth of a structured logging format, we implemented several tools that can directly ingest and visualize *qlog* files in our open-source *qvis* toolsuite [52]. Using a few general-purpose tools (e.g., a sequence diagram, high-level statistics overview), users can form a hypothesis of the potential problem. More focused tools (e.g., for congestion control, multiplexing and packetization, see §3) can then be used to drill deeper.

While at that time *we* were of course convinced of the usefulness of our proposed approach, it was firstly difficult to estimate **how essential these tools would really be** for QUIC debugging. Secondly and subsequently, it was unclear **if QUIC developers were willing to adopt *qlog***. These are two of the three questions this follow-up work aims to answer.

ANRW '20, July 2020, Online (Meetecho), Spain

ACM ISBN 978-1-4503-8039-3/20/07…$15.00
https://doi.org/10.1145/3404868.3406663

The third question is **whether *qlog* can scale**, and can be used not only to debug the initial implementations, but also to analyse issues in their ensuing real-world deployments. This is needed as more traditional methods to handle this use case are difficult or impossible to use for QUIC [46]. To understand this, first consider that live deployment troubleshooting is typically done by two types of actors. Firstly, the 'endpoint owners', who manage load balancers, Content Delivery Network (CDN) nodes, origin servers, etc. Secondly, the 'network operators', who maintain the intermediate network infrastructure. For analysing deployments, lower-layer tooling is prevalent (e.g., ipfix, netflow [22, 23]), yet to debug complex issues, insight at especially the transport layer is often needed. For TCP, most approaches again rely on capturing its wire image, either at endpoints or from passive on-path measurements. Core network health metrics, like latency and packet loss rate, are deduced from TCP metadata (e.g., by correlating TCP sequence and acknowledgement numbers) and used to localize network issues [18, 32, 47, 79]. Crucially however, these latter methods no longer work for QUIC, as it end-to-end encrypts most of this transport-level metadata [77]. As such, QUIC traffic would either have to be decrypted live or be stored fully encrypted for post-hoc decryption and analysis. The latter could lead to huge storage requirements for QUIC, whereas for TCP 'frame snapshots' can be used [88], storing just the first 100 or so bytes of each packet. More problematic however, are the privacy and security implications of such a scheme. Storing the per-connection decryption keys undoes many of their ephemeral benefits and decrypting QUIC traffic not only reveals transport metadata, but also full application layer payloads. Aggravating matters, to make this usable for network operators, endpoint owners would have to actively share the decryption keys or decrypted traces.

One solution to this issue is to add unencrypted metadata to QUIC packet headers. This has however been met with trepidation from the QUIC working group, and even the inclusion of a single 'spin bit' for latency measurement was heavily debated [2, 3]. Another solution, especially for the endpoint owners, could be to use *qlogs*, as they can easily be constructed to only contain the necessary metadata, saving storage and bypassing many privacy issues. They can also contain internal application state, allowing for deeper root-cause analysis. Yet, for this to be feasible, we first need to determine if *qlog*'s JSON-based endpoint logging indeed scales.

In the rest of this text, we address these three questions, starting with *qlog*'s and *qvis*' adoption and scalability in §2. We show a high investment from the QUIC community, with 12 out of 18 active QUIC stacks [4] supporting the format (notably including Cloudflare, Mozilla, Node.js [5, 7, 8]). Facebook's [6] use of *qlog* and *qvis* at Internet scale (to fine-tune their QUIC+H3 deployment for their mobile applications), gives us insight into *qlog*'s scalability, as they log over 30 billion *qlog* events per day. We then discuss in §3 how tools like our *qvis* visualizations have been used by us and others to debug issues encountered with multiplexing, packetization, congestion control and multipath extension design. We conclude in §4 that our approach indeed delivers on its potential, but that there is still a way to go towards practical use at scale.

2 QLOG AND QVIS USAGE IN PRACTICE

To assess *qlog* and *qvis*' adoption and use by others we use a two-pronged approach. Firstly, we conduct a month long (March 2020) expert online survey among QUIC implementers and researchers. We inquire after (reasons for) logging format availability, debugging practices and future plans in this space. A total of 28 participants was recruited via the IETF QUIC mailing list and via direct requests. Our respondents include at least one core developer from all but 2 of the 18 active QUIC implementations, in addition to 6 academic researchers, which we consider a representative sample of the IETF QUIC community. The survey questions and full results are omitted here for space considerations, but are available on our website [56]. Secondly, as Facebook is the first party to use *qlog* and *qvis* at a large scale, we conduct a 1.5 hour long semi-structured interview with a senior Facebook engineer on the QUIC team. We inquire after reasons for choosing *qlog*, deployment and scalability details and encountered issues.

Our survey results show a high uptake of *qlog* and *qvis* in the community, with 12 of 18 stacks outputting the format and 18 of 28 participants using *qvis*. Per our survey, this broad adoption is driven by two main factors: firstly, the ability to use and create tools and visualizations for debugging (which we discuss in §3), and secondly, the flexibility of the *qlog* format, which is leveraged in four main ways:

Firstly, there is the ability to easily define custom events. Due to *qlog*'s use of JSON, new event categories, types and metadata can trivially be added, modified and extended, which many developers have done for implementation-specific events. It also helped in debugging new QUIC features such as the ACK frequency, Datagram and Loss Bits extensions [33, 42, 64], as well as Multipath (§3.4). In *qvis*, most visualizations also show these custom events, preventing users from having to wait for a *qlog* or *qvis* update. Conversely, implementers can choose *not* to log certain event types, allowing them to reduce file sizes or implement only the events they need.

