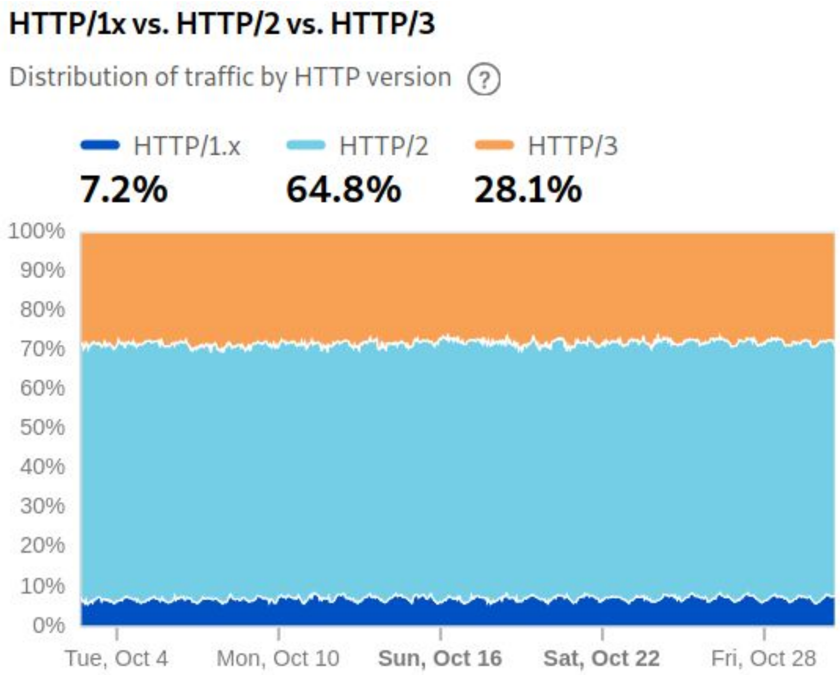
serde for SERious DEbugging

Lucas Pardue

HTTP Workshop 2022

HTTP is increasingly binary and multiplexed



HTTP/1x vs. HTTP/2 vs. HTTP/3

Distribution of traffic by HTTP version ?

HTTP/1.x	HTTP/2	HTTP/3
7.2%	64.8%	28.1%

Data from https://radar.cloudflare.com/adoption-and-usage?range=28d captured on October 31, 2022

ERR_SPDY_PROTOCOL_ERROR

ERR_HTTP2_PROTOCOL_ERROR

ERR_QUIC_PROTOCOL_ERROR

Question to the room:

What kinds of behaviours in HTTP/2 or HTTP/3 tend to lead to these sorts of problems?

“Logs or it didn’t happen”

Every dev, ever and always

qlog - structured logging by endpoints

Implementations often have logging that can enhance or augment packet captures.

A common logging format can encourage an ecosystem of analysis tools. E.g. what is an endpoint producing and why is it doing that?

draft-ietf-quic-qlog-main-schema: a base schema defined in Concise Data Definition Language (CDDL; RFC 8610). Highly extensible. Many possible serialization formats.

Draft-ietf-quic-qlog-quic-events, draft-ietf-quic-qlog-h3-events: concrete definitions to cover events related to packets and frames, security, congestion control etc.

qlog CDDL examples

```
HTTPFrameCreated = {
    stream_id: uint64
    ? length: uint64
    frame: $HTTPFrame
    ? raw: RawInfo
}

HTTPFrameParsed = {
    stream_id: uint64
    ? length: uint64
    frame: $HTTPFrame
    ? raw: RawInfo
}
```

```
; The HTTPFrame is any key-value map (e.g., JSON object)
$HTTPFrame /= {
    * text => any
}

$HTTPFrame /= HTTPBaseFrames

HTTPBaseFrames =  HTTPDataFrame / HTTPHeadersFrame /
             HTTPCancelPushFrame / HTTPSettingsFrame /
             HTTPPushPromiseFrame / HTTPGoawayFrame /
             HTTPMaxPushIDFrame / HTTPReservedFrame /
             HTTPUnknownFrame


HTTPHeadersFrame = {
    frame_type: "headers"
    headers: [* HTTPField]
}

HTTPField = {
    name: text
    value: text
}
```

qlog example

Client: `QLOGDIR=qlogs quiche-client --no-verify --wire-version 1 https://127.0.0.1:4433/index.html`

