structured event logging

The philosophical update

Robin Marx  robin.marx@kuleuven.be
The story so far

\[ q\text{log} \] = 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)

3 separate documents:
- Main schema
- QUIC and TLS events
- HTTP/3 and QPACK events

metadata schema + serialization format

https://github.com/quicwg/qlog
Schema vs Serialization Format

```json
{
  "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]
          ]
        }
      ]
    }
  }],
  ...
}
```

```javascript
class AckFrame{
    frame_type:string = "ack";
    ack_delay?:float; // in ms
    acked_ranges?:Array<[uint64, uint64]>[uint64];
    ect1?:uint64;
    ect0?:uint64;
    ce?:uint64;
}
```

https://github.com/quicwg/qlog
Schema vs Serialization Format

```json
{
  "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]
        }
      ]
    }
  }]
}
...
```

```javascript
class AckFrame{
  frame_type:string = "ack";
  ack_delay?:float; // in ms
  acked_ranges?:Array<[uint64, uint64]>[uint64];
  ecc1?:uint64;
  ecc0?:uint64;
  ce?:uint64;
}
```
Schema vs Serialization Format

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

```typescript
class AckFrame{
  frame_type: string = "ack";
  ack_delay?: float; // in ms
  acked_ranges?: Array<[uint64, uint64] | [uint64]>
  ect1?: uint64;
  ect0?: uint64;
  ce?: uint64;
}
```

qlog_format?: string = "JSON" | "NDJSON";

https://github.com/quicwg/qlog
Today

What do we actually standardize and why?

https://github.com/quicwg/qlog/issues
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… need to define our own “Streaming JSON”

Alternatives:
- CBOR
- Protobuffers/flatbuffers/...
- PCAPNG
- ...

Binary format - https://github.com/quicwg/qlog/issues/30
Revisit serialization format - 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?
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 JSON is feasible
  - mvfst, quic-go

- Log optimized, **convert**
  - quickly, picoquic
  - chromium (kind of)

- **Compress**

<table>
<thead>
<tr>
<th>Format</th>
<th>Raw (MB)</th>
<th>%</th>
<th>Gzip5 (MB)</th>
<th>%</th>
<th>Brotl4 (MB)</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>pcap</td>
<td>561.57</td>
<td>203.45</td>
<td>529.01</td>
<td>191.65</td>
<td>528.85</td>
<td>191.60</td>
</tr>
<tr>
<td>qlog</td>
<td>276.02</td>
<td>100.00</td>
<td>19.15</td>
<td>6.94</td>
<td>19.40</td>
<td>7.03</td>
</tr>
<tr>
<td>cbor</td>
<td>215.53</td>
<td>78.08</td>
<td>17.78</td>
<td>6.44</td>
<td>18.90</td>
<td>6.85</td>
</tr>
<tr>
<td>qlog_lookup</td>
<td>155.89</td>
<td>56.48</td>
<td>17.25</td>
<td>6.25</td>
<td>17.99</td>
<td>6.52</td>
</tr>
<tr>
<td>cbor_lookup</td>
<td>90.85</td>
<td>32.91</td>
<td>15.18</td>
<td>5.50</td>
<td>13.18</td>
<td>4.77</td>
</tr>
<tr>
<td>protobuf</td>
<td>66.15</td>
<td>23.96</td>
<td>14.56</td>
<td>5.27</td>
<td>10.71</td>
<td>3.88</td>
</tr>
</tbody>
</table>
Part 1: Proposal

Stick to JSON + “Streaming JSON”

- 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 Streaming JSON properly ourselves then…
  - Can still be identical to NDJSON’s format! Or use another delimiter or …

Revisit serialization format - https://github.com/quicwg/qlog/issues/144
Part 2: which events do we include?

+ Custom events!

Tools MUST deal with unknown 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]
        ]
      }
    ]
  }
}
```

```
{  
  "time": 15001,
  "name": "recovery:metrics_updated",
  "data": {
    "min_rtt": 25,
    "smoothed_rtt": 30,
    "latest_rtt": 25,
    "congestion_window": 60,
    "bytes_in_flight": 77000,
  }
}
```
Part 2: 2 sides of the same coin

Note: we also have a separate `packet_lost` event
Part 2: 3 sides of the same... triangle?

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

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

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

Only newly ACKed

No packet header

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

provide clearer usage advice - https://github.com/quicwg/qlog/issues/53
frames_processed fails to capture - https://github.com/quicwg/qlog/issues/154
Part 2: Re-use event types

When handling header:
```
{
  "time": 15000,
  "name": "transport:packet_received",
  "data": {
    "header": {
      "packet_type": "lrrt",
      "packet_number": 25
    }
  }
}
```

When handling payload:
```
{
  "time": 15000,
  "name": "transport:packet_received",
  "data": {
    "header": {
      "packet_number": 25
    },
    "frames": [
      {
        "frame_type": "ack",
        "acked_ranges": [
          [10,15],
          [17,20]
        ]
      }
    ]
  }
}
```

frames_processed fails to capture - https://github.com/quicwg/qlog/issues/154

Tool couples based on PN instead of frames_processed
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*

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”*
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”*

\[ \text{packet_sent + congestion_metrics_updated} \]
\[ = \text{no more frames_processed} \]

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 **standardize**?

Proposal 1: JSON + “Streaming JSON”

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

getting rough consensus on these impacts ~75% of open issues
EXTRA
Schema vs Serialization Format

JSON and NDJSON

```javascript
qlog_format?: string = "JSON" | "NDJSON";
```

class AckFrame{
    frame_type: string = "ack";
    ack_delay?: float; // in ms
    acked_ranges?: Array<[uint64, uint64]>[[uint64]];
    ect1?: uint64;
    ect0?: uint64;
    ce?: uint64;
}
```

Data Definition Language

not for today ;)

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": {...}
      }
    ],
    ...
  ]
}
```

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

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
- quickly / H2O
- f5
- ...

- neqo
- picoquic
- ats
- applequic
- ...

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

draft-02: JSON

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

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

**draft-02:** JSON + NDJSON

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

- mvfst
- aioquic
- quicly / H2O
- f5
- neqo
- picoquic
- ats
- applequic
- quic-go
- ngtcp2
- quiche
- haskell
- kwik

https://github.com/quicwg/qlog