Hypertext Transfer Protocol (HTTP) over QUIC
draft-ietf-quic-http-05

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.

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].

   Field definitions are given in Augmented Backus-Naur Form (ABNF), as
   defined in [RFC5234].

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 50781 at the same
hostname by including the following header in any response:

Alt-Svc: hq=":50781"

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.

Servers MAY serve HTTP/QUIC on any UDP port. Servers MUST use the
same port across all IP addresses that serve a single domain, and
SHOULD NOT change this port.

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=":49288";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 5.2.5) MUST be sent as the
initial frame of the HTTP control stream (Stream ID 1, 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 0 for crypto operations (the handshake, crypto
config updates). Stream 1 is reserved for sending and receiving HTTP
control frames, and is analogous to HTTP/2's Stream 0. This control
stream is considered critical to the HTTP connection. If the 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
single stream: This means that the client's first request occurs on
QUIC stream 3, the second on stream 5, and so on. The server's first
push consumes stream 2.

This stream carries frames related to the request/response (see
Section 5.2). When a 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). Streams which terminate
abruptly may be reset at any point in the frame.

Streams SHOULD be used sequentially, with no gaps. Streams used for
pushed resources MAY be initiated out-of-order, but stream IDs SHOULD
be allocated to promised resources sequentially.

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 stream is closed in the appropriate direction.

4.1. Stream 1: Control Stream

Since most connection-level concerns will be managed by QUIC, the
primary use of Stream 1 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 QUIC stream. A server sends
an HTTP response on the same stream as the request.

An HTTP message (request or response) consists of:

1. one header block (see Section 5.2.2) containing the message
    headers (see [RFC7230], Section 3.2),

2. the payload body (see [RFC7230], Section 3.3), sent as a series
    of DATA frames (see Section 5.2.1),

3. optionally, one header block 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 containing
the message headers of informational (1xx) HTTP responses (see
[RFC7230], Section 3.2 and [RFC7231], Section 6.2).

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
following the body. 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
discard any subsequent header blocks.

An HTTP request/response exchange fully consumes a QUIC stream.
After sending a request, a client closes the stream for sending;
after sending a response, the server closes the stream for sending
and the QUIC stream is 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 triggering a QUIC STOP_SENDING with error code
HTTP_EARLY_RESPONSE, sending a complete response, and cleanly closing
its streams. Clients MUST NOT discard complete responses as a result
of having their request terminated abruptly, though clients can
always discard responses at their discretion for other reasons.
Servers MUST NOT abort a response in progress as a result of
receiving a solicited RST_STREAM.

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.
QUIC anticipates moving to a modified version of HPACK without this
assumption. In the meantime, by fixing the size of the dynamic table
at zero, HPACK can be used in an unordered environment.

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.

   All DATA frames on the request stream correspond to data sent on the
   TCP connection. Any DATA frame sent by the client is transmitted by
   the proxy to the TCP server; data received from the TCP server is
   packaged into DATA frames by the proxy. Note that the size and
   number of TCP segments is not guaranteed to map predictably to the
   size and number of HTTP DATA or QUIC STREAM frames.

   The TCP connection can be closed by either peer. When the client
   half-closes the request 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 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. Request Prioritization

   HTTP/QUIC uses the priority scheme described in [RFC7540],
   Section 5.3. In this priority scheme, a given request can be
   designated as dependent upon another request, which expresses the
   preference that the latter stream (the "parent" request) be allocated
   resources before the former stream (the "dependent" request). Taken
   together, the dependencies across all requests 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
   requests.

   HTTP/2 defines its priorities in terms of streams whereas HTTP over
   QUIC identifies requests. The PRIORITY frame Section 5.2.3
   identifies a request either by identifying the stream that carries a
   request or by using a Push ID (Section 5.2.6). Other than the means
   of identifying requests, the prioritization system is identical to
   that in HTTP/2.

   Only a client can send PRIORITY frames. A server MUST NOT send a
   PRIORITY frame.