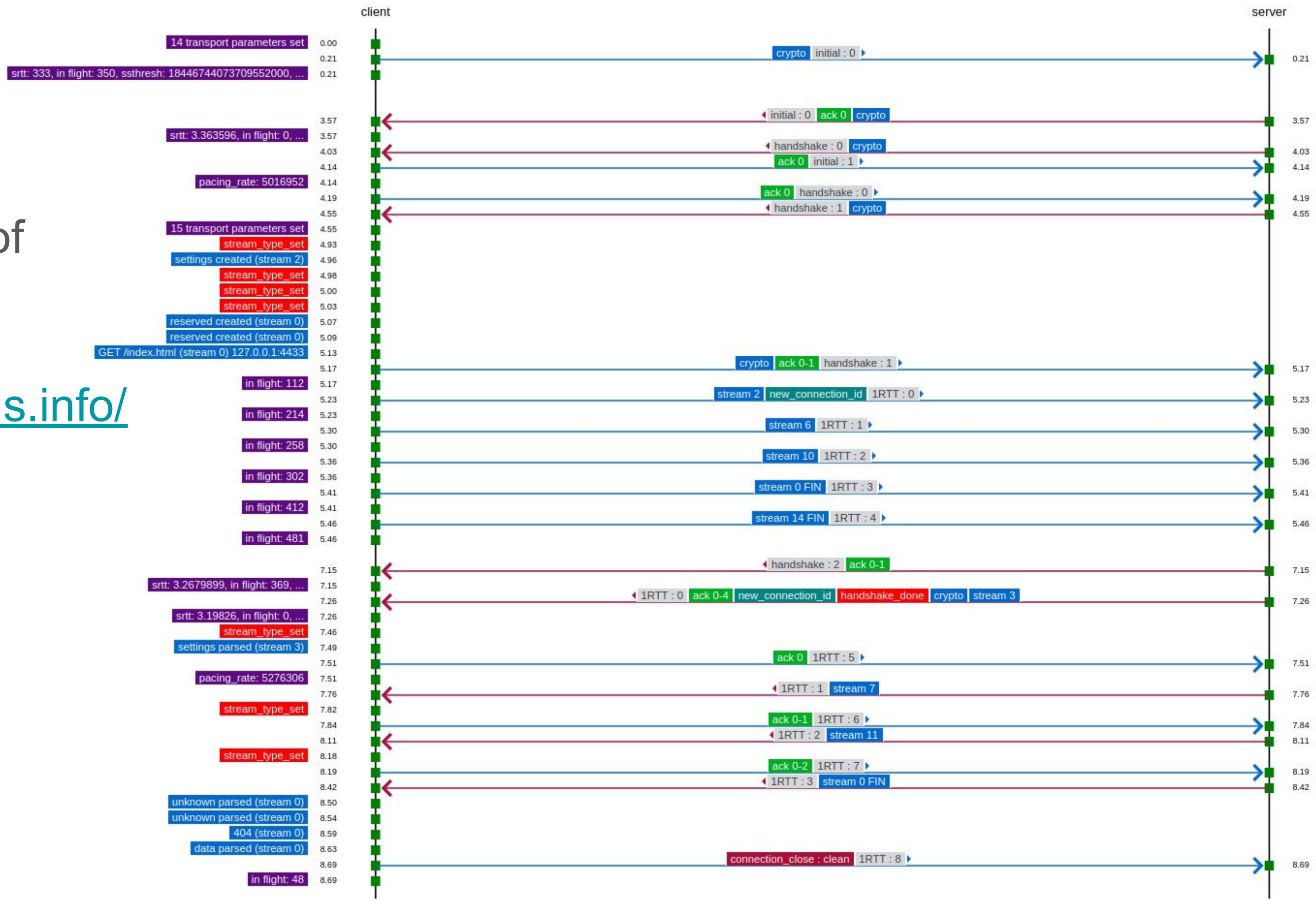
Server: `QLOGDIR=qlogs quiche-server --no-retry`

```
{"qlog_version":"0.3","qlog_format":"JSON-SEQ","title":"quiche-client qlog","description":"quiche-client qlog
id=9463b9d6695a7b2d189da2871fc255977bc7c6f8","trace":{"vantage_point":{"type":"client"},"title":"quiche-client
qlog","description":"quiche-client qlog id=9463b9d6695a7b2d189da2871fc255977bc7c6f8","configuration":{"time_offset":0.0}}}
{"time":0.0,"name":"transport:parameters_set","data":{"owner":"local","tls_cipher":"None","disable_active_migration":true,"max_idle_t
imeout":30000,"max_udp_payload_size":1350,"ack_delay_exponent":3,"max_ack_delay":25,"active_connection_id_limit":2,"initial_max_data"
:10000000,"initial_max_stream_data_bidi_local":1000000,"initial_max_stream_data_bidi_remote":1000000,"initial_max_stream_data_uni":10
00000,"initial_max_streams_bidi":100,"initial_max_streams_uni":100}}
{"time":0.207949,"name":"transport:packet_sent","data":{"header":{"packet_type":"initial","packet_number":0,"version":"1","scil":20,"
dcil":16,"scid":"9463b9d6695a7b2d189da2871fc255977bc7c6f8","dcid":"6c94d2c299cbff6253a202bcb20ceb42"},"raw":{"length":350,"payload_le
ngth":287},"send_at_time":0.207949,"frames":[{"frame_type":"crypto","offset":0,"length":283}]}}
{"time":0.207949,"name":"recovery:metrics_updated","data":{"smoothed_rtt":333.0,"rtt_variance":166.5,"congestion_window":13500,"bytes
_in_flight":350,"ssthresh":18446744073709551615}}
{"time":3.5715451,"name":"transport:packet_received","data":{"header":{"packet_type":"initial","packet_number":0,"version":"1","scil"
:20,"dcil":20,"scid":"78015def011d1adf3af94c44067955dd4d52fc70","dcid":"9463b9d6695a7b2d189da2871fc255977bc7c6f8"},"raw":{"length":12
00,"payload_length":117},"frames":[{"frame_type":"ack","ack_delay":0.305,"acked_ranges":[[0,0]]},{"frame_type":"crypto","offset":0,"l
ength":90}]}}
```

