QUIC Wire Layout Specification

This document is intended to mirror the rapidly evolving QUIC implementation found in the Chromium.org code repository. Due to the rapid development speed, this document may periodically be outdated. This is a very living document, as the QUIC protocol is also evolving based on results of experiments.

You may find the QUIC FAQ useful for a very brief overview. Alternatively, the rationalization, and justification for most of this wire layout specification can be found in the QUIC Design Document and Rationalization.

Detailed explanation of the cryptographic elements of QUIC are found in the QUIC Crypto document, which is generally very precise and up-to-date, but is continuing to undergo careful scrutiny and review. Information about the mechanism used to negotiate the QUIC version used for a connection can be found in the QUIC Protocol Version Negotiation document.

Definitions

- *Client*: The endpoint initiating the QUIC session.
- *Connection*: A bi-directional exchange of packets between a client and a server, identified by a Connection ID.
- *Endpoint*: Either the client or server of a connection.
- *Connection ID*: A unique identifier for a particular connection.
- *Packet*: A UDP packet containing framed data sent over a QUIC session.
- *Server*: The endpoint not initiating the QUIC session.
- *Session*: A sequence of packets sent over a single connection. This is the same as the "framing layer".
- *Stream*: A bi-directional flow of bytes across a virtual channel within a QUIC session.

Framing Layer

Connections

The QUIC framing layer (or "session") runs atop UDP, unlike how HTTP works today. The client is expected to be the QUIC session initiator. Because it runs on UDP, the underlying transport is not reliable, and the QUIC layer will handle retransmission of dropped packets for reliable streams. Unlike HTTP, the underlying connection for QUIC is guaranteed to be persistent. The HTTP “Connection” header therefore does not apply. For best performance, it is expected that clients will not close open sessions until the user navigates away from all web pages referencing a session, or until the server closes the sessions. Sessions should remain open until they become idle for a pre-negotiated period of time. When a server decides to terminate an idle session, it should not notify the client to avoid waking up the radio on mobile devices.

Framing

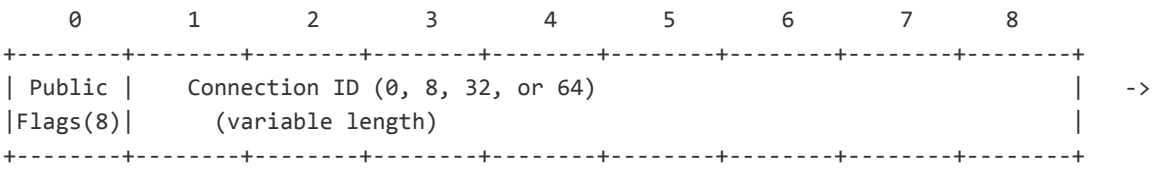
Once the session is established, clients and servers exchange UDP packets. There are two types of packets: data packets and FEC packets. Packets always have a common header which is between 2 and 19 bytes, followed by an authenticated and encrypted blob. When the blob is decrypted, the results shown below include a 1 or 2 byte header, followed by additional data. FEC packets contain parity data which allows dropped packets to be recovered. Data packets contain stream payload data, ACK data, and control data (e.g. streams or session termination).

The simple header is designed to make reading and writing of packets easy.

All integer values, including length, version, and type, are in little-endian byte order, **NOT** network byte order. QUIC does not enforce alignment of types in dynamically sized frames.

QUIC Packet Header

All packets begin with a common header



The two bits that signify sequence number length are only meaningful for data packets. For public reset and version negotiation packets (when sent by the server), which don't have a sequence number, these two bits are not used and must both be set to 0.

Public Flags: There are 5 groups of bits, consisting of two single bits, two pairs of used bits, and one pair of unused bits.

- LSB 0x1 has value 1 iff the packet contains a Quic Version. This bit must be set by a client in all packets until confirmation from a server arrives agreeing to the proposed version is received by the client. A server indicates agreement on a version by sending packets without setting this bit.
- Bit at location, 0x2, is set to indicate that the packet is a Public Reset packet.
- Pair of bits, included in 0xC, together indicate the size of the connection ID that is present in the packet, but should be set to set to 0xC in all packets until agreeably negotiated to a different value,

for a given direction (e.g., client may request fewer bytes of the connection id be presented). Within this 2 bit mask:
    - 0xC indicates that an 8 byte connection id is present;
    - 0x8 indicates that a 4 byte connection id is present;
    - 0x4 indicates that a 1 byte connection id is used;
    - 0x0 indicates that the connection id is omitted.
- Pair of bits included in 0x30 indicate the number of low-order-bytes of the packet sequence number that are present in each packet. Within this 2 bit mask:
    - 0x30 indicates that 6 bytes of the sequence number is present
    - 0x20 indicates that 4 bytes of the sequence number is present
    - 0x10 indicates that 2 bytes of the sequence number is present
    - 0x00 indicates that 1 byte of the sequence number is present
- Pair of bits include in 0xC0 are currently unused, and should be must be set to 0.

Connection ID: This is an unsigned 64 bit statistically random number selected by the client that is the identifier of the connection. Because QUIC sessions are designed to remain established even if the client roams, the IP 4-tuple (source IP, source port, destination IP, destination port) may be insufficient to identify the connection. For each transmission direction, when less uniqueness is sufficient to identify the connection, a truncated transmitted connection ID length is negotiable.

