

Workgroup: Network Working Group

Internet-Draft:

draft-mcquistin-quic-augmented-diagrams-05

Published: 25 October 2021

Intended Status: Experimental

Expires: 28 April 2022

Authors: S. McQuistin

V. Band

University of Glasgow

University of Glasgow

D. Jacob

C. S. Perkins

University of Glasgow

University of Glasgow

Describing QUIC's Protocol Data Units with Augmented Packet Header Diagrams

Abstract

This document describes the core transport protocol data units used in the QUIC protocol using a machine-readable augmented packet header diagram format. It is intended as an example of the augmented packet header diagram language, and not as a contribution to the development of the QUIC protocol.

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 28 April 2022.

Copyright Notice

Copyright (c) 2021 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](#)
- [2. Header and Packet Protection](#)
- [3. Variable Length Integer](#)
- [4. Stateless Reset](#)
- [5. Version Negotiation Packet](#)
- [6. Long Header Packets](#)
 - [6.1. Initial Packet](#)
 - [6.2. 0RTT Packet](#)
 - [6.3. Handshake Packet](#)
 - [6.4. Retry Packet](#)
- [7. Short Header Packets](#)
- [8. Frames and Frame Formats](#)
 - [8.1. PADDING frame](#)
 - [8.2. PING frame](#)
 - [8.3. ACK frame](#)
 - [8.4. RESET STREAM frame](#)
 - [8.5. STOP SENDING frame](#)
 - [8.6. CRYPTO frame](#)
 - [8.7. NEW TOKEN frame](#)
 - [8.8. STREAM frame](#)
 - [8.9. MAX DATA frame](#)
 - [8.10. MAX STREAM DATA frame](#)
 - [8.11. MAX STREAMS frame](#)
 - [8.12. DATA BLOCKED frame](#)
 - [8.13. STREAM DATA BLOCKED frame](#)
 - [8.14. STREAMS BLOCKED frame](#)
 - [8.15. NEW CONNECTION ID frame](#)
 - [8.16. RETIRE CONNECTION ID frame](#)
 - [8.17. PATH CHALLENGE frame](#)
 - [8.18. PATH RESPONSE frame](#)
 - [8.19. CONNECTION CLOSE frame](#)
 - [8.20. HANDSHAKE DONE frame](#)
- [9. Informative References](#)
- [Appendix A. Source code repository](#)
- [Authors' Addresses](#)

1. Introduction

The augmented packet header diagram format [[AUGMENTED-DIAGRAMS](#)] enables documents to specify the syntax of protocol data units in a way that enables support for automated parser generation, while maintaining human readability.

To demonstrate how this approach can be applied, and the value that it can provide, this document describes QUIC [[QUIC-TRANSPORT](#)] using the augmented packet header diagram format. It is intended as an example of the augmented packet header diagram language, and not as a contribution to the development of the QUIC protocol.

This document is not an exhaustive description of the QUIC protocol. It contains only those elements necessary to demonstrate the augmented packet header diagram format, and should be read as an example of the use of that format.

This document describes the QUIC protocol. The QUIC protocol uses Stateless Reset Packets, Protected Packets, Retry Packets, and Version Negotiation Packets.

2. Header and Packet Protection

A Protected Packet is either a Protected Long Header Packet or a Protected Short Header Packet.

An Unprotected Packet is either a Long Header Packet or a Short Header Packet.

An Unprotected Packet is parsed from a Protected Packet using the `remove_protection` function. The `remove_protection` function is defined as:

```
func remove_protection(from: Protected Packet) -> Unprotected Packet:
    remove header protection from protected_packet
    remove packet protection from protected_packet
    construct appropriate packet type
    return Unprotected Packet
```

An Unprotected Packet is serialised to a Protected Packet using the `apply_protection` function. The `apply_protection` function is defined as:

```
func apply_protection(to: Unprotected Packet)
    -> Protected Packet:
    apply packet protection to payload
    apply header protection to first_byte and packet_number
    construct appropriate Protected Packet based on first_byte
    return Protected Packet
```

3. Variable Length Integer

A Variable Length Integer is formatted as follows:

```

0               1               2               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|Len|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                               Value                               ...
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

```

where:

Len: 2 bits. The base 2 logarithm of the integer encoding length in bytes.

Value: $((2^{\text{Len}})*8)-2$ bits. The integer value encoded in network byte order.

4. Stateless Reset

A Stateless Reset Packet is formatted as follows:

```

0               1               2               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|0|1|                               Unpredictable Bits                               ...
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|
+
|
+                               Stateless Reset Token                               +
|
+
|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

```

where:

Header Form (HF): 1 bit; HF == 0. The most significant bit (0x80) of byte 0 (the first byte) is set to 0 for stateless reset packets.