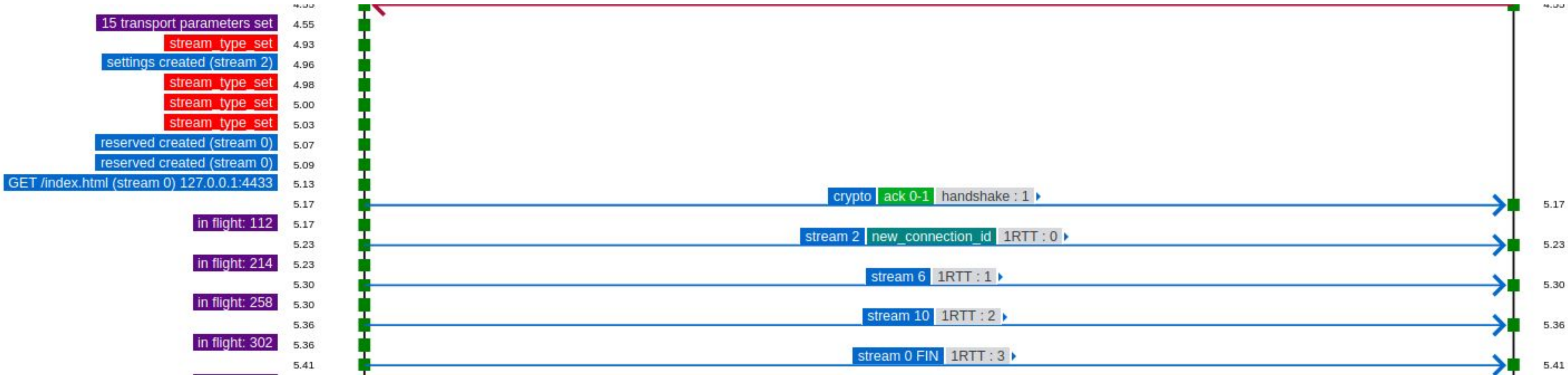
qvis

Making sense out of oodles of data

https://qvis.quictools.info/


client
server
14 transport parameters set 0.00
0.21 crypto initial : 0 0.21
srtt: 333, in flight: 350, ssthresh: 18446744073709552000, ... 0.21
3.57 initial : 0 ack 0 crypto 3.57
srtt: 3.363596, in flight: 0, ... 3.57
4.03 handshake : 0 crypto 4.03
4.14 ack 0 initial : 1 4.14
pacing_rate: 5016952 4.14
4.19 ack 0 handshake : 0 4.19
4.55 handshake : 1 crypto 4.55
15 transport parameters set 4.55
stream_type_set 4.93
settings created (stream 2) 4.96
stream_type_set 4.98
stream_type_set 5.00
stream_type_set 5.03
reserved created (stream 0) 5.07
reserved created (stream 0) 5.09
GET /index.html (stream 0) 127.0.0.1:4433 5.13
5.17 crypto ack 0-1 handshake : 1 5.17
in flight: 112 5.17
5.23 stream 2 new_connection_id 1RTT : 0 5.23
in flight: 214 5.23
5.30 stream 6 1RTT : 1 5.30
in flight: 258 5.30
5.36 stream 10 1RTT : 2 5.36
in flight: 302 5.36
5.41 stream 0 FIN 1RTT : 3 5.41
in flight: 412 5.41
5.46 stream 14 FIN 1RTT : 4 5.46
in flight: 481 5.46
7.15 handshake : 2 ack 0-1 7.15
srtt: 3.2679899, in flight: 369, ... 7.15
7.26 1RTT : 0 ack 0-4 new_connection_id handshake_done crypto stream 3 7.26
srtt: 3.19826, in flight: 0, ... 7.26
stream_type_set 7.46
settings parsed (stream 3) 7.49
7.51 ack 0 1RTT : 5 7.51
pacing_rate: 5276306 7.51
7.76 1RTT : 1 stream 7 7.76
stream_type_set 7.82
7.84 ack 0-1 1RTT : 6 7.84
8.11 1RTT : 2 stream 11 8.11
stream_type_set 8.18
8.19 ack 0-2 1RTT : 7 8.19
8.42 1RTT : 3 stream 0 FIN 8.42
unknown parsed (stream 0) 8.50
unknown parsed (stream 0) 8.54
404 (stream 0) 8.59
data parsed (stream 0) 8.63
8.69 connection_close : clean 1RTT : 8 8.69
in flight: 48 8.69