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_ENABLE_PUSH setting in the
   SETTINGS frame (see Section 3), which is disabled by default.

   As with server push for HTTP/2, the server initiates a server push by
   sending a PUSH_PROMISE frame that includes request header fields
   attributed to the request. The PUSH_PROMISE frame is sent on a
   response stream. Unlike HTTP/2, the PUSH_PROMISE does not reference
   a stream; when a server fulfills a promise, the stream that carries
   the stream headers references the PUSH_PROMISE. This allows a server
   to fulfill promises in the order that best suits its needs.

   The server push response is conveyed on a push stream. A push stream
   is a server-initiated stream. A push stream includes a header (see
   Figure 1) that identifies the PUSH_PROMISE that it fulfills. This
   header consists of a 32-bit Push ID, which identifies a server push
   (see Section 5.2.6).

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                         Push ID (32)                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
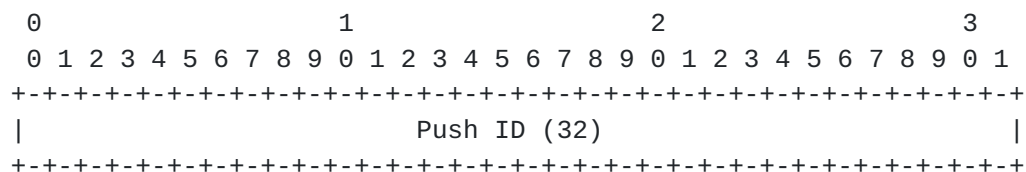
                       Figure 1: Push Stream Header

   Each Push ID MUST only be used once in a push stream header. If a
   push stream header includes a Push ID that was used in another push
   stream header, the client MUST treat this as a connection error of
   type HTTP_DUPLICATE_PUSH. The same Push ID can be used in multiple
   PUSH_PROMISE frames (see Section 5.2.6).

   After the push stream header, a push contains a response
   (Section 4.2), with response headers, a response body (if any)
   carried by DATA frames, then trailers (if any) carried by HEADERS
   frames.

   If a promised server push is not needed by the client, the client
   SHOULD send a CANCEL_PUSH frame; if the push stream is already open,
   a QUIC STOP_SENDING frame with an appropriate error code can be used
   instead (e.g., HTTP_PUSH_REFUSED, HTTP_PUSH_ALREADY_IN_CACHE; see
   Section 6). This asks the server not to transfer the data and
   indicates that it will be discarded upon receipt.

5. HTTP Framing Layer

   Frames are used on each stream. 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:

```
     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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |          Length (16)          |   Type (8)    |   Flags (8)   |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                       Frame Payload (*)                    ...
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
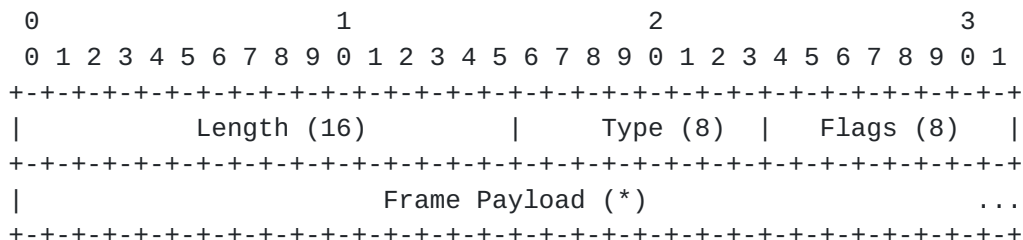
                      Figure 2: HTTP/QUIC frame format

5.2. Frame Definitions

5.2.1. DATA

   DATA frames (type=0x0) convey arbitrary, variable-length sequences of
   octets associated with an HTTP request or response payload.

   The DATA frame defines no flags.

   DATA frames MUST be associated with an HTTP request or response. If
   a DATA frame is received on the control stream, the recipient MUST
   respond with a connection error (Section 6) of type
   HTTP_WRONG_STREAM.

   DATA frames MUST contain a non-zero-length payload. If a DATA frame
   is received with a payload length of zero, the recipient MUST respond
   with a stream error (Section 6) of type HTTP_MALFORMED_DATA.

5.2.2. HEADERS

   The HEADERS frame (type=0x1) is used to carry part of a header set,
   compressed using HPACK Section 4.2.1.

   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.

   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.

5.2.3. PRIORITY

   The PRIORITY (type=0x02) frame specifies the sender-advised priority
   of a stream and is substantially different in format from [RFC7540].
   In order to ensure that prioritization is processed in a consistent
   order, PRIORITY frames MUST be sent on the control stream. A

PRIORITY frame sent on any other stream MUST be treated as a
HTTP_WRONG_STREAM error.

The format has been modified to accommodate not being sent on a
request stream, to allow for identification of server pushes, and the
larger stream ID space of QUIC. The semantics of the Stream
Dependency, Weight, and E flag are otherwise the same as in HTTP/2.

The flags defined are:

PUSH_PRIORITIZED (0x04): Indicates that the Prioritized Stream is a
   server push rather than a request.

PUSH_DEPENDENT (0x02): Indicates a dependency on a server push.

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