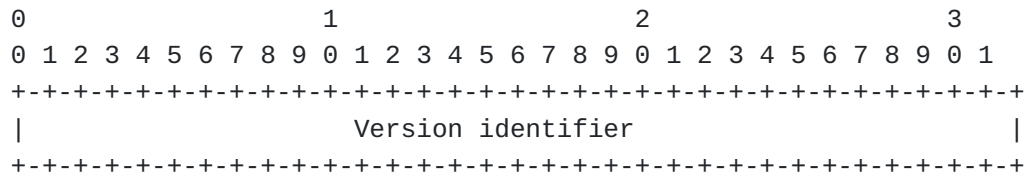
Fixed Bit (FB): 1 bit; FB == 1. The next bit (0x40) of byte 0 is set to 1. Packets containing a zero value for this bit are not valid packets in this version and MUST be discarded.

Unpredictable Bits (UB): UB#Size > 38. The remainder of the first byte and an arbitrary number of bytes following it that are set to unpredictable values.

Stateless Reset Token: 128 bits. The last 16 bytes of the datagram contain a Stateless Reset Token.

5. Version Negotiation Packet

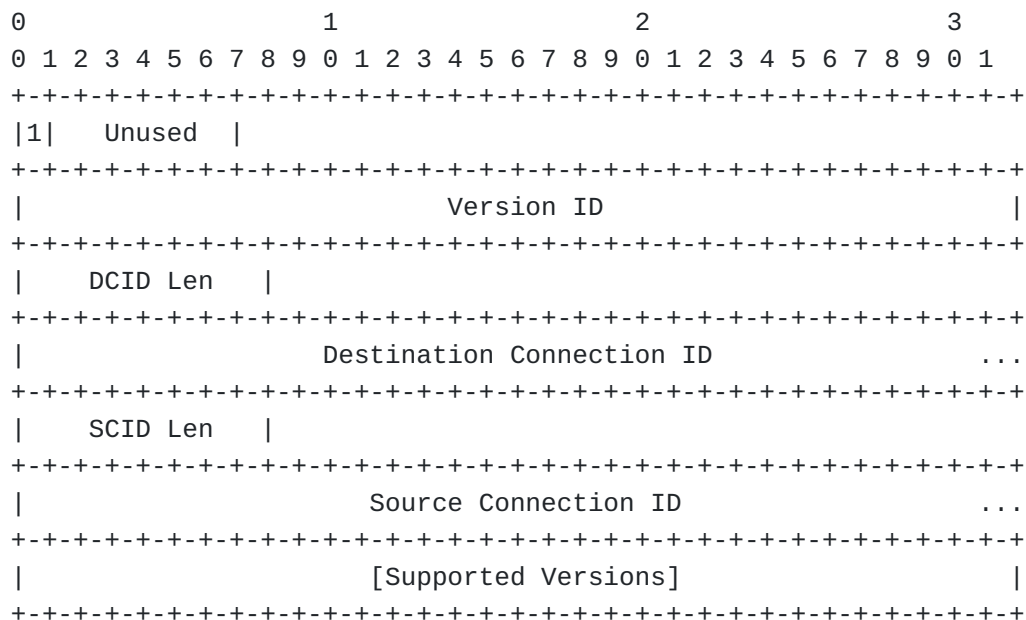
A Version is formatted as follows:



where:

Version identifier (ID): 32 bits. 32-bit version identifier.

A Version Negotiation Packet is formatted as follows:



where:

Header Form (HF): 1 bit; HF == 1. The most significant bit (0x80) of byte 0 (the first byte) is set to 1 for version negotiation packets.

Unused (T): 6 bits. The value in the Unused field is selected randomly by the server. Clients **MUST** ignore the value of this field. Servers **SHOULD** set the most significant bit of this field

(0x40) to 1 so that Version Negotiation packets appear to have the Fixed Bit field.

Version ID (VID): 1 Version; VID.ID == 0. The Version field of a Version Negotiation packet MUST be set to 0x00000000.

DCID Len (DLen): 1 byte. This field is as previously define.
However, as future versions of QUIC may support Connection IDs larger than the version 1 limit, Version Negotiation packets could carry Connection IDs that are longer than 20 bytes.

Destination Connection ID: DLen bytes. The Destination Connection ID field is between 0 and 2^8-1 bytes in length.