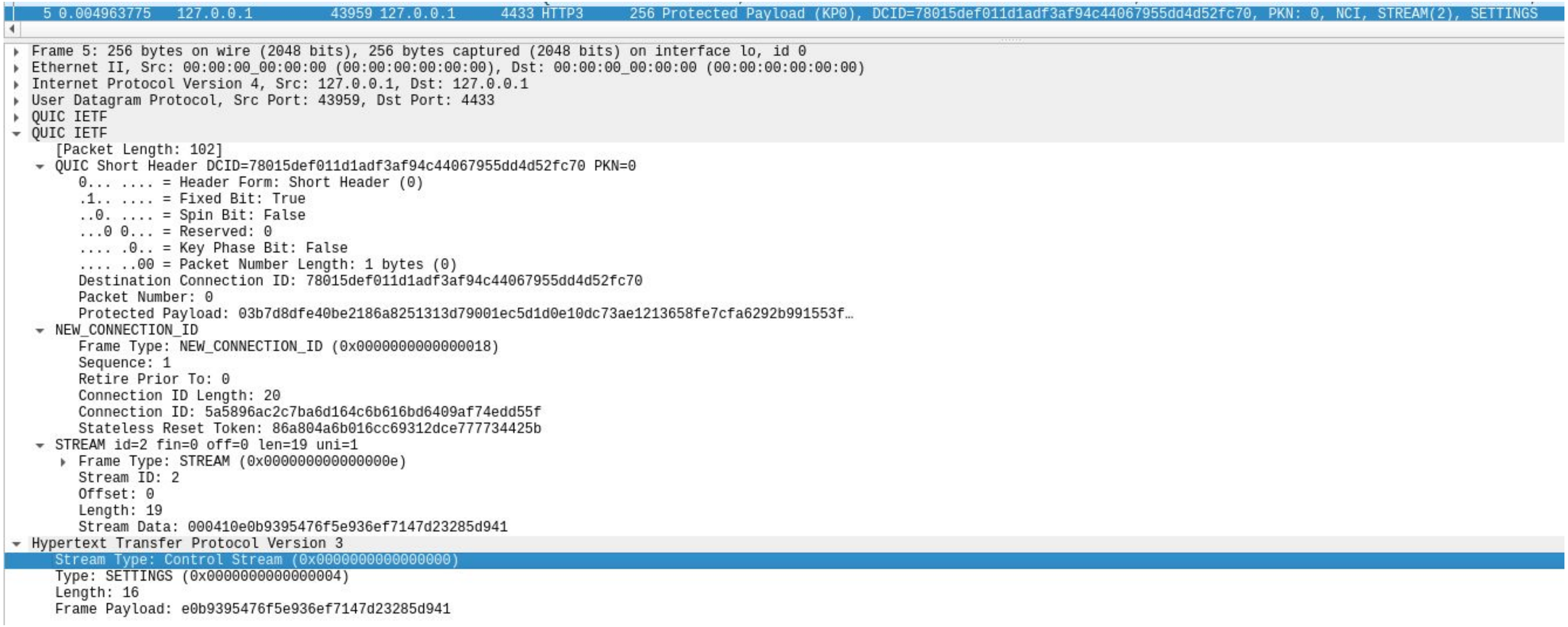
Streams example: HTTP/3


15 transport parameters set 4.55
stream_type_set 4.93
settings created (stream 2) 4.96
stream_type_set 4.98
stream_type_set 5.00
stream_type_set 5.03
reserved created (stream 0) 5.07
reserved created (stream 0) 5.09
GET /index.html (stream 0) 127.0.0.1:4433 5.13
5.17
in flight: 112 5.17
5.23
in flight: 214 5.23
5.30
in flight: 258 5.30
5.36
in flight: 302 5.36
5.41
crypto ack 0-1 handshake : 1 5.17
stream 2 new_connection_id 1RTT : 0 5.23
stream 6 1RTT : 1 5.30
stream 10 1RTT : 2 5.36
stream 0 FIN 1RTT : 3 5.41

Control stream on ID 2. QPACK streams on ID 6 and 10.

Request stream on ID 0. GET request for /index.html. Stream is FIN’d to indicate request message is complete

Streams example: HTTP/3

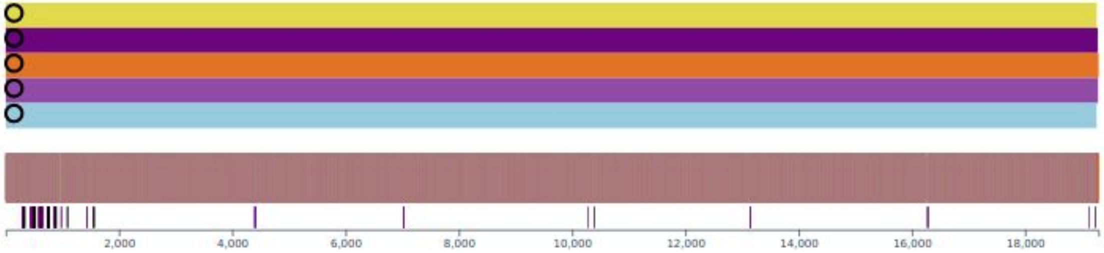


```
    5 0.004963775    127.0.0.1          43959 127.0.0.1      4433 HTTP3      256 Protected Payload (KP0), DCID=78015def011d1adf3af94c44067955dd4d52fc70, PKN: 0, NCI, STREAM(2), SETTINGS

Frame 5: 256 bytes on wire (2048 bits), 256 bytes captured (2048 bits) on interface lo, id 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 43959, Dst Port: 4433
QUIC IETF
QUIC IETF
   [Packet Length: 102]
   QUIC Short Header DCID=78015def011d1adf3af94c44067955dd4d52fc70 PKN=0
      0... .... = Header Form: Short Header (0)
      .1.. .... = Fixed Bit: True
      ..0. .... = Spin Bit: False
      ...0 0... = Reserved: 0
      .... .0.. = Key Phase Bit: False
      .... ..00 = Packet Number Length: 1 bytes (0)
      Destination Connection ID: 78015def011d1adf3af94c44067955dd4d52fc70
      Packet Number: 0
      Protected Payload: 03b7d8dfe40be2186a8251313d79001ec5d1d0e10dc73ae1213658fe7cfa6292b991553f…
   NEW_CONNECTION_ID
      Frame Type: NEW_CONNECTION_ID (0x0000000000000018)
      Sequence: 1
      Retire Prior To: 0
      Connection ID Length: 20
      Connection ID: 5a5896ac2c7ba6d164c6b616bd6409af74edd55f
      Stateless Reset Token: 86a804a6b016cc69312dce777734425b
   STREAM id=2 fin=0 off=0 len=19 uni=1
      Frame Type: STREAM (0x000000000000000e)
      Stream ID: 2
      Offset: 0
      Length: 19
      Stream Data: 000410e0b9395476f5e936ef7147d23285d941
Hypertext Transfer Protocol Version 3
   Stream Type: Control Stream (0x0000000000000000)
   Type: SETTINGS (0x0000000000000004)
   Length: 16
   Frame Payload: e0b9395476f5e936ef7147d23285d941
```