```
    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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                 Prioritized Request ID (32)                  |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                  Stream Dependency ID (32)                   |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |   Weight (8)  |
   +-+-+-+-+-+-+-+-+
```
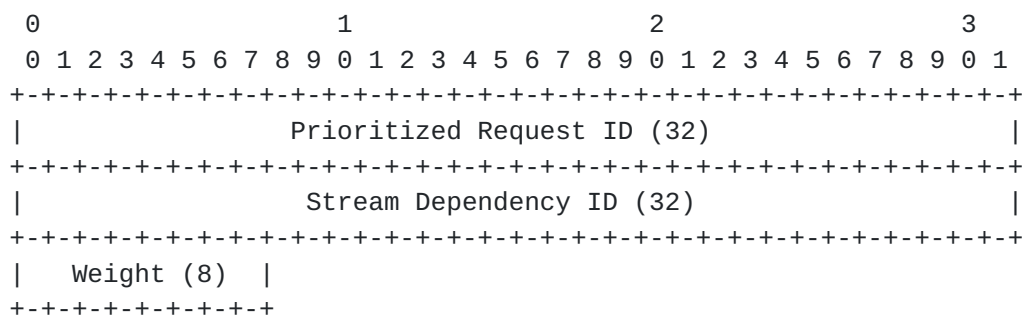
                    Figure 3: PRIORITY frame payload

The PRIORITY frame payload has the following fields:

Prioritized Request ID: A 32-bit identifier for a request. This
   contains the stream ID of a request stream when the
   PUSH_PRIORITIZED flag is clear, or a Push ID when the
   PUSH_PRIORITIZED flag is set.

Stream Dependency ID: A 32-bit stream identifier for a dependent
   request. This contains the stream ID of a request stream when the
   PUSH_DEPENDENT flag is clear, or a Push ID when the PUSH_DEPENDENT
   flag is set. A request stream ID of 0 indicates a dependency on
   the root stream. For details of dependencies, 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 identifies a request to priotize, and a request upon
which that request is dependent. A Prioritized Request ID or Stream
Dependency ID identifies a client-initiated request using the
corresponding stream ID when the corresponding PUSH_PRIORITIZED or
PUSH_DEPENDENT flag is not set. Setting the PUSH_PRIORITIZED or
PUSH_DEPENDENT flag causes the Prioritized Request ID or Stream
Dependency ID (respectively) to identify a server push using a Push
ID (see Section 5.2.6 for details).

A PRIORITY frame MAY identify a Stream Dependency ID using a stream
ID of 0; as in [RFC7540], this makes the request dependent on the
root of the dependency tree.

Stream ID 0 and stream ID 1 cannot be reprioritized. A Prioritized
Request ID that identifies Stream 0 or 1 MUST be treated as a
connection error of type HTTP_MALFORMED_PRIORITY.

A PRIORITY frame that does not reference a request MUST be treated as
a HTTP_MALFORMED_PRIORITY error, unless it references stream ID 0. A
PRIORITY that sets a PUSH_PRIORITIZED or PUSH_DEPENDENT flag, but
then references a non-existent Push ID MUST be treated as a
HTTP_MALFORMED_PRIORITY error.

The length of a PRIORITY frame is 9 octets. A PRIORITY frame with
any other length MUST be treated as a connection error of type
HTTP_MALFORMED_PRIORITY.

5.2.4. CANCEL_PUSH

The CANCEL_PUSH frame (type=0x3) is used to request cancellation of
server push prior to the push stream being created. The CANCEL_PUSH
frame identifies a server push request by Push ID (see
Section 5.2.6).

When a server receives this frame, it aborts sending the response for
the identified server push. If the server has not yet started to
send the server push, it can use the receipt of a CANCEL_PUSH frame
to avoid opening a stream. If the push stream has been opened by the
server, the server SHOULD sent a QUIC RST_STREAM frame on those
streams and cease transmission of the response.

A server can send this frame to indicate that it won't be sending a
response prior to creation of a push stream. Once the push stream
has been created, sending CANCEL_PUSH has no effect on the state of
the push stream. A QUIC RST_STREAM frame SHOULD be used instead to
cancel transmission of the server push response.

A CANCEL_PUSH frame is sent on the control stream. Sending a
CANCEL_PUSH frame on a stream other than the control stream MUST be
treated as a stream error of type HTTP_WRONG_STREAM.

The CANCEL_PUSH frame has no defined flags.

The CANCEL_PUSH frame carries a 32-bit Push ID that identifies the