Secondly, this flexibility goes beyond just QUIC+H3, as *qlog* can support additional network protocols. Some have already done this for DNS-over-QUIC [40], while others envision utilizing it for WebTransport, MASQUE and QUIC tunnel [66, 71, 84]. We ourselves have laid the groundwork for supporting TCP+TLS+H2 in *qlog* and *qvis* [9]. In this case, most of the basic events are obtained by transforming packets from (decrypted) pcap files into their *qlog* event counterparts. Fine-grained internal TCP state (e.g., congestion window,

RTT estimates) is retrieved by injecting eBPF probes inside the Linux kernel [34, 69], an approach that others have used as well [44, 78]. Both types of data are then spliced together into a single qlog trace. Eventually, we will also extract internal H2 state [1, 80], but even without this we have successfully analysed several TCP+TLS+H2 deployments with the same tools we use for QUIC+H3 (see §3.1 - §3.3).

Thirdly, the ability to aggregate connection traces (e.g., from multiple endpoints) into a single *qlog* file has much potential. It makes it easier to evaluate end-to-end behaviour in complex multi-tier setups, something which Facebook is experimenting with and which several companies indicate they do not even have for their TCP+TLS+H2 setups. The *qvis* sequence diagram (see §3.4) can also visualize these complex interactions across endpoints, showing accurate packet loss, reordering and RTT interplays.

A fourth and final aspect is that *qlog*'s machine readable format allows other uses besides logging and visualizing. For example, some implementers utilize it as part of their (unit) testing pipeline, validating protocol behaviour by observing events in the *qlog* output [6, 48]. QUIC-Tracker and QUIC-Network-Simulator [67, 73] are also considering using it to verify interoperability testing results. Another example is Facebook, which stores all *qlog* events in a relational database. This allows them to easily query for traces with specific behaviour (e.g., high percentage of 'packet_lost' events).

Knowing now why most QUIC implementers choose to support *qlog*, we should consider why some (6/18) do not. In the survey, some large companies such as Google and Microsoft indicate their preference for an in-house format. Others hesitate assigning *qlog* a high priority, waiting for a student developer or on the availability of libraries [53, 63], despite also indicating they suffer from the lack of additional debuggability. They also fear that both the initial implementation and its later maintenance entail a considerable time investment. This is somewhat contradicted by our own experiences: our *qlog* integration in PQUIC [28] is isolated and made flexible so that additional plugins can easily inject new events (§3.4).

A final argument heard against *qlog* is that its use of JSON might not scale [72]. We chose JSON because it is flexible, has excellent built-in support in most programming environments, and allows plaintext search. However, especially larger companies fear the format is too verbose (leading to large file sizes) and too slow to (de)serialize to use in production. They advocate using a more optimized binary format [29, 83], even though these typically lack many of JSON's benefits. Facebook, the only party with experience deploying *qlog* at scale, posits a more nuanced view. They find *qlog* is indeed two to three times as large, and takes 50% longer to serialize, than their previous in-house binary format. However, this overhead is manageable on the server-side. They *qlog* close to 10% of all QUIC connections, selected via random sampling, scaling to over 30 billion daily *qlog* events [11]. Contrarily, on the client-side, the large size does often prevent them from uploading full-length *qlogs* via the users' often constrained cellular links. Still, they would not want to move to a binary format if it meant losing flexibility and feel the CPU overhead can be reduced by developing a *qlog*-specific JSON serializer.

It is clear that *qlog* would benefit from a solution which balances flexibility and efficiency. After evaluating several options [50, 72], we settled on adding a two-pronged "optimized mode". Firstly, we employ logical compression by replacing repeated values with an index into a dynamic dictionary (akin to H3's QPACK [45]). Secondly, we use CBOR [17] to encode this smaller *qlog* and its dictionary. CBOR is JSON's direct binary counterpart, compresses well and retains flexibility [31, 68]. For reference, the *qlog* file for a 500MB download is normally 276MB, but only 91MB in optimized mode, which easily compresses down to 15MB, while the optimized (but less flexible) binary protobuf equivalent [29] ends at a compressed 14MB. In contrast, even the compressed *pcap* exceeds 500MB, showing this alternative is indeed much more difficult to scale. Finally, note that even web-based tools like *qvis* scale to loading hundreds of MB of JSON.

3 VISUALIZATION CASE STUDIES

The behaviours and cross-layer interactions of network protocols can be difficult to discern from textual logs, but even a simple visualization often provides immediate insight into a problem. Per our survey, one of the main reasons to use a structured and standardized format like *qlog* is indeed the ability to both easily create custom tools, and re-use existing ones like *qvis* [52]. We have extended *qvis* substantially since its introduction, implementing five powerful tools in over 10.000 lines of open source TypeScript code [51]. This section shows new examples of how we and others [21, 54, 75] have used *qlog*-based tools to find bugs and inefficiencies in 16 QUIC+H3 stacks and five TCP+TLS+H2 implementations, and to validate our own protocol improvements. As interactive tools can be challenging to illustrate in screenshots, readers are encouraged to explore the discussed examples (and more) in *qvis* via our web page https://qlog.edm.uhasselt.be/anrw [56].

3.1 Stream Multiplexing and Prioritization

Modern protocol stacks often multiplex data from several parallel "streams" onto one connection (e.g., HTML, CSS and image files when loading a web page). This multiplexing can happen in various ways (e.g., files are sent sequentially as a whole or are scheduled via Round-Robin (RR) after being subdivided in chunks) and is typically steered using a prioritization system (i.e., H2's complex dependency tree [15]). However, correctly implementing H2 prioritization is difficult, which has lead to degraded Web page load performance [25, 60, 87].

