QUIC
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

M. Bishop, Ed.
Microsoft

March 13, 2017

Hypertext Transfer Protocol (HTTP) over QUIC
draft-ietf-quic-http-02

Abstract

The QUIC transport protocol has several features that are desirable in a transport for HTTP, such as stream multiplexing, per-stream flow control, and low-latency connection establishment. This document describes a mapping of HTTP semantics over QUIC. This document also identifies HTTP/2 features that are subsumed by QUIC, and describes how HTTP/2 extensions can be ported to QUIC.

Note to Readers

Discussion of this draft takes place on the QUIC working group mailing list (quic@ietf.org), which is archived at https://mailarchive.ietf.org/arch/search/?email_list=quic .

Working Group information can be found at https://github.com/quicwg ; source code and issues list for this draft can be found at https://github.com/quicwg/base-drafts/labels/http .

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 http://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 September 14, 2017.

Table of Contents

1. Introduction

The QUIC transport protocol has several features that are desirable in a transport for HTTP, such as stream multiplexing, per-stream flow control, and low-latency connection establishment. This document describes a mapping of HTTP semantics over QUIC, drawing heavily on the existing TCP mapping, HTTP/2. Specifically, this document identifies HTTP/2 features that are subsumed by QUIC, and describes how the other features can be implemented atop QUIC.

QUIC is described in [QUIC-TRANSPORT]. For a full description of HTTP/2, see [RFC7540].

1.1. Notational Conventions

The words "MUST", "MUST NOT", "SHOULD", and "MAY" are used in this document. It's not shouting; when they are capitalized, they have the special meaning defined in [RFC2119].

2. QUIC Advertisement

An HTTP origin advertises the availability of an equivalent HTTP/QUIC endpoint via the Alt-Svc HTTP response header or the HTTP/2 ALTSVC frame ([RFC7838]), using the ALPN token defined in Section 3.

For example, an origin could indicate in an HTTP/1.1 or HTTP/2 response that HTTP/QUIC was available on UDP port 443 at the same hostname by including the following header in any response:

```
Alt-Svc: hq=":443"
```

On receipt of an Alt-Svc header indicating HTTP/QUIC support, a client MAY attempt to establish a QUIC connection to the indicated host and port and, if successful, send HTTP requests using the mapping described in this document.

Connectivity problems (e.g. firewall blocking UDP) can result in QUIC connection establishment failure, in which case the client SHOULD continue using the existing connection or try another alternative endpoint offered by the origin.

2.1. QUIC Version Hints

This document defines the "quic" parameter for Alt-Svc, which MAY be used to provide version-negotiation hints to HTTP/QUIC clients. QUIC versions are four-octet sequences with no additional constraints on format. Syntax:

```
quic = version-number
version-number = 1*8HEXDIG; hex-encoded QUIC version
```

Leading zeros SHOULD be omitted for brevity. When multiple versions are supported, the "quic" parameter MAY be repeated multiple times in a single Alt-Svc entry. For example, if a server supported both version 0x00000001 and the version rendered in ASCII as "Q034", it could specify the following header:

```
Alt-Svc: hq=":443";quic=1;quic=51303334
```

Where multiple versions are listed, the order of the values reflects the server's preference (with the first value being the most preferred version). Origins SHOULD list only versions which are supported by the alternative, but MAY omit supported versions for any reason.

3. Connection Establishment

HTTP/QUIC connections are established as described in [QUIC-TRANSPORT]. During connection establishment, HTTP/QUIC support is indicated by selecting the ALPN token "hq" in the crypto handshake.

While connection-level options pertaining to the core QUIC protocol are set in the initial crypto handshake, HTTP-specific settings are conveyed in the SETTINGS frame. After the QUIC connection is established, a SETTINGS frame (Section 5.2.3) MUST be sent as the initial frame of the HTTP control stream (StreamID 3, see Section 4). The server MUST NOT send data on any other stream until the client's SETTINGS frame has been received.

3.1. Draft Version Identification

> *RFC Editor's Note:* Please remove this section prior to publication of a final version of this document.

Only implementations of the final, published RFC can identify themselves as "hq". Until such an RFC exists, implementations MUST NOT identify themselves using this string.

Implementations of draft versions of the protocol MUST add the string "-" and the corresponding draft number to the identifier. For example, draft-ietf-quic-http-01 is identified using the string "hq-01".

Non-compatible experiments that are based on these draft versions MUST append the string "-" and an experiment name to the identifier. For example, an experimental implementation based on draft-ietf-quic-http-09 which reserves an extra stream for unsolicited transmission of 1980s pop music might identify itself as "hq-09-rickroll". Note that any label MUST conform to the "token" syntax defined in Section 3.2.6 of [RFC7230]. Experimenters are encouraged to coordinate their experiments on the quic@ietf.org mailing list.

4. Stream Mapping and Usage

A QUIC stream provides reliable in-order delivery of bytes, but makes no guarantees about order of delivery with regard to bytes on other streams. On the wire, data is framed into QUIC STREAM frames, but this framing is invisible to the HTTP framing layer. A QUIC receiver buffers and orders received STREAM frames, exposing the data contained within as a reliable byte stream to the application.

QUIC reserves Stream 1 for crypto operations (the handshake, crypto config updates). Stream 3 is reserved for sending and receiving HTTP control frames, and is analogous to HTTP/2's Stream 0. This connection control stream is considered critical to the HTTP connection. If the connection control stream is closed for any reason, this MUST be treated as a connection error of type QUIC_CLOSED_CRITICAL_STREAM.

