Towards QUIC debuggability

Robin Marx*
Wim Lamotte
Hasselt University – tUL – EDM
Diepenbeek, Belgium

Jonas Reynders
Kevin Pittevils
Hasselt University – tUL
Diepenbeek, Belgium

Peter Quax
Hasselt University – tUL – Flanders
Make – EDM
Diepenbeek, Belgium

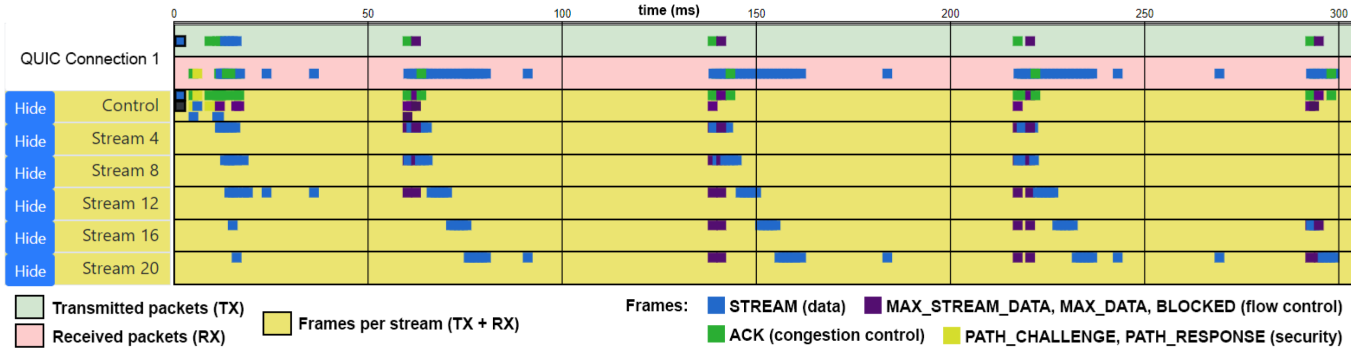

Figure 1: The QUICvis timeline-view, showing the parallel transfer of five resources.

ABSTRACT

QUIC has been called the mother of all web protocols, as it deeply integrates aspects of TCP (reliability, flow control, congestion control, loss recovery), TLS (handshake, encryption keys) and HTTP/2 (streams, prioritization) together into one cross-layer implementation over UDP. However, such ambition comes at the cost of high complexity, which in turn leads to misinterpretations, bugs and unwanted behaviour in implementations. This was also witnessed in the recently standardized HTTP/2 protocol.

We posit that QUIC should thus take a proactive approach in ensuring its testability and debuggability. To that end, this work introduces the first version of a common logging format for QUIC endpoints, called qlog. This format allows the capture of internal QUIC state that is not visible on the network. It is easily deployable and empowers the creation of reusable (visual) tools to aid in interpreting QUIC's behaviour. We implement and evaluate three such tools (a timeline, sequence diagram and congestion/flow control graph) in the proposed QUICvis toolset and show their usefulness in comparing behaviours across three competing QUIC implementations, as well as in performing root cause analysis on bugs and issues. We hope this work will foster the discussion on QUIC debuggability and that it will raise community awareness.

*Robin Marx is a SB PhD fellow at FWO, Research Foundation Flanders, #1S02717N. Contact:robin.marx@uhasselt.be

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
EPIQ'18, December 4, 2018, Heraklion, Greece
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6082-1/18/12...$15.00
https://doi.org/10.1145/3284850.3284851

CCS CONCEPTS

• **Networks** → **Protocol testing and verification**; **Transport protocols**; **Network performance analysis**; *Network performance evaluation*; *Network measurement*;

KEYWORDS

QUIC; Transport protocol; Logging; Interactive visualization

ACM Reference Format:
Robin Marx, Wim Lamotte, Jonas Reynders, Kevin Pittevils, and Peter Quax. 2018. Towards QUIC debuggability. In *Workshop on the Evolution, Performance, and Interoperability of QUIC (EPIQ'18), December 4, 2018, Heraklion, Greece.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3284850.3284851

1 INTRODUCTION

The QUIC protocol [10] is a complex beast. One of its goals is to finally solve some of the long-term issues plaguing TCP (such as Head-of-Line blocking due to packet loss, and low flexibility due to ossification [16]). As such, it is essentially a reinterpretation of what we typically think of as the features of a reliable and congestion controlled transport layer, but on top of UDP. However, QUIC does not stop there, as it aims to also incorporate best practices from security and application-level protocols. In practice, this means a (partial) integration with the TLS/1.3 protocol (e.g., allowing 0-Round-Trip-Time (RTT) connection setups) and a mapping of many of the concepts in the HTTP/2 standard, as primitives such as streams are moved down into the QUIC transport layer [10]. To remain flexible and be able to evolve quickly, the protocol is currently also fully implemented in user-space and almost entirely end-to-end encrypted, leaking only minimal information on the wire [16]. This all means that much existing functionality (e.g., reliability, congestion control, encryption logic, compression, HTTP mapping) has to be implemented from scratch, as little existing code can be re-used. Although Google has shown that this is certainly possible in their original work on Google

QUIC (gQUIC)[16], the soon to be standardized IETF version (iQUIC [10], which this work focuses on) introduces many changes and has already led to 15+ work-in-progress implementations [23], each with their own idiosyncrasies.