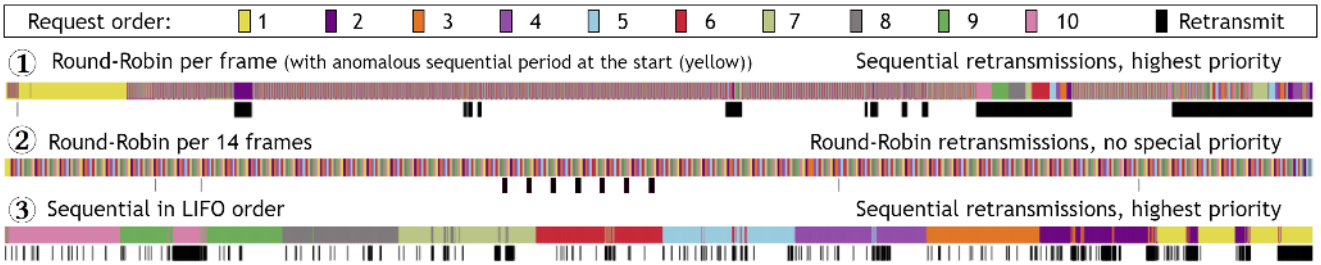

Figure 1: Multiplexing behaviour across three different QUIC stacks when downloading 10 1MB files in parallel. Each small colored rectangle is one payload frame belonging to a file. Black areas indicate which frames above them contain retransmitted data. Data arrives from left to right.

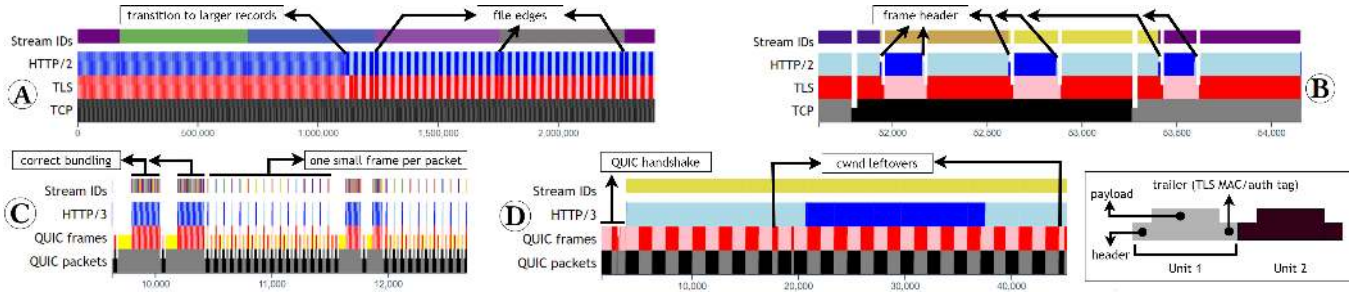

Figure 2: Traces from four different TCP/QUIC servers visualized in the packetization diagram. The x-axis is in bytes received. Alternating colors on each row indicate the switch to a new TCP/QUIC packet, TLS record, QUIC/HTTP frame or HTTP stream. Elements that align vertically are packed into the lower layer's payload.

Partially due to a lack of tooling, these issues were only discovered years after H2's standardisation [26, 87] (and remain to this day [74]), which prompted a redesign of priorities in H3 [54, 61]. Several H3 developers are now using the *qvis* multiplexing diagram (Figure 1) to verify their implementations of this simplified setup. Given a receiver-side trace, it shows H2/H3's response payload carrying frames, appended on a horizontal line with coloring to discriminate the stream each belongs to. By looking at color patterns, the multiplexing behaviour can quickly be deduced: RR schemes show frequent color changes (①,②), while long contiguous swaths (③) mean sequential transfers. It is also easy to see abnormalities: ① normally uses RR but has a long sequential period at the start (which turned out to be due to unexpected interactions with flow control), while ③ unintentionally sent data in Last-In First-Out order, the worst-case for web performance [76] (this bug was subsequently fixed [39]). For QUIC, the diagram also highlights retransmissions. While lost TCP packets are always retransmitted with the highest priority in the original packet order, QUIC's independent streams [43] give it more freedom. For example, while ③ is similar to TCP (later streams are interrupted with retransmissions of earlier ones), ② instead interleaves retransmissions with *new* data, and ① even changes its multiplexing behaviour from RR to sequential for lost data (so individual streams can make as much progress as possible). Even though none of the stacks do this today, QUIC retransmissions can be coupled to H3 prioritization (i.e., new high priority data could precede low priority retransmissions), potentially leading to improved Web performance. As this approach can incur high scheduling variability, the multiplexing diagram is important to verify correct behaviour.

3.2 Packetization and Framing

The multiplexing diagram in §3.1 was mainly concerned with the high-level ordering of HTTP DATA frames. Yet in the lower layers, these and other HTTP frames are subdivided into smaller protocol units for transport. H2 frames are packed in one or more TLS records, which are in turn distributed across TCP packets. QUIC foregoes TLS records [77], instead packing H3 frames in QUIC STREAM frames, and finally QUIC packets [43]. How these units are sized and combined can significantly impact the protocol's efficiency, as each subdivision adds some bytes of overhead (e.g., packet, record, and frame headers). Additionally, it can also carry security risks: if the edges of HTTP frames align directly with lower layer edges, attackers could in some cases derive HTTP resource sizes, usable in fingerprinting and CRIME-style attacks [30, 82].

The *qvis* packetization diagram (Figure 2) reveals the data packing particularities by vertically aligning each protocol layer's units. It clearly distinguishes payloads from overhead

(making the latter show up as vertical white areas) and alternates colors to show unit edges. The top row ("Stream IDs") is similar to the multiplexing diagram (see Figure 1, §3.1).