When HTTP headers and data are sent over QUIC, the QUIC layer handles most of the stream management. An HTTP request/response consumes a pair of streams: This means that the client's first request occurs on QUIC streams 5 and 7, the second on stream 9 and 11, and so on. The server's first push consumes streams 2 and 4. This amounts to the second least-significant bit differentiating the two streams in a request.

The lower-numbered stream is called the message control stream and carries frames related to the request/response, including HEADERS. The higher-numbered stream is the data stream and carries the request/response body with no additional framing. Note that a request or response without a body will cause this stream to be half-closed in the corresponding direction without transferring data.

Because the message control stream contains HPACK data which manipulates connection-level state, the message control stream MUST NOT be closed with a stream-level error. If an implementation chooses to reject a request with a QUIC error code, it MUST trigger a QUIC RST_STREAM on the data stream only. An implementation MAY close (FIN) a message control stream without completing a full HTTP message if the data stream has been abruptly closed. Data on message control streams MUST be fully consumed, or the connection terminated.

All message control streams are considered critical to the HTTP connection. If a message control stream is terminated abruptly for any reason, this MUST be treated as a connection error of type HTTP_RST_CONTROL_STREAM. When a message control stream terminates cleanly, if the last frame on the stream was truncated, this MUST be treated as a connection error (see HTTP_MALFORMED_* in Section 6.1).

Pairs of streams must be utilized sequentially, with no gaps. The data stream is opened at the same time as the message control stream is opened and is closed after transferring the body. The data stream is closed immediately after sending the request headers if there is no body.

HTTP does not need to do any separate multiplexing when using QUIC - data sent over a QUIC stream always maps to a particular HTTP transaction. Requests and responses are considered complete when the corresponding QUIC streams are closed in the appropriate direction.

4.1. Stream 3: Connection Control Stream

Since most connection-level concerns will be managed by QUIC, the primary use of Stream 3 will be for the SETTINGS frame when the connection opens and for PRIORITY frames subsequently.

4.2. HTTP Message Exchanges

A client sends an HTTP request on a new pair of QUIC streams. A server sends an HTTP response on the same streams as the request.

An HTTP message (request or response) consists of:

1. one header block (see Section 5.2.1) on the control stream containing the message headers (see [RFC7230], Section 3.2),

2. the payload body (see [RFC7230], Section 3.3), sent on the data stream,

3. optionally, one header block on the control stream containing the trailer-part, if present (see [RFC7230], Section 4.1.2).

In addition, prior to sending the message header block indicated above, a response may contain zero or more header blocks on the control stream containing the message headers of informational (1xx) HTTP responses (see [RFC7230], Section 3.2 and [RFC7231], Section 6.2).

The data stream MUST be half-closed immediately after the transfer of the body. If the message does not contain a body, the corresponding data stream MUST still be half-closed without transferring any data. The "chunked" transfer encoding defined in Section 4.1 of [RFC7230] MUST NOT be used.

Trailing header fields are carried in an additional header block on the message control stream. Such a header block is a sequence of HEADERS frames with End Header Block set on the last frame. Senders MUST send only one header block in the trailers section; receivers MUST decode any subsequent header blocks in order to maintain HPACK decoder state, but the resulting output MUST be discarded.

An HTTP request/response exchange fully consumes a pair of streams. After sending a request, a client closes the streams for sending; after sending a response, the server closes its streams for sending and the QUIC streams are fully closed.

A server can send a complete response prior to the client sending an entire request if the response does not depend on any portion of the request that has not been sent and received. When this is true, a server MAY request that the client abort transmission of a request without error by sending a RST_STREAM with an error code of NO_ERROR after sending a complete response and closing its stream. Clients MUST NOT discard responses as a result of receiving such a RST_STREAM, though clients can always discard responses at their discretion for other reasons.

4.2.1. Header Compression

HTTP/QUIC uses HPACK header compression as described in [RFC7541]. HPACK was designed for HTTP/2 with the assumption of in-order delivery such as that provided by TCP. A sequence of encoded header

blocks must arrive (and be decoded) at an endpoint in the same order in which they were encoded. This ensures that the dynamic state at the two endpoints remains in sync.

QUIC streams provide in-order delivery of data sent on those streams, but there are no guarantees about order of delivery between streams. To achieve in-order delivery of HEADERS frames in QUIC, the HPACK-bearing frames contain a counter which can be used to ensure in-order processing. Data (request/response bodies) which arrive out of order are buffered until the corresponding HEADERS arrive.

This does introduce head-of-line blocking: if the packet containing HEADERS for stream N is lost or reordered then the HEADERS for stream N+4 cannot be processed until it has been retransmitted successfully, even though the HEADERS for stream N+4 may have arrived.

DISCUSS: Keep HPACK with HOLB? Redesign HPACK to be order-invariant? How much do we need to retain compatibility with HTTP/2's HPACK?

4.2.2. The CONNECT Method

The pseudo-method CONNECT ([RFC7231], Section 4.3.6) is primarily used with HTTP proxies to establish a TLS session with an origin server for the purposes of interacting with "https" resources. In HTTP/1.x, CONNECT is used to convert an entire HTTP connection into a tunnel to a remote host. In HTTP/2, the CONNECT method is used to establish a tunnel over a single HTTP/2 stream to a remote host for similar purposes.