QUIC Version: A 32 bit opaque tag that represents the version of the QUIC protocol. Only present if the public flags contain FLAG_VERSION (i.e public_flags & FLAG_VERSION !=0). A client may set this flag, and include EXACTLY one proposed version, as well as including arbitrary data (conforming to that version). A server may set this flag when the client-proposed version was unsupported,, and may then provide a list (0 or more) of acceptable versions, but MUST not include any data after the version(s). Examples of version values in recent experimental versions include “Q015” which corresponds to byte 9 containing ‘Q”, byte 10 containing ‘0”, etc. [See list of changes in various versions listed at the end of this document.]

Sequence Number: The lower 8, 16, 32, or 48 bits of the sequence number, based on which FLAG_?BYTE_SEQUENCE_NUMBER flag is set in the public flags. See “Sequence numbers” below.

Private Flags:

- 0x01 = FLAG_ENTROPY - for data packets, signifies that this packet contains the 1 bit of entropy, for fec packets, contains the xor of the entropy of protected packets.
- 0x02 = FLAG_FEC_GROUP - indicates whether the fec byte is present.
- 0x04 = FLAG_FEC - signifies that this packet represents an FEC packet.

FEC (FEC Group Number Offset): An FEC Group Number is the Packet Sequence Number of the first packet in the FEC group. The FEC Group Number Offset is an 8 bit unsigned value which should be subtracted from the current packet’s Packet Sequence Number to yield the FEC Group Number for this packet. This is only present if the private flags contain FLAG_FEC_GROUP. All packets within a single FEC group must have Sequence Numbers encoded into an identical number of bytes (i.e., the Sequence Number coding must not change during a group).

Sequence numbers

Each QUIC data packet (as opposed to public reset and version negotiation packets) is assigned a sequence number by the sender. The first packet sent by an endpoint shall have a sequence number of 1,

and each subsequent packet shall have a sequence number one larger than that of the previous packet.

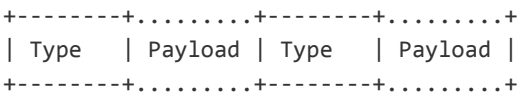
The lower 64 bits of the sequence number may be used as part of a cryptographic nonce; therefore, a QUIC endpoint must not send a packet with a sequence number that cannot be represented in 64 bits. If a QUIC endpoint transmits a packet with a sequence number of (2^64-1), that packet must include a CONNECTION_CLOSE frame with an error code of QUIC_SEQUENCE_NUMBER_LIMIT_REACHED, and the endpoint must not transmit any additional packets.

At most the lower 48 bits of a sequence number are transmitted. To enable unambiguous reconstruction of the sequence number by the receiver, a QUIC endpoint must not transmit a packet whose sequence number is larger by (2^(bitlength-2)) than the largest sequence number for which an acknowledgement is known to have been transmitted by the receiver. Therefore, there must never be more than (2^46) packets in flight.

Any truncated sequence number shall be inferred to have the value closest to the one more than the largest known sequence number of the endpoint which transmitted the packet that originally contained the truncated sequence number. The transmitted portion of the sequence number matches the lowest bits of the inferred value.

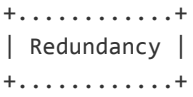
Data Packets

Data packets (those packets with FLAG_DATA set) have a payload that is a series of type-prefixed frames. The format of frame types is defined later.



FEC Packets

FEC packets (those packets with FLAG_FEC set) have a payload that simply contains an XOR of the null-padded payload of each Data Packet in the FEC group.



Streams

Streams are independent sequences of bi-directional data cut into stream frames. Streams can be created either by the client or the server; can concurrently send data interleaved with other streams; and can be cancelled. The usage of streams with SPDY is that SPDY streams map to QUIC streams, using the QUIC stream ID, cancel using QUIC's RST_STREAM frame etc. See SPDY layering section for details.

Stream creation

Stream creation is done implicitly, by sending a STREAM frame for a given stream. If the server is initiating

the stream, the Stream-ID must be even. If the client is initiating the stream, the Stream-ID must be odd. 0 is not a valid Stream-ID. Stream 1 is reserved for the crypto handshake, which should be the first client-initiated stream. Stream 3 is reserved for transmitting compressed headers for all other streams, ensuring reliable in-order delivery and processing of headers.

Stream-IDs from each side of the connection must increase monotonically as new streams are ceated. E.g. Stream 2 may be created after stream 3, but stream 7 must not be created after stream 9. The peer may *receive* streams out of order. For example, if a server receives packet 10 including frames for stream 9 before it receives packet 9 including frames for stream 7, it should handle this gracefully.

If the endpoint receiving a STREAM frame does not want to accept the stream, it can immediately respond with a RST_STREAM frame. Note, however, that the initiating endpoint may have already sent data on the stream as well; this data must be ignored.