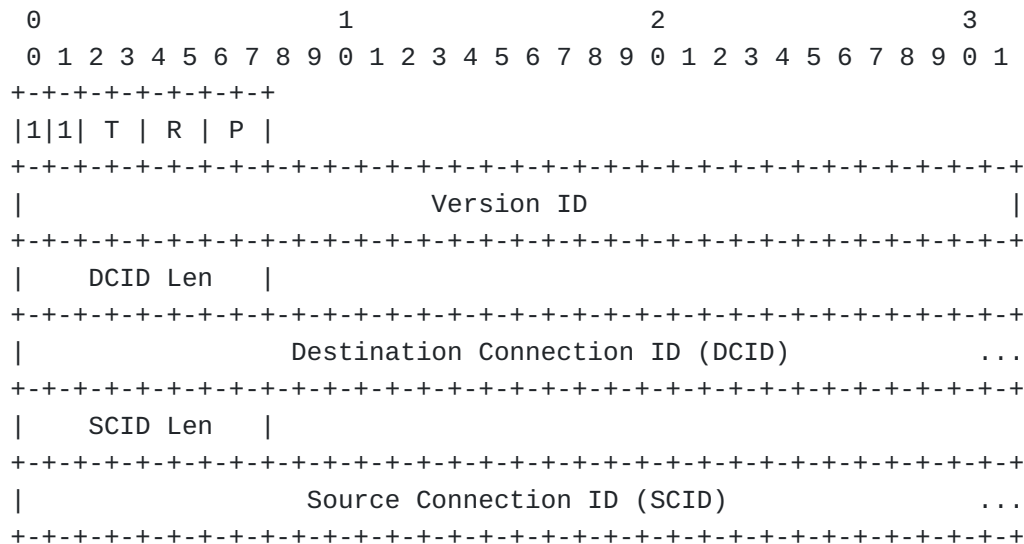
SCID Len (SLen): 1 byte. This field is as previously define.
However, as future versions of QUIC may support Connection IDs larger than the version 1 limit, Version Negotiation packets could carry Connection IDs that are longer than 20 bytes.

Source Connection ID: SLen bytes. The Source Connection ID field is between 0 and 2^8-1 bytes in length.

Supported Versions: [Version]. The remainder of the Version Negotiation packet is a list of 32-bit versions which the server supports.

6. Long Header Packets

A Long Header is formatted as follows:



where:

Header Form (HF): 1 bit; HF == 1. The most significant bit (0x80) of byte 0 (the first byte) is set to 1 for long headers.

Fixed Bit (FB): 1 bit; FB == 1.

The next bit (0x40) of byte 0 is set to 1. Packets containing a zero value for this bit are not valid packets in this version and MUST be discarded.

Long Packet Type (T): 2 bits. The next two bits (those with a mask of 0x30) of byte 0 contain a packet type.

Reserved Bits (R): 2 bits. Two bits (those with a mask of 0x0c) of byte 0 are reserved across multiple packet types. These bits are protected using header protection.

Packet Number Length (P): 2 bits. In packet types which contain a Packet Number field, the least significant two bits (those with a mask of 0x03) of byte 0 contain the length of the packet number, encoded as an unsigned, two-bit integer that is one less than the length of the packet number field in bytes.

Version ID (VID): 1 Version. This field indicates which version of QUIC is in use and determines how the rest of the protocol fields are interpreted.

DCID Len (DLen): 1 byte; DLen <= 20. This field contains the length, in bytes, of the Destination Connection ID field that follows it. This length is encoded as an 8-bit unsigned integer. In QUIC version 1, this value MUST NOT exceed 20. Endpoints that receive a version 1 long header with a value larger than 20 MUST drop the packet. Servers SHOULD be able to read longer connection IDs from other QUIC versions in order to properly form a version negotiation packet.

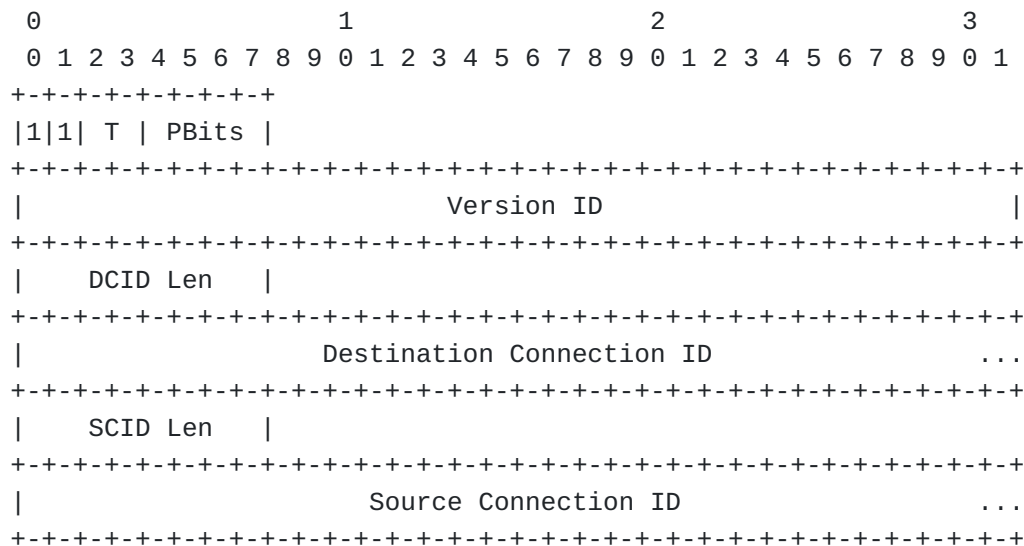
Destination Connection ID (DCID): DLen bytes. The Destination Connection ID field is between 0 and 20 bytes in length.