A CONNECT request in HTTP/QUIC functions in the same manner as in HTTP/2. The request MUST be formatted as described in [RFC7540], Section 8.3. A CONNECT request that does not conform to these restrictions is malformed. The message data stream MUST NOT be closed at the end of the request.

A proxy that supports CONNECT establishes a TCP connection ([RFC0793]) to the server identified in the ":authority" pseudo-header field. Once this connection is successfully established, the proxy sends a HEADERS frame containing a 2xx series status code to the client, as defined in [RFC7231], Section 4.3.6, on the message control stream.

All QUIC STREAM frames on the message data stream correspond to data sent on the TCP connection. Any QUIC STREAM frame sent by the client is transmitted by the proxy to the TCP server; data received from the TCP server is written to the data stream by the proxy. Note that the

size and number of TCP segments is not guaranteed to map predictably to the size and number of QUIC STREAM frames.

The TCP connection can be closed by either peer. When the client half-closes the data stream, the proxy will set the FIN bit on its connection to the TCP server. When the proxy receives a packet with the FIN bit set, it will half-close the corresponding data stream. TCP connections which remain half-closed in a single direction are not invalid, but are often handled poorly by servers, so clients SHOULD NOT half-close connections on which they are still expecting data.

A TCP connection error is signaled with RST_STREAM. A proxy treats any error in the TCP connection, which includes receiving a TCP segment with the RST bit set, as a stream error of type HTTP_CONNECT_ERROR (Section 6.1). Correspondingly, a proxy MUST send a TCP segment with the RST bit set if it detects an error with the stream or the QUIC connection.

4.3. Stream Priorities

HTTP/QUIC uses the priority scheme described in [RFC7540] Section 5.3. In this priority scheme, a given stream can be designated as dependent upon another stream, which expresses the preference that the latter stream (the "parent" stream) be allocated resources before the former stream (the "dependent" stream). Taken together, the dependencies across all streams in a connection form a dependency tree. The structure of the dependency tree changes as PRIORITY frames add, remove, or change the dependency links between streams.

For consistency’s sake, all PRIORITY frames MUST refer to the message control stream of the dependent request, not the data stream.

4.4. Server Push

HTTP/QUIC supports server push as described in [RFC7540]. During connection establishment, the client indicates whether it is willing to receive server pushes via the SETTINGS_DISABLE_PUSH setting in the SETTINGS frame (see Section 3), which defaults to 1 (true).

As with server push for HTTP/2, the server initiates a server push by sending a PUSH_PROMISE frame containing the StreamID of the stream to be pushed, as well as request header fields attributed to the request. The PUSH_PROMISE frame is sent on the control stream of the associated (client-initiated) request, while the Promised Stream ID field specifies the Stream ID of the control stream for the server-initiated request.

The server push response is conveyed in the same way as a non-server-push response, with response headers and (if present) trailers carried by HEADERS frames sent on the control stream, and response body (if any) sent via the corresponding data stream.

5. HTTP Framing Layer

Frames are used only on the connection (stream 3) and message (streams 5, 9, etc.) control streams. Other streams carry data payload and are not framed at the HTTP layer.

This section describes HTTP framing in QUIC and highlights some differences from HTTP/2 framing. For more detail on differences from HTTP/2, see Section 7.1.

5.1. Frame Layout

All frames have the following format:

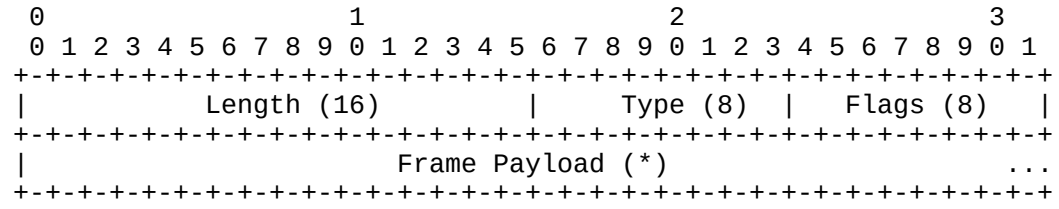

Figure 1: HTTP/QUIC frame format

5.2. Frame Definitions

5.2.1. HEADERS

The HEADERS frame (type=0x1) is used to carry part of a header set, compressed using HPACK [RFC7541].

One flag is defined:

End Header Block (0x4): This frame concludes a header block.

A HEADERS frame with any other flags set MUST be treated as a connection error of type HTTP_MALFORMED_HEADERS.

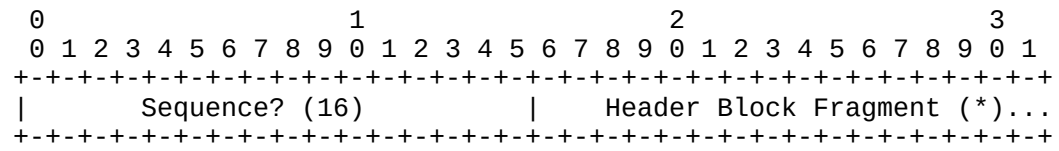

Figure 2: HEADERS frame payload

The HEADERS frame payload has the following fields:

Sequence Number: Present only on the first frame of a header block sequence. This MUST be set to zero on the first header block sequence, and incremented on each header block.

The next frame on the same stream after a HEADERS frame without the EHB flag set MUST be another HEADERS frame. A receiver MUST treat the receipt of any other type of frame as a stream error of type HTTP_INTERRUPTED_HEADERS. (Note that QUIC can intersperse data from other streams between frames, or even during transmission of frames, so multiplexing is not blocked by this requirement.)