Stream data exchange

Once a stream is created, it can be used to send arbitrary amounts of data. Generally this means that a series of frames will be sent on the stream until a frame containing the fin bit is set. Once the FIN has been sent, the stream is considered to be half-closed. (See Stream half-close)

Stream half-close

When one side of the stream sends a frame with FIN set to true, the stream is considered to be half-closed from that side. The sender of the FIN is indicating that no further data will be sent from the sender on this stream. When both sides have half-closed, the stream is considered to be closed. (See Stream close)

Stream close

There are four ways that streams can be terminated:

1. **Normal termination**
   Normal stream termination occurs when both sender and receiver have half-closed the stream by sending a FIN. Ideally the FIN will be sent with the last data for the stream, for optimal byte packing.
2. **Abrupt termination**
   Either the client or server can send a RST_STREAM frame for a stream at any time. A RST_STREAM frame contains an error code to indicate the reason for failure. When a RST_STREAM frame is sent from the stream originator, it indicates a failure to complete the stream and that no further data will be sent on the stream. When a RST_STREAM frame is sent from the stream receiver, the sender, upon receipt, should stop sending any data on the stream. The stream receiver should be aware that there is a race between data already in transit from the sender and the time the RST_STREAM frame is received.
3. **QUIC connection teardown**
   If the QUIC connection is torn down via a CONNECTION_CLOSE frame while unterminated streams are active (no RST_STREAM frames have been sent or received for the stream), then the endpoint must assume that the stream was abnormally interrupted and may be incomplete.
4. **Implicit timeout**
   The default idle timeout for a QUIC connection is 30 seconds, and is a required parameter(“ICSL”) in connection negotiation. The maximum is 10 minutes. If there is no network activity for the

duration of the idle timeout, the connection is closed. By default a CONNECTION_CLOSE frame will be sent. A silent close option can be enabled when it is expensive to send an explicit close, such as mobile networks that must wake up the radio.

Frames

Frame types

There are two interpretations for the Frame Type byte, resulting in two frame types: Special Frame Types, and Regular Frame Types. Special Frame Types encode both a Frame Type and corresponding flags all in the Frame Type byte, while Regular Frame Types use the Frame Type byte simply.

Currently defined Special Frame Types are:

Type-field value	Control Frame-type
1fdooossB	STREAM
01ntllmmB	ACK
001xxxxxB	CONGESTION_FEEDBACK

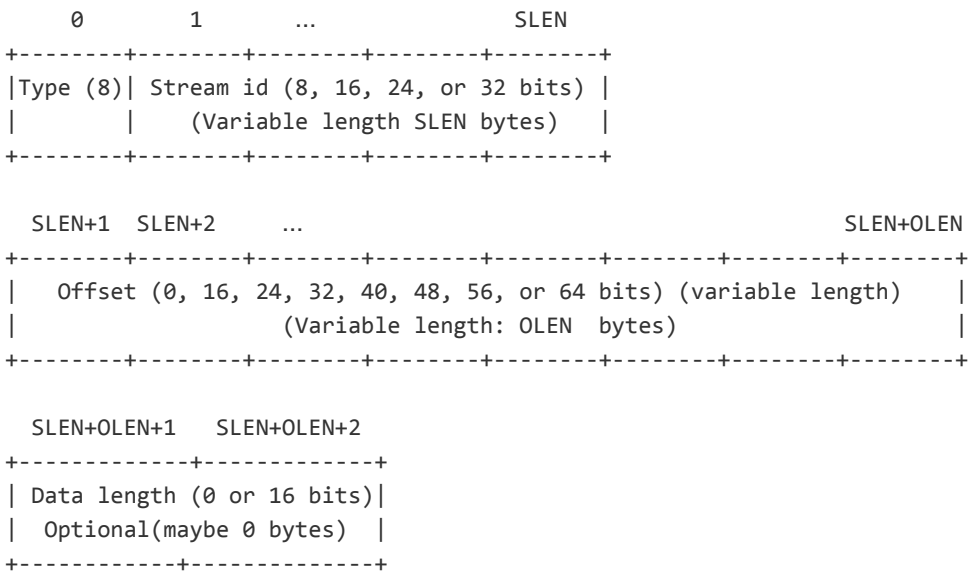
Currently defined Regular Frame Types are:

Type-field value	Control Frame-type
00000000B (0x00)	PADDING
00000001B (0x01)	RST_STREAM
00000010B (0x02)	CONNECTION_CLOSE
00000011B (0x03)	GOAWAY
00000100B (0x04)	WINDOW_UPDATE
00000101B (0x05)	BLOCKED
00000110B (0x06)	STOP_WAITING
00000110B (0x07)	PING

STREAM Frame

The STREAM frame is used to both implicitly create a stream and to send data on it.

The type-field value of the stream frame contains flags to encode the StreamID, Offset, Data Length, and Fin, read from right to left.



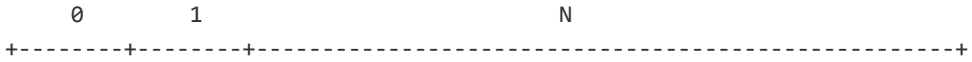
Frame type: an 8 bit value specifying the length of the stream id, the offset length, whether there is a data length, and whether it's a fin, from right to left. The leftmost bit if set to 1 indicates that this is a stream frame.
Stream Id: An 8, 16, 24, or 32 bit unsigned ID unique to this stream. The encoded length is specified by the two bits in the type-field value, with 00 corresponding to 8 bits and 11 corresponding to 32 bits.
Fin: The fin bit is set to 0 if the sender may send more data on this stream. 1 indicates the sender is done sending on this stream and wishes to half-close.
Offset: A 0, 16, 24, 32, 40, 48, 56, or 64 bit unsigned number specifying the byte offset in the stream for this block of data. The encoded length is specified by the three offset bits in the type-field value, with 000 corresponding to 0 bits and 111 corresponding to 64 bits.
Data length: An optional uint16 representing length of the data following this stream frame. This length is present when the d bit is 1, and otherwise the rest of the packet contains data for the stream. The option to omit the length should only be used when the packet is a "full packet," to avoid the risk of corruption via padding.