With such an enormous undertaking, it is highly likely that there will be various bugs and unexpected behaviours in the young iQUIC implementations for some time to come [20]. Furthermore, as QUIC will continue to evolve, adding capabilities such as multipath [6], Forward Error Correction (FEC) [16] and mappings for other application-layer protocols (e.g., WebRTC), implementations can undergo significant changes even after reaching initial maturity. In such a highly dynamic environment, it is important that these implementations can be easily tested and debugged, as many different actors will attempt to figure out which implementation to use and/or how to fine-tune that code for their specific use cases. Overall, making QUIC debuggable is not just useful in the early phases (e.g., initial implementations, first deployments), but also long afterwards (e.g., fine tuning live deployments, testing new features, academic research, live network operations).

However, as QUIC is a highly optimized, binary and end-to-end encrypted protocol, proper tooling support is necessary for efficient debugging. While some software is available that can help in analyzing QUIC traffic (e.g., Wireshark), it is not focused on QUIC specifically, nor does it cater for combining contextual information across the various layers that QUIC integrates (transport, security and application). This makes it difficult to understand complex interactions (e.g., 0-RTT connection establishment combined with HTTP Server Push). Additionally, much of QUIC's internal state used for decision making (e.g., congestion control state) is not explicitly communicated over the network. Given this state of affairs, we posit that the time is right to start working towards a common, standardized endpoint logging format (§3) and a set of shareable and easily reusable visual tools (§4). We support our statement by evaluating various iQUIC implementations (§5) and open source our work at https://quic.edm.uhasselt.be.

2 BACKGROUND AND RELATED WORK

2.1 A motivating analogue: HTTP/2

Much of our motivation for this work comes from our experience with the HTTP/2 protocol, whose standardization trajectory was very similar to QUIC's (both evolving from work at Google [16]). Even though HTTP/2 was already standardized in 2015, is conceptually much simpler than QUIC and various mature, interoperable implementations exist, significant operational differences remain. Unexpected issues and bugs keep popping up even now (e.g., Server Push is inconsistent [12], prioritization support is lacking or faulty [8, 9, 19, 33], dynamic HPACK compression is missing [32]) and few implementations support the full specification (correctly) [13]. Finding these implementation discrepancies can be complex in practice, mainly due to a lack of a common logging format and specialized tools. We posit that a more extensive focus on debuggability from the early phases can help prevent long running issues in QUIC.

Packet traces (e.g., .pcap files, Wireshark) are useful in this regard, but they only convey information that is actually put on the wire, hiding certain details (e.g., whether the server actually adheres to priority directives or not). Chrome is the only browser providing easily accessible HTTP/2 logs (at chrome://net-internals/#http2), but only barely exposes additional state (e.g., whether a pushed stream was successfully adopted or not). A proposal [2] was made for a standard HTTP/2 server side debugging state format, but it exposes only limited and fairly high-level information and is not currently supported by all (commodity) web-server implementations.

We ourselves developed several visual tools to help with debugging HTTP/2 subsystems. The H2vis project [5] has both a priority tree visualization and a timeline on which individual TCP packets are shown next to HTTP/2 frames, split out per-stream. These tools have helped us to efficiently discover differences in various implementations and led to the uncovering of several bugs [33]. While H2vis only uses packet traces, others developed similar tools based on the chrome devtools output [25]. We do not know of any similarly complex tools attempting to parse and display various server (debug) log formats directly or even use the proposed standard format [2]. Instead, to gain deeper insights, we and other academic work have had to turn to custom or adjusted HTTP/2 implementations to obtain additional debug output or test specific scenarios [13, 30, 33].

2.2 QUIC research

With regards to gQUIC, several papers have started to assess its inner workings and performance. Even though there are only a handful of up-to-date open source implementations, almost all previous publications focus on high-level performance gains or losses and perform no root cause analyses of the observed behaviours [3, 4, 17]. A notable exception is the work of Kakhki et al. [14], in which they had to rely on custom tools and manual code instrumentation to generate state transition diagrams for gQUIC's congestion control algorithms. We believe availability of diagnostic tools would encourage researchers to go deeper, even for larger-scale studies [27].

Looking at iQUIC, there is still a lack of academic work, as the implementations are not yet robust enough to be thoroughly (performance) tested. Still, implementers have started using tools to assess their progress. For example, the quic-tracker tool [20] runs conformance tests against available experimental public iQUIC endpoints. The tool mainly registers and displays .pcap data and basic client logs. When a test fails, it often has to be re-run and the variously-formatted corresponding server side logs are looked up manually to determine the root cause. Similarly, when testing iQUIC cross-implementation interoperability, developers often rely on direct chat conversations and live, concerted debugging in a Slack group [24] to diagnose problems, as it is difficult to interpret problem areas in other people's logs, which are often also lacking key information. While this method may work in these early days, as the number of developers, users and use cases [6, 7, 21] for iQUIC grows, a more scalable approach will be needed.