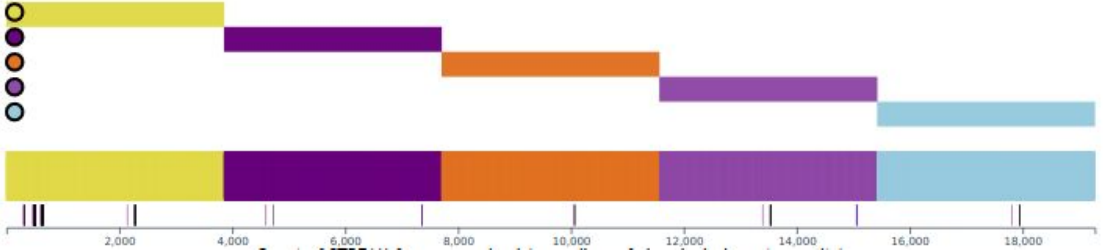
Example: HTTP/3 prioritization shown in qvis

5 concurrent transfers of 5 MB, all urgency=1

quiche (before priorities)
round-robin



quiche (now)
FIFO



qvis is great (but has some limitations)

Browser-based tool means it’s instantly accessible

Large qlogs slow to load or process. Too large they’ll crash the browser process

Qvis is quite an interactive tool. Good for point investigations. Less good for bulk investigations over many files.

Qlog is usually just JSON or JSON-SEQ. Pre-processing with tools like jq can help a lot but are not schema aware.

Quiche qlogs

https://crates.io/crates/qlog - A general rust crate for serialization and deserialization of qlog. Quiche uses this for qlog serialization to JSON-SEQ. Firefox uses the crate too.

CDDL

```
HTTPFrameCreated = {
    stream_id: uint64
    ? length: uint64
    frame: $HTTPFrame
    ? raw: RawInfo
}
```

Rust

```
#[serde_with::skip_serializing_none]
#[derive(Serialize, Deserialize, Clone,
PartialEq, Eq, Debug)]
pub struct H3FrameCreated {
    pub stream_id: u64,
    pub length: Option<u64>,
    pub frame: Http3Frame,
    pub raw: Option<RawInfo>,
}
```

Rust Serde

https://serde.rs/ - is a framework for serializing and deserializing Rust data structures efficiently and generically.

The Serde ecosystem consists of data structures that know how to serialize and deserialize themselves along with data formats that know how to serialize and deserialize other things. Serde provides the layer by which these two groups interact with each other, allowing any supported data structure to be serialized and deserialized using any supported data format.

JSON, Postcard, CBOR, YAML, MessagePack, TOML, Pickle, ROS, BSON, Avro, JSON5, URL, Envy, S-expressions, D-Bus, FlexBuffers, Bencode, DynamoDb, Hjson, …