A full header block is contained in a sequence of zero or more HEADERS frames without EHB set, followed by a HEADERS frame with EHB set.

On receipt, header blocks (HEADERS, PUSH_PROMISE) MUST be processed by the HPACK decoder in sequence. If a block is missing, all subsequent HPACK frames MUST be held until it arrives, or the connection terminated.

When the Sequence counter reaches its maximum value (0xFFFF), the next increment returns it to zero. An endpoint MUST NOT wrap the Sequence counter to zero until the previous zero-value header block has been confirmed received.

5.2.2. PRIORITY

The PRIORITY (type=0x02) frame specifies the sender-advised priority of a stream and is substantially different from [RFC7540]. In order to support ordering, it MUST be sent only on the connection control stream. The format has been modified to accommodate not being sent on-stream and the larger stream ID space of QUIC.

The semantics of the Stream Dependency, Weight, and E flag are the same as in HTTP/2.

The flags defined are:

E (0x01): Indicates that the stream dependency is exclusive (see [RFC7540] Section 5.3).

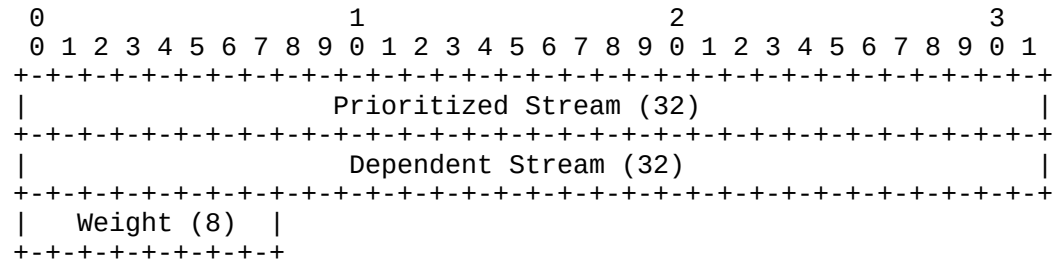

Figure 3: PRIORITY frame payload

The HEADERS frame payload has the following fields:

Prioritized Stream: A 32-bit stream identifier for the message control stream whose priority is being updated.

Stream Dependency: A 32-bit stream identifier for the stream that this stream depends on (see Section 4.3 and {!RFC7540} Section 5.3).

Weight: An unsigned 8-bit integer representing a priority weight for the stream (see [RFC7540] Section 5.3). Add one to the value to obtain a weight between 1 and 256.

A PRIORITY frame MUST have a payload length of nine octets. A PRIORITY frame of any other length MUST be treated as a connection error of type HTTP_MALFORMED_PRIORITY.

5.2.3. SETTINGS

The SETTINGS frame (type=0x4) conveys configuration parameters that affect how endpoints communicate, such as preferences and constraints on peer behavior, and is substantially different from [RFC7540]. Individually, a SETTINGS parameter can also be referred to as a "setting".

SETTINGS parameters are not negotiated; they describe characteristics of the sending peer, which can be used by the receiving peer. However, a negotiation can be implied by the use of SETTINGS - a peer uses SETTINGS to advertise a set of supported values. The recipient can then choose which entries from this list are also acceptable and proceed with the value it has chosen. (This choice could be announced in a field of an extension frame, or in its own value in SETTINGS.)

Different values for the same parameter can be advertised by each peer. For example, a client might permit a very large HPACK state table while a server chooses to use a small one to conserve memory.

Parameters MUST NOT occur more than once. A receiver MAY treat the presence of the same parameter more than once as a connection error of type HTTP_MALFORMED_SETTINGS.

The SETTINGS frame defines no flags.

The payload of a SETTINGS frame consists of zero or more parameters, each consisting of an unsigned 16-bit setting identifier and a length-prefixed binary value.

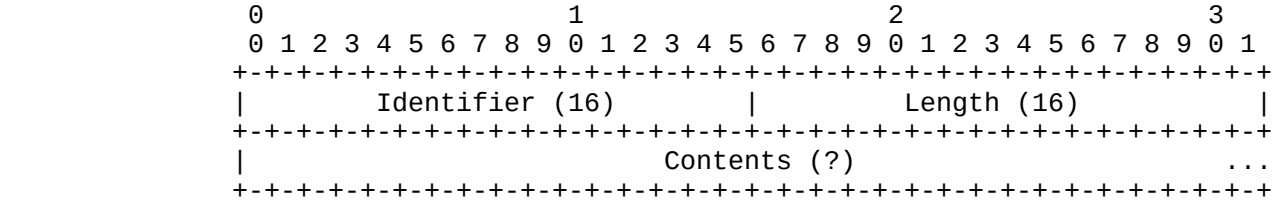

Figure 4: SETTINGS value format

A zero-length content indicates that the setting value is a Boolean and true. False is indicated by the absence of the setting.

Non-zero-length values MUST be compared against the remaining length of the SETTINGS frame. Any value which purports to cross the end of the frame MUST cause the SETTINGS frame to be considered malformed and trigger a connection error of type HTTP_MALFORMED_SETTINGS.

An implementation MUST ignore the contents for any SETTINGS identifier it does not understand.