2.3 Adding measurability to QUIC

Other work has also picked up on the (passive) measurability issues with QUIC's encrypted wire image. For example, Kazuho Oku proposed a specialized METRICS packet [18], containing information on number of packets sent and lost, smoothed RTTs and packet reordering. In the proposal, on-path devices have to actively request METRICS packets from the endpoints. A subsequent discussion on the QUIC mailing list [22] showed that other stakeholders were

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

```
1   {"connectionid": "0x763f8eaf61aa3ffe84270c0644bdbd2b0d", "starttime": 1543917600,
2    "fields":
3       ["time","category",  "type",             "trigger",          "data"],
4    "events": [
5       [50,    "TLS",       "0RTT_KEY",         "PACKET_RX",        {"key": ...}],
6       [51,    "HTTP",      "STREAM_OPEN",      "PUSH",             {"id": 0, "headers": ...}],
7       ...
8       [200,   "TRANSPORT", "PACKET_RX",        "STREAM",           {"nr": 50, "contents": "GET /ping.html", ...}],
9       [201,   "HTTP",      "STREAM_OPEN",      "GET",              {"id": 16, "headers": ...}],
10      [201,   "TRANSPORT", "STREAMFRAME_NEW",  "PACKET_RX",        {"id": 16, "contents": "pong", ...}],
11      [201,   "TRANSPORT", "PACKET_NEW",       "PACKET_RX",        {"nr": 67, "frames": [16, ...], ...}],
12      [203,   "RECOVERY",  "PACKET_QUEUED",    "CWND_EXCEEDED",    {"nr": 67, "cwnd": 14600, ...}],
13      [250,   "TRANSPORT", "ACK_NEW",          "PACKET_RX",        {"nr": 51, "acked": 60, ...}],
14      [251,   "RECOVERY",  "CWND_UPDATE",      "ACK_NEW",          {"nr": 51, "cwnd": 20780, ...}],
15      [252,   "TRANSPORT", "PACKET_TX",        "CWND_UPDATE",      {"nr": 67, "frames": [16, ...], ...}],
16      ...
17      [1001,  "RECOVERY",  "LOSS_DETECTED",    "ACK_NEW",          {"nr": a, "frames": ...}],
18      [2002,  "RECOVERY",  "PACKET_NEW",       "EARLY_RETRANS",    {"nr": x, "frames": ...}],
19      [3003,  "RECOVERY",  "PACKET_NEW",       "TAIL_LOSS_PROBE",  {"nr": y, "frames": ...}],
20      [4004,  "RECOVERY",  "PACKET_NEW",       "TIMEOUT",          {"nr": z, "frames": ...}]
21   ]}
```

averse to this solution, as they would prefer passive instead of active measurement. Nevertheless, they acknowledged the usefulness of debugging modalities, especially at the application layer.

The consensus of the working group evolved towards two main measurability provisions. Firstly, the so-called spinbit proposal [28], which adds up to three [29] bits of information to QUIC's packet headers. By flipping these spinbits in predictable ways, on-path observers can estimate RTTs between endpoints. However, even this relatively simple signal caused pushback from stakeholders reluctant to expose even this basic information on the network. At the time of writing, the spinbit proposal is still pending further applicability research. Secondly, Explicit Congestion Notification (ECN) is integrated via the ACK mechanism [10], but this relies on existing ECN functionality in the IP layer.

Parallel academic work proposes more generic approaches, the leading example being the PLUS project [15]. PLUS prepends an extra header with measurement information (similar to the contents of the METRICS packet [18]) to select (UDP) packets, thus adding a separate "path layer" between the network and transport layers. However, even though the authors provide a PLUS implementation for gQUIC, it is difficult to assess if and when this approach could be practically deployed, as it requires large changes to existing infrastructure and middleboxes, which could take a long time to find adoption.

Finally, the proposed measurement data in all these proposals only covers a subset of the state needed to fully debug a protocol as complex as QUIC. Recognizing the need for additional information for gQUIC, in parallel to this work Google developed the quic-trace utilitites [31], consisting of a logging format and basic visualizations. Their approach focuses heavily on processing and rendering (very) large traces, by using a binary logging format and optimized OpenGL renderer, while we opt for a human readable format (§3) and web-based tooling (§4). It also mainly targets congestion-control related metrics, while we include more application-layer information. We believe our visions are complementary and can be combined and grow together.

In conclusion, the short-term proposals (e.g., spinbit) are very limited in scope, while the more extensive mechanisms (e.g., PLUS) will probably take a longer time to find adoption in real networks. As such, there is a need for an easily deployable, yet comprehensive data gathering setup.

3 FLEXIBLE ENDPOINT LOGGING

As nor packet traces, nor existing proposals (§2.3) provide a comprehensive method for gathering QUIC debug data, this section proposes the basis for an extensive QUIC endpoint logging format named qlog. Using a single, standardized logging format enables automated aggregation of results from a large amount of tests across implementations (e.g., during a full-factorial evaluation) and the creation of reusable toolsets (§4) and shareable (research) datasets as well. It also opens up the possibility of easier post-hoc conformance validation (as opposed to online testing [20]) and conformance testing of internal server behaviour from client-side test setups.