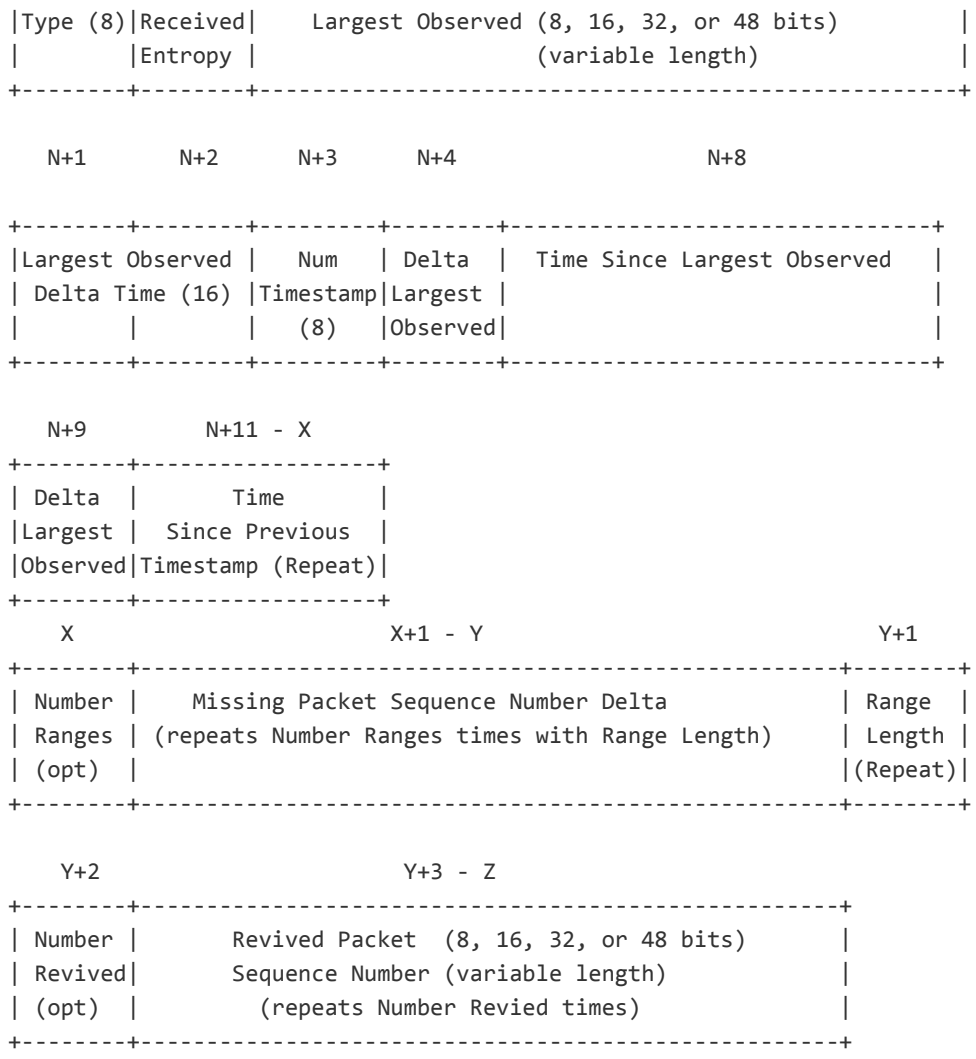
A stream frame MUST have non-zero data length, or have the FIN bit set.

ACK Frame

The ACK frame is sent to inform the peer which packets have been received, as well as which packets are still considered missing by the receiver (the contents of missing packets may need to be resent).

The ack frame re-uses the idea of two bits in the flags to indicate 1, 2, 4, or 6 byte sequence number deltas.





Frame type: Each ack frame starts with an 8 bit value specifying that this is a ack frame (01ntllmmB).

The last 6 bits are flags:
n = 1 bit to indicate whether the frame has any missing packet ranges (Nack ranges).
t = 1 bit to indicate whether the ack frame has been truncated.
ll = 2 bits to indicate the largest observed sequence number length.
mm = 2 bits to indicate the sequence number length of deltas between missing packet ranges.

Received Entropy: An 8 bit unsigned value specifying the cumulative hash of entropy in all received packets up to the largest observed packet.
Largest Observed: A 1, 2, 4, or 6 byte unsigned value representing the largest sequence number the peer has observed. When an ACK frame is truncated, it indicates a sequence number greater than the specified largest observed has been received, but information about those additional receptions can't fit into this frame (typically due to packet size restrictions).
Largest Observed Delta Time: A 16 bit unsigned float with 11 explicit bits of mantissa and 5 bits of explicit exponent, specifying the time elapsed in microseconds from when largest observed was received until this Ack frame was sent. The bit format is loosely modeled after IEEE 754. For example, 1 microsecond is