Zoomed out, this setup shows macro-level trends. For example, the blue H2 frames in Ⓐ are the same size as the red TLS records they are packed in, as their edges align exactly. The connection also starts with small TLS records (hence the blurriness in the screenshot), but after about 1.1MB it switches to larger records (which are more efficient to encrypt [35, 36]). Additionally, each file (top row) ends on a much smaller TLS record, making file sizes easier to estimate by an attacker.

Zoomed in, lower level details can be discerned. In Ⓑ, the H2 layer strangely forces a flush of all outstanding data into a new TLS record whenever a new 9-byte H2 frame header is written. This is not only highly inefficient, it can also again reveal resource sizes. Our H3 stress test in Ⓒ requests 1000 small files of 10 bytes each. We expect the implementation to bundle as many of them as possible in one packet, which it fails to do at times, unintentionally generating some tiny QUIC packets. This result caused the developers to revise their bundling logic. In Ⓓ, a few smaller than average packets can be seen after approximately 10 and 30 sent QUIC packets. This is because per specification [41], QUIC's congestion window (cwnd) is not expressed in packets (as it is in TCP) but in bytes. Ⓓ's implementation aggressively fills its byte-allowance completely, even if that means generating smaller packets. Interestingly, our tests revealed that contrary to the specification, almost half of all QUIC implementations bypass this inefficiency and round up their cwnd to full-sized packets (similar to TCP). After we pointed this out to Ⓓ's developer, he decided to switch to this alternate approach as well [81].

For the future, examining packetization will be useful for debugging new application protocols (e.g., DNS-over-QUIC [40] or DNS-over-HTTP/3) and when using QUIC as an encrypted tunnel (e.g., MASQUE, QUIC tunnel [37, 66, 71]).

3.3 Congestion Control

Even after decades of evolution, congestion control approaches (CCs) are still a topic of active research and innovation [13, 85]. Bugs are still being found (e.g., Google found a decade-old bug in the Cubic CC when implementing QUIC [49]), CCs are still being fine tuned [70] and new CCs are being developed (e.g., COPA, BBRv2 [14, 20]). This is only expected to continue and even increase with QUIC, which is more open to experimentation than TCP due to its user-space implementations [49].

Yet, this experimentation might be stifled by the fact that the CC is one of the most complex components to implement correctly. This was also echoed in our survey, where debugging CCs was quoted as the main reason to create custom visualizations based on *qlog*. Several participants have created ad-hoc tools, implemented in a matter of minutes, to plot CC-related variables to observe their evolution over time.

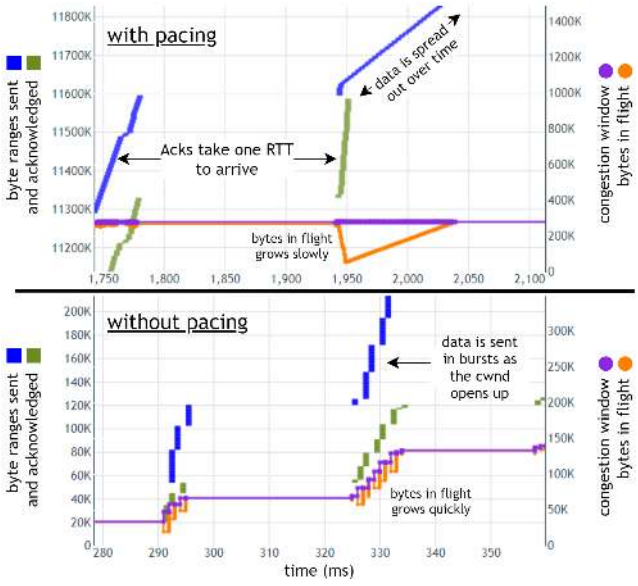

Figure 3: Detail of the congestion control graph.

The *qvis* suite includes a more comprehensive congestion control graph, partly shown in Figure 3. Like tcptrace's [62] Time-Sequence Diagram, it plots data sent, acknowledgements received and flow control limits on a timeline. However, if given a full sender-side *qlog* file, it can also plot accurate internal state typically not available for TCP traces (e.g., congestion window, exact bytes in flight, employed RTT measurements, when and why packets were determined lost). Figure 3 shows how this rich *qlog* data can be used to verify the CC behaviour with and without pacing enabled. Pacing is the practice of spreading out packets across an RTT instead of sending them in short bursts, and is thought to reduce packet loss [12]. With pacing, the bytes in flight grow slowly over time as data is spread out, while without pacing, it jumps up quickly.

Using both *qvis* and custom tools, several bugs were found. As an example, Facebook diagnosed their BBR code not entering the probeRTT state at the right time. They also identified large-scale pacing issues between their transatlantic data centers due to errors in RTT measurement. Cloudflare used *qvis* to debug their Cubic CC with 'hystart' implementation [21]. Others mention finding bugs in QUIC's retransmission logic during its complex handshake. While developing PQUIC [28], we ourselves found the network emulation tool "mininet" to queue up an infinite amount of packets when using the default settings (ignoring the `max_queue_size` parameter). This was visible as a slow start phase and an ever-increasing RTT spanned the entire transfer. Additionally, we detected very short kernel freezes that were clearly visible as a narrow notch in the congestion graph. We also used the tool to validate our implementation of new CCs such as Westwood and BBR [19, 59] in PQUIC.