As the iQUIC specification is not yet finalized, we are unable to present a full schema definition or specification for qlog here. Instead, a "living document" schema for the qlog format, which will evolve together with iQUIC, can be found on our website, together with additional examples.

qlog is an incremental logging format, adding a new log entry per event, thus leading to a complete ledger of all individual events. Each event is accompanied by additional metadata, to allow quick filtering on high-level contextual criteria (e.g., time period, category, event type) and more detailed information is optionally added per event. We also specify an additional "trigger" field, which indicates the reason for a specific event occurring. This enables easy tracking of high-level decisions, as the same event type can be triggered by a variety of sources. This is different from normal logging, which primarily dumps the packet contents and tools rely on heuristics to interpret their meaning in-context [1]. It is also different from the HTTP/2 standard logging proposal [2], which does not incrementally log events but instead gives a global state snapshot each time it is requested, possibly leading to missing observations if the log fetch frequency is low. With our approach, full endpoint state can be reconstructed at all times.

An illustrative, non-exhaustive (i.e., not all possible entries are shown) example of qlog, highlighting different event and trigger types, can be seen in Listing 1. The center part (lines 8-15) shows a new packet being blocked due to congestion (l12) and then finally sent after a received ACK frame enlarges the congestion windows (CWND, l14). Lines 18-20 show how the same event type (PACKET_NEW) can be triggered by three different loss detection/prevention mechanisms. Lines 6 and 9 in turn show that STREAM_OPEN

can happen due to an HTTP GET request or a Server PUSH directive. Alternately, lines 10, 11 and 13 show that a single trigger can lead to multiple different events.

Our setup allows us to be very selective in what we log and when. For example, by toggling categories and event types, the endpoint itself can decide whether it logs everything, only transport-related information, only the raw packets, only congestion/flow control, whether it includes encryption keys in the logs, etc. We can also selectively log only connections from certain clients or with given connection IDs. This allows our setup to cater to a wide range of use cases and helps limit the logging overhead.

To access the server side qlog output, we take inspiration from the HTTP/2 proposal [2], which uses a well-known url (which, for qlog, could be https://example.com/.well-known/hq/state) to access the logs for the connection on which the state is requested. We also allow retrieving information for any other single active connection (e.g., /state?connid=XYZ), and provide a way to list all available connections (e.g., /state/list), to make debugging larger setups easier. As this might easily expose sensitive information to unwanted parties, endpoints can decide not to expose the latter two options or to mandate the request to be accompanied by a secret token or password indicated in the server configuration file (e.g., /state?connid=XYZ&token=53CR3T). This prevents unwanted access by an (on-path) attacker, as the request URL is encrypted on the wire. Alternatively, the logs themselves could be encrypted. It would be best practice, especially for commodity web servers, to provide secure default settings (e.g., disabling TLS key logging) and to enforce best practices when enabling logs, ensuring the server administrator really intended to expose this information. A very similar access method is possible on the client side (e.g., Google Chrome already allows viewing logs via chrome://net-internals/#protocol). Automated tests (e.g., using webpagetest.org) could easily gather the qlog output from both the browser and server and visualize it.

The design of qlog adheres largely to the Principles for Measurability by Allman et al. [1], excluding those that are precluded by QUIC's encryption-related design choices (e.g., in-band visibility and co-operation with middleboxes). Firstly, the variety of event types and indication of event triggers, makes the format very explicit. Secondly, qlog is economic, as logging can be selectively enabled/disabled. Finally, we grant the QUIC endpoints full control on which data they expose and add security features to this effect.

qlog is quickly deployable at both client and server side and can provide much needed support during the early days of iQUIC, while other solutions (§2.3) can replace or complement our proposal later on. To speed up adoption and aid implementers who do not wish to write their own qlog codebase, we plan to provide an open source library with bindings for multiple languages. We propose the use of JSON as the carrier format, as it is human-readable and simplifies the development of shareable web-based tools, but other more optimized substrates are possible (e.g., protocol buffers, as used by quic-trace [31]). However, this should not be needed for many use cases, as JSON data can be incrementally streamed, parsed and potentially aggregated to enable processing of larger datasets and it lends itself well to (gzip) compression for storage and transfer. Some readability could also be sacrificed for performance (e.g., by replacing strings for categories, types, etc. with numerical ENUM references). These decisions are pending community feedback.

4 INTERACTIVE VISUALIZATIONS

Whether we use heterogeneous logging formats or a single, common format such as qlog (§3), we are still left with text-based artefacts that can be difficult and tedious to interpret. Interactive (visual) tooling can be employed to abstract, filter and contextualize this information. Software such as Wireshark does help with some information hiding and making packet flows clearer, but more powerful visualizations can make this process much more tangible and efficient. This section discusses the merits of three such tools.

The sequence diagram (Figure 3) draws arrows between two endpoints indicating individual packet transmission and reception events. It is similar to the well-known Wireshark / tcptrace view, but adds an indication of network latency by slanting the arrows. This in turn highlights the usefulness of having a separate client- and serverside log: as opposed to a single in-between trace, we can correlate events by packet number to calculate exact RTTs for each packet (i.e., we know exactly the processing overhead that contributes to the delay, which is hidden using the spinbit approach and which QUIC's ACK frames only expose for a single packet via the ack_delay field [10]). It also allows us to give more insight into packet reordering (clearly visible when arrows travelling in the same direction cross each other, see Figure 3 Ⓧ). Lost packets (which otherwise might remain invisible to an in-network observer, depending on its location) are indicated as half-drawn lines, ending abruptly halfway through (not shown in the image). Finally, re-transmits are highlighted and the original (lost) packet(s) are indicated in the sidebar (also not shown in the image). This is useful as, unlike TCP, QUIC uses a new packet number when re-sending lost data, and individual frames from one lost packet can be re-distributed across multiple new packets. If the source log contains that information, a click on an individual event displays additional metadata (e.g., why a re-transmit was triggered, flow control state at that time). This diagram helps analysis of small packet flows (e.g., handshake), individual streams and packet loss/re-ordering impact. While this type of graph is a staple in various tools, the use of two independent logs increases its usefulness and accuracy considerably.