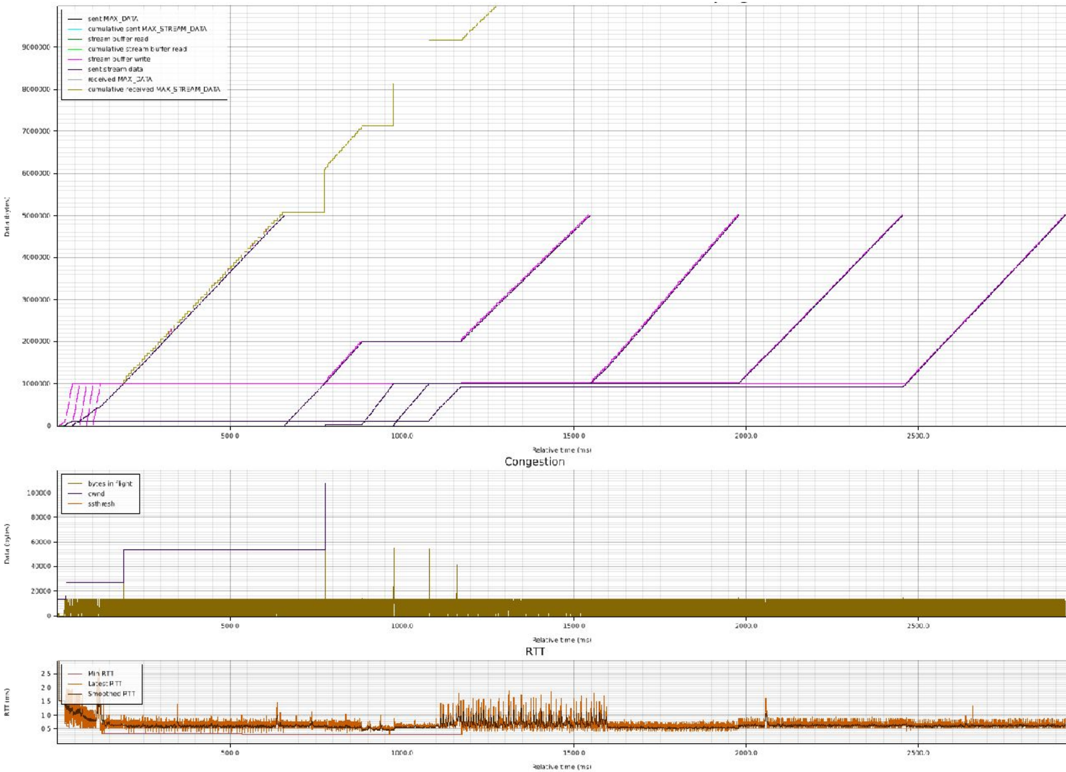
Roundtrip example

```
use serde::{Serialize, Deserialize};

#[derive(Serialize, Deserialize, Debug)]
struct Point {
        x: i32,
        y: i32,
}

fn main() {
        let point = Point { x: 1, y: 2 };

        // Convert the Point to a JSON string.
        let serialized = serde_json::to_string(&point).unwrap();

        // Prints serialized = {"x":1,"y":2}
        println!("serialized = {}", serialized);

        // Convert the JSON string back to a Point.
        let deserialized: Point = serde_json::from_str(&serialized).unwrap();

        // Prints deserialized = Point { x: 1, y: 2 }
        println!("deserialized = {:?}", deserialized);
}
```

qlog-dancer

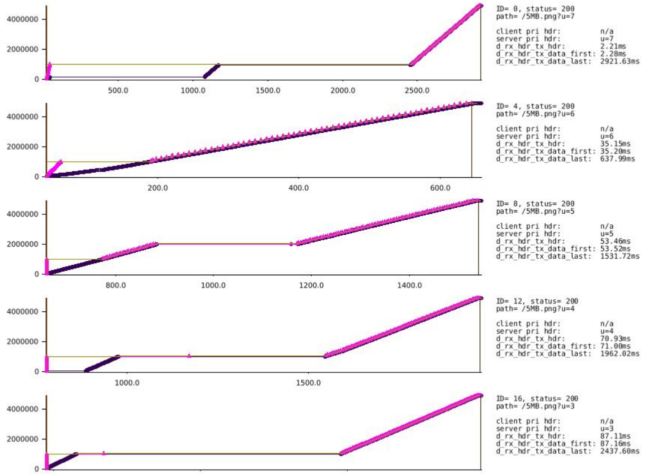
An internal tool. qlog crate powers deserialization. Application responsible for file handling, data munging and plotting (using [Plotters](#) library).

Builds on the shoulders of giants for congestion control oriented plots.

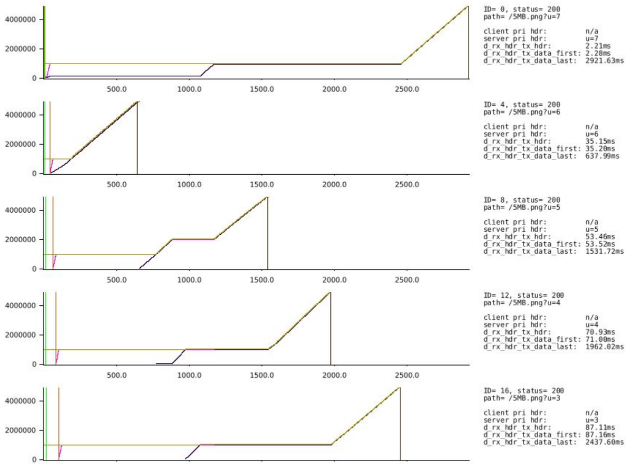


Qlog-dancer streams

Lots of spooky behavioural oddities happen when we cross the streams.



