Understanding QUIC and HTTP/3
with qlog and quicvis

Robin Marx - @programmingart
(Hasselt University – Belgium)
QUIC and HTTP/3: Too big to fail?!

The new QUIC and HTTP/3 protocols are coming and they are the bee’s knees! Combining lessons and best practices from over 30 years of networking, the new protocol stack offers major improvements to performance, privacy, security and flexibility.

Much has been said about the potential benefits of QUIC, most of it based on Google’s experience with an early version of the protocol. However, its potential shortcomings are rarely talked about and little is yet known about the properties of the upcoming, standardized versions (as they are still under active development). This post takes a (nuanced) “devil’s advocate” viewpoint and looks at how QUIC and HTTP/3 might still fail in practice, despite the large amount of current enthusiasm. In all fairness, I will also mention counter arguments to each point and let the reader make up their own mind, hopefully after plenty of additional discussion.

Note: if you’re not really sure what QUIC and HTTP/3 are in the first place, it’s best to get up to speed a bit before reading this post, which assumes some familiarity with the topic. Some resources that might help you with that:

- Mathias Oeniar’s blog post
- Cloudflare’s write-up
- Robert Graham’s comments
- Daniel Stenberg (@baeders)’s HTTP/3 explained
- Mailing list explanation and blog post by Patrick McManus
- And my own talk from DeltaVConf this year

Robin Marx is a Web Performance PhD candidate at Hasselt University, Belgium. He is mainly looking into HTTP/2 and QUIC performance, and maintains the TypeScript QUIC implementation Quicker. In a previous life he was a multiplayer game programmer and co-founder of LuGus Studios. YouTube videos of Robin are either humorous technical talks or him hitting other people with lorgnards.
QUIC and HTTP/3 are quite complex...

- **V1**
  - Congestion control + loss detection
  - Fairness issues
  - Flow control
  - Encryption and integrity protection
  - Connection migration
  - 0-RTT support
  - Independent streams
    - Partly re-introduces HOL-blocking in HTTP/3
  - Low overhead, binary encoding
  - DoS prevention
  - Stateless Retry
  - Retransmission logic
    - Brings up interesting prioritization questions
  - ...

- **Postponed to V2**
  - Multipath
  - Forward error correction
  - Unreliable data transfer
  - Support for other crypto besides TLS 1.3
  - Most (all?) non-HTTP/3 applications
    - IoT, realtime media, ...
  - How to expose all of this to the developer?
    - E.g., TAPS, WebTransport, ...
I'VE BEEN STUDYING QUIC FOR 2 YEARS

I STILL UNDERSTAND ONLY HALF OF IT
If QUIC is to become the **dominant** transport protocol...

- **Students** should learn QUIC basics no later than 3\(^{rd}\) bachelor
- **Researchers** should dig deep into all aspects, in specific settings
- **Application developers** should be able to debug behaviour in complex setups

*Right now, in-depth QUIC knowledge resides with 50 – 100 people worldwide*

- We need **examples**
- We need documentation
- We need **tools**

https://github.com/rmarx/quicker
Quicker: TypeScript implementation

https://github.com/rmarx/quicker/tree/congestionControl
https://github.com/rmarx/quicker/tree/http3-19
https://github.com/DaanDeMeyer/h3c
https://github.com/DaanDeMeyer/chromium
QUIC tools

not in .pcap

https://quic.edm.uhasselt.be
QUIC logging: The Wild Wild West
Standardized QUIC endpoint logging format

