[qlog]

structured event logging

The philosophical update

Robin Marx robin.marx@kuleuven.be

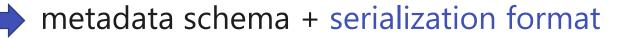
The story so far

Log events directly inside the endpoint implementations

- Packet captures require <u>full</u> decryption → worse for privacy/security
- Can add additional information (e.g., congestion window)

3 separate documents:

- Main schema
- QUIC and TLS eventsHTTP/3 and QPACK events



```
"metadata": {...},
"events": [{
  "time": 15000,
  "name": "transport:packet_received",
  "data": {
    "header": {
      "packet_type": "1rtt",
      "packet_number": 25
    },
    "frames": [
      "frame_type": "ack",
      "acked_ranges": [
        [10,15],
        [17,20]
    }]
  }},
```

```
class AckFrame{
    frame_type:string = "ack";

    ack_delay?:float; // in ms

    acked_ranges?:Array<[uint64, uint64]|[uint64]>;

    ect1?:uint64;
    ect0?:uint64;
    ce?:uint64;
}
```

```
"metadata": {...},
                                                         JSON
"events": [{
  "time": 15000,
  "name": "transport:packet_received",
  "data": {
    "header": {
      "packet_type": "1rtt",
      "packet_number": 25
    },
    "frames": [
                                                         class AckFrame{
                                                            frame_type:string = "ack";
      "frame_type": "ack",
                                                            ack_delay?:float; // in ms
      "acked_ranges": [
        [10,15],
                                                            acked_ranges?:Array<[uint64, uint64]|[uint64]>;
        [17,20]
                                                            ect1?:uint64;
    }]
                                                            ect0?:uint64;
  }},
                                                            ce?:uint64;
```

```
JSON and NDJSON
"time": 15000,
"name": "transport:packet_received",
                                                       qlog_format?:string = "JSON" | "NDJSON";
"data": {
  "header": {
    "packet_type": "1rtt",
    "packet_number": 25
  "frames": [
                                                    class AckFrame{
                                                       frame type:string = "ack";
    "frame_type": "ack",
                                                       ack_delay?:float; // in ms
    "acked_ranges": [
      [10,15],
                                                       acked ranges?:Array<[uint64, uint64]|[uint64]>;
      [17,20]
                                                       ect1?:uint64;
 }]
                                                       ect0?:uint64;
}},
                                                       ce?:uint64;
```

```
JSON and NDJSON
"time": 15000,
"name": "transport:packet_received",
                                                       qlog_format?:string = "JSON" | "NDJSON";
"data": {
  "header": {
    "packet_type": "1rtt",
    "packet_number": 25
  "frames": [
                                                   class AckFrame{
                                                       frame type:string = "ack";
    "frame_type": "ack",
                                                       ack_delay?:float; // in ms
    "acked_ranges": [
      [10,15],
                                                       acked ranges?:Array<[uint64, uint64]|[uint64]>;
      [17,20]
                                                       ect1?:uint64;
 }]
                                                       ect0?:uint64;
}},
                                                                               Data Definition Language
                                                       ce?:uint64;
                                                                                       not for today;)
```

Today

What do we actually standardize and why?

Part 1: The JSON in the room

JSON pros:

- Broadly supported → browser-based tooling, scripting libraries
- Plaintext → re-use existing tools (jq, sed/awk/grep/..., YOU), fprintf("%s")

JSON cons:

- Slow
- Verbose
- NDJSON isn't actually standardized anywhere yet...

Alternatives:

- CBOR
- Protobuffers/flatbuffers/...
- PCAPNG

- ...

Part 1: What is the goal for qlog?

Optimize for interoperable/reusable tools?

VS

Optimize for direct output/storage/transfer?

Part 1: What is the goal for qlog?

Optimize for interoperable/reusable tools?

Optimize for direct output/storage/transfer?

Is this even needed?

