QUIC and HTTP/3 event definitions for qlog
draft-marx-qlog-event-definitions-quic-h3-00

Abstract

This document describes concrete qlog event definitions and their
metadata for QUIC and HTTP/3-related events. These events can then
be embedded in the higher level schema defined in draft-marx-quic-
logging-main-schema-latest.

Status of This Memo

This Internet-Draft is submitted in full conformance with the
provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering
Task Force (IETF). Note that other groups may also distribute
working documents as Internet-Drafts. The list of current Internet-
Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months
and may be updated, replaced, or obsoleted by other documents at any
time. It is inappropriate to use Internet-Drafts as reference
material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 4, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(https://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.
Table of Contents

1. Introduction ................................................. 3
   1.1. Notational Conventions ................................. 3
2. Overview .................................................... 3
3. QUIC event definitions ....................................... 4
   3.1. CONNECTIVITY ........................................... 4
       3.1.1. CONNECTION_ATTEMPT .............................. 4
       3.1.2. CONNECTION_NEW .................................. 4
       3.1.3. CONNECTION_ID_UPDATE ......................... 5
       3.1.4. MIGRATION-related events ..................... 5
       3.1.5. CONNECTION_CLOSED .............................. 5
   3.2. SECURITY ................................................. 5
       3.2.1. HEADER_DECRYPT_ERROR .......................... 5
       3.2.2. PACKET_DECRYPT_ERROR ......................... 5
       3.2.3. KEY_UPDATE ...................................... 5
       3.2.4. KEY_RETIRED ..................................... 5
       3.2.5. CIPHER_UPDATE .................................. 5
   3.3. TRANSPORT .............................................. 5
       3.3.1. PACKET_SENT .................................... 5
       3.3.2. PACKET_RECEIVED ................................ 6
       3.3.3. PACKET_DROPPED .................................. 6
       3.3.4. VERSION_UPDATE .................................. 7
       3.3.5. TRANSPORT_PARAMETERS_UPDATE ............... 7
       3.3.6. ALPN_UPDATE ..................................... 7
       3.3.7. STREAM_STATE_UPDATE ............................ 7
       3.3.8. FLOW_CONTROL_UPDATE ............................ 8
   3.4. RECOVERY ................................................. 8
       3.4.1. CC_STATE_UPDATE ................................ 8
       3.4.2. METRIC_UPDATE ................................... 8
       3.4.3. LOSS_ALARM_SET .................................. 9
       3.4.4. LOSS_ALARM_FIRED ............................... 9
       3.4.5. PACKET_LOST .................................... 9
       3.4.6. PACKET_ACKNOWLEDGED ........................... 9
       3.4.7. PACKET_RETRANSMIT ............................... 10
   4. HTTP/3 event definitions .................................. 10
   4.1. HTTP ...................................................... 10
   4.2. QPACK .................................................... 10
   4.3. PRIORITIZATION ........................................ 10
   4.4. PUSH ...................................................... 10
5. Security Considerations .................................... 10
6. IANA Considerations ........................................ 10
7. References .................................................. 10
   7.1. Normative References .................................. 10
   7.2. URIs ..................................................... 10
Appendix A. QUIC DATA type definitions ........................ 11
   A.1. PacketType ............................................. 11
   A.2. PacketHeader ........................................... 11
1. Introduction

Feedback and discussion welcome at https://github.com/quiclog/internet-drafts

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Overview

This document describes the values of the qlog CATEGORY, EVENT_TYPE, TRIGGER and DATA fields and their semantics for the QUIC and HTTP/3 protocols. The definitions included in this file are assumed to be used in qlog’s "trace" containers, where the trace’s "protocol_type" field MUST be set to "QUIC_HTTP3".

This document is based on draft-20 of the QUIC and HTTP/3 I-Ds QUIC-TRANSPORT [QUIC-HTTP].