Store
Process
Aggregate

Visualize
Analyze

Share

https://quic.edm.uhasselt.be
Our proposal: qlog

JSON:
- **Easy to use** in web-based tools (and most programming languages)
- **Human-readable**
- **Minimally verbose while keeping flexibility** (vs csv)
qlog: simple to filter (both when reading and writing)

```
{
  "connectionid": "0x763f8eaf61a3ffe84270c0644b6db20d",
  "starttime": 1543917600,
  "fields": [
    {
      "time",
      "category",
      "type",
      "trigger",
      "data"
    }
  ],
  "events": [
    {
      "id": 50,
      "TLS",
      "ORTT_KEY",
      "PACKET_RX",
      {
        "key": ...
      }
    },
    {
      "id": 51,
      "HTTP",
      "STREAM_OPEN",
      "PUSH",
      {
        "id": 0,
        "headers": ...
      }
    },
    {
      "id": 200,
      "TRANSPORT",
      "PACKET_RX",
      "STREAM",
      {
        "nr": 50,
        "contents": "GET /ping.html",
        "headers": ...
      }
    },
    {
      "id": 201,
      "HTTP",
      "STREAM_OPEN",
      "GET",
      {
        "id": 16,
        "headers": ...
      }
    },
    {
      "id": 201,
      "TRANSPORT",
      "STREAMFRAME_NEW",
      "PACKET_RX",
      {
        "id": 16,
        "contents": "pong",
        "headers": ...
      }
    },
    {
      "id": 201,
      "TRANSPORT",
      "PACKET_NEW",
      "PACKET_RX",
      {
        "nr": 67,
        "frames": [16, ...]
      }
    },
    {
      "id": 203,
      "RECOVERY",
      "PACKET_QUEUE",
      "CWND_EXCEEDED",
      {
        "nr": 67,
        "cwnd": 14600,
        "frames": [16, ...]
      }
    },
    {
      "id": 250,
      "TRANSPORT",
      "ACK_NEW",
      "PACKET_RX",
      {
        "nr": 51,
        "acked": 60,
        "frames": [16, ...]
      }
    },
    {
      "id": 251,
      "RECOVERY",
      "CWND_UPDATE",
      "ACK_NEW",
      {
        "nr": 51,
        "cwnd": 20780,
        "frames": [16, ...]
      }
    },
    {
      "id": 252,
      "TRANSPORT",
      "PACKET_TX",
      "CWND_UPDATE",
      {
        "nr": 67,
        "frames": [16, ...]
      }
    },
    {
      "id": 1001,
      "RECOVERY",
      "LOSS_DETECTED",
      "ACK_NEW",
      {
        "nr": a,
        "frames": ...
      }
    },
    {
      "id": 2002,
      "RECOVERY",
      "PACKET_NEW",
      "EARLY_RETRANS",
      {
        "nr": x,
        "frames": ...
      }
    },
    {
      "id": 3003,
      "RECOVERY",
      "PACKET_NEW",
      "TAILLOSS_PROBE",
      {
        "nr": y,
        "frames": ...
      }
    },
    {
      "id": 4004,
      "RECOVERY",
      "PACKET_NEW",
      "TIMEOUT",
      {
        "nr": z,
        "frames": ...
      }
    }
  ]
}
```

"HTTP_STREAM_OPEN" VS "HTTP", "STREAM_OPEN"

Also saves on verbosity: akin to csv column names
qlog: clear cause and effect

Mainly when cause of event isn’t clear from context
However, also easier for tooling: focus on certain triggers
→ Explicit vs implicit/heuristic logging
qlog : structured metadata

```
{"connectionid": "0x763f8eaf61a3ffe84270c0644b6bd2b0d", "starttime": 1543917600,
"fields":
 ["time", "category", "type", "trigger", "data"],
"events": [
 [50, "TLS", "0RTT_KEY", "PACKET_RX", {"key": ...}],
 [51, "HTTP", "STREAM_OPEN", "PUSH", {"id": 0, "headers": ...}],
 [200, "TRANSPORT", "PACKET_RX", "STREAM", {"nr": 50, "contents": "GET /ping.html", "id": 16, "headers": ...}],
 [201, "HTTP", "STREAM_OPEN", "GET", {"id": 16, "contents": "pong", ...}],
 [201, "TRANSPORT", "PACKET_RX", "PACKET_RX", {"nr": 67, "frames": [16, ...], ...}],
 [203, "RECOVERY", "PACKET_QUEUE", "ACK_NEW", {"nr": 67, "cwnd": 14600, ...}],
 [250, "TRANSPORT", "ACK_NEW", "PACKET_RX", {"nr": 51, "acked": 60, ...}],
 [251, "RECOVERY", "CWND_UPDATE", "ACK_NEW", {"nr": 51, "cwnd": 20780, ...}],
 [252, "TRANSPORT", "PACKET_TX", "CWND_UPDATE", {"nr": 67, "frames": [16, ...], ...}],
 [1001, "RECOVERY", "LOSS_DETECTED", "ACK_NEW", {"nr": a, "frames": ...}],
 [1002, "RECOVERY", "PACKET_NEW", "EARLY_RETRANS", {"nr": x, "frames": ...}],
 [3003, "RECOVERY", "PACKET_NEW", "TAIL_LOSS_PROBE", {"nr": y, "frames": ...}],
 [4004, "RECOVERY", "PACKET_NEW", "TIMEOUT", {"nr": z, "frames": ...]}
]}
```

INITIAL 15 1523 VS type="initial", nr=15, size=1523
Standardized general purpose endpoint logging format?