SETTINGS frames always apply to a connection, never a single stream. A SETTINGS frame MUST be sent as the first frame of the connection control stream (see Section 4) by each peer, and MUST NOT be sent subsequently or on any other stream. If an endpoint receives an SETTINGS frame on a different stream, the endpoint MUST respond with a connection error of type HTTP_SETTINGS_ON_WRONG_STREAM. If an endpoint receives a second SETTINGS frame, the endpoint MUST respond with a connection error of type HTTP_MULTIPLE_SETTINGS.

The SETTINGS frame affects connection state. A badly formed or incomplete SETTINGS frame MUST be treated as a connection error (Section 5.4.1) of type HTTP_MALFORMED_SETTINGS.

5.2.3.1. Integer encoding

Settings which are integers are transmitted in network byte order. Leading zero octets are permitted, but implementations SHOULD use only as many bytes as are needed to represent the value. An integer MUST NOT be represented in more bytes than would be used to transfer the maximum permitted value.

5.2.3.2. Defined SETTINGS Parameters

The following settings are defined in HTTP/QUIC:

SETTINGS_HEADER_TABLE_SIZE (0x1): An integer with a maximum value of 2^32 - 1.

SETTINGS_DISABLE_PUSH (0x2): Transmitted as a Boolean; replaces SETTINGS_ENABLE_PUSH

SETTINGS_MAX_HEADER_LIST_SIZE (0x6): An integer with a maximum value of 2^32 - 1.

5.2.3.3. Usage in 0-RTT

When a 0-RTT QUIC connection is being used, the client’s initial requests will be sent before the arrival of the server’s SETTINGS frame. Clients SHOULD cache at least the following settings about servers:

- SETTINGS_HEADER_TABLE_SIZE
- SETTINGS_MAX_HEADER_LIST_SIZE

Clients MUST comply with cached settings until the server’s current settings are received. If a client does not have cached values, it SHOULD assume the following values:

- SETTINGS_HEADER_TABLE_SIZE: 0 octets
- SETTINGS_MAX_HEADER_LIST_SIZE: 16,384 octets

Servers MAY continue processing data from clients which exceed its current configuration during the initial flight. In this case, the client MUST apply the new settings immediately upon receipt.

If the connection is closed because these or other constraints were violated during the 0-RTT flight (e.g. with HTTP_HPACK_DECOMPRESSION_FAILED), clients MAY establish a new connection and retry any 0-RTT requests using the settings sent by

the server on the closed connection. (This assumes that only requests that are safe to retry are sent in 0-RTT.) If the connection was closed before the SETTINGS frame was received, clients SHOULD discard any cached values and use the defaults above on the next connection.

5.2.4. PUSH_PROMISE

The PUSH_PROMISE frame (type=0x05) is used to carry a request header set from server to client, as in HTTP/2. It defines no flags.

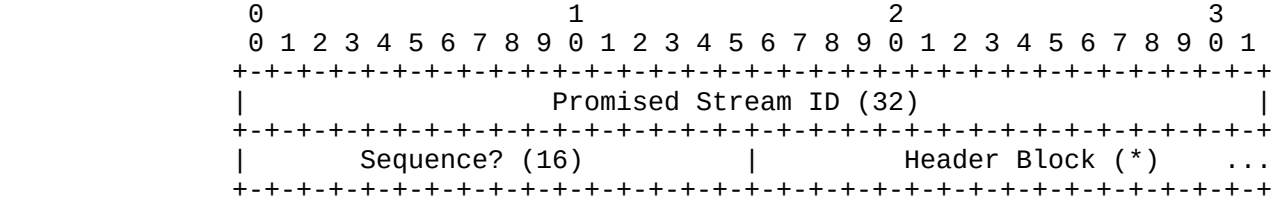

Figure 5: PUSH_PROMISE frame payload

The payload consists of:

Promised Stream ID: A 32-bit Stream ID indicating the QUIC stream on which the response headers will be sent. (The response body stream is implied by the headers stream, as defined in Section 4.)

HPACK Sequence: A sixteen-bit counter, equivalent to the Sequence field in HEADERS

Payload: HPACK-compressed request headers for the promised response.

6. Error Handling

QUIC allows the application to abruptly terminate individual streams or the entire connection when an error is encountered. These are referred to as "stream errors" or "connection errors" and are described in more detail in [QUIC-TRANSPORT].

HTTP/QUIC requires that only data streams be terminated abruptly. Terminating a message control stream will result in an error of type HTTP_RST_CONTROL_STREAM.

This section describes HTTP-specific error codes which can be used to express the cause of a connection or stream error.

6.1. HTTP-Defined QUIC Error Codes

QUIC allocates error codes 0x0000-0x3FFF to application protocol definition. The following error codes are defined by HTTP for use in QUIC RST_STREAM, GOAWAY, and CONNECTION_CLOSE frames.

HTTP_PUSH_REFUSED (0x01): The server has attempted to push content which the client will not accept on this connection.

HTTP_INTERNAL_ERROR (0x02): An internal error has occurred in the HTTP stack.

HTTP_PUSH_ALREADY_IN_CACHE (0x03): The server has attempted to push content which the client has cached.

HTTP_REQUEST_CANCELLED (0x04): The client no longer needs the requested data.

HTTP_HPACK_DECOMPRESSION_FAILED (0x05): HPACK failed to decompress a frame and cannot continue.

HTTP_CONNECT_ERROR (0x06): The connection established in response to a CONNECT request was reset or abnormally closed.

HTTP_EXCESSIVE_LOAD (0x07): The endpoint detected that its peer is exhibiting a behavior that might be generating excessive load.