represented as 0x1, which has an exponent of zero, presented in the 5 high order bits, and mantissa of 1, presented in the 11 low order bits. When the explicit exponent is greater than zero, an implicit high-order 12th bit of 1 is assumed in the mantissa. For example, a floatingvalue of 0x800 has an explicit exponent of 1, as well as an explicit mantissa of 0, but then has an effective mantissa of 4096 (12th bit is assumed to be 1). Additionally, the actual exponent is one-less than the explicit exponent, and the value represents 4096 microseconds. Any values larger than the representable range are clamped to 0xFFFF.
Num Timestamp: An 8 bit unsigned value specifying the number of TCP timestamps that are included in this frame. There will be this many pairs of <sequence number, timestamp> following in the timestamps.
Delta Largest Observed: An 8 bit unsigned value specifying the sequence number delta from the first timestamp to the largest observed.
Time Since Largest Observed: A 32 bit unsigned value specifying the first timestamp. This is the time delta in microseconds from the time the receiver's packet framer was created.
Time Since Previous Timestamp: A 16 bit unsigned value specifying the first timestamp. This is the time delta from the previous timestamp.
Num Ranges: An optional 8 bit unsigned value specifying the number of missing packet ranges between largest observed and least unacked. Only present if the n flag bit is 1.
Missing Packet Sequence Number Delta: A 1, 2, 4, or 6 byte sequence number delta. For the first missing packet range, it is a delta from the largest observed. For subsequent nack ranges, it is the number of packets received between ranges. In the case of the first nack range, a value of 0 specifies that the packet reported as the largest observed is missing. In the case of the later nack ranges, a value of 0 indicates the missing packet ranges are contiguous (used only when more than 256 packets in a row were lost).
Range Length: An 8 bit unsigned value specifying one less than the number of sequential nacks in the range.
Num Revived: An 8 bit unsigned value specifying the number of revived packets. Just like Num Ranges, this is only present if the n flag bit is 1.
Revived Packet Sequence Number: A 1, 2, 4, or 6 byte unsigned value representing a packet the peer has revived via FEC. Its byte length is identical to the Largest Observed field. All packets numbers in this list are sorted in ascending order (smallest first) and must also be in the list of missing packets.

Entropy Accumulation

The entropy bits for a subset of packets (known to a receiver or sender) are accumulated into an 8 bit unsigned value, and similarly presented in both a STOP_WAITING frame and an ACK frame. If we defined E(k) to be the FLAG_ENTROPY bit present in packet sequence number k, then the k'th packet's contribution C(k) is defined to be E(k) left shifted by k mod 8 bits. The accumulated entropy is then the bitwise-XOR sum of the contributions C(k), for all packets in the desired subset.

STOP_WAITING Frame

The STOP_WAITING frame is sent to inform the peer that it should not continue to wait for packets with sequence numbers earlier than a specified value.

The sequence number is encoded in 1, 2, 4 or 6 bytes, using the same coding length as is specified for the sequence number for the enclosing packet's header [See Public Flags].

```
     0        1        2        3        4        5        6      7
+--------+--------+--------+--------+--------+--------+--------+--------+
```

```
|Type (8)|Sent    |   Least unacked delta (8, 16, 32, or 48 bits)       |
|        |Entropy |                       (variable length)             |
+--------+--------+--------+--------+--------+--------+--------+---------
```

Frame type: an 8-bit value specifying that this is a stop waiting frame (0x06).

Sent Entropy: An 8 bit unsigned value specifying the cumulative hash of entropy in all sent packets up to the packet with sequence number one less than the least unacked packet. [See “Entropy Accumulation” section in the ACK frame section for details of this calculation.]
Least Unacked Delta: A variable length sequence number delta with the same length as the packet header’s sequence number. In the case of an FEC revived packet, the same length as the other packets in the FEC group. Subtract it from the header's packet sequence number to determine the least unacked. The resulting least unacked is the smallest sequence number of any packet for which the sender is still awaiting an ack. If the receiver is missing any packets smaller than this value, the receiver should consider those packets to be irrecoverably lost.

WINDOW_UPDATE Frame
The WINDOW_UPDATE frame is used to inform the peer of an increase in an endpoint’s flow control receive window. The stream ID can be 0, indicating this WINDOW_UPDATE applies to the connection level flow control window, or > 0 indicating that the specified stream should increase its flow control window.

An absolute byte offset is specified, and the receiver of a WINDOW_UPDATE frame may only send up to that number of bytes on the specified stream. Violating flow control by sending further bytes will result in the receiving endpoint closing the connection.

On receipt of multiple WINDOW_UPDATE frames for a specific stream ID, it is only necessary to keep track of the maximum byte offset.

Both stream and session windows start with a default value of 16 KB, but this is typically increased during the handshake. To do this, an endpoint should include SFCW (Stream Flow Control Window) and CFCW (Connection/Session Flow Control Window) tags in the CHLO/SHLO (tags are described in the crypto doc). The value associated with each tag should be the number of bytes for initial stream window and initial connection window respectively.

```
    0        1        2        3         4        5        6        7
+--------+--------+--------+--------+--------+--------+--------+--------+
|Type(8) |          Stream ID (32 bits)      |       Byte offset (64 bits) ->
+--------+--------+--------+--------+--------+--------+--------+--------+
     8        9        10       11     12
+--------+--------+--------+--------+--------+
   Byte offset continued                     |
+--------+--------+--------+--------+--------+
```

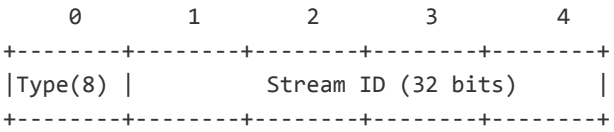
Stream ID: ID of the stream whose flow control windows is begin updated, or 0 to specify the connection-level flow control window.
Byte offset: A 64-bit unsigned integer indicating the absolute byte offset of data which can be sent on the

given stream. In the case of connection level flow control, the cumulative number of bytes which can be sent on all currently open streams.