- Why just for QUIC and HTTP/3?
  - TCP endpoint states
  - RTP / WebRTC / DTLS
  - Anything really...

- Wait... doesn’t this exist yet?
  - Turns out: no...
Current state of tooling and logging

- **Public**
  - .pcap-based (e.g., tcptrace, wireshark)
  - In-browser devtools (but very high-level)

- **Private**
  - Many more proprietary/internal tools
  - Focused on individual implementation/logging

- In many (academic) cases: none
  - Wrong interpretations of results
  - Important bugs can remain undetected for a long time

We seem to rely on a limited amount of (not thoroughly tested) implementations, which are understood only through their (high-level), end-to-end behaviour

https://github.com/andydavies/http2-prioritization-issues
https://twitter.com/AndyDavies/status/106591677408346112
Standardized *general purpose* endpoint logging format

- Discussed at IETF 104
  - Too early for its own working group
  - Use QUIC as incubator / concrete use case

1. **High-level schema**
   - Semi-protocol agnostic
   - Take into account a variety of use cases

2. **QUIC + H3 event definitions**
   - Names + metadata semantics for each type of event
   - Later also:
     - Method of access
     - Security and privacy considerations

High-level logging schema

- **Main tenets**
  - *Flexibility* in the format, complexity in the tooling
  - Extensible but *pragmatic* (e.g., no complex fixed schema with extension points)
  - *Streamable*, event-based
  - Aggregation and *transformation* friendly
  - Explicit and *human-readable*
1. Flexibility: included fields depend on use case

- **Fields** are now more dynamic
  - Depend on use case
  - Split in per-event and shared fields

  - **common_fields**
    - Value listed once in header

  - **event_fields**
    - Value listed separately for each event instance
1. Flexibility: mingle different types of log together.
1. Flexibility : file size optimization

```json
{
    "common_fields": {
        "protocol_type": ["QUIC_HTTP3", "TCP"],
        "group_ids": [
            { "ip1": "10.0.6.137", "ip2": "52.58.13.57", "port1": 56522, "port2": 443 }
        ]
    }
}
```

**Reference-by-index**
- Smaller files, less readability
- Could be done for any field
- Done in Chromium's internal format Netlog

https://github.com/quiclog/internet-drafts/issues/8
1. Flexibility: file size optimization to the extreme

```
57, "TRANSPORT", "FRAME CREATED", "TRIGGER", ["frame_type": "STREAM", "packet_number": 15, "contents": [...]],
58, "TRANSPORT", "FRAME CREATED", "TRIGGER", ["frame_type": "STREAM", "packet_number": 16, "contents": [...]]
```
1. Flexibility: file size optimization to the extreme

"data_fields": {
    "TRANSPORT+FRAME_CREATED": [
        "frame_type",
        "packet_number",
        "contents"
    ]
}

“Fully self-describing format”
→ Extensible, but pragmatic

https://github.com/quiclog/internet-drafts/issues/8
2. Streamability

- “Live debugging”: tool updates as events come in
- JSON is not a streamable format per se

```json
{"connectionid": "0x763f8eaf61a3ffe84270c0644b0d", "starttime": 1543917600,
"fields":
["time", "category", "type", "trigger", "data"],
"events": [
[50, "TLS", "ORRT_KEY", "PACKET_RX", {"key": ...}],
[51, "HTTP", "STREAM_OPEN", "PUSH", {"id": 0, "headers": ...}],
[1001, "RECOVERY", "LOSS_DETECTED", "ACK_NEW", {"nr": a, "frames": ...}],
[2002, "RECOVERY", "PACKET_NEW", "EARLY RETRANS", {"nr": x, "frames": ...}],
[3003, "RECOVERY", "PACKET_NEW", "TAIL LOSS_PROBE", {"nr": y, "frames": ...}],
[4004, "RECOVERY", "PACKET_NEW", "TIMEOUT", {"nr": z, "frames": ...}]
}
```