This document uses the "TypeScript" language [1] to describe its schema in. We use TypeScript because it is less verbose than JSON-schema and almost as expressive. It also makes it easier to include these definitions directly into a web-based tool. The main conventions a reader should be aware of are:

- obj? : this object is optional

Marx                     Expires January 4, 2020                [Page 3]
3. QUIC event definitions

- TODO: flesh out the definitions for most of these

- TODO: add all definitions for HTTP3 and QPACK events

3.1. CONNECTIVITY

3.1.1. CONNECTION_ATTEMPT

TODO: specify how this works with happy eyeballs

3.1.2. CONNECTION_NEW
3.1.3. CONNECTION_ID_UPDATE

TODO: mention that CIDs can be logged in hex

3.1.4. MIGRATION-related events

e.g., PATH_UPDATE

TODO: read up on the draft how migration works and whether to best
fit this here or in TRANSPORT

3.1.5. CONNECTION_CLOSED

3.2. SECURITY

3.2.1. HEADER_DECRYPT_ERROR

{ mask, error }

3.2.2. PACKET_DECRYPT_ERROR

{ key, error }

3.2.3. KEY_UPDATE

{ type = "Initial | handshake | 1RTT", value }

3.2.4. KEY_RETIRED

{ value } # initial encryption level is implicitly deleted

3.2.5. CIPHER_UPDATE

3.3. TRANSPORT

3.3.1. PACKET_SENT

Triggers:

o "DEFAULT"

o "RETRANSMIT_REORDERING" // draft-19 6.1.1

o "RETRANSMIT_TIMEOUT" // draft-19 6.1.2

o "RETRANSMIT_CRYPTO" // draft-19 6.2

o "RETRANSMIT_PTO" // draft-19 6.3
o  "CC_BANDWIDTH_PROBE" // needed for some CCs to figure out
    bandwidth allocations when there are no normal sends

Data:

{  
    packet_type:PacketType,  
    header:PacketHeader,  
    frames:Array<QuicFrame>
}

Notes:

o  We don’t explicitly log the encryption_level or
   packet_number_space: the packet_type specifies this by inference
   (assuming correct implementation)

3.3.2.  PACKET_RECEIVED

Triggers:

o  "DEFAULT"

Data:

{  
    packet_type:PacketType,  
    header:PacketHeader,  
    frames:Array<QuicFrame>
}

Notes:

o  We don’t explicitly log the encryption_level or
   packet_number_space: the packet_type specifies this by inference
   (assuming correct implementation)

3.3.3.  PACKET_DROPPED

Can be due to several reasons * TODO: How does this relate to
HEADER_DECRYPT_ERROR and PACKET_DECRYPT_ERROR? * TODO: if a packet is
dropped because we don’t have a connection for it, how can we add it
to a given trace in the overall qlog file? Need a sort of catch-call
trace in each file? * TODO: differentiate between DATAGRAM_DROPPED
and PACKET_DROPPED? Same with PACKET_RECEIVED and DATAGRAM_RECEIVED?
3.3.4. VERSION_UPDATE

TODO: maybe name VERSION_SELECTED?

3.3.5. TRANSPORT_PARAMETERS_UPDATE

3.3.6. ALPN_UPDATE

TODO: should this be in HTTP?

{ alpn:string }

3.3.7. STREAM_STATE_UPDATE

{ 
  old:string,
  new:string
}

Possible values:

- IDLE
- OPEN
- CLOSED
- HALF_CLOSED_REMOTE
- HALF_CLOSED_LOCAL
- DESTROYED // memory actually freed
- Ready
- Send
- Data Sent
- Reset Sent
- Data Rcvd
- Reset Rcvd
- Rcv
- Size Known
o Data Rcvd

o Data Read

o Reset Read

TODO: do we need all of these? How do implementations actually handle this in practice?