The timeline (Figures 1 and 2) hides most smaller details and allows the user to focus on the bigger picture, as individual packets and frames are shown as small, (potentially overlapping) squares (coloured based on their contents). Each UDP QUIC connection can be expanded to individual streams and their frames (a conceptual "control stream" groups connection-level and handshake data), see Figure 1. The timeline further allows us to easily compare various traces. For example, Figure 2 shows us client, server and packet logs for two different implementations, all side-by-side. This makes it easy to notice (large) differences between implementations, individual runs or parallel separate connections. This comparison is useful in debugging multipath setups, fairness of congestion control algorithms or multiple concurrent QUIC connections. Especially this type of analysis is difficult using tools like Wireshark.

The timeline can also help debug more specific issues. It provides different operation modes that highlight or toggle context-sensitive information. For example, in flow control mode, frames such as MAX_STREAM_DATA are clearly highlighted. For compression

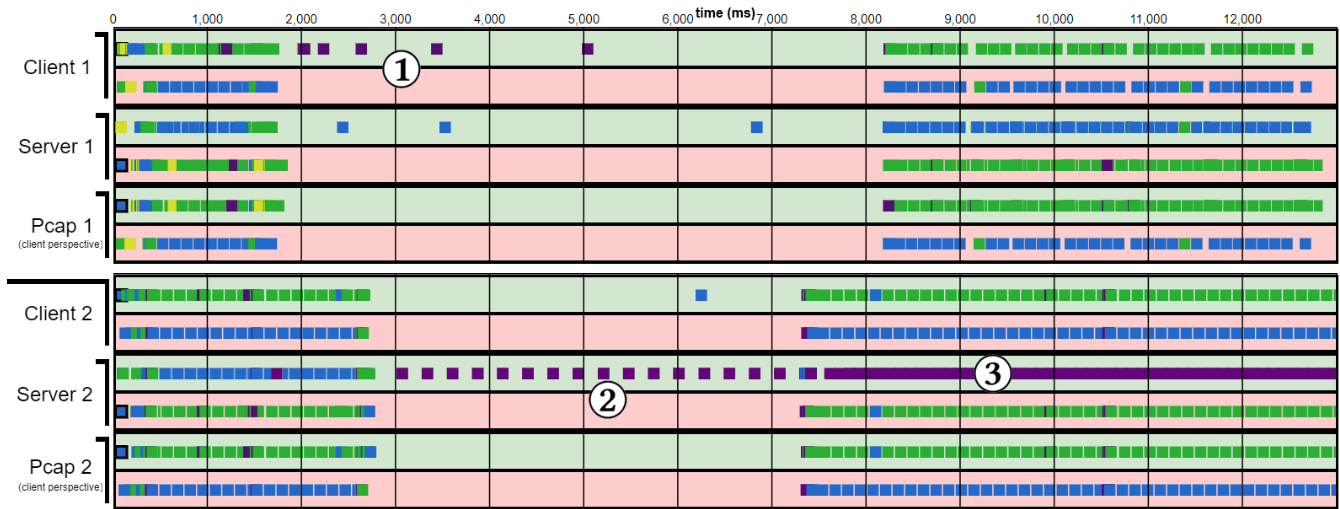

Figure 2: The QUICvis timeline, showing network interruption traces for 2 implementations. Legend: see Figure 1.

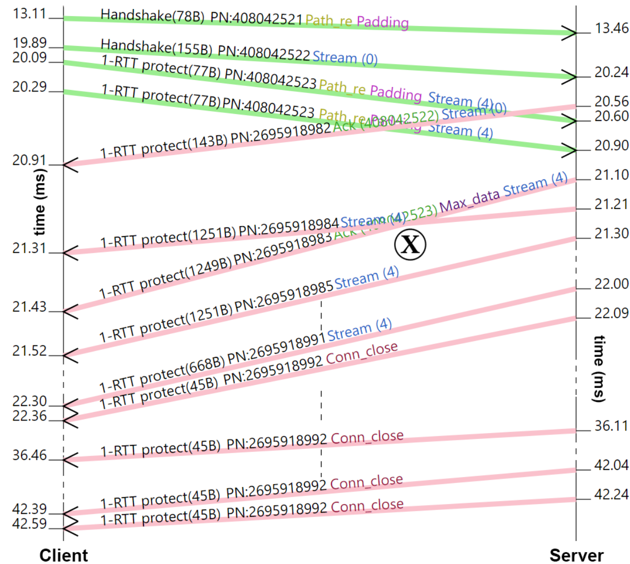

Figure 3: The QUICvis sequence diagram, showing duplicate CONNECTION_CLOSE packets. ACK-only packets are not shown for clarity.

mode, updates to QPACK's dynamic table are added to the timeline as markers. In congestion mode, endpoint estimates for goodput, latency/jitter, loss rate etc. are shown at each chosen point in time. Furthermore, selecting an ACK frame highlights the packet(s) it acknowledges, as well as earlier lost versions of itself (if any). Finally, the user can easily zoom and pan the timeline; clicking a square provides detailed information about its contents, and packets across different traces can easily be compared.