SCID Len (SLen): 1 byte; SLen <= 20. This field contains the length, in bytes, of the Source Connection ID field that follows it. This length is encoded as an 8-bit unsigned integer. In QUIC version 1, this value MUST NOT exceed 20 bytes. Endpoints that receive a version 1 long header with a value larger than 20 MUST drop the packet. Servers SHOULD be able to read longer connection IDs from other QUIC versions in order to properly form a version negotiation packet.

Source Connection ID (SCID): SLen bytes. The Source Connection ID field is between 0 and 20 bytes in length.

A Long Header Packet is one of: an Initial Packet, a 0RTT Packet, a Handshake Packet, or a Retry Packet.

A Protected Long Header is formatted as follows:



where:

Header Form (HF): 1 bit; HF == 1. The most significant bit (0x80) of byte 0 (the first byte) is set to 1 for long headers.

Fixed Bit (FB): 1 bit; FB == 1. The next bit (0x40) of byte 0 is set to 1. Packets containing a zero value for this bit are not valid packets in this version and MUST be discarded.

Long Packet Type (T): 2 bits. The next two bits (those with a mask of 0x30) of byte 0 contain a packet type.

Protected Bits (PBits): 4 bits. 4 bits protected using header protection.

Version ID (VID): 1 Version. This field indicates which version of QUIC is in use and determines how the rest of the protocol fields are interpreted.

DCID Len (DLen): 1 byte; DLen <= 20. This field contains the length, in bytes, of the Destination Connection ID field that follows it. This length is encoded as an 8-bit unsigned integer. In QUIC version 1, this value MUST NOT exceed 20. Endpoints that receive a version 1 long header with a value larger than 20 MUST drop the packet. Servers SHOULD be able to read longer connection IDs from other QUIC versions in order to properly form a version negotiation packet.

Destination Connection ID: DLen bytes. The Destination Connection ID field is between 0 and 20 bytes in length.

SCID Len (SLen): 1 byte; SLen <= 20. This field contains the length, in bytes, of the Source Connection ID field that follows it. This length is encoded as an 8-bit unsigned integer. In QUIC

version 1, this value MUST NOT exceed 20 bytes. Endpoints that receive a version 1 long header with a value larger than 20 MUST drop the packet. Servers SHOULD be able to read longer connection IDs from other QUIC versions in order to properly form a version negotiation packet.

Source Connection ID: SLen bytes. The Source Connection ID field is between 0 and 20 bytes in length.

A Protected Long Header Packet is one of: a Protected Initial Packet, a Protected 0RTT Packet, or a Protected Handshake Packet.

6.1. Initial Packet

An Initial Packet is formatted as follows:

[illegible]

where:

Long Header (LH): 1 Long Header; LH.T == 0. An Initial packet uses long headers with a type value of 0x0. On receipt, the value of LH.DCID is stored as Initial DCID.

Token Length (TL): 1 Variable Length Integer. A variable-length integer specifying the length of the Token field, in bytes.

Token: TL bytes; present only when TL > 0. The value of the token that was previously provided in a Retry packet or NEW_TOKEN frame.

Length: 1 Variable Length Integer. The length of the remainder of the packet (that is, the Packet Number and Payload fields) in bytes, encoded as a variable-length integer.

Packet Number: LH.P+1 bytes.

The packet number field.

Payload: [Frame]. The payload field, comprised of multiple frames.

A Protected Initial packet is formatted as follows:

[illegible]

where:

Long Header (LH): 1 Long Header; LH.T == 0. An Initial packet uses long headers with a type value of 0x0.

Token Length (TL): 1 Variable Length Integer. A variable-length integer specifying the length of the Token field, in bytes.