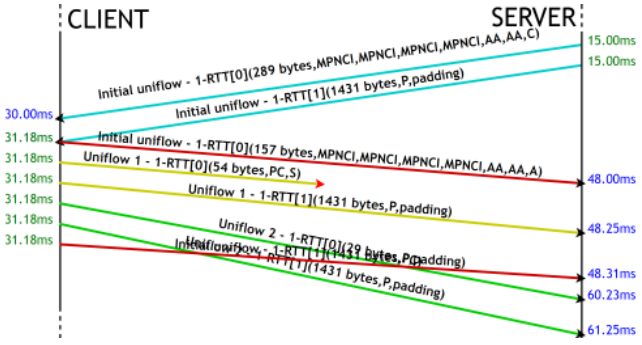

Figure 4: Sequence diagram of a partial MP-QUIC uniflow establishment, with reordering and packet loss.

3.4 Multipath QUIC

One key feature of QUIC is its ability to migrate a connection from one set of IP addresses and ports to another [43]. Multipath QUIC (MP-QUIC) is a proposed extension improving this mechanism by allowing the simultaneous use of several network paths by modelling them as "uniflows" [24, 27]. An MP-QUIC implementation utilizes four new key components (see below), that each add significant complexity to QUIC's packet sending and processing logic, and thus require insight into large amounts of internal state to properly debug.

To this end, when developing the Multipath extension in our Pluginized QUIC (PQUIC) implementation [28], we have leveraged *qlog*'s flexibility and substantially extended it. We added custom *qlog* events for the new multipath-related state and re-scoped existing connection-level events to the path level by adding a "uniflow ID". Subsequently, as most of the *qvis* tools available today were still under development at that time, we developed two custom multipath-enabled tools. The first is a simplified version of *qvis*' sequence diagram (Figure 4). By correlating both client and server-side *qlogs* (a feature missing from many TCP-centric tools), it can accurately show packet loss, RTT, and reordering and it uses colored arrows to indicate separate uniflows. A packet's full content can be viewed by hovering over its summary text. The second is a general timeline tool allowing the visualization of any *qlog* event data, selected through the use of a simple grammar (e.g., it plots congestion control state per path, providing similar functionality to mptcptrace [38]). We have used these two tools to debug the four key MP-QUIC components:

First, the **path manager**, which decides which potential network paths are actually usable at any time by using a link failure detection heuristic. By matching the time at which our experiments triggered a link failure with the time we observed the path manager retiring a uniflow and probing another one, we detected and corrected false positives in this heuristic.

Second, the **packet scheduler**, which decides which of several uniflows should be chosen for sending a packet. In the sequence diagram tool, we observed packets being re-injected into uniflows that were not validated after a link failure. The per-uniflow colors revealed these issues in a matter of seconds.

Third, the **frame scheduler**, which composes a packet from a series of frames based on the selected uniflow. By observing the Stream IDs of STREAM frames sent on each path using the timeline tool, we were able to quickly assess the correctness of our path-aware frame scheduler.

Fourth, the **congestion controller**, which was adapted to retain separate state for each network path. By plotting the RTT estimates for each path in an asymmetric path scenario, we found MP_ACK frames being incorrectly handled. The timeline plot clearly exhibited two modes in the raw RTT estimates, showing signals from both paths being mixed together. This finding partly motivated the change from bidirectional paths to uniflows in the latest MP-QUIC design [24].

4 CONCLUSION & CHALLENGES

With this work, we feel we have adequately answered two of the three open questions listed in the introduction. Firstly, it is clear that a substantial part of the QUIC community (including Facebook, Mozilla, and Cloudflare) has adopted *qlog* and *qvis*, both for debugging initial implementations and for supporting (planned) deployments at scale (60% of participants from large companies).

Secondly, what we discussed in this work was only a small selection of the results and insights we and others obtained by using *qlog* and *qvis*. Still, we hope the reader will agree that these examples show that our approach indeed radically improves the ability to implement, debug and evolve not just QUIC+H3, but other protocols as well. Additionally, we and our survey participants feel the ability to easily observe QUIC's complex behaviour will be central in bringing it to the wider masses, helping to educate newcomers, as well as facilitating further academic research.

The last question, whether the structured endpoint logging approach could scale and replace packet captures in large deployments, has however only been partially answered. We feel we have shown that it is indeed feasible for endpoint owners, as also evidenced by Facebook's deployment. We also intend to add changes (§2) to *qlog* to make it easier, though their effectiveness will also have to be proven. However, it is difficult to see how this can work directly for network operators, as they lack the ability to easily produce *qlogs*. Potentially, endpoint owners could share (aggregated/limited versions of) *qlogs* with these intermediaries, but this would require additional infrastructure for log transport, storage, aggregation and, most importantly, access control and privacy ensurance. As such, we hope our results will aid further IETF discussion on comparing our approach with the alternative of adding additional plaintext metadata to QUIC's packet header for the network operator use case. Finally, we hope the IETF can help us explore applying *qlog* to more protocols.

ACKNOWLEDGMENTS

Robin Marx is a SB PhD fellow at FWO, Research Foundation Flanders, #1S02717N. Maxime Piraux's contribution is partially supported by funding from the Walloon Government (DGO6) within the MQUIC project. The authors would like to thank our shepherd Simone Ferlin for her helpful insight and suggestions. We also appreciate the help of Olivier Bonaventure, Lucas Pardue, Mariano Di Martino, Joris Herbots, Wouter Vanmontfort, Jens Bruggemans, Pallieter Verlinden, Jimmy Cleuren, and Quentin De Coninck in reviewing earlier versions of this work and Jonas Reynders, Jeremy Lainé, Marten Seemann, Christian Huitema, Matt Joras, Nick Banks, Dmitri Tikhonov, Jana Iyengar, Mirja Kühlewind, Brian Trammell, and many other members of the QUIC community for their efforts on this topic in general.