BLOCKED Frame

The BLOCKED frame is used to indicate to the remote endpoint that this endpoint is ready to send data (and has data to send), but is currently flow control blocked. If the stream ID field is > 0, this indicates that the connection is flow control blocked on a specific stream. If the stream ID field = 0, this indicates that the connection is flow control blocked at the connection level.

This is a purely informational frame, which is extremely useful for debugging purposes. A receiver of a BLOCKED frame should simply discard it (after possibly printing a helpful log message).



CONGESTION_FEEDBACK Frame

The CONGESTION_FEEDBACK frame is an experimental frame currently not used. It is intended to provide extra congestion feedback information outside the scope of the standard ack frame.

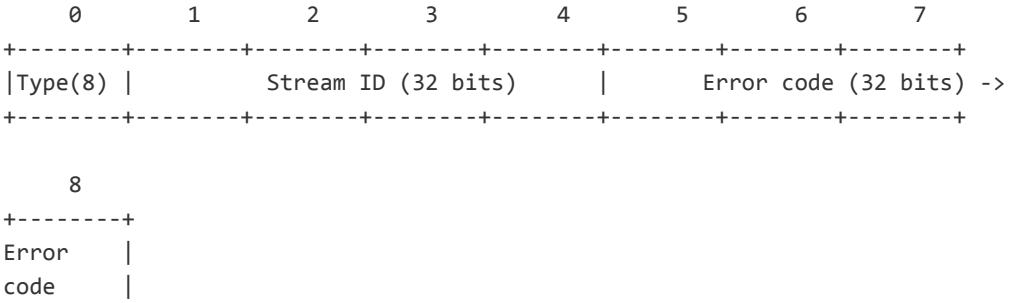
Frame type: Each congestion control frame starts with an 8 bit value specifying that this is a congestion control frame (001xxxxxB). The last 5 bits are reserved for future use.

PADDING Frame

The PADDING frame pads a packet with 0x00 bytes. When this frame is encountered, the rest of the packet is expected to be padding bytes. The frame contains 0x00 bytes and extends to the end of the QUIC packet.
Frame type: an 8-bit value specifying that this is a padding frame (0x00).

RST_STREAM Frame

The RST_STREAM frame allows for abnormal termination of a stream. When sent by the creator of a stream, it indicates the creator wishes to cancel the stream. When sent by the receiver of a stream, it indicates an error or that the receiver did not want to accept the stream, so the stream should be closed.



```
+--------+
```

Frame type: an 8 bit value specifying that this is a stream rst frame (0x4)
Stream Id: A 32 bit ID unique to this stream.
Error code: A 32-bit QuicErrorCode which indicates why the stream is being closed.

CONNECTION_CLOSE frame

The CONNECTION_CLOSE frame allows for notification that the connection is being aborted: it can be compared to closing a TCP connection. If there are streams in flight, those streams are all implicitly closed when the connection is closed. Ideally, a GOAWAY frame would be sent with enough time that all streams are torn down.

```
     0        1        2        3         4        5        6      7        X
+--------+--------+--------+--------+--------+--------+--------+--------+--------+
|Type(8) |         Error code (32 bits)      | Reason phrase   |  Reason phrase  |
|        |                                   | length (16 bits)|(variable length)|
+--------+--------+--------+--------+--------+--------+--------+--------+--------+
```

Frame type: an 8 bit value specifying that this is a connection close frame (0x5)
Error code: 32 bits containing the QuicErrorCode which indicates why the connection is being closed.
Reason phrase length: a uint16 specifying the length of the reason phrase. This may be zero if the sender chooses to not give details beyond the error code.
Reason phrase: An optional human-readable explanation for why the connection was closed.

GOAWAY Frame

The GOAWAY frame allows for notification that the connection should stop being used, and will likely be aborted in the future. Any active streams will continue to be processed, but the sender of the GOAWAY will not initiate any additional streams, and will not accept any new streams.

```
     0        1        2        3         4        5        6        7
+--------+--------+--------+--------+--------+--------+--------+--------+
|Type(8) |         Error code (32 bits)      | Last Good Stream id (32 bits) ->
+--------+--------+--------+--------+--------+--------+--------+--------+

     8        9        10        X
+--------+--------+--------+--------+--------+
 stream  | Reason phrase   |  Reason phrase    ->
   id    | length (16 bits)|(variable length)
+--------+--------+--------+--------+--------+
```

Frame type: an 8 bit value specifying that this is a goaway frame (0x6)
Error code: 32 bits containing the QuicErrorCode which indicates why the connection is being closed.
Last good stream Id: The last stream id which was accepted by the sender of the GOAWAY message. If no streams were replied to, this value MUST be 0.
Reason phrase: An optional human-readable explanation for why the connection was closed.

Reason phrase length: a uint16 specifying the length of the reason phrase. This may be zero if the sender chooses to not give details beyond the error code.
Reason phrase: An optional human-readable explanation for why the connection was closed.

PING frame

The PING frame can be used by an endpoint to verify that a peer is still alive. The PING frame contains no payload. The receiver of a PING frame simply needs to ACK the packet containing this frame.