HTTP_VERSION_FALLBACK (0x08): The requested operation cannot be served over HTTP/QUIC. The peer should retry over HTTP/2.

HTTP_MALFORMED_HEADERS (0x09): A HEADERS frame has been received with an invalid format.

HTTP_MALFORMED_PRIORITY (0x0A): A PRIORITY frame has been received with an invalid format.

HTTP_MALFORMED_SETTINGS (0x0B): A SETTINGS frame has been received with an invalid format.

HTTP_MALFORMED_PUSH_PROMISE (0x0C): A PUSH_PROMISE frame has been received with an invalid format.

HTTP_INTERRUPTED_HEADERS (0x0E): A HEADERS frame without the End Header Block flag was followed by a frame other than HEADERS.

HTTP_SETTINGS_ON_WRONG_STREAM (0x0F): A SETTINGS frame was received on a request control stream.

HTTP_MULTIPLE_SETTINGS (0x10): More than one SETTINGS frame was received.

HTTP_RST_CONTROL_STREAM (0x11): A message control stream closed abruptly.

7. Considerations for Transitioning from HTTP/2

HTTP/QUIC is strongly informed by HTTP/2, and bears many similarities. This section points out important differences from HTTP/2 and describes how to map HTTP/2 extensions into HTTP/QUIC.

7.1. HTTP Frame Types

Many framing concepts from HTTP/2 can be elided away on QUIC, because the transport deals with them. Because frames are already on a stream, they can omit the stream number. Because frames do not block multiplexing (QUIC's multiplexing occurs below this layer), the support for variable-maximum-length packets can be removed. Because stream termination is handled by QUIC, an END_STREAM flag is not required.

Frame payloads are largely drawn from [RFC7540]. However, QUIC includes many features (e.g. flow control) which are also present in HTTP/2. In these cases, the HTTP mapping does not re-implement them. As a result, several HTTP/2 frame types are not required in HTTP/QUIC. Where an HTTP/2-defined frame is no longer used, the frame ID has been reserved in order to maximize portability between HTTP/2 and HTTP/QUIC implementations. However, even equivalent frames between the two mappings are not identical.

Many of the differences arise from the fact that HTTP/2 provides an absolute ordering between frames across all streams, while QUIC provides this guarantee on each stream only. As a result, if a frame type makes assumptions that frames from different streams will still be received in the order sent, HTTP/QUIC will break them.

For example, implicit in the HTTP/2 prioritization scheme is the notion of in-order delivery of priority changes (i.e., dependency tree mutations): since operations on the dependency tree such as reparenting a subtree are not commutative, both sender and receiver must apply them in the same order to ensure that both sides have a consistent view of the stream dependency tree. HTTP/2 specifies priority assignments in PRIORITY frames and (optionally) in HEADERS frames. To achieve in-order delivery of priority changes in HTTP/QUIC, PRIORITY frames are sent on the connection control stream and the PRIORITY section is removed from the HEADERS frame.

Other than this issue, frame type HTTP/2 extensions are typically portable to QUIC simply by replacing Stream 0 in HTTP/2 with Stream 3 in HTTP/QUIC.

Below is a listing of how each HTTP/2 frame type is mapped:

DATA (0x0): Instead of DATA frames, HTTP/QUIC uses a separate data stream. See Section 4.

HEADERS (0x1): As described above, the PRIORITY region of HEADERS is not supported. A separate PRIORITY frame MUST be used. Padding is not defined in HTTP/QUIC frames. See Section 5.2.1.

PRIORITY (0x2): As described above, the PRIORITY frame is sent on the connection control stream. See Section 5.2.2.

RST_STREAM (0x3): RST_STREAM frames do not exist, since QUIC provides stream lifecycle management.

SETTINGS (0x4): SETTINGS frames are sent only at the beginning of the connection. See Section 5.2.3 and Section 7.2.

PUSH_PROMISE (0x5): See Section 5.2.4.

PING (0x6): PING frames do not exist, since QUIC provides equivalent functionality.

GOAWAY (0x7): GOAWAY frames do not exist, since QUIC provides equivalent functionality.

WINDOW_UPDATE (0x8): WINDOW_UPDATE frames do not exist, since QUIC provides flow control.

CONTINUATION (0x9): CONTINUATION frames do not exist; instead, larger HEADERS/PUSH_PROMISE frames than HTTP/2 are permitted, and HEADERS frames can be used in series.

The IANA registry of frame types has been updated in Section 9.3 to include references to the definition for each frame type in HTTP/2 and in HTTP/QUIC. Frames not defined as available in HTTP/QUIC SHOULD NOT be sent and SHOULD be ignored as unknown on receipt.

7.2. HTTP/2 SETTINGS Parameters

An important difference from HTTP/2 is that settings are sent once, at the beginning of the connection, and thereafter cannot change. This eliminates many corner cases around synchronization of changes.

Some transport-level options that HTTP/2 specifies via the SETTINGS frame are superseded by QUIC transport parameters in HTTP/QUIC. The HTTP-level options that are retained in HTTP/QUIC have the same value as in HTTP/2.

Below is a listing of how each HTTP/2 SETTINGS parameter is mapped:

SETTINGS_HEADER_TABLE_SIZE: See Section 5.2.3.2.

SETTINGS_ENABLE_PUSH: See SETTINGS_DISABLE_PUSH in Section 5.2.3.2.