REFERENCES

[1] 2012. NetLog: Chrome's network logging system. https://www.chromium.org/developers/design-documents/network-stack/netlog.
[2] 2018. IETF 101 - spin bit discussion. https://github.com/quicwg/wg-materials/blob/master/ietf101/minutes.md.
[3] 2018. IETF 103 - spin bit discussion. https://github.com/quicwg/wg-materials/blob/master/ietf103/minutes.md.
[4] 2020. Active QUIC implementations. https://github.com/quicwg/base-drafts/wiki/Implementations.
[5] 2020. Cloudflare quiche. https://github.com/cloudflare/quiche.
[6] 2020. Facebook mvfst. https://github.com/facebookincubator/mvfst.
[7] 2020. Mozilla neqo. https://github.com/mozilla/neqo.
[8] 2020. Node.js QUIC. https://github.com/nodejs/quic.
[9] 2020. qlog for TCP+TLS+HTTP/2 proof of concept. https://github.com/quiclog/qvis/blob/master/visualizations/src/components/filemanager/pcapconverter/qlog_tcp_tls_h2.ts.
[10] 2020. Wireshark. https://www.wireshark.org/.
[11] Lior Abraham, John Allen, Oleksandr Barykin, Vinayak Borkar, Bhuwan Chopra, Ciprian Gerea, Daniel Merl, Josh Metzler, David Reiss, Subbu Subramanian, et al. 2013. Scuba: diving into data at Facebook. *Proceedings of the VLDB Endowment* 6, 11 (2013), 1057–1067.
[12] Amit Aggarwal, Stefan Savage, and Thomas Anderson. 2000. Understanding the performance of TCP pacing. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies.* IEEE, 1157–1165.
[13] Rasool Al-Saadi, Grenville Armitage, Jason But, and Philip Branch. 2019. A survey of delay-based and hybrid TCP congestion control algorithms. *IEEE Communications Surveys & Tutorials* 21, 4 (2019), 3609–3638.
[14] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical delay-based congestion control for the internet. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18).* 329–342.
[15] M. Belshe, R. Peon, and M. Thomson. 2015. *Hypertext Transfer Protocol Version 2 (HTTP/2).* RFC 7540. RFC Editor. https://www.rfc-editor.org/rfc/rfc7540.txt
[16] Mike Bishop. 2020. *Hypertext Transfer Protocol Version 3 (HTTP/3).* Internet-Draft draft-ietf-quic-http-27. IETF Secretariat. https://tools.ietf.org/html/draft-ietf-quic-http-27
[17] C. Bormann and P. Hoffman. 2013. *Concise Binary Object Representation (CBOR).* RFC 7049. RFC Editor.
[18] Fabio Bulgarella, Mauro Cociglio, Giuseppe Fioccola, Guido Marchetto, and Riccardo Sisto. 2019. Performance measurements of QUIC communications. In *Proceedings of the Applied Networking Research Workshop.* 8–14.
[19] Neal Cardwell, Yuchung Cheng, Soheil Yeganeh, and Van Jacobson. 2017. *BBR Congestion Control.* Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-00. IETF Secretariat. https://tools.ietf.org/html/draft-cardwell-iccrg-bbr-congestion-control-00
[20] N Cardwell, Yuchung Cheng, S Hassas Yeganeh, Ian Swett, Victor Vasiliev, Priyaranjan Jha, Yousuk Seung, Matt Mathis, and Van Jacobson. 2019. BBRv2: A Model-Based Congestion Control. In *Presentation in ICCRG at IETF 104th meeting.*
[21] Junho Choi. 2020. CUBIC and HyStart++ Support in quiche. https://blog.cloudflare.com/cubic-and-hystart-support-in-quiche/.
[22] B. Claise. 2004. *Cisco Systems NetFlow Services Export Version 9.* RFC 3954. RFC Editor.
[23] B. Claise, B. Trammell, and P. Aitken. 2013. *Specification of the IP Flow Information Export (IPFIX) Protocol.* RFC 7011. RFC Editor.
[24] Quentin De Coninck, François Michel, and Olivier Bonaventure. 2020. *Multipath Extensions for QUIC (MP-QUIC).* Internet-Draft draft-deconinck-quic-multipath-04. IETF Secretariat. https://tools.ietf.org/html/draft-deconinck-quic-multipath-04
[25] Andy Davies. 2019. Preloading Fonts and the Puzzle of Priorities. https://andydavies.me/blog/2019/02/12/preloading-fonts-and-the-puzzle-of-priorities/.
[26] Andy Davies and Patrick Meenan. 2018. Tracking HTTP/2 Prioritization Issues. https://github.com/andydavies/http2-prioritization-issues.
[27] Quentin De Coninck and Olivier Bonaventure. 2017. Multipath QUIC: Design and Evaluation. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies.* ACM, 160–166.
[28] Quentin De Coninck, François Michel, Maxime Piraux, Florentin Rochet, Thomas Given-Wilson, Axel Legay, Olivier Pereira, and Olivier Bonaventure. 2019. Pluginizing QUIC. In *Proceedings of the ACM Special Interest Group on Data Communication.* ACM, 59–74.
[29] Google Developers. 2020. Protocol Buffers. https://developers.google.com/protocol-buffers.
[30] Mariano Di Martino, Peter Quax, and Wim Lamotte. 2019. Realistically Fingerprinting Social Media Webpages in HTTPS Traffic. In *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES '19).* 10. https://doi.org/10.1145/3339252.3341478
[31] J. Dickinson, J. Hague, S. Dickinson, T. Manderson, and J. Bond. 2019. *Compacted-DNS (C-DNS): A Format for DNS Packet Capture.* RFC 8618. RFC Editor.
[32] A. Ferrieux, I. Hamchaoui, I. Lubashev, and D. Tikhonov. 2020. *Packet Loss Signaling for Encrypted Protocols.* Internet-Draft draft-ferrieuxhamchaoui-quic-lossbits-03. IETF Secretariat. https://tools.ietf.org/html/draft-ferrieuxhamchaoui-quic-lossbits-03
[33] Alexandre Ferrieux, Isabelle Hamchaoui, Igor Lubashev, and Dmitri Tikhonov. 2020. *Packet Loss Signaling for Encrypted Protocols.* Internet-Draft draft-ferrieuxhamchaoui-quic-lossbits-03. IETF Secretariat. https://tools.ietf.org/html/draft-ferrieuxhamchaoui-quic-lossbits-03
[34] Matt Fleming. 2017. A thorough introduction to eBPF. *Linux Weekly News* (December 2017). https://lwn.net/Articles/740157/.
[35] John Graham-Cumming. 2016. Optimizing TLS over TCP to reduce latency. https://blog.cloudflare.com/optimizing-tls-over-tcp-to-reduce-latency.
[36] Ilya Grigorik. 2013. Optimizing TLS Record Size and Buffering Latency. https://www.igvita.com/2013/10/24/optimizing-tls-record-size-and-buffering-latency.
[37] Russell Harkanson, Yoohwan Kim, Ju-Yeon Jo, and Khanh Pham. 2019. Effects of TCP Transfer Buffers and Congestion Avoidance Algorithms