These two characters are apparently pretty important

- “Solution”: streaming JSON parser

https://github.com/quiclog/internet-drafts/issues/2
3. Aggregation friendly: multiple traces in 1 file

- **vantage_point**
  - Combine traces from several locations for *end-to-end overview*
  - Also possible: split out per protocol, per flow, type of event, ...
    - e.g., trace 1 = HTTP/2 from server, trace 2 = TCP from eBPF
3. Aggregation friendly: Tooling support

```
"summary": {
  "trace_count": number, // amount of traces in this file
  "max_duration": string, // time duration of the longest trace
  "max_outgoing_loss_rate": number, // highest loss rate for outgoing packets over all traces
  "total_event_count": number // total number of events across all traces
}
```

```
"configuration": {
  "time_units": "ms",
  "time_offset": 100,
  "quicvis.timeline.settings": {
    "xmin": 1000,
    "xmax": 2000,
    "streams.enabled": [1,5,9],
    "color.scheme": "HIGHLIGHT_LOSS"
  }
}
```

Quickly sift through hundreds of logs (put on top + streaming parser)

Immediately clear what other person should be looking at
4. Transformation friendly

- “No one is going to output qlog directly”
  - Liar! But even then: they don’t need to
  - “common logging format” → “common tool input format”

https://quic.edm.uhasselt.be/qtr-to-qlog/
https://github.com/quiclog/pcap2qlog
Many more open questions

- Textual vs binary (readability vs file size savings/logging perf)
- Are separate triggers useful?
- Preventing proliferation of something2qlog converters?
- Fine-grainedness of events
- Privacy aspects

- Single format for many use cases?
  - Even within QUIC: CDN vs Facebook app vs browsers vs IoT vs ... 
  - Why doesn’t this exist yet?
Many more open questions

- Textual vs binary (readability vs file size savings/logging perf)
- Are separate triggers useful?
- Preventing proliferation of something2qlog converters?
- Fine-grainedness of events
- Privacy aspects

- Single format for many use cases?
  - Even within QUIC: CDN vs Facebook app vs browsers vs IoT vs ...
  - Why doesn’t this exist yet?

- How the hell do you publish papers on this topic?
Let’s talk tools!

DEMO TIME
Also plenty of open questions about tools

- How to handle overlapping data?
- Many small tools vs a few mega-tools?
- Truly re-usable and integrate-able tools
  - More of a software engineering challenge...
- Need to know what you are looking for up-front...
  - Tools that automatically identify problematic areas in a trace?
- Tools need to indicate which events they rely on
- Chicken or the egg: tools or qlog support?

- Which tools would you use? (which do you use today?)
QUIC visualization: bug/behaviour examples
Extra slides / potential question support
QUICvis examples: connectivity lost

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

Transmitted packets (TX)
Received packets (RX)
Frames per stream (TX + RX)
QUICvis examples:
Duplicate packet nr
### QUICvis: Flow and congestion control logic

<table>
<thead>
<tr>
<th>QUIC Connection 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hide</strong></td>
</tr>
<tr>
<td><strong>Hide</strong></td>
</tr>
<tr>
<td><strong>Hide</strong></td>
</tr>
<tr>
<td><strong>Hide</strong></td>
</tr>
<tr>
<td><strong>Hide</strong></td>
</tr>
<tr>
<td><strong>Hide</strong></td>
</tr>
</tbody>
</table>

- **Transmitted packets (TX)**
- **Received packets (RX)**
- **Frames per stream (TX + RX)**
- **Frames:**
  - STREAM (data)
  - MAX_STREAM_DATA, MAX_DATA, BLOCKED (flow control)
  - ACK (congestion control)
  - PATH_CHALLENGE, PATH_RESPONSE (security)

**I**: Data segment (STREAM frame)

- **F**: 4K
- **R**: 8K
- **U**: 12K
QUICvis: Flow and congestion control logic

10% packet loss

Stream: 4 8 12 16 20
- MAX_STREAM_DATA (stream level)
- Data segment (STREAM frame)

MAX_DATA (connection level)
- Congestion window (CWND)
- Data allowance (CWND - bytes in flight)
Sending data along with BLOCKED, going over the limit

Client sends erroneous flow control allowances

Server sends BLOCKED, accompanied by STREAM, going over the max_data

- **payloadinfo**
  - **Max_data**
    - frametype: 4
    - maximum_data: 102400
  - **Max_str_data**
    - frametype: 5
    - stream_id: 4
    - maximum_data: 204800