server push that is being cancelled (see Section 5.2.6).

If the client receives a CANCEL_PUSH frame, that frame might identify
a Push ID that has not yet been mentioned by a PUSH_PROMISE frame.

A server MUST treat a CANCEL_PUSH frame payload that is other than 4
octets in length as a connection error of type
HTTP_MALFORMED_CANCEL_PUSH.

5.2.5. SETTINGS

The SETTINGS frame (type=0x4) conveys configuration parameters that
affect how endpoints communicate, such as preferences and constraints
on peer behavior, and is 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 be willing to consume very large
response headers, while servers are more cautious about request size.

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.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Identifier (16)      |          Length (16)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Contents (?)                       ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
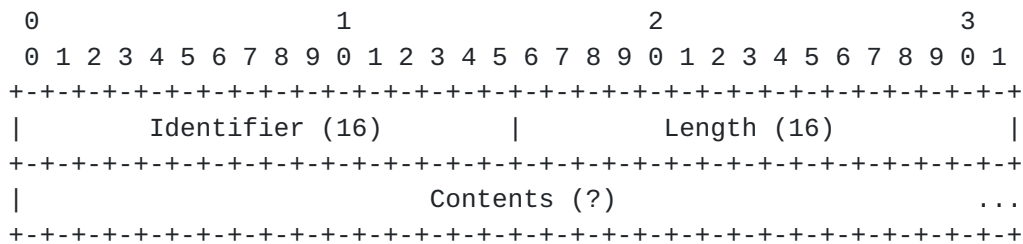
Figure 4: SETTINGS value format

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 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_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 6) of type HTTP_MALFORMED_SETTINGS.

5.2.5.1. 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.

5.2.5.2. Defined SETTINGS Parameters

The following settings are defined in HTTP/QUIC:

SETTINGS_HEADER_TABLE_SIZE (0x1): An integer with a maximum value of
   $2^{32} - 1$. This value MUST be zero.

SETTINGS_ENABLE_PUSH (0x2): Transmitted as a Boolean

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

5.2.5.3. 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:

o SETTINGS_HEADER_TABLE_SIZE

o 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:

o SETTINGS_HEADER_TABLE_SIZE: 0 octets

o 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.

5.2.6. PUSH_PROMISE

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

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Push ID (32)                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Header Block (*)                      ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
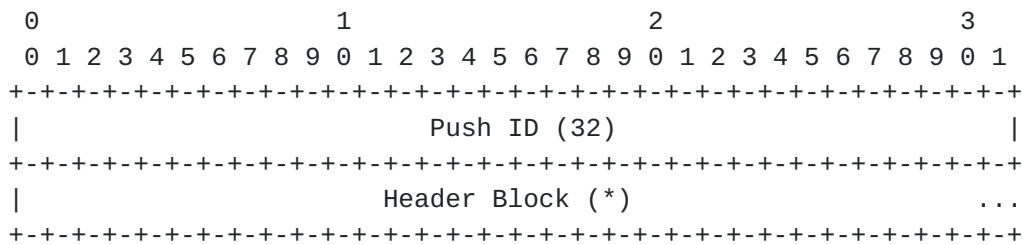
Figure 5: PUSH_PROMISE frame payload

The payload consists of:

Push ID: A 32-bit identifier for the server push request. A push ID
   is used in push stream header (Section 4.4), CANCEL_PUSH frames
   (Section 5.2.4), and PRIORITY frames (Section 5.2.3).

Header Block: HPACK-compressed request headers for the promised
   response.

A server MAY use the same Push ID in multiple PUSH_PROMISE frames.
This allows the server to use the same server push in response to
multiple concurrent requests. Referencing the same server push
ensures that a PUSH_PROMISE can be made in relation to every response
in which server push might be needed without duplicating pushes.

A server that uses the same Push ID in multiple PUSH_PROMISE frames
MUST include the same header fields each time. The octets of the
header block MAY be different due to differing encoding, but the
header fields and their values MUST be identical. Note that ordering
of header fields is significant. A client MUST treat receipt of a
PUSH_PROMISE with conflicting header field values for the same Push
ID as a connection error of type HTTP_MALFORMED_PUSH_PROMISE.

Allowing duplicate references to the same Push ID is primarily to
reduce duplication caused by concurrent requests. A server SHOULD
avoid reusing a Push ID over a long period. Clients are likely to
consume server push responses and not retain them for reuse over