SETTINGS_MAX_CONCURRENT_STREAMS: QUIC requires the maximum number of incoming streams per connection to be specified in the initial transport handshake. Specifying SETTINGS_MAX_CONCURRENT_STREAMS in the SETTINGS frame is an error.

SETTINGS_INITIAL_WINDOW_SIZE: QUIC requires both stream and connection flow control window sizes to be specified in the initial transport handshake. Specifying SETTINGS_INITIAL_WINDOW_SIZE in the SETTINGS frame is an error.

SETTINGS_MAX_FRAME_SIZE: This setting has no equivalent in HTTP/QUIC. Specifying it in the SETTINGS frame is an error.

SETTINGS_MAX_HEADER_LIST_SIZE: See Section 5.2.3.2.

Settings defined by extensions to HTTP/2 MAY be expressed as integers with a maximum value of 2^32-1, if they are applicable to HTTP/QUIC, but SHOULD have a specification describing their usage. Fields for this purpose have been added to the IANA registry in Section 9.4.

7.3. HTTP/2 Error Codes

QUIC has the same concepts of "stream" and "connection" errors that HTTP/2 provides. However, because the error code space is shared between multiple components, there is no direct portability of HTTP/2 error codes.

The HTTP/2 error codes defined in Section 7 of [RFC7540] map to QUIC error codes as follows:

NO_ERROR (0x0): QUIC_NO_ERROR

PROTOCOL_ERROR (0x1): No single mapping. See new HTTP_MALFORMED_* error codes defined in Section 6.1.

INTERNAL_ERROR (0x2) HTTP_INTERNAL_ERROR in Section 6.1.

FLOW_CONTROL_ERROR (0x3): Not applicable, since QUIC handles flow control. Would provoke a QUIC_FLOW_CONTROL_RECEIVED_TOO_MUCH_DATA from the QUIC layer.

SETTINGS_TIMEOUT (0x4): Not applicable, since no acknowledgement of SETTINGS is defined.

STREAM_CLOSED (0x5): Not applicable, since QUIC handles stream management. Would provoke a QUIC_STREAM_DATA_AFTER_TERMINATION from the QUIC layer.

FRAME_SIZE_ERROR (0x6) No single mapping. See new error codes defined in Section 6.1.

REFUSED_STREAM (0x7): Not applicable, since QUIC handles stream management. Would provoke a QUIC_TOO_MANY_OPEN_STREAMS from the QUIC layer.

CANCEL (0x8): HTTP_REQUEST_CANCELLED in Section 6.1.

COMPRESSION_ERROR (0x9): HTTP_HPACK_DECOMPRESSION_FAILED in Section 6.1.

CONNECT_ERROR (0xa): HTTP_CONNECT_ERROR in Section 6.1.

ENHANCE_YOUR_CALM (0xb): HTTP_EXCESSIVE_LOAD in Section 6.1.

INADEQUATE_SECURITY (0xc): Not applicable, since QUIC is assumed to provide sufficient security on all connections.

HTTP_1_1_REQUIRED (0xd): HTTP_VERSION_FALLBACK in Section 6.1.

Error codes defined by HTTP/2 extensions need to be re-registered for HTTP/QUIC if still applicable. See Section 9.5.

8. Security Considerations

The security considerations of HTTP over QUIC should be comparable to those of HTTP/2.

The modified SETTINGS format contains nested length elements, which could pose a security risk to an uncautious implementer. A SETTINGS frame parser MUST ensure that the length of the frame exactly matches the length of the settings it contains.

9. IANA Considerations

9.1. Registration of HTTP/QUIC Identification String

This document creates a new registration for the identification of HTTP/QUIC in the "Application Layer Protocol Negotiation (ALPN) Protocol IDs" registry established in [RFC7301].

The "hq" string identifies HTTP/QUIC:

Protocol: HTTP over QUIC

Identification Sequence: 0x68 0x71 ("hq")

Specification: This document

9.2. Registration of QUIC Version Hint Alt-Svc Parameter

This document creates a new registration for version-negotiation hints in the "Hypertext Transfer Protocol (HTTP) Alt-Svc Parameter" registry established in [RFC7838].

Parameter: "quic"

Specification: This document, Section 2.1

9.3. Existing Frame Types

This document adds two new columns to the "HTTP/2 Frame Type" registry defined in [RFC7540]:

Supported Protocols: Indicates which associated protocols use the frame type. Values MUST be one of:

* "HTTP/2 only"

* "HTTP/QUIC only"

* "Both"

HTTP/QUIC Specification: Indicates where this frame’s behavior over QUIC is defined; required if the frame is supported over QUIC.

Values for existing registrations are assigned by this document:

Frame Type	Supported Protocols	HTTP/QUIC Specification
DATA	HTTP/2 only	N/A
HEADERS	Both	Section 5.2.1
PRIORITY	Both	Section 5.2.2
RST_STREAM	HTTP/2 only	N/A
SETTINGS	Both	Section 5.2.3
PUSH_PROMISE	Both	Section 5.2.4
PING	HTTP/2 only	N/A
GOAWAY	HTTP/2 only	N/A
WINDOW_UPDATE	HTTP/2 only	N/A
CONTINUATION	HTTP/2 only	N/A

The "Specification" column is renamed to "HTTP/2 specification" and is only required if the frame is supported over HTTP/2.