on the End-to-End Throughput of TCP-over-TCP Tunnels. In *16th International Conference on Information Technology-New Generations (ITNG 2019)*. Springer, 401–408.

[38] Benjamin Hesmans and Olivier Bonaventure. 2014. Tracing multipath TCP connections. In *Proceedings of the 2014 ACM Conference on SIGCOMM*. 361–362.

[39] Christian Huitema. 2020. Files are being sent LIFO. https://github.com/private-octopus/picoquic/issues/768.

[40] Christian Huitema, Melinda Shore, Allison Mankin, Sara Dickinson, and Jana Iyengar. 2019. *Specification of DNS over Dedicated QUIC Connections*. Internet-Draft draft-huitema-quic-dnsoquic-06. IETF Secretariat. https://tools.ietf.org/html/draft-huitema-quic-dnsoquic-06

[41] Jana Iyengar and Ian Swett. 2020. *QUIC Loss Detection and Congestion Control*. Internet-Draft draft-ietf-quic-recovery-27. IETF Secretariat. https://tools.ietf.org/html/draft-ietf-quic-recovery-27

[42] Jana Iyengar and Ian Swett. 2020. *Sender Control of Acknowledgement Delays in QUIC*. Internet-Draft draft-iyengar-quic-delayed-ack-00. IETF Secretariat. https://tools.ietf.org/html/draft-iyengar-quic-delayed-ack-00

[43] Jana Iyengar and Martin Thomson. 2020. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Internet-Draft draft-ietf-quic-transport-27. IETF Secretariat. https://tools.ietf.org/html/draft-ietf-quic-transport-27

[44] Keertan Kini. 2017. *Vessel: a lightweight container for network analysis*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[45] Charles Krasic, Mike Bishop, and Alan Frindell. 2020. *QPACK: Header Compression for HTTP/3*. Internet-Draft draft-ietf-quic-qpack-14. IETF Secretariat. https://tools.ietf.org/html/draft-ietf-quic-qpack-14

[46] Mirja Kuehlewind and Brian Trammell. 2020. *Manageability of the QUIC Transport Protocol*. Internet-Draft draft-ietf-quic-manageability-06. IETF Secretariat. https://tools.ietf.org/html/draft-ietf-quic-manageability-06

[47] Mirja Kühlewind, Tobias Bühler, Brian Trammell, Stephan Neuhaus, Roman Müntener, and Gorry Fairhurst. 2017. A path layer for the Internet: Enabling network operations on encrypted protocols. In *2017 13th International Conference on Network and Service Management (CNSM)*. IEEE, 1–9.

[48] Jeremy Lainé. 2020. aioquic. https://github.com/aiortc/aioquic.

[49] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The QUIC transport protocol: Design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 183–196.

[50] Robin Marx. 2020. qlog format converters. https://github.com/quiclog/pcap2qlog/tree/binary/src/converters.

[51] Robin Marx. 2020. qvis toolsuite code. https://github.com/quiclog/qvis.

[52] Robin Marx. 2020. qvis toolsuite live. https://qvis.edm.uhasselt.be.

[53] Robin Marx. 2020. TypeScript qlog implementation. https://github.com/quiclog/qlog/tree/master/TypeScript.

[54] Robin Marx., Tom De Decker., Peter Quax., and Wim Lamotte. 2019. Of the Utmost Importance: Resource Prioritization in HTTP/3 over QUIC. In *Proceedings of the 15th International Conference on Web Information Systems and Technologies - Volume 1: WEBIST,*. INSTICC, SciTePress, 130–143. https://doi.org/10.5220/0008191701300143

[55] 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*. 1–7.

[56] Robin Marx and Maxime Piraux. 2020. Artefacts for this paper. https://qlog.edm.uhasselt.be/anrw.

[57] Robin Marx and Kevin Pittevils. 2019. quicker, a QUIC implementation in TypeScript. https://github.com/rmarx/quicker.

[58] Robin Marx, Marten Seemann, and Jeremy Lainé. 2019. The IETF I-D documents for the qlog format. https://github.com/quiclog/internet-drafts.