time. Clients that see a PUSH_PROMISE that uses a Push ID that they
have since consumed and discarded are forced to ignore the
PUSH_PROMISE.

5.2.7. GOAWAY

The GOAWAY frame (type=0x7) is used to initiate graceful shutdown of
a connection by a server. GOAWAY allows a server to stop accepting
new requests while still finishing processing of previously received
requests. This enables administrative actions, like server
maintenance. GOAWAY by itself does not close a connection. (Note

that clients do not need to send GOAWAY to gracefully close a
connection; they simply stop making new requests.)

The GOAWAY frame does not define any flags, and the payload is a QUIC
stream identifier. The GOAWAY frame applies to the connection, not a
specific stream. An endpoint MUST treat a GOAWAY frame on a stream
other than the control stream as a connection error (Section 6) of
type HTTP_WRONG_STREAM.

New client requests might already have been sent before the client
receives the server's GOAWAY frame. The GOAWAY frame contains the
stream identifier of the last client-initiated request that was or
might be processed in this connection, which enables client and
server to agree on which requests were accepted prior to the
connection shutdown. This identifier MAY be lower than the stream
limit identified by a QUIC MAX_STREAM_ID frame, and MAY be zero if no
requests were processed. Servers SHOULD NOT increase the
MAX_STREAM_ID limit after sending a GOAWAY frame.

Note: In this context, "processed" means that some data from the
   stream was passed to some higher layer of software that might have
   taken some action as a result.

Once sent, the server will refuse requests sent on streams with an
identifier higher than the included last stream identifier. Clients
MUST NOT send new requests on the connection after receiving GOAWAY,
although requests might already be in transit. A new connection can
be established for new requests.

If the client has sent requests on streams with a higher stream
identifier than indicated in the GOAWAY frame, those requests were
not and will not be processed. Endpoints SHOULD reset any streams
above this ID with the error code HTTP_REQUEST_CANCELLED. Servers
MAY also reset streams below the indicated ID with
HTTP_REQUEST_CANCELLED if the associated requests were not processed.

The client can treat requests cancelled by the server as though they
had never been sent at all, thereby allowing them to be retried later
on a new connection. Automatically retrying other requests is not
possible, unless this is otherwise permitted (e.g. idempotent actions
like GET, PUT, or DELETE). Requests on stream IDs less than or equal
to the stream ID in the GOAWAY frame might have been processed; their
status cannot be known until they are completed successfully, reset,
or the connection terminates.

Servers SHOULD send a GOAWAY frame when the closing of a connection
is known in advance, even if the advance notice is small, so that the
remote peer can know whether a stream has been partially processed or

not. For example, if an HTTP client sends a POST at the same time
that a server closes a QUIC connection, the client cannot know if the
server started to process that POST request if the server does not
send a GOAWAY frame to indicate what streams it might have acted on.

For unexpected closures caused by error conditions, a QUIC
CONNECTION_CLOSE frame MUST be used. However, a GOAWAY MAY be sent
first to provide additional detail to clients. If a connection
terminates without a GOAWAY frame, the last stream identifier is