The recovery graph (Figure 4) helps show the complex interplay of congestion control and flow control, as it is often unclear which of these mechanisms is limiting data flow. The top part of the graph focuses on flow control, and shows the MAX_STREAM_DATA limits per-stream, as well as STREAM frames sent and for which offsets in that stream they carry data [10]. Equally coloured STREAM frames occupying the same y-range are re-transmits, as they contain data for the same byte offset within the stream, see Figure 4 Ⓡ. When the stacked frames reach the allowance lines, it means that the stream was flow control limited (Figure 4 Ⓕ) and a MAX_STREAM_DATA increase is needed (Figure 4 Ⓤ).

The bottom part focuses on congestion control (though it also shows the connection-level MAX_DATA flow control variable). The evolution of the congestion window (cwnd) is clearly visible, as well as which proportion of said window is currently free to be used to send data. If this "data allowance" is zero (Figure 4 Ⓒ), the connection is congestion control limited and the receipt of (an) ACK frame(s) is needed to increase the cwnd and/or reduce bytes in flight. This graph helps perform root cause analysis for slow data transfers. For example, in Figure 4 Ⓛ, streams 16 and 20 are delayed because of congestion, not faulty flow control. This kind of cross-correlative graph is often lacking from other tools.

5 EVALUATION

As the iQUIC specification is still in full flux, it is difficult to fully develop or evaluate qlog and our visualizations at this time. Nevertheless, in the open source QUICvis (https://quic.edm.uhasselt.be) project , we provide initial implementations of qlog and web-based versions of the three discussed visualizations (§4). We currently support loading .pcap files, a basic qlog format and also the custom logging formats from three iQUIC implementations (i.e., our own Quicker (https://github.com/rmarx/quicker), Quant (https://github.com/NTAP/quant) and Ngtcp2 (https://github.com/ngtcp2/ngtcp2)) to assess the investment of work needed without a unified format.

For our evaluation, we ran a variety of specification conformance tests and simple fuzzing tests and determined the three QUIC implementations' robustness and compliance. This has led to several deeper insights and unexpected findings. For example, the iQUIC specification (we have evaluated its 11th draft [11]) disallows the

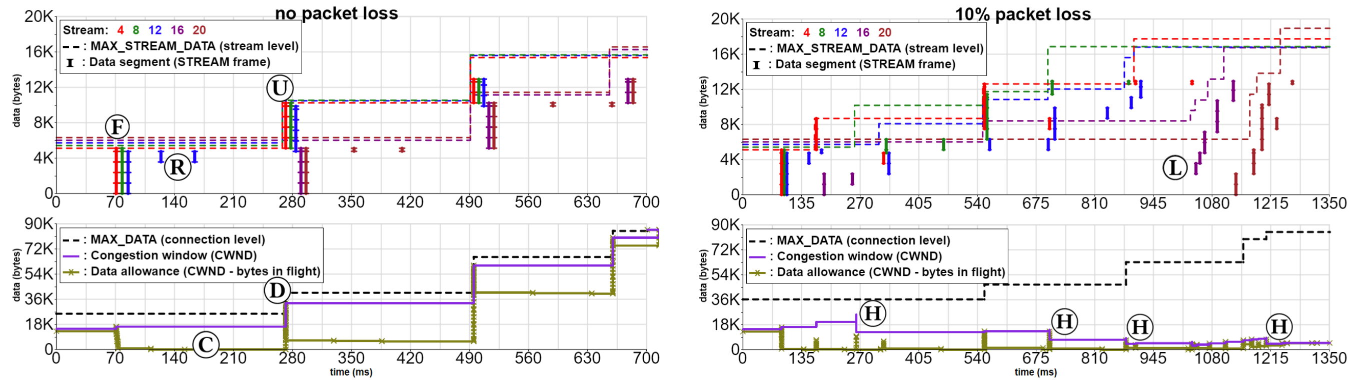

Figure 4: The QUICvis recovery graph, showing five parallel streams with no packet loss (left) and 10% loss (right). **(To prevent overlapping, MAX_STREAM_DATA lines are offset vertically and STREAM frames horizontally, where necessary).**

re-use of packet numbers and requires endpoints to ignore duplicate packet numbers. One of the observed codebases instead explicitly closes the connection with a PROTOCOL_VIOLATION error. Ironically, it repeatedly signals this with several CONNECTION_CLOSE frames, each in its own individual packet, all of which share the same packet number (see bottom of Figure 3).

A second test assessed the impact of a complete network interruption. Figure 2 compares client, server and network logs for two QUIC implementations. The top is the "normal" case: client and server individually attempt to continue by re-transmitting packets (as no timely ACK was received). The bottom implementation is faulty: it is too aggressive in re-sending the unACKed purple flow control packet ② (notice how the top implementation does have a proper back-off period ①). Even worse, when the connection is restored, it ignores congestion control and sends all pending data at once, triggering flow control allowance requests (BLOCKED frames) ③.