3.3.8. FLOW_CONTROL_UPDATE

o type = connection

o type = stream + id = streamid

TODO: check state machine in QUIC transport draft

3.4. RECOVERY

3.4.1. CC_STATE_UPDATE

{
    old:string,
    new:string
}

3.4.2. METRIC_UPDATE

{
    cwnd?: number;
    bytes_in_flight?:number;

    min_rtt?:number;
    smoothed_rtt?:number;
    latest_rtt?:number;
    max_ack_delay?:number;

    rtt_variance?:number;
    ssthresh?:number;

    pacing_rate?:number;
}

This event SHOULD group all possible metric updates that happen at or around the same time in a single event (e.g., if min_rtt and smoothed_rtt change at the same time, they should be bundled in a single METRIC_UPDATE entry, rather than split out into two).
Consequently, a METRIC_UPDATE is only guaranteed to contain at least one of the listed metrics.

Note: to make logging easier, implementations MAY log values even if they are the same as previously reported values (e.g., two subsequent METRIC_UPDATE entries can both report the exact same value for min_rtt). However, applications SHOULD try to log only actual updates to values.

- TODO: split these up into separate events? e.g., CWND_UPDATE, BYTES_IN_FLIGHT_UPDATE, ...
- TODO: move things like pacing_rate, cwnd, bytes_in_flight, ssthresh, etc. to CC_STATE_UPDATE?
- TODO: what types of CC metrics do we need to support by default (e.g., cubic vs bbr)

3.4.3. LOSS_ALARM_SET

3.4.4. LOSS_ALARM_FIRED

3.4.5. PACKET_LOST

Data:

```json
{
    packet_number: string
}
```

Triggers:

- "UNKNOWN",
- "REORDERING_THRESHOLD",
- "TIME_THRESHOLD"

3.4.6. PACKET_ACKNOWLEDGED

TODO: must this be a separate event? can’t we get this from logged ACK frames? (however, explicitly indicating this and logging it in the ack handler is a better signal that the ACK actually had the intended effect than just logging its receipt)
3.4.7. PACKET_RETRANSMIT

TODO: only if a packet is retransmit in-full, which many stacks don’t do. Need something more flexible.

4. HTTP/3 event definitions

4.1. HTTP

4.2. QPACK

4.3. PRIORITIZATION

4.4. PUSH

5. Security Considerations

TBD

6. IANA Considerations

TBD

7. References

7.1. Normative References

[QUIC-HTTP]

[QUIC-TRANSPORT]


7.2. URIs

[1] https://www.typescriptlang.org/
Appendix A. QUIC DATA type definitions

A.1. PacketType

enum PacketType {
  INITIAL,
  HANDSHAKE,
  ZERO_RTT = "0RTT",
  ONE_RTT = "1RTT",
  RETRY,
  VERSION_NEGOTIATION,
  UNKNOWN
}

A.2. PacketHeader

class PacketHeader {
  packet_number: string;
  packet_size?: number;
  payload_length?: number;
  // only if present in the header
  // if correctly using NEW_CONNECTION_ID events,
  // dcid can be skipped for 1RTT packets
  version?: string;
  scli?: string;
  dcli?: string;
  scld?: string;
  dclid?: string;
  // Note: short vs long header is implicit through PacketType
}

A.3. QUIC Frames

type QuicFrame = AckFrame | StreamFrame | ResetStreamFrame | ConnectionCloseFrame | MaxDataFrame | MaxStreamDataFrame | UnknownFrame;

A.3.1. AckFrame
class AckFrame{
    frame_type:string = "ACK";
    ack_delay:string;
    // first number is "from": lowest packet number in interval
    // second number is "to": up to and including // highest packet number in interval
    // e.g., looks like [[1,2],[4,5]]
    acked_ranges:Array<[number, number]>;
    ect1?:string;
    ect0?:string;
    ce?:string;
}

Note: the packet ranges in AckFrame.acked_ranges do not necessarily have to be ordered (e.g., [[5,9],[1,4]] is a valid value).

Note: the two numbers in the packet range can be the same (e.g., [120,120] means that packet with number 120 was ACKed). TODO: maybe make this into just [120]?