Token: TL bytes; present only when TL > 0. The value of the token that was previously provided in a Retry packet or NEW_TOKEN frame.

Length: 1 Variable Length Integer. The length of the remainder of the packet (that is, the Packet Number and Payload fields) in bytes, encoded as a variable-length integer.

Protected Packet Number: LH.P+1 bytes. The packet number field, with header protection.

Protected Payload: (Length-(LH.P+1)) bytes. The protected payload field.

6.2. 0RTT Packet

A **ORTT** Packet is formatted as follows:

A 0-RTT packet uses

A Protected Handshake packet is formatted as follows:

Key Phase (K): 1 bit. The next bit (0x04) of byte 0 indicates the key phase, which allows a recipient of a packet to identify the packet protection keys that are used to protect the packet. This bit is protected using header protection.

Packet Number Length (P): 2 bits.

In packet types which contain a Packet Number field, the least significant two bits (those with a mask of 0x03) of byte 0 contain the length of the packet number, encoded as an unsigned, two-bit integer that is one less than the length of the packet number field in bytes. These bits are protected using header protection.

Destination Connection ID: 20 bytes. The Destination Connection ID is a connection ID that is chosen by the intended recipient of the packet.

Packet Number: P+1 bytes. The packet number field is 1 to 4 bytes long. The packet number has confidentiality protection separate from packet protection. The length of the packet number field is encoded in Packet Number Length field.

Payload: [Frame]. The payload field, comprised of multiple frames.

A Protected Short Header Packet is formatted as follows:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+
|0|1|S| R |K| P |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               Destination Connection ID                               ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               Protected Packet Number                               ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               :
:                               Protected Payload                               :
:                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

where:

Header Form (HF): 1 bit; HF == 0. The most significant bit (0x80) of byte 0 (the first byte) is set to 0 for short headers.

Fixed Bit (FB): 1 bit; FB == 1. The next bit (0x40) of byte 0 is set to 1. Packets containing a zero value for this bit are not valid packets in this version and MUST be discarded.

Spin Bit (S): 1 bit. The third most significant bit (0x20) of byte 0 is the latency spin bit.

Reserved Bits (R): 2 bits. The next two bits (those with a mask of 0x18) of byte 0 are reserved. These bits are protected using

header protection. The value included prior to protection MUST be set to 0.

Key Phase (K): 1 bit. The next bit (0x04) of byte 0 indicates the key phase, which allows a recipient of a packet to identify the packet protection keys that are used to protect the packet. This bit is protected using header protection.

Packet Number Length (P): 2 bits. In packet types which contain a Packet Number field, the least significant two bits (those with a mask of 0x03) of byte 0 contain the length of the packet number, encoded as an unsigned, two-bit integer that is one less than the length of the packet number field in bytes. These bits are protected using header protection.

Destination Connection ID: 20 bytes. The Destination Connection ID is a connection ID that is chosen by the intended recipient of the packet.

Packet Number: P+1 bytes. The packet number field is 1 to 4 bytes long. The packet number has confidentiality protection separate from packet protection. The length of the packet number field is encoded in Packet Number Length field.

Protected Payload: [Frame]. Packets with a short header always include a 1-RTT protected payload.

8. Frames and Frame Formats

A Frame is one of: a PADDING Frame, a PING Frame, an ACK Frame, a RESET_STREAM Frame, a STOP_SENDING Frame, a CRYPTO Frame, a NEW_TOKEN Frame, a STREAM Frame, a MAX_DATA Frame, a MAX_STREAM_DATA Frame, a MAX_STREAMS Frame, a DATA_BLOCKED Frame, a STREAM_DATA_BLOCKED Frame, a STREAMS_BLOCKED Frame, a NEW_CONNECTION_ID Frame, a PATH_CHALLENGE Frame, a PATH_RESPONSE Frame, a CONNECTION_CLOSE Frame, or a HANDSHAKE_DONE Frame.

8.1. PADDING frame

A PADDING Frame is formatted as follows:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           0           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

where:

Frame Type (FT): 1 Variable Length Integer; FT.Value == 0.

Frame type, set to 0 for PADDING frames.

8.2. PING frame

A PING Frame is formatted as follows:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           1           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

where:

Frame Type (FT): 1 Variable Length Integer; FT.Value == 1. Frame type, set to 1 for PING frames.

8.3. ACK frame

An ACK Range is formatted as follows:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     Gap                                     ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     ACK Packet Range                         ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

where:

Gap: 1 Variable Length Integer. A variable-length integer indicating the number of contiguous unacknowledged packets preceding the packet number one lower than the smallest in the preceding ACK Range.