The PING frame should be used to keep a connection alive when a stream is open. The default is to do this after 15 seconds of quiescence, which is much shorter than most NATs time out.

```
     0
+--------+
|Type(8) |
+--------+
```

QuicErrorCodes

The number to code mappings for QuicErrorCodes are currently defined in [quic_protocol.h]
TODO: **hard code numbers and add them here**

0: **QUIC_NO_ERROR.** There was no error. This is not valid for RST_STREAM frames or CONNECTION_CLOSE frames
QUIC_STREAM_DATA_AFTER_TERMINATION: There were data frames after the a fin or reset.
QUIC_SERVER_ERROR_PROCESSING_STREAM: There was some server error which halted stream processing.
QUIC_MULTIPLE_TERMINATION_OFFSETS: The sender received two mismatching fin or reset offsets for a single stream.
QUIC_BAD_APPLICATION_PAYLOAD: The sender received bad application data.
QUIC_INVALID_PACKET_HEADER: The sender received a malformed packet header.
QUIC_INVALID_FRAME_DATA: The sender received an frame data. The more detailed error codes below are prefered where possible.
QUIC_INVALID_FEC_DATA: FEC data is malformed.
QUIC_INVALID_RST_STREAM_DATA: Stream rst data is malformed
QUIC_INVALID_CONNECTION_CLOSE_DATA: Connection close data is malformed.
QUIC_INVALID_ACK_DATA: Ack data is malformed.
QUIC_DECRYPTION_FAILURE: There was an error decrypting.
QUIC_ENCRYPTION_FAILURE: There was an error encrypting.
QUIC_PACKET_TOO_LARGE: The packet exceeded kMaxPacketSize.
QUIC_PACKET_FOR_NONEXISTENT_STREAM: Data was sent for a stream which did not exist.
QUIC_CLIENT_GOING_AWAY: The client is going away (browser close, etc.)
QUIC_SERVER_GOING_AWAY: The server is going away (restart etc.)
QUIC_INVALID_STREAM_ID: A stream ID was invalid.
QUIC_TOO_MANY_OPEN_STREAMS: Too many streams already open.

QUIC_CONNECTION_TIMED_OUT: We hit our prenegotiated (or default) timeout
QUIC_CRYPTO_TAGS_OUT_OF_ORDER: Handshake message contained out of order tags.
QUIC_CRYPTO_TOO_MANY_ENTRIES: Handshake message contained too many entries.
QUIC_CRYPTO_INVALID_VALUE_LENGTH: Handshake message contained an invalid value length.
QUIC_CRYPTO_MESSAGE_AFTER_HANDSHAKE_COMPLETE: A crypto message was received after the handshake was complete.
QUIC_INVALID_CRYPTO_MESSAGE_TYPE: A crypto message was received with an illegal message tag.
QUIC_SEQUENCE_NUMBER_LIMIT_REACHED: Transmitting an additional packet would cause a sequence number to be reused.

Encryption

All QUIC packets are encrypted except for the initial packets where the shared secret is being established. The details of QUIC crypto can be found here

Initial Packet Null Encryption

The initial packets, containing CHLO, SHLO, and REJ messages are not encrypted, but do include a message authentication hash. The body of the packet, placed identically to where the encrypted payload would otherwise appear, consists of a 12 byte hash, followed by the plaintext payload. The hash is computed over the concatenation of the associated data, plus plaintext, in that order. The hash is an FNV1a-128 hash, serialized in little endian order, with only the first 12 bytes of the presented in the packet.

Priority

TODO: implement

QUIC will use the SPDY prioritization mechanism. Roughly speaking, a stream may be dependent on another stream. In this situation, the “parent” stream should effectively starve the “child” stream. In addition, parent streams have an explicit priority. Parent streams should not starve other parent streams, but should make progress proportional to their relative priority.

SPDY Layering over QUIC

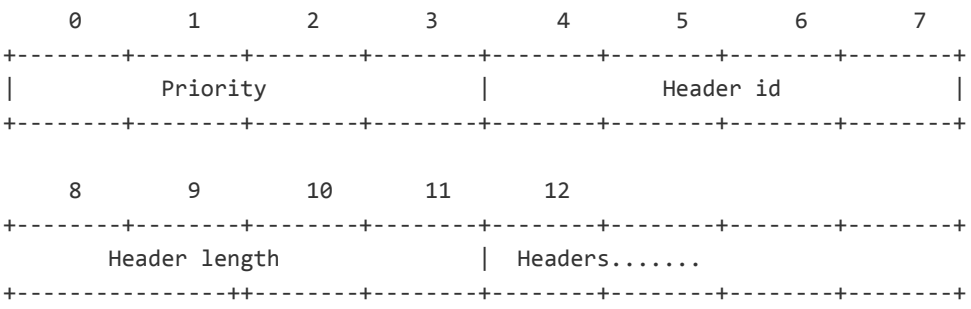
Stream management

When SPDY is sent over QUIC, the QUIC layer handles most of the stream management. SPDY stream IDs are replaced by QUIC stream IDs, and stream establishment is done by sending data on that stream. There’s no explicit framing: the data sent over the QUIC stream simply consists of SPDY headers followed by the body. As with SPDY, the requests and responses are considered complete when the appropriate sender sends a frame with the FIN bit set to true. Flow control will also be handled at the QUIC layer.

Streams can be closed off if both sides set the fin flag, or via either side sending a RST_STREAM frame.