A.3.2. StreamFrame

class StreamFrame{
    frame_type:string = "STREAM";
    id:string;
    // These two MUST always be set
    // If not present in the Frame type, log their default values
    offset:string;
    length:string;
    // this MAY be set any time, but MUST only be set if the value is "true"
    // if absent, the value MUST be assumed to be "false"
    fin:boolean;
}

A.3.3. ResetStreamFrame

class ResetStreamFrame{
    frame_type:string = "RESET_STREAM";
    id:string;
    error_code:ApplicationError | number;
    final_offset:string;
}
A.3.4. ConnectionCloseFrame

type ErrorSpace = "TRANSPORT" | "APPLICATION";

class ConnectionCloseFrame{
    frame_type:string = "CONNECTION_CLOSE";
    error_space:ErrorSpace;
    error_code:TransportError | ApplicationError | number;
    reason:string;
    trigger_frame_type?:number; // TODO: should be more defined, but we don’t have a FrameType enum atm...
}

A.3.5. MaxDataFrame

class MaxDataFrame{
    stream_type:string = "MAX_DATA";
    maximum:string;
}

A.3.6. MaxStreamDataFrame

class MaxStreamDataFrame{
    stream_type:string = "MAX_STREAM_DATA";
    id:string;
    maximum:string;
}

A.3.7. UnknownFrame

class UnknownFrame{
    frame_type:string = "UNKNOWN";
}

A.3.8. TransportError
enum TransportError {
    NO_ERROR,
    INTERNAL_ERROR,
    SERVER_BUSY,
    APPLICATION_FLOW_CONTROL_ERROR, // 0x3
    STREAM_FLOW_CONTROL_ERROR, // 0x4
    STREAM_STATE_ERROR,
    FINAL_SIZE_ERROR,
    FRAME_ENCODING_ERROR,
    TRANSPORT_PARAMETER_ERROR,
    PROTOCOL_VIOLATION,
    INVALID_MIGRATION,
    CRYPTO_ERROR
}

Appendix B. HTTP/3 DATA type definitions

B.1. ApplicationError

enum ApplicationError{
    HTTP_NO_ERROR,
    HTTP_WRONG_SETTING_DIRECTION,
    HTTP_PUSH_REFUSED,
    HTTP_INTERNAL_ERROR,
    HTTP_PUSH_ALREADY_IN_CACHE,
    HTTP_REQUEST_CANCELLED,
    HTTP_INCOMPLETE_REQUEST,
    HTTP_CONNECT_ERROR,
    HTTP_EXCESSIVE_LOAD,
    HTTP_VERSION_FALLBACK,
    HTTP_WRONG_STREAM,
    HTTP_LIMIT_EXCEEDED,
    HTTP_DUPLICATE_PUSH,
    HTTP_UNKNOWN_STREAM_TYPE,
    HTTP_WRONG_STREAM_COUNT,
    HTTP_CLOSED_CRITICAL_STREAM,
    HTTP_WRONG_STREAM_DIRECTION,
    HTTP_EARLY_RESPONSE,
    HTTP_MISSING_SETTINGS,
    HTTP_UNEXPECTED_FRAME,
    HTTP_REQUEST_REJECTED,
    HTTP_GENERAL_PROTOCOL_ERROR,
    HTTP_MALFORMED_FRAME
}

TODO: HTTP_MALFORMED_FRAME is not a single value, but can include the frame type in its definition. This means we need more flexible error
logging. Best to wait until h3-draft-21, which will include substantial changes to error codes.

Appendix C. Change Log

C.1. Since draft-marx-qlog-event-definitions-quic-h3-00-00:

- None yet.

Appendix D. Design Variations

TBD

Appendix E. Acknowledgements

Thanks to Jana Iyengar, Brian Trammell, Dmitri Tikhonov, Stephen Petrides, Jari Arkko, Marcus Ihlar, Victor Vasiliev, Mirja Kuehlewind and Lucas Pardue for their feedback and suggestions.

Author’s Address

Robin Marx
Hasselt University

Email: robin.marx@uhasselt.be