A third test downloaded five files simultaneously to assess how stream data is multiplexed and how flow and congestion control is performed in practice. Figure 4 Ⓕ shows that one QUIC implementation sends as much data as possible for stream 4 until the flow control allowance for that stream is reached, after which it switches to the next stream. Only when the MAX_STREAM_DATA is increased by the client, does the server send the next batch of data for stream 4 Ⓤ. Streams 16 and 20 are delayed, but not because of flow control: as the implementation employs a TCP NewReno-alike congestion control approach [10], the congestion window (cwnd) is small in the slow start phase. The three first streams completely fill the cwnd, leaving no data allowance for streams 16 and 20 Ⓒ. Only when the cwnd doubles in size after receiving the first ACKs Ⓓ, do streams 16 and 20 get bandwidth. The middle part of the qlog example in Listing 1 conceptually shows how this type of situation would appear in the logs. The exact same scenario is also shown in the timeline in Figure 1, where the stream-per-stream sending is clearly visible on the x-axis. However, there it would have been more difficult to assess the reasons for the per-stream bandwidth distribution, showing the value in combining different visualizations. The same test is also shown on the right side of Figure 4, but then on a link with 10% packet loss. There, we clearly see the cwnd being halved after loss (multiple times) Ⓗ and streams 16 and 20 being delayed even longer Ⓛ, as the earlier streams get precedence. Note that a mechanism such as HTTP/2 priorities can change how this bandwidth allocation happens, and could allow streams 16 and 20 to send data earlier.

Various other tests were performed, among others looking at packet coalescing behaviour, ACK generation delays and loss detection timeouts, but are omitted here due to space limitations. We conclude that most current iQUIC implementations will take considerable time to become production ready. Further details and results can be found in Jonas Reynders' thesis [26]. We have not yet evaluated very large traces (e.g., several hours of a real-life deployment). However, our current implementations do combine data filtering/hiding with flexible zooming/panning and offset overlapping data to allow for easier interpretation and to improve tool interactivity. Additional work will be needed though, especially to support fast web-based comparisons of multiple long-running event logs.

6 DISCUSSION AND CONCLUSION

A protocol as complex as QUIC will be challenging to debug, test, deploy and evolve. Now is the ideal time to think about these issues, as the first version of the specification is (almost) finished and implementations start to expand. We aim to kickstart the discussion by proposing a unified logging format, qlog (§3), which can be easily deployed and allows both high-level and low-level state tracking. We believe a standardized logging format is important, as it allows the creation of reusable tools that can be employed in large-scale, full-factorial evaluations across QUIC implementations. We have implemented several such tools (§4) and have used them to analyze the results of various specification conformance tests (§5). The visualizations were a great help in assessing overall behaviour and in performing root cause analysis for a variety of uncovered bugs and behaviours. While qlog and our visualizations are not absolutely necessary to reach similar conclusions, they significantly speed up the process, especially when analyzing and comparing many or complex traces. We have also shown the added value of having separate client and server side logs, as opposed to a single (network) trace. Finally, while tooling itself does not necessarily require a uniform logging format, we remark that there are large discrepancies in the details present in each observed individual format and that, if implementations aim to eventually support the same state information in their logs, it makes sense to do this in a common way. We hope to work towards this goal in co-operation with other contributors.

REFERENCES

[1] Mark Allman, Robert Beverly, and Brian Trammell. 2017. Principles for Measurability in Protocol Design. *SIGCOMM Computer Communication Review* 47, 2 (May 2017), 2–12. https://doi.org/10.1145/3089262.3089264

[2] Cory Benfield and Brad Fitzpatrick. 2016. *HTTP/2 Implementation Debug State*. Internet-Draft. IETF Secretariat. http://www.ietf.org/internet-drafts/draft-benfield-http2-debug-state-01.txt

[3] Divyashri Bhat, Amr Rizk, and Michael Zink. 2017. Not So QUIC: A Performance Study of DASH over QUIC. In *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'17)*. ACM, 13–18. https://doi.org/10.1145/3083165.3083175

[4] Sarah Cook, Bertrand Mathieu, Patrick Truong, and Isabelle Hamchaoui. 2017. QUIC: Better For What And For Whom?. In *IEEE International Conference on Communications (ICC'17)*. IEEE. https://hal.archives-ouvertes.fr/hal-01565785

[5] Daan De Meyer. 2017. H2vis. Online, https://github.com/rmarx/h2vis. (September 2017).

[6] 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 (CoNEXT'17)*. ACM, 160–166. https://doi.org/10.1145/3143361.3143370

[7] Yufeng Duan, Massimo Gallo, Stefano Traverso, Rafael Laufer, and Paolo Giaccone. 2017. Towards a Scalable Modular QUIC Server. In *Proceedings of the Workshop on Kernel-Bypass Networks (KBNets '17)*. ACM, 19–24. https://doi.org/10.1145/3098583.3098587

[8] Erik Witt. 2018. Chrome's Service Workers Break HTTP/2 Priorities. Online, https://medium.baqend.com/chromes-service-workers-break-http-2-priorities-649c4e0fa930. (August 2018).