ACK Packet Range: 1 Variable Length Integer. A variable-length integer indicating the number of contiguous acknowledged packets preceding the largest packet number, as determined by the preceding Gap.

An ECN Count is formatted as follows:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                               ECT0 Count                               ...
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                               ECT1 Count                               ...
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                               ECNCE Count                             ...
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

```

where:

ECT0 Count: 1 Variable Length Integer. A variable-length integer representing the total number of packets received with the ECT(0) codepoint in the packet number space of the ACK frame.

ECT1 Count: 1 Variable Length Integer. A variable-length integer representing the total number of packets received with the ECT(1) codepoint in the packet number space of the ACK frame.

ECNCE Count: 1 Variable Length Integer. A variable-length integer representing the total number of packets received with the CE codepoint in the packet number space of the ACK frame.

An ACK Frame is formatted as follows:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  Frame Type  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                               Largest Acknowledged                               ...
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                               ACK Delay                                         ...
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                               ACK Range Count                                   ...
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                               First ACK Range                                   ...
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                               [Other ACK Ranges]                               ...
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                               ECN Counts                                         ...
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

```

where:

Frame Type (FT): 1 Variable Length Integer; (FT.Value == 3) || (FT.Value == 4).

Frame type, set to 3 or 4 for ACK frames.

A variable-length integer representing the largest packet number the peer is acknowledging; this is usually the largest packet number that the peer has received prior to generating the ACK frame. Unlike the packet number in the QUIC long or short header, the value in an ACK frame is not truncated.

ACK Range Count: 1 Variable Length Integer. A variable-length integer specifying the number of Gap and ACK Range fields in the frame.

Other ACK Ranges: [ACK Range]. Contains additional ranges of packets which are alternately not acknowledged and acknowledged.

8.4. RESET_STREAM frame

```

0                                     1                                   2                                    3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+
|         4             |
+-+-+-+-+-+-+-+-+
|                               Stream ID                             ...
+-+-+-+-+-+-+-+-+
|                   Application Error Code                           ...
+-+-+-+-+-+-+-+-+
|                               Final Size                            ...
+-+-+-+-+-+-+-+-+

```

Frame Type (FT): 1 Variable Length Integer; FT.Value == 4. Frame type, set to 4 for RESET_STREAM frames.

Stream ID: 1 Variable Length Integer. A variable-length integer encoding of the Stream ID of the stream being terminated.

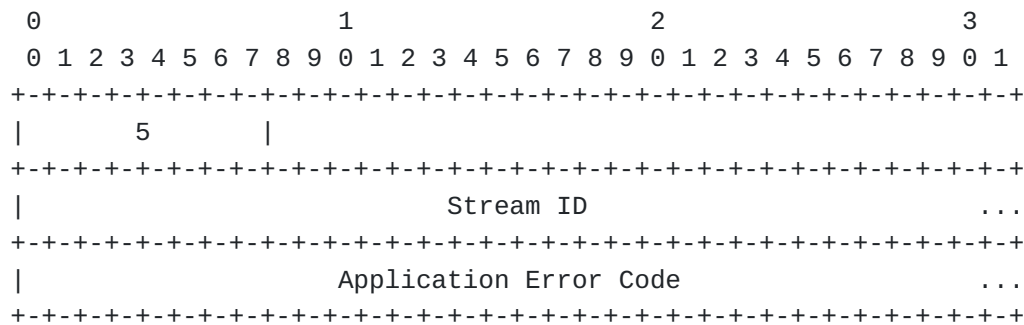
Application Error Code: 1 Variable Length Integer.

A variable-length integer containing the application protocol error code which indicates why the stream is being closed.

Final Size: 1 Variable Length Integer. A variable-length integer indicating the final size of the stream by the RESET_STREAM sender, in unit of bytes.

8.5. STOP_SENDING frame

A STOP_SENDING Frame is formatted as follows:



where:

Frame Type (FT): 1 Variable Length Integer; FT.Value == 5. Frame type, set to 5 for STOP_SENDING frames.

Stream ID: 1 Variable Length Integer. A variable-length integer carrying the Stream ID of the stream being ignored.

Application Error Code: 1 Variable Length Integer. A variable-length integer containing the application-specified reason the sender is ignoring the stream.