- Direct JSON is feasible
 - mvfst, quic-go
- Log optimized, convert
 - quicly, picoquic
 - chromium (kind of)

Compress

500MB_0ms_lsquic							
format	raw (MB)	%	gzip6 (MB)	%	brotli4 (MB)	%	
pcap	561.57	203.45	529.01	191.65	528.85	191.60	4
qlog	276.02	100.00	19.15	6.94	19.40	7.03	
cbor	215.53	78.08	17.78	6.44	18.90	6.85	
qlog_lookup	155.89	56.48	17.25	6.25	17.99	6.52	
cbor_lookup	90.85	32.91	15.18	5.50	13.18	4.77	4
protobuf	66.15	23.96	14.56	5.27	10.71	3.88	

Part 1: Proposal

Stick to JSON + NDJSON

- Optimize for text-based and browser-based processing
- Even loading large JSON files should be feasible
 - Not in qvis/browser, but surely in native apps

- Other documents can later define CBOR/PCAPNG/Protobuf/... if needed
 - Take care to make schema as generic as possible to allow easy mapping
 - You're free to use another format in your implementation (duh) and then write converter

- We do need to define NDJSON (or similar) properly ourselves then...

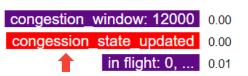
Part 2: which events do we include?

```
wire image
"time": 15000,
"name": "transport:packet_received",
"data": {
    "header": {
       "packet_type": "1rtt",
        "packet_number": 25
    "frames": [
       "frame_type": "ack",
        "acked_ranges": [
           [10,15],
           [17,20]
    }]
}}
```

```
internal state
"time": 15001,
"name": "recovery:metrics_updated",
"data": {
  "min_rtt": 25,
  "smoothed_rtt": 30,
  "latest_rtt": 25,
  "congestion_window": 60,
  "bytes_in_flight": 77000,
```

+ Custom events!

Tools MUST deal with unknown events



Part 2: 2 sides of the same coin

```
wire image
"time": 15000,
"name": "transport:packet_received",
"data": {
    "header": {
        "packet_type": "1rtt",
        "packet_number": 25
    "frames": [
        "frame_type": "ack",
        "acked_ranges": [
            [10,15],
            [17,20]
     }]
}}
```

```
state changes

{
"time": 15000,
"name": "transport:packets_acked",
"data": {
    "packet_numbers": [17,20]
}
Only newly
ACKed
```

Note: we also have a separate packet_lost event

Part 2: 3 sides of the same... triangle?

```
state changes
                                                wire image
"time": 15000,
                                                "time": 15000,
"name": "transport:packet_received",
                                                "name": "transport:packets_acked",
"data": {
                                                                                                Only newly
                                                "data": {
   "header": {
                                                  "packet_numbers": [17,20]
                                                                                                ACKed
       "packet_type": "1rtt",
        "packet_number": 25
    "frames": [
                                                   partial wire image
       "frame_type": "ack",
        "acked_ranges": [
           [10,15],
                                                   "time": 15000,
           [17,20]
                                                   "name": "transport:frames_processed",
                                                   "data": {
                                                                                               No packet
                                                     "frames": {
    }]
                                                                                               header
                                                        "frame_type": "ack",
}}
                                                        "acked_ranges": [
                                                           [10, 15],
                                                           [17,20]
                                                                                              Is this too tied to
                                                                                               implementation
                                                                                                  specifics?
```

Part 2: 4 sides of ... I give up

```
wire image
"time": 15000,
"name": "transport:packet_received",
"data": {
   "header": {
       "packet_type": "1rtt",
       "packet_number": 25
   "frames": [
       "frame_type": "ack",
       "acked_ranges": [
           [10,15],
           [17,20]
    }]
}}
```

```
"optimized" partial wire image
"time": 15000,
"name":"transport:frames_created",
"data":{
  "default_frame": {
    "frame_type": "stream",
   "stream_id":0,
   "length": 1000
 },
  "frames":[
   {"offset": 2000 },
   {"offset": 3000 },
   {"offset": 4000, "length": 500}
```

Often sending similar STREAM frames

Part 2: Explosion of events

All useful, but confusing

- qlog implementers: what to log when/where?
- Tool creators: which events to use? What if contradictions?
 - If tools only support a subset, what's the use of standardizing more?