[9] Fedor Indutny. 2017. spdy for NodeJS. Online, https://github.com/spdy-http2/spdy-transport/blob/master/lib/spdy-transport/connection.js#L339. (May 2017).

[10] Jana Iyengar and Martin Thomson. 2018. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Internet-Draft 14. IETF Secretariat. http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-14

[11] Jana Iyengar and Martin Thomson. 2018. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Internet-Draft 11. IETF Secretariat. http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-11

[12] Jake Archibald. 2017. HTTP/2 Push. Online, https://jakearchibald.com/2017/h2-push-tougher-than-i-thought/. (May 2017).

[13] Muhui Jiang, Xiapu Luo, Tungngai Miu, Shengtuo Hu, and Weixiong Rao. 2017. Are HTTP/2 Servers Ready Yet?. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS'17)*. IEEE, 1661–1671. https://doi.org/10.1109/ICDCS.2017.279

[14] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. 2017. Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols. In *Proceedings of the 2017 Internet Measurement Conference (IMC '17)*. ACM, 290–303. https://doi.org/10.1145/3131365.3131368

[15] 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'17)*. 1–9. https://doi.org/10.23919/CNSM.2017.8255973

[16] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. 2017. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*. ACM, 183–196. https://doi.org/10.1145/3098822.3098842

[17] Péter Megyesi, Zsolt Krämer, and Sándor Molnár. 2016. How quick is QUIC?. In *IEEE International Conference on Communications (ICC'16)*. IEEE. https://doi.org/10.1109/ICC.2016.7510788

[18] Kazuho Oku. 2018. *Performance Metrics Subprotocol for QUIC*. Internet-Draft. IETF Secretariat. http://www.ietf.org/internet-drafts/draft-kazuho-quic-perf-metrics-00.txt

[19] Patrick Meenan. 2018. Optimizing HTTP/2 prioritization with BBR and tcp_notsent_lowat. Online, https://blog.cloudflare.com/http-2-prioritization-with-nginx. (October 2018).

[20] Maxime Piraux. 2018. *A test suite for QUIC*. Master's thesis. Ecole polytechnique de Louvain, Université catholique de Louvain, Belgium. http://hdl.handle.net/2078.1/thesis:14585

[21] Peng Qian, Ning Wang, and Rahim Tafazolli. 2018. Achieving Robust Mobile Web Content Delivery Performance Based on Multiple Coordinated QUIC Connections. *IEEE Access* 6 (2018), 11313–11328. https://doi.org/10.1109/ACCESS.2018.2804222

[22] QUICwg. 2018. Mailinglist. Online, https://mailarchive.ietf.org/arch/browse/quic/?q=quic-perf-metrics. (February 2018).

[23] QUICwg. 2018. QUIC implementations. Online, https://github.com/quicwg/base-drafts/wiki/Implementations. (February 2018).

[24] QUICwg. 2018. Slack chat group. Online, https://quicdev.slack.com/archives/C6ALXAB7A/p1535980138000100. (August 2018).

[25] Rebecca Murphey. 2016. chrome-http2-log-parser. Online, https://github.com/rmurphey/chrome-http2-log-parser. (October 2016).

[26] Jonas Reynders. 2018. *QUIC insight*. Bachelor's Thesis. Expertisecentre for Digital Media, Hasselt University, Belgium. https://quic.edm.uhasselt.be

[27] Jan Rüth, Ingmar Poese, Christoph Dietzel, and Oliver Hohlfeld. 2018. A First Look at QUIC in the Wild. In *International Conference on Passive and Active Network Measurement (PAM'18)*. Springer, 255–268. https://doi.org/10.1007/978-3-319-76481-8$_1$9

[28] Brian Trammell, Piet Vaere, Roni Even, Giuseppe Fioccola, Thomas Fossati, Marcus Ihlar, Al Morton, and Stephan Emile. 2018. *Adding Explicit Passive Measurability of Two-Way Latency to the QUIC Transport Protocol*. Internet-Draft draft-trammell-quic-spin-03. IETF Secretariat. http://www.ietf.org/internet-drafts/draft-trammell-quic-spin-03.txt

[29] Piet De Vaere, Tobias Bühler, Mirja Kühlewind, and Brian Trammell. 2018. Three Bits Suffice: Explicit Support for Passive Measurement of Internet Latency in QUIC and TCP. In *Internet Measurement Conference (IMC'18)*. https://mami-project.eu/wp-content/uploads/2018/09/spinbit.pdf

[30] Jeroen van der Hooft, Stefano Petrangeli, Tim Wauters, Rafael Huysegems, Tom Bostoen, and Filip De Turck. 2018. An HTTP/2 Push-Based Approach for Low-Latency Live Streaming with Super-Short Segments. *Journal of Network and Systems Management* 26, 1 (01 Jan 2018), 51–78. https://doi.org/10.1007/s10922-017-9407-2

[31] Victor Vasiliev. 2018. QUIC trace utilities. Online, https://github.com/google/quic-trace. (September 2018).

[32] Vlad Krasnov. 2016. HPACK, The silent killer feature of HTTP/2. Online, https://blog.cloudflare.com/hpack-the-silent-killer-feature-of-http-2/. (November 2016).

[33] 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 (WWW '18)*. ACM, 1755–1764. https://doi.org/10.1145/3178876.3186181