8.6. CRYPTO frame

A CRYPTO Frame is formatted as follows:

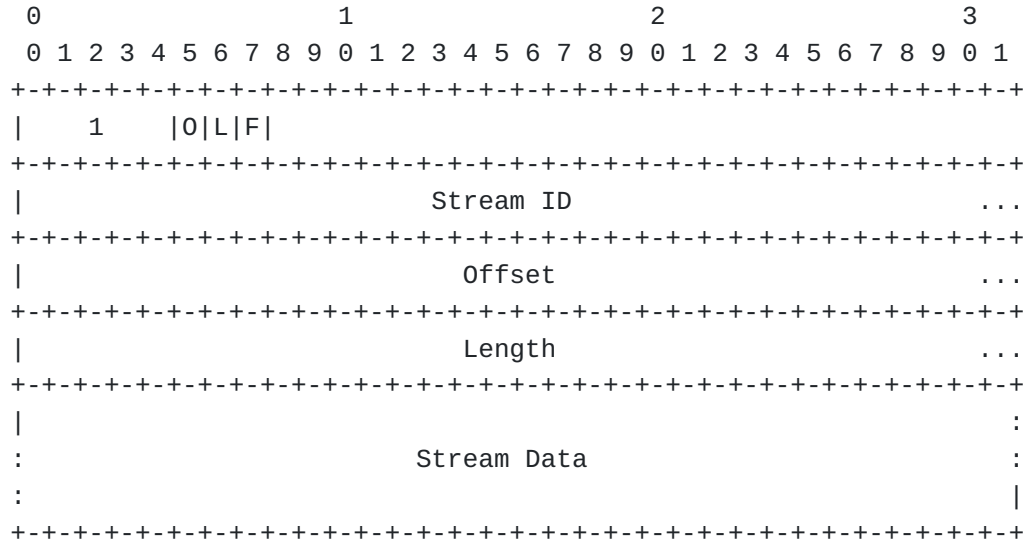
Token Length (TL): 1 Variable Length Integer.

A variable-length integer specifying the length of the token in bytes.

Token: TL bytes. An opaque blob that the client may use with a future Initial packet.

8.8. STREAM frame

A STREAM Frame is formatted as follows:



where:

Unused: 5 bits; Unused == 1. Five high-order bits in frame type field; set to 1 for STREAM frames.

OFF bit (0): 1 bit. Set to indicate that there is an Offset field present.

LEN bit (L): 1 bit. Set to indicate that there is a Length field present.

FIN bit (F): 1 bit. Set only on frames that contain the final size of the stream.

Stream ID: 1 Variable Length Integer. A variable-length integer indicating the stream ID of the stream.

Offset: 1 Variable Length Integer; present only when 0 == 1. A variable-length integer specifying the byte offset in the stream for the data in this STREAM frame.

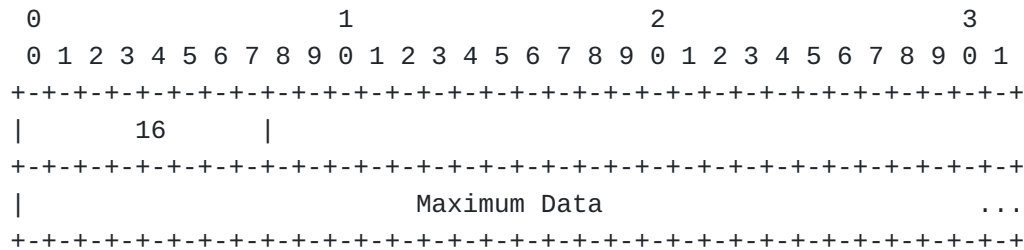
Length: 1 Variable Length Integer; present only when L == 1. A variable-length integer specifying the length of the Stream Data

field in this STREAM frame. This field is present when the LEN bit is set to 1. When the LEN bit is set to 0, the Stream Data field consumes all the remaining bytes in the packet.

Stream Data: Length bytes. The bytes from the designated stream to be delivered.

8.9. MAX_DATA frame

A MAX_DATA Frame is formatted as follows:



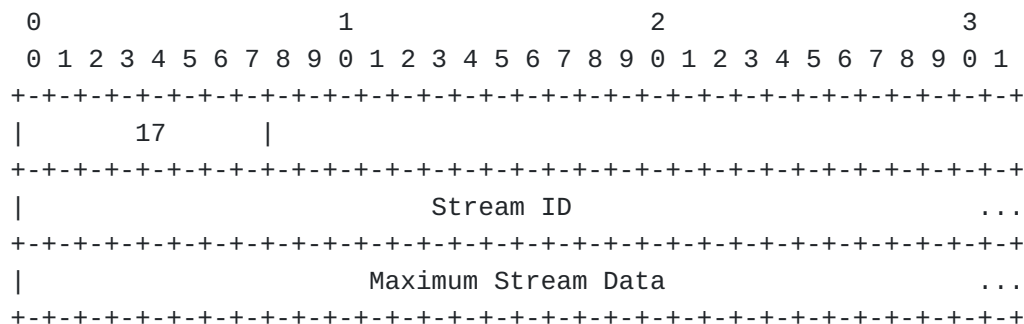
where:

Frame Type (FT): 1 Variable Length Integer; FT.Value == 16. Frame type, set to 16 for MAX_DATA frames.