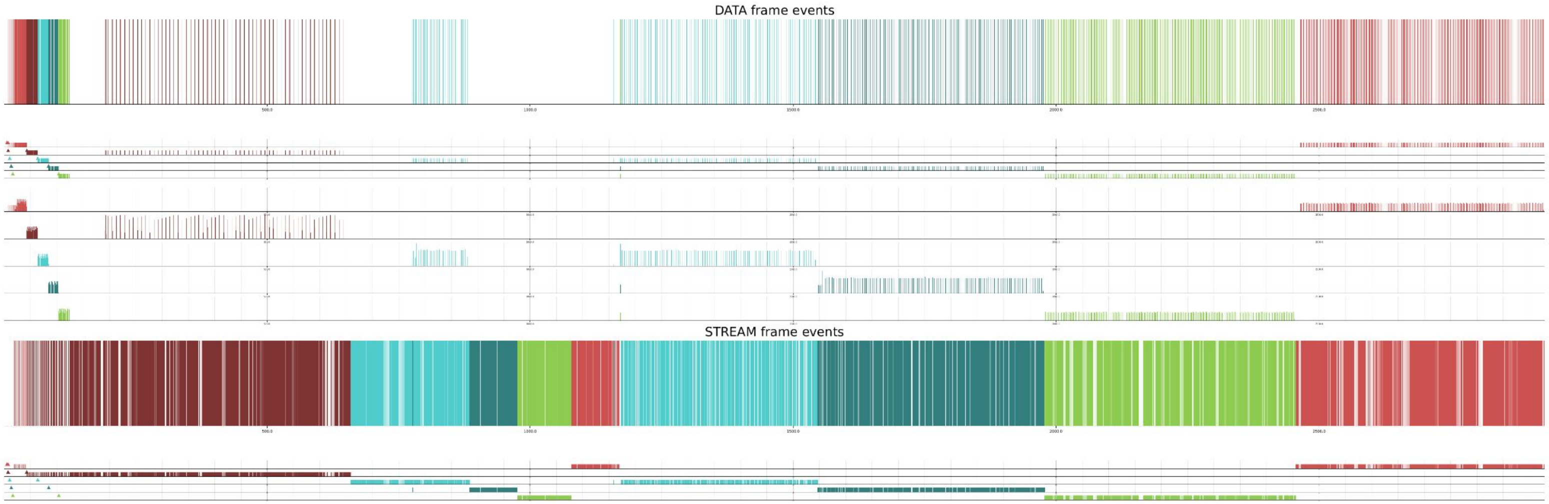
```
ID= 0, status= 200
path= /5MB.png?u=7

client pri hdr:          n/a
server pri hdr:          u=7
d_rx_hdr_tx_hdr:         2.21ms
d_rx_hdr_tx_data_first: 2.28ms
d_rx_hdr_tx_data_last:  2921.63ms

ID= 4, status= 200
path= /5MB.png?u=6

client pri hdr:          n/a
server pri hdr:          u=6
d_rx_hdr_tx_hdr:         35.15ms
d_rx_hdr_tx_data_first: 35.20ms
d_rx_hdr_tx_data_last:  637.99ms

ID= 8, status= 200
path= /5MB.png?u=5

client pri hdr:          n/a
server pri hdr:          u=5
d_rx_hdr_tx_hdr:         53.46ms
d_rx_hdr_tx_data_first: 53.52ms
d_rx_hdr_tx_data_last:  1531.72ms

ID= 12, status= 200
path= /5MB.png?u=4

client pri hdr:          n/a
server pri hdr:          u=4
d_rx_hdr_tx_hdr:         70.93ms
d_rx_hdr_tx_data_first: 71.00ms
d_rx_hdr_tx_data_last:  1962.02ms

ID= 16, status= 200
path= /5MB.png?u=3

client pri hdr:          n/a
server pri hdr:          u=3
d_rx_hdr_tx_hdr:         87.11ms
d_rx_hdr_tx_data_first: 87.16ms
d_rx_hdr_tx_data_last:  2437.60ms
```



```
ID= 0, status= 200
path= /5MB.png?u=7

client pri hdr:          n/a
server pri hdr:          u=7
d_rx_hdr_tx_hdr:         2.21ms
d_rx_hdr_tx_data_first: 2.28ms
d_rx_hdr_tx_data_last:  2921.63ms

ID= 4, status= 200
path= /5MB.png?u=6

client pri hdr:          n/a
server pri hdr:          u=6
d_rx_hdr_tx_hdr:         35.15ms
d_rx_hdr_tx_data_first: 35.20ms
d_rx_hdr_tx_data_last:  637.99ms

ID= 8, status= 200
path= /5MB.png?u=5

client pri hdr:          n/a
server pri hdr:          u=5
d_rx_hdr_tx_hdr:         53.46ms
d_rx_hdr_tx_data_first: 53.52ms
d_rx_hdr_tx_data_last:  1531.72ms

ID= 12, status= 200
path= /5MB.png?u=4

client pri hdr:          n/a
server pri hdr:          u=4
d_rx_hdr_tx_hdr:         70.93ms
d_rx_hdr_tx_data_first: 71.00ms
d_rx_hdr_tx_data_last:  1962.02ms

ID= 16, status= 200
path= /5MB.png?u=3

client pri hdr:          n/a
server pri hdr:          u=3
d_rx_hdr_tx_hdr:         87.11ms
d_rx_hdr_tx_data_first: 87.16ms
d_rx_hdr_tx_data_last:  2437.60ms
```

Qlog-dancer streams

Lots of spooky behavioural oddities happen when we cross the streams.



A real-world failure (1)

Reporter: Downloads are in Chrome are **behaving weirdly**. Seems it is worse with larger files.

Responder: **Logs or it didn’t happen**. Also, got a repro?