Stream parsing

Because there's no SYN stream for frames, some of the metadata in the TCP version of spdy is pre-pended to the beginning of the spdy stream as follows:



Priority type: a uint32 representing the stream's priority (spdy3 style)
Header id: uint32 specifying the header id (see "compression" below)
Header length: a uint32 specifying the length of the compressed headers
Headers: spdy3-style compressed headers
Body: the body (if any) of the request

Compression

QUIC uses the same HPACK header compression as found in HTTP/2, which unfortunately means that QUIC headers are head-of-line blocking because header blocks must be decompressed in the order they were compressed. The order of header blocks may not match the order of streams; server-side especially streams may responded to out-of-order (if headers for stream 5 are ready to go before headers for stream 3, for example). To make sure the headers are decompressed in the order they were compressed, headers are delivered on a dedicated headers stream, with stream id 3, in the form of HTTP/2 HEADERS frames..

The long term plan is to tweak the compressor and decompressor in QUIC so that the compressed output does not depend on unacked previous compressed state. This could be done, perhaps, by creating "checkpoints" of HPACK state which are updated when headers have been ACK'd. When compressing headers QUIC would only compress relative to the previous "checkpoint". This is all speculative for now.

Deployment

Unlike SPDY, one can not upgrade to QUIC as part of the SSL handshake: by the time the handshake has started, client-server interaction is already doing SSL and TCP, neither of which applies to QUIC. Instead, the alternate-protocol header is used.

To specify QUIC as an alternate protocol available on port 123, use:

```
Alternate-Protocol: 123:quic
```

When a client receives a Alternate-Protocol header advertising QUIC, it should attempt to use QUIC for future secure connections on that domain. Note that there is no guarantee that the port is open between

client and server, or that intermediate servers will route packets. If the client begins communicating over QUIC and does not get responses in a timely fashion it should fail gracefully back over to the prior protocol. It should persist the failure for the browser session so that it doesn't re-attempt in the near future.

Note that the server may reply with multiple field values or a comma-separated field value for Alternate-Protocol (and in this manner indicate the various SPDY versions and transports it supports).

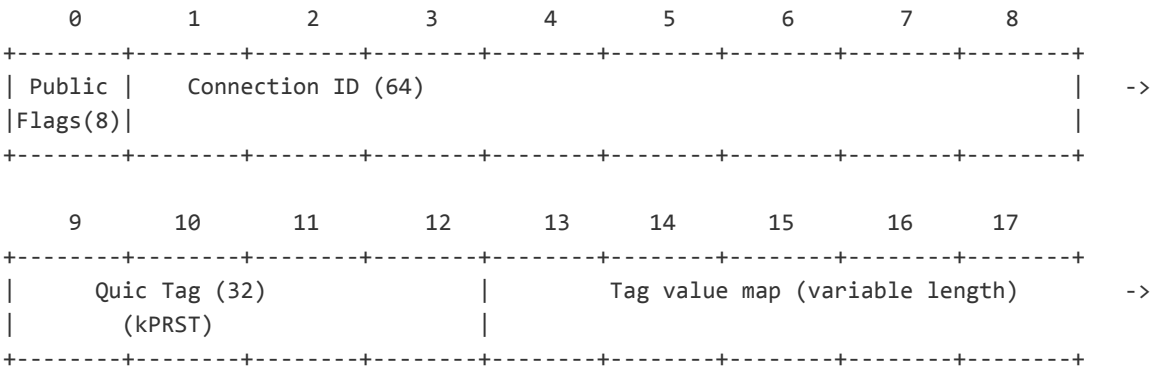
A server can also send a header to notify that QUIC should not be used on this domain. If it sends the alternate-protocol-required header, the client should remember to not use QUIC on that domain in future, and not do any UDP probing to see if QUIC is available.

To mandate https rather than QUIC for a given domain, one could send:

```
Alternate-Protocol-Required: 443:https
```

Public Reset Packet

Public reset packets begin with an 8-bit public flags and 64-bit Connection ID. The rest of the public reset packets is encoded as if it were a crypto handshake message of the tag kPRST:



Tag value map: The tag value map contains the following tag-values:

- kRNON (public reset nonce proof) - a 64-bit unsigned integer. Mandatory.
- kRSEQ (rejected sequence number) - a 64-bit sequence number. Mandatory.
- kCADR (client address) - the observed client IP address and port number. Optional.

TODO: public reset packet should include authenticated (destination) server IP/port.

Recent changes by version:

- Q009: added priority as the first 4 bytes on spdy streams.
- Q010: renumber the various frame types
- Q011: shrunk the fnv128 hash on NULL encrypted packets from 16 bytes to 12 bytes.
- Q012: optimize the ack frame format to reduce the size and better handle ranges of nacks, which should make truncated acks virtually impossible. Also adding an explicit flag for truncated acks and moving the ack outside of the connection close frame.
- Q013: Compressed headers for *all* data streams are serialized into a reserved stream. This

ensures serialized handling of headers, independent of stream cancellation notification.

- Q014: Added WINDOW_UPDATE and BLOCKED frames, no behavioral change.
- Q015: Removes the accumulated_number_of_lost_packets field from the TCP and inter arrival congestion feedback frames and adds an explicit list of recovered packets to the ack frame.
- Q016: Breaks out the sent_info field from the ACK frame into a new STOP_WAITING frame.
- Changed GUID to Connection ID
- Q017: Adds stream level flow control
- Q018: Added a PING frame
- Q019: Adds session/connection level flow control
- Q020: Allow endpoints to set different stream/session flow control windows
- Q021: Crypto and headers streams are flow controlled (at stream level)
- Q023: Ack frames include packet timestamps
- Q024: HTTP/2-style header compression
- Q025: HTTP/2-style header keys. Removal of error_details from the RST_STREAM frame.