effectively the highest possible stream identifier (as determined by
QUIC's MAX_STREAM_ID).

An endpoint MAY send multiple GOAWAY frames if circumstances change.
For instance, an endpoint that sends GOAWAY without an error code
during graceful shutdown could subsequently encounter an error
condition. The last stream identifier from the last GOAWAY frame
received indicates which streams could have been acted upon.
Endpoints MUST NOT increase the value they send in the last stream
identifier, since the peers might already have retried unprocessed
requests on another connection.

A client that is unable to retry requests loses all requests that are
in flight when the server closes the connection. A server that is
attempting to gracefully shut down a connection SHOULD send an
initial GOAWAY frame with the last stream identifier set to the
current value of QUIC's MAX_STREAM_ID and SHOULD NOT increase the
MAX_STREAM_ID thereafter. This signals to the client that a shutdown
is imminent and that initiating further requests is prohibited.
After allowing time for any in-flight requests (at least one round-
trip time), the server MAY send another GOAWAY frame with an updated
last stream identifier. This ensures that a connection can be
cleanly shut down without losing requests.

6. Error Handling

QUIC allows the application to abruptly terminate (reset) 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].

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 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_MALFORMED_DATA (0x0D): A DATA 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_WRONG_STREAM (0x0F): A frame was received on stream where it is
    not permitted.

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

HTTP_DUPLICATE_PUSH (0x11): Multiple push streams used the same Push
   ID.

7. Considerations for Transitioning from HTTP/2

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

HTTP/QUIC begins from the premise that HTTP/2 code reuse is a useful
feature, but not a hard requirement. HTTP/QUIC departs from HTTP/2
primarily where necessary to accommodate the differences in behavior
between QUIC and TCP (lack of ordering, support for streams). We
intend to avoid gratuitous changes which make it difficult or
impossible to build extensions with the same semantics applicable to
both protocols at once.

These departures are noted in this section.

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 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 1
in HTTP/QUIC. HTTP/QUIC extensions will not assume ordering, but
would not be harmed by ordering, and would be portable to HTTP/2 in
the same manner.

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

DATA (0x0): Padding is not defined in HTTP/QUIC frames. See
   Section 5.2.1.

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 5.2.2.

PRIORITY (0x2): As described above, the PRIORITY frame is sent on
   the control stream. See Section 5.2.3.

RST_STREAM (0x3): RST_STREAM frames do not exist, since QUIC
   provides stream lifecycle management. The same code point is used
   for the CANCEL_PUSH frame (Section 5.2.4).

SETTINGS (0x4): SETTINGS frames are sent only at the beginning of
   the connection. See Section 5.2.5 and Section 7.2.

PUSH_PROMISE (0x5): The PUSH_PROMISE does not reference a stream;
   instead the push stream references the PUSH_PROMISE frame using a
   Push ID. See Section 5.2.6.

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

GOAWAY (0x7): GOAWAY is sent only from server to client and does not
   contain an error code. See Section 5.2.7.