Reporter: Downloads aren’t captured in HAR files*. I can’t get any other logs. I can describe the repro.

Responder: What about Chrome netlog?

Reporter: **What is Chrome netlog**?

Responder: chrome://net-export/

Reporter: OK, I captured you a netlog at the point the problem happened. While I was doing a dozen other things in the same browsing session.

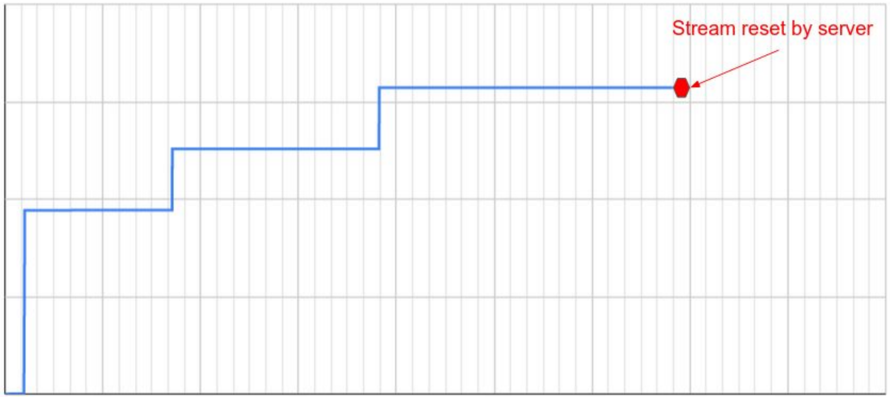
Responder: D’oh. Don’t worry, I’ll recreate the repro and make my own logs.

* Narrator: we will never find out why HAR doesn’t cover downloads. But also, for any HTTP/2 or HTTP/3 related error, HARs tend to be next to a bit rubbish.

A real-world failure (2)

```
#! /bin/bash

RAND_VAL=$RANDOM
PROFILE_PATH="${HOME}/.temp-chrome-${RAND_VAL}"
#PROFILE_PATH=$HOME/temp-chrome
mkdir $PROFILE_PATH
echo "using temporary profile at ${PROFILE_PATH}"

NETLOG_FILE="${HOME}/netlog_${RAND_VAL}.json"
echo "logging netlog to ${NETLOG_FILE}"

# launch a fresh Chrome profile without annoying first-time checks and have it
immediately start netlogging
google-chrome --user-data-dir=$PROFILE_PATH --disable-fre --no-default-browser-check
--no-first-run --log-net-log=$NETLOG_FILE --auto-open-devtools-for-tabs $1

# tidy up leftovers we don't want to persist
rm -r $PROFILE_PATH
```

A real-world failure (3)



Stream reset by server

- Netlog viewer crashes opening this jumbo log
- Manually data munge
- In file, lookup HTTP2_SESSION_RECV_DATA in the constants block and note it's number (it was 209 for me but it might vary?),
- do the same for HTTP2_SESSION_RECV_RST_STREAM (199 for me).
- Then delete the constants block from the netlog
- Run this query "jq '.events[] | select(.params.stream_id==7 and .type==209) | [.time, .params.size] | @tsv' chrome-net-export-log-lucas.json > stream_data.tsv".
- Plot the data in some tool like google sheets
- Find a way to never have to do this manually again.

Apply learnings from qlog to netlog

Qlog and netlog both JSON-ish formats

Qlog-dancer can parsing large files fast. Reuse its framework.

Just need to see the netlog schema and write some serde-compatible structures.

I HAVE A LOG FORMAT THAT CAPTURES ALL THE DATA YOU NEED
AND IT HAS A DOCUMENTED SCHEMA, RIGHT?
imgflip.com
AND IT HAS A DOCUMENTED SCHEMA, RIGHT?

Reverse-engineering netlog to serde

```
#[derive(Serialize, Deserialize, Debug, Default)]
pub struct Http3DataFrameReceivedParams {
     pub payload_length: u64,
     pub stream_id: u64,
}

#[derive(Serialize, Deserialize, Debug, Default)]
pub struct Http3DataFrameReceivedEvent {
     pub params: Http3DataFrameReceivedParams,
}
```

Reverse-engineering netlog to serde

```
fn parse_netlog_h3_event(
     session: &mut Vec<(f32, netlog::Event)>,
     event_hdr: &netlog::EventHeader,
     event: &[u8], verbose: bool,
) {
     match event_hdr.ty_string.as_str() {
          "HTTP3_DATA_FRAME_RECEIVED" => {
          let ev: Http3DataFrameReceivedEvent =
               serde_json::from_slice(event).unwrap();
          session.push((
               netlog_time_delta(event_hdr),
               netlog::Event::H3(h3::Event::Http3DataFrameReceived(ev)),
          ));
     },

     _ =>()
}
```

Summary

It would be great if there was one true logging format. But that’s not realistic.

Well defined logging format can encourage an ecosystem of analysis tools. E.g. what is an endpoint producing and why is it doing that?

Tools compliment each other. Tools provide more value when they address user needs. Keeping tools internal reduces user base.

Is there any interest in some of the work presented? E.g., open sourcing of qlog-dancer or netlog library, trying to define schema for netlog or HAR?