[59] Saverio Mascolo, Claudio Casetti, Mario Gerla, Medy Y Sanadidi, and Ren Wang. 2001. TCP Westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of the 7th annual international conference on Mobile computing and networking*. 287–297.

[60] Patrick Meenan. 2019. Better HTTP/2 Prioritization for a Faster Web. https://blog.cloudflare.com/better-http-2-prioritization-for-a-faster-web/.

[61] Kazuho Oku and Lucas Pardue. 2020. *Extensible Prioritization Scheme for HTTP*. Internet-Draft draft-ietf-httpbis-priority-00. IETF Secretariat. https://tools.ietf.org/html/draft-ietf-httpbis-priority-00

[62] Shawn Ostermann. 2005. Tcptrace.

[63] Lucas Pardue. 2020. qlog Rust crate. https://crates.io/crates/qlog.

[64] Tommy Pauly, Eric Kinnear, and David Schinazi. 2020. *An Unreliable Datagram Extension to QUIC*. Internet-Draft draft-ietf-quic-datagram-00. IETF Secretariat. https://tools.ietf.org/html/draft-ietf-quic-datagram-00

[65] Hagen Paul Pfeifer. 2013. Captcp. http://research.protocollabs.com/captcp/.

[66] Maxime Piraux and Olivier Bonaventure. 2020. *Tunneling Internet protocols inside QUIC*. Internet-Draft draft-piraux-quic-tunnel-01. IETF Secretariat. https://tools.ietf.org/html/draft-piraux-quic-tunnel-01

[67] Maxime Piraux, Quentin De Coninck, and Olivier Bonaventure. 2018. Observing the evolution of QUIC implementations. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*. 8–14.

[68] Shahid Raza, Joel Hoeglund, Goeran Selander, John Mattsson, and Martin Furuhed. 2019. *CBOR Profile of X.509 Certificates*. Internet-Draft draft-raza-ace-cbor-certificates-03. IETF Secretariat. https://tools.ietf.org/html/draft-raza-ace-cbor-certificates-03

[69] Jonas Reynders. 2020. QUICSim. https://github.com/moonfalir/quicSim-docker.

[70] Jan Rüth, Ike Kunze, and Oliver Hohlfeld. 2019. An empirical view on content provider fairness. In *2019 Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 177–184.

[71] David Schinazi. 2020. *The MASQUE Protocol*. Internet-Draft draft-schinazi-masque-protocol-01. IETF Secretariat. https://tools.ietf.org/html/draft-schinazi-masque-protocol-01

[72] Marten Seemann. 2019. Consider moving qlog to a binary format. https://github.com/quiclog/internet-drafts/issues/30.

[73] Marten Seemann and Jana Iyengar. 2020. Network Simulator for QUIC benchmarking. https://github.com/marten-seemann/quic-network-simulator.

[74] James Snell. 2020. Node.js does not support HTTP/2 priorities. https://twitter.com/jasnell/status/1245410283582918657.

[75] Daniel Stenberg. 2020. qlog with curl. https://daniel.haxx.se/blog/2020/05/07/qlog-with-curl/.

[76] Ian Swett and Robin Marx. 2019. HTTP Priority design team update - IETF 107. https://github.com/httpwg/wg-materials/blob/gh-pages/ietf106/priorities.pdf.

[77] Martin Thomson and Sean Turner. 2020. *Using TLS to Secure QUIC*. Internet-Draft draft-ietf-quic-tls-27. IETF Secretariat. https://tools.ietf.org/html/draft-ietf-quic-tls-27

[78] Olivier Tilmans and Olivier Bonaventure. 2019. COP2: Continuously Observing Protocol Performance. *arXiv preprint arXiv:1902.04280* (2019).

[79] B. Trammell and M. Kuehlewind. 2018. *The QUIC Latency Spin Bit*. Internet-Draft draft-ietf-quic-spin-exp-01. IETF Secretariat. https://tools.ietf.org/html/draft-ietf-quic-spin-exp-01

[80] Tatsuhiro Tsujikawa. 2015. Nghttp2: HTTP/2 C library and tools. https://github.com/nghttp2/nghttp2.

[81] Tatsuhiro Tsujikawa. 2020. Round up cwnd left to the maximum UDP packet size. https://github.com/ngtcp2/ngtcp2/commit/0a28514bbbb37d85dc6e262235768716666919 2a.
[82] Mathy Vanhoef and Tom Van Goethem. 2016. HEIST: HTTP Encrypted Information can be Stolen through TCP-windows. In *Black Hat US Briefings, Location: Las Vegas, USA*.
[83] Kenton Varda. 2020. Cap'n Proto. https://capnproto.org/.
[84] Victor Vasiliev. 2019. *The WebTransport Protocol Framework*. Internet-Draft draft-vvv-webtransport-overview-01. IETF Secretariat. https://tools.ietf.org/html/draft-vvv-webtransport-overview-01
[85] Ranysha Ware, Matthew K Mukerjee, Srinivasan Seshan, and Justine Sherry. 2019. Beyond Jain's Fairness Index: Setting the Bar For The Deployment of Congestion Control Algorithms. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*. 17–24.
[86] QUIC wg. 2020. QUIC Working Group adopted documents. https://datatracker.ietf.org/wg/quic/documents/.
[87] Maarten Wijnants, Robin Marx, Peter Quax, and Wim Lamotte. 2018. HTTP/2 Prioritization and Its Impact on Web Performance. In *Proceedings of the 2018 World Wide Web Conference* (Lyon, France) *(WWW '18)*. 1755–1764. https://doi.org/10.1145/3178876.3186181
[88] Wireshark. 2010. Frame snapshot length. https://wiki.wireshark.org/SnapLen.