Maximum Data: 1 Variable Length Integer. A variable-length integer indicating the maximum amount of data that can be sent on the entire connection, in units of bytes.

8.10. MAX_STREAM_DATA frame

A MAX_STREAM_DATA Frame is formatted as follows:



where:

Frame Type (FT): 1 Variable Length Integer; FT.Value == 17. Frame type, set to 17 for MAX_STREAM_DATA frames.

Stream ID: 1 Variable Length Integer. The stream ID of the stream that is affected encoded as a variable-length integer.

Maximum Stream Data: 1 Variable Length Integer.

A variable-length integer indicating the maximum amount of data that can be sent on the identified stream, in units of bytes.

8.11. MAX_STREAMS frame

A MAX_STREAMS Frame is formatted as follows:

```

0               1               2               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           FT           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               Maximum Streams                               ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

where:

Frame Type (FT): 1 Variable Length Integer; (FT.Value == 18) || (FT.Value == 19).

Frame type, set to 18 or 19 for MAX_STREAMS frames.

Maximum Streams: 1 Variable Length Integer. A count of the cumulative number of streams of the corresponding type that can be opened over the lifetime of the connection.

8.12. DATA_BLOCKED frame

A DATA_BLOCKED Frame is formatted as follows:

```

0               1               2               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           20           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               Data Limit                               ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

where:

Frame Type (FT): 1 Variable Length Integer; FT.Value == 20. Frame type, set to 20 for DATA_BLOCKED frames.

Data Limit: 1 Variable Length Integer. A variable-length integer indicating the connection-level limit at which blocking occurred.

8.13. STREAM_DATA_BLOCKED frame

A STREAM_DATA_BLOCKED Frame is formatted as follows:

where:

Frame Type (FT): 1 Variable Length Integer; FT.Value == 27. Frame type, set to 27 for PATH_RESPONSE frames.

Data: 64 bits. This 8-byte field contains arbitrary data.

8.19. CONNECTION_CLOSE frame

A CONNECTION CLOSE Frame is formatted as follows:

[illegible]

where:

```
Frame Type (FT): 1 Variable Length Integer; FT.Value == 28 ||
FT.Value == 29.
```

Frame type, set to 28 or 29 for CONNECTION_CLOSE frames.

Error Code: 1 Variable Length Integer. A variable length integer error code which indicates the reason for closing this connection.

Frame Type: 1 Variable Length Integer; present only when FT.Value == 28.

A variable-length integer encoding the type of frame that triggered the error.

Phrase Length (Length): 1 Variable Length Integer. A variable-length integer specifying the length of the reason phrase in bytes.

Reason Phrase: Length bytes. A human-readable explanation for why the connection was closed.

8.20. HANDSHAKE_DONE frame

A HANDSHAKE_DONE Frame is formatted as follows:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|          30          |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

where:

Frame Type (FT): 1 Variable Length Integer; FT.Value == 30. Frame type, set to 30 for HANDSHAKE_DONE frames.

9. Informative References

[QUIC-TRANSPORT] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-27, 21 February 2020, <<http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-27.txt>>.

[AUGMENTED-DIAGRAMS] McQuistin, S., Band, V., Jacob, D., and C. S. Perkins, "Describing Protocol Data Units with Augmented Packet Header Diagrams", Work in Progress, Internet-Draft, draft-mcquistin-augmented-ascii-diagrams-09, 25 October 2021, <<http://www.ietf.org/internet-drafts/draft-mcquistin-augmented-ascii-diagrams-09.txt>>.

Appendix A. Source code repository

The source for this draft is available from <https://github.com/glasgow-ipl/draft-mcquistin-quic-augmented-diagrams>.

The source code for tooling that can be used to parse this document is available from <https://github.com/glasgow-ipl/ips-protodesc-code>.

Authors' Addresses

Stephen McQuistin
University of Glasgow
School of Computing Science
Glasgow
G12 8QQ
United Kingdom

Email: sm@smcquistin.uk

Vivian Band

University of Glasgow
School of Computing Science
Glasgow
G12 8QQ
United Kingdom

Email: vivianband0@gmail.com

Dejice Jacob
University of Glasgow
School of Computing Science
Glasgow
G12 8QQ
United Kingdom

Email: d.jacob.1@research.gla.ac.uk

Colin Perkins
University of Glasgow
School of Computing Science
Glasgow
G12 8QQ
United Kingdom

Email: csp@csp Perkins.org