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.

Frame types defined by extensions to HTTP/2 need to be separately
registered for HTTP/QUIC if still applicable. The IDs of frames
defined in [RFC7540] have been reserved for simplicity. See
Section 9.3.

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 5.2.5.2.

SETTINGS_ENABLE_PUSH: See Section 5.2.5.2.

SETTINGS_MAX_CONCURRENT_STREAMS: QUIC controls the largest open
   stream ID as part of its flow control logic. 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 5.2.5.2.

Settings need to be defined separately for HTTP/2 and HTTP/QUIC. The
IDs of settings defined in [RFC7540] have been reserved for
simplicity. See 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 need to be defined for HTTP/2 and HTTP/QUIC separately.
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. Frame Types

This document establishes a registry for HTTP/QUIC frame type codes.
The "HTTP/QUIC Frame Type" registry manages an 8-bit space. The
"HTTP/QUIC Frame Type" registry operates under either of the "IETF
Review" or "IESG Approval" policies [RFC5226] for values between 0x00
and 0xef, with values between 0xf0 and 0xff being reserved for
Experimental Use.

While this registry is separate from the "HTTP/2 Frame Type" registry
defined in [RFC7540], it is preferable that the assignments parallel
each other. If an entry is present in only one registry, every
effort SHOULD be made to avoid assigning the corresponding value to
an unrelated operation.

New entries in this registry require the following information:

Frame Type: A name or label for the frame type.

Code: The 8-bit code assigned to the frame type.

Specification: A reference to a specification that includes a
   description of the frame layout, its semantics, and flags that the
   frame type uses, including any parts of the frame that are
   conditionally present based on the value of flags.

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

Frame Type	Code	Specification
DATA	0x0	Section 5.2.1
HEADERS	0x1	Section 5.2.2
PRIORITY	0x2	Section 5.2.3
CANCEL_PUSH	0x3	Section 5.2.4
SETTINGS	0x4	Section 5.2.5
PUSH_PROMISE	0x5	Section 5.2.6
Reserved	0x6	N/A
GOAWAY	0x7	Section 5.2.7
Reserved	0x8	N/A
Reserved	0x9	N/A

9.4. Settings Parameters

This document establishes a registry for HTTP/QUIC settings. The
"HTTP/QUIC Settings" registry manages a 16-bit space. The "HTTP/QUIC
Settings" registry operates under the "Expert Review" policy

[RFC5226] for values in the range from 0x0000 to 0xefff, with values
between and 0xf000 and 0xffff being reserved for Experimental Use.
The designated experts are the same as those for the "HTTP/2
Settings" registry defined in [RFC7540].

While this registry is separate from the "HTTP/2 Settings" registry
defined in [RFC7540], it is preferable that the assignments parallel
each other. If an entry is present in only one registry, every
effort SHOULD be made to avoid assigning the corresponding value to
an unrelated operation.

New registrations are advised to provide the following information:

Name: A symbolic name for the setting. Specifying a setting name is
   optional.

Code: The 16-bit code assigned to the setting.

Specification: An optional reference to a specification that
   describes the use of the setting.

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

```
+----------------------+------+----------------+
| Setting Name         | Code | Specification  |
+----------------------+------+----------------+
| HEADER_TABLE_SIZE    | 0x1  | Section 5.2.5.2 |
|                      |      |                |
| ENABLE_PUSH          