[qlog]

structured event logging

The philosophical update

Robin Marx robin.marx@kuleuven.be

The story so far

[qlog] = QUIC Logging

Log events directly inside the endpoint implementations

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

Protocol agnostic in 3 parts:

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

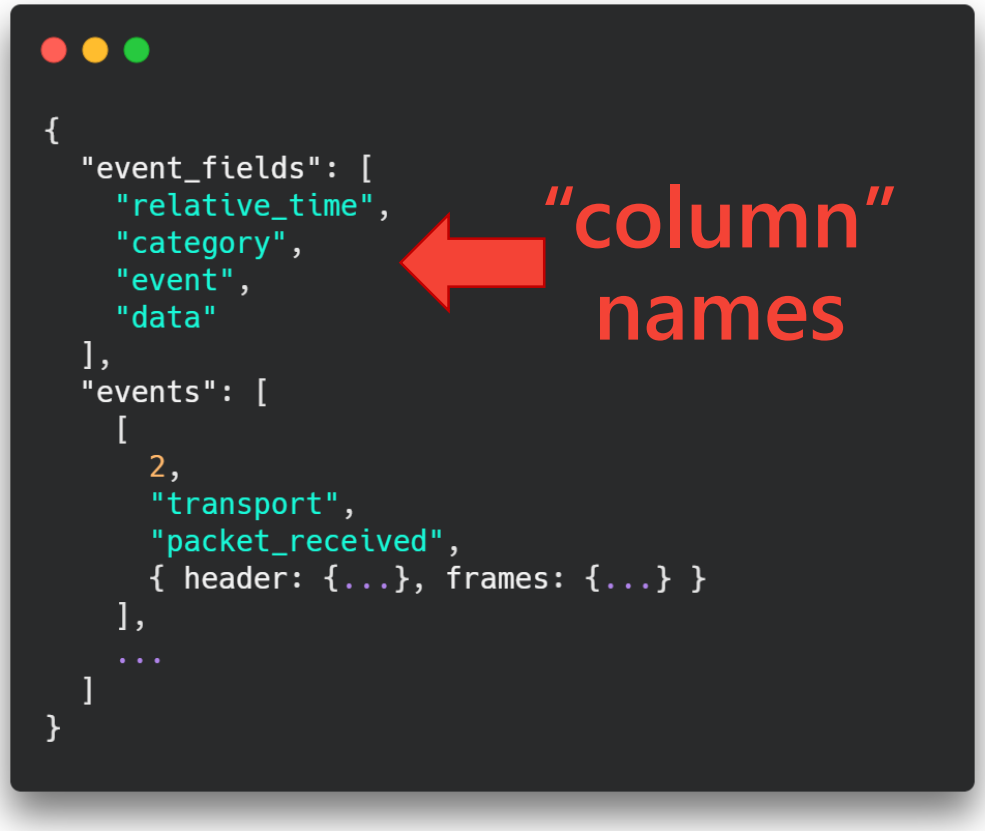
https://github.com/quicwg/qlog

What do we actually standardize and why?

Part 1: what does it look like?

Part 1: what does it look like?

draft-01: csv + JSON



- mvfst
- aioquic
- quicly / H2O
- f5
- neqo
- picoquic
- ats
- ...

https://github.com/quicwg/qlog

Part 1: what does it look like?

draft-01: csv + JSON

```
{
  "event_fields": [
    "relative_time",
    "category",
    "event",
    "data"
  ],
  "events": [
    [
      2,
      "transport",
      "packet_received",
      { header: {...}, frames: {...} }
    ],
    ...
  ]
}
```

"column" names

- mvfst
- aioquic
- quicly / H2O
- f5
- neqo
- picoquic
- ats
- ...

draft-02: JSON

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

- quic-go
- ngtcp2
- quiche
- haskell
- kwik

https://github.com/quicwg/qlog

Part 1: what does it look like?

draft-01: csv + JSON

```
{
  "event_fields": [
    "relative_time",
    "category",
    "event",
    "data"
  ],
  "events": [
    [
      2,
      "transport",
      "packet_received",
      { header: {...}, frames: {...} }
    ],
    ...
  ]
}
```

"column" names

- mvfst
- aioquic
- quicly / H2O
- f5
- neqo
- picoquic
- ats
- ...

draft-02: JSON + NDJSON

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

- quic-go
- ngtcp2
- quiche
- haskell
- kwik

https://github.com/quicwg/qlog

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
- …

https://github.com/quicwg/qlog/issues/30
https://github.com/quicwg/qlog/issues/144

Part 1: What is the goal for qlog?

Optimize for interoperable/reusable tools?

VS

Optimize for direct output/storage/transfer?

https://github.com/quicwg/qlog/issues/144#issuecomment-815018003

Part 1: What is the goal for qlog?

Optimize for interoperable/reusable tools?

VS

Optimize for direct output/storage/transfer?

Is this even needed?