9.4. Settings Parameters

This document adds two new columns to the "HTTP/2 Settings" registry defined in [RFC7540]:

Supported Protocols: Indicates which associated protocols use the setting. Values MUST be one of:

* "HTTP/2 only"

* "HTTP/QUIC only"

* "Both"

HTTP/QUIC Specification: Indicates where this setting’s behavior over QUIC is defined; required if the frame is supported over QUIC.

Values for existing registrations are assigned by this document:

Setting Name	Supported Protocols	HTTP/QUIC Specification
HEADER_TABLE_SIZE	Both	Section 5.2.3.2
ENABLE_PUSH / DISABLE_PUSH	Both	Section 5.2.3.2
MAX_CONCURRENT_STREAMS	HTTP/2 Only	N/A
INITIAL_WINDOW_SIZE	HTTP/2 Only	N/A
MAX_FRAME_SIZE	HTTP/2 Only	N/A
MAX_HEADER_LIST_SIZE	Both	Section 5.2.3.2

The "Specification" column is renamed to "HTTP/2 Specification" and is only required if the setting is supported over HTTP/2.

9.5. Error Codes

This document establishes a registry for HTTP/QUIC error codes. The "HTTP/QUIC Error Code" registry manages a 30-bit space. The "HTTP/QUIC Error Code" registry operates under the "Expert Review" policy [RFC5226].

Registrations for error codes are required to include a description of the error code. An expert reviewer is advised to examine new registrations for possible duplication with existing error codes. Use of existing registrations is to be encouraged, but not mandated.

New registrations are advised to provide the following information:

Name: A name for the error code. Specifying an error code name is optional.

Code: The 30-bit error code value.

Description: A brief description of the error code semantics, longer if no detailed specification is provided.

Specification: An optional reference for a specification that defines the error code.

The entries in the following table are registered by this document.

Name	Code	Description	Specification
HTTP_PUSH_REFUSED	0x01	Client refused pushed content	Section 6.1
HTTP_INTERNAL_ERROR	0x02	Internal error	Section 6.1
HTTP_PUSH_ALREADY_IN_CACHE	0x03	Pushed content already cached	Section 6.1
HTTP_REQUEST_CANCELLED	0x04	Data no longer needed	Section 6.1
HTTP_HPACK_DECOMPRESSION_FAILED	0x05	HPACK cannot continue	Section 6.1
HTTP_CONNECT_ERROR	0x06	TCP reset or error on CONNECT request	Section 6.1
HTTP_EXCESSIVE_LOAD	0x07	Peer generating excessive load	Section 6.1
HTTP_VERSION_FALLBACK	0x08	Retry over HTTP/2	Section 6.1
HTTP_MALFORMED_HEADERS	0x09	Invalid HEADERS frame	Section 6.1
HTTP_MALFORMED_PRIORITY	0x0A	Invalid PRIORITY frame	Section 6.1
HTTP_MALFORMED_SETTINGS	0x0B	Invalid SETTINGS frame	Section 6.1

HTTP_MALFORMED_PUSH_PROMISE	0x0C	Invalid PUSH_PROMISE frame	Section 6.1
HTTP_INTERRUPTED_HEADERS	0x0E	Incomplete HEADERS block	Section 6.1
HTTP_SETTINGS_ON_WRONG_STREAM	0x0F	SETTINGS frame on a request control stream	Section 6.1
HTTP_MULTIPLE_SETTINGS	0x10	Multiple SETTINGS frames	Section 6.1
HTTP_RST_CONTROL_STREAM	0x11	Message control stream was RST	Section 6.1

10. References

10.1. Normative References

[QUIC-TRANSPORT]
Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport".

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

[RFC7541] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <http://www.rfc-editor.org/info/rfc7541>.

[RFC7838] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <http://www.rfc-editor.org/info/rfc7838>.

10.2. Informative References

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <http://www.rfc-editor.org/info/rfc7301>.

Appendix A. Contributors

The original authors of this specification were Robbie Shade and Mike Warres.

Appendix B. Change Log

RFC Editor's Note: Please remove this section prior to publication of a final version of this document.

B.1. Since draft-ietf-quic-http-01:

- SETTINGS changes (#181):
  - SETTINGS can be sent only once at the start of a connection; no changes thereafter
  - SETTINGS_ACK removed

    * Settings can only occur in the SETTINGS frame a single time

    * Boolean format updated

  - Alt-Svc parameter changed from "v" to "quic"; format updated (#229)

  - Closing the connection control stream or any message control stream is a fatal error (#176)

  - HPACK Sequence counter can wrap (#173)

  - 0-RTT guidance added

  - Guide to differences from HTTP/2 and porting HTTP/2 extensions added (#127,#242)

B.2. Since draft-ietf-quic-http-00:

  - Changed "HTTP/2-over-QUIC" to "HTTP/QUIC" throughout (#11,#29)

  - Changed from using HTTP/2 framing within Stream 3 to new framing format and two-stream-per-request model (#71,#72,#73)

  - Adopted SETTINGS format from draft-bishop-httpbis-extended-settings-01

  - Reworked SETTINGS_ACK to account for indeterminate inter-stream order (#75)

  - Described CONNECT pseudo-method (#95)

  - Updated ALPN token and Alt-Svc guidance (#13,#87)

  - Application-layer-defined error codes (#19,#74)

B.3. Since draft-shade-quic-http2-mapping-00:

  - Adopted as base for draft-ietf-quic-http.

  - Updated authors/editors list.

Author’s Address

Mike Bishop (editor)
Microsoft

Email: Michael.Bishop@microsoft.com