We need guidelines/design philosphy

When should something be a new event / re-use event / be custom event?

Part 2: Re-use event types

```
instead of frames_processed
                                                             "time": 15000,
"name": "transport:packet_received",
                                                              "time": 15000,
"data": {
                                                              "name": "transport:packet_received",
   "header": {
                                                              "data": {
       "packet_type": "1rtt",
                                                                 "header": {
       "packet_number": 25
                                                                     "packet_number": 25
                                                 Tool
                                                                 "frames": [
}}
                                               couples
                                                                     "frame_type": "ack",
                                                                     "acked_ranges": [
                                              based on
                                                                         [10,15],
                                                                         [17,20]
                                                  PN
                                                                  }]
                                                             }}
```

When handling header

When handling payload

Part 2: Proposal

Pragmatism: rules with exceptions

- 1. Stay as close to wire image as possible
 - Only deviate for internal state
 - Makes tools mostly usable on pcaps as well

```
packet_sent +
congestion_metrics_updated
```

Part 2: Proposal

Pragmatism: rules with exceptions

- 1. Stay as close to wire image as possible
 - Only deviate for internal state
 - Makes tools mostly usable on pcaps as well

packet_sent +
congestion_metrics_updated

- 2. Prevent duplicate info logging
 - Only deviate for non-trivial internal state changes
 - packets_acked would be a good "exception to the rule"
 - QPACK wire image vs "dynamic_table_contents"

packets_acked

Part 2: Proposal

Pragmatism: rules with exceptions

- 1. Stay as close to wire image as possible
 - Only deviate for internal state
 - Makes tools mostly usable on pcaps as well

packet_sent +
congestion_metrics_updated

- 2. Prevent duplicate info logging
 - Only deviate for non-trivial internal state changes

packets_acked

- packets_acked would be a good "exception to the rule"
- QPACK wire image vs "dynamic_table_contents"

= no more frames_processed

If implementations need split (re-used) events/other logic:

→ Write custom converter to "proper" qlog for tools that don't support those

What do we actually <u>standardize</u>?

Proposal 1: JSON + NDJSON

Proposal 2: limit event options, similar to draft-01

getting consensus on these impacts ~75% of open issues

EXTRA

Part 1: what does it look like?

draft-01: csv + JSON

```
"event_fields": [
                        "column"
   "relative_time",
   "category",
   "event",
                           names
   "data"
 "events": [
     "transport",
     "packet_received",
     { header: {...}, frames: {...} }
```

```
mvfst
aioquic
quicly / H2O
f5
applequic
```

Part 1: what does it look like?

draft-01: csv + JSON

```
"event_fields": [
                         "column"
   "relative_time",
   "category",
   "event",
                           names
   "data"
 "events": [
     "transport",
     "packet_received",
     { header: {...}, frames: {...} }
```

```
mvfst - neqo
aioquic - picoquic
quicly / H2O - ats
f5 - applequic
```

draft-02: JSON

```
"events": [
    "time": 2,
    "name": "transport:packet_received",
    "data": {
      header: {...},
      frames: {...}
  },
```

```
quic-gongtcp2quichehaskellkwik
```

Part 1: what does it look like?

draft-01: csv + JSON

```
"event_fields": [
                         "column"
   "relative_time",
   "category",
   "event",
                           names
   "data"
 "events": [
     "transport",
     "packet_received",
     { header: {...}, frames: {...} }
```

```
- mvfst - neqo
- aioquic - picoquic
- quicly / H2O - ats
- f5 - applequic
```

draft-02: JSON + NDJSON

```
"time": 2,
  "name": "transport:packet_received",
  "data": {
    header: {...},
    frames: {...}
},
```

```
- quic-go
```

- ngtcp2
- quiche
- haskell
- kwik