- **Blocked**
  - frametype: 8
  - offset: 102400

- **Padding**
  - frametype: 0
  - length: 51

- **Stream**
  - frametype: 22
  - type_flags: { "off_flag": true, "len_flag": true, "fin_flag": false }
  - stream_id: 4
  - offset: 101460
  - length: 1140
  - stream_data: 626f726973206e69736920757420616c6174696f6e6172
Keep sending data VS flood of BLOCKED
Server retransmits too much, client answers to each blocked
Pacing (network, not server)

Server sends all at once

Client and network see very spaced-out
Pacing (server, not network)

Server sends interleaved itself

Server sends all at once at first
Extra slides
QUIC and HTTP/3

- Many people will be looking into the behavior
  - Initial implementations + conformance testing (current stage)
  - Early and at-scale deployments
  - Academic research (and teaching!)

- Cycle starts over with new features in v2
Many use cases

- Debugging
- Live deployment
- Education
- New feature development
- Large scale verification
In the wild, things start getting hairy real quick: bufferbloat

![Diagram showing network traffic flow from users to Origin via the Internet and Caching Proxy.]

**Expected:**

- uncached.js?v=xyz
- data.json
- picture1.jpg
- picture2.jpg

**Actual:**

- uncached.js?v=xyz
- data.json
- picture1.jpg
- picture2.jpg


https://github.com/andydavies/http2-prioritization-issues
Standard logging: existing alternatives

- HTTP/2 debug state
  - .json response for .well-known/h2/state
  - High-level summary of internal h2 state
  - Poll-based, manually diff changes between states

```json
"streams": {
  "5": {
    "state": "HALF_CLOSED_REMOTE",
    "flowIn": 65535,
    "flowOut": 6291456,
    "dataIn": 0,
    "dataOut": 0,
    "paddingIn": 0,
    "paddingOut": 0,
    "created": 1470835059.619137
  },
  "7": {
    "state": "OPEN",
    "flowIn": 65535,
    "flowOut": 6291456,
    "queuedData": 59093,
  }
},
```

- Low overhead
- Coarse grained

Standard logging: existing alternatives

- **NetLog (Chromium)**
  - .json log of **full browser window**
  - Medium-level (no congestion stuff, prioritization, loss, ...)
  - **Event-based**, one entry for every state change

```
@186143 [st= 40] QUIC_SESSION_STREAM_FRAME_RECEIVED
  --> fin = true
  --> length = 0
  --> offset = "0"
  --> stream_id = 5

@186143 [st= 40] QUIC_CHROMIUM_CLIENT_STREAM_READ_RESPONSE_HEADERS
  --> :status: 304
  age: 187
  alt-svc: quic=":443"; ma=2592000; v="46,44,43,39"
  etag: "1552399307"

@186158 [st= 55] QUIC_CHROMIUM_CLIENT_STREAM_SEND_REQUEST_HEADERS
  --> :authority: i.yimg.com
  :method: GET
  :path: /v1/v8QiKkd4-ms/hqdefault.jpg?sqp=-oaymwEYI\n  :scheme: https
  accept: image/webp,image/apng,image/*;*/q=0.8
  accept-encoding: gzip, deflate, br
  accept-language: en-US,en;q=0.9,nl;q=0.8
  referer: https://www.youtube.com/
  user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; )
  --> quic_priority = 3
  --> quic_stream_id = 7
```

- Event correlation to "sources"
- Event **phase**: start, end, none

Finer grained

High overhead
Standard logging: existing alternatives

- quic-trace
  - .json response (from protocolbuffer)
  - Low-level (focus on congestion control and loss)
  - Event-based, one entry for every state change

```
enum EventType {
  UNKNOWN_EVENT = 0;
  PACKET_SENT = 1;
  PACKET_RECEIVED = 2;
  PACKET_LOST = 3;
  APPLICATIONLIMITED = 4;
  EXTERNAL_PARAMETERS = 5;
};

enum TransmissionReason {
  NORMAL_TRANSMISSION = 0;
  TAILLOSS_PROBE = 1;
  RTO_TRANSMISSION = 2;
  PROBING_TRANSMISSION = 3;
};
```

- Reasons logged explicitly

Finer grained

High overhead

https://github.com/google/quic-trace