- Direct 2 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
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
protobuf	66.15	23.96	14.56	5.27	10.71	3.88

https://crates.io/crates/qlog
https://github.com/quicwg/qlog/issues/30
https://github.com/quicwg/qlog/issues/144#issuecomment-815018003

Part 1: Proposal

Prioritize tools, stick to JSON + NDJSON

- Other documents can later define CBOR/PCAPNG/Protobuf/… if needed

- We do need to define NDJSON proper ourselves then…

- *Implies switch to CDDL to describe events in drafts*
  - *Currently, we're using a custom DDL based on TypeScript*

- Even loading large JSON files should be feasible
  - Not in qvis/browser, but surely in native apps

https://github.com/quicwg/qlog/issues/143

Part 2: what's inside it?

Part 2: What we have now

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: What we also have now

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

Part 2: What we also also have now

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

partial wire image

```
{
"time": 15000,
"name": "transport:frames_processed",
"data": {
  "frames": {
      "frame_type": "ack",
      "acked_ranges": [
          [10,15],
          [17,20]
      ]
   }
}
```

No packet header

Part 2: What people are proposing

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

https://github.com/quicwg/qlog/issues/163

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?

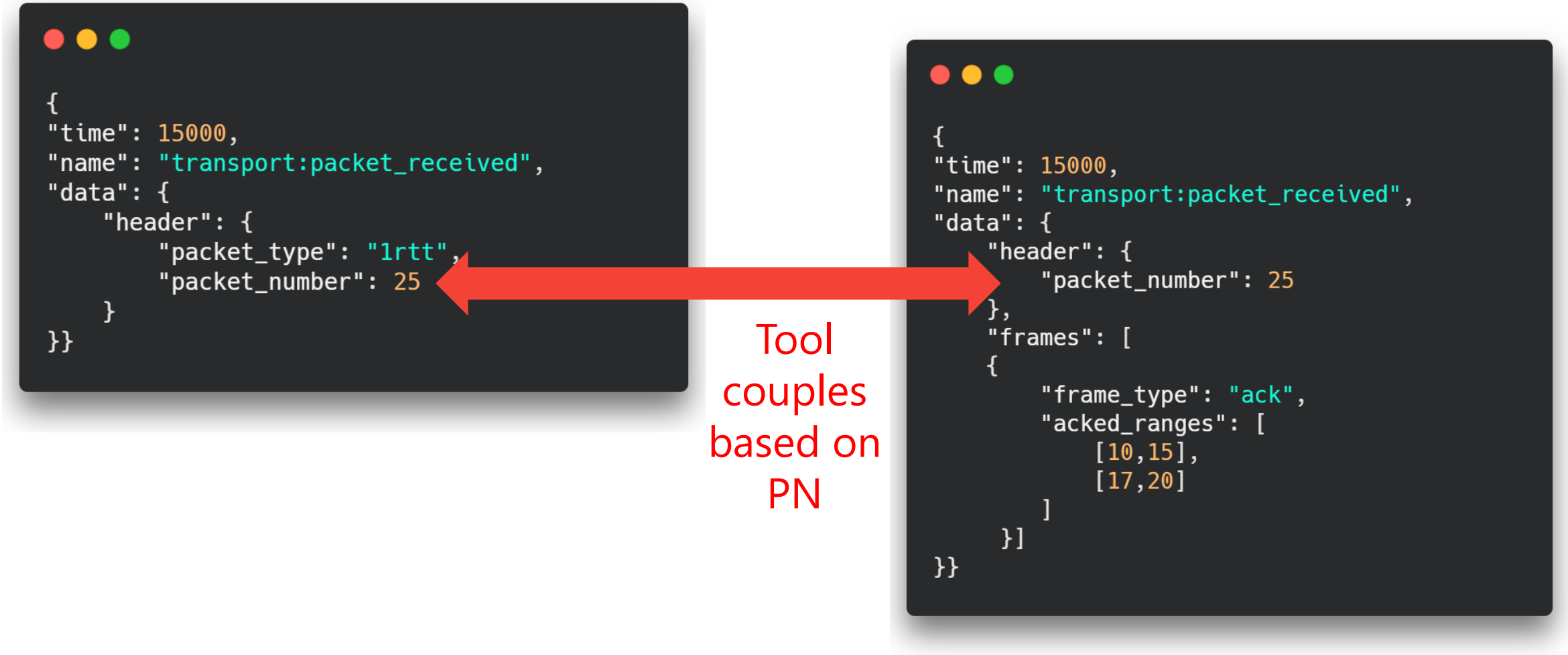
    - *If tools only support a subset, what's the use of standardizing more?*

We need guidelines/design philosphy

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

https://github.com/quicwg/qlog/issues/154

Part 2: Re-use event types



When handling header

When handling payload

https://github.com/quicwg/qlog/issues/154

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*

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"*

If implementations need split (re-used) events/other logic:
→ Write custom converter to "proper" qlog for tools that don't support those

In short

Proposal 1: JSON + NDJSON

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

- TODO: not sure if we can call for “consensus” here, but really would like to hear the thoughts in the room about this
  - Not sure how to best approach that
  - Don’t want to be left in limbo AGAIN → been bringing this up for 9 months, need decision or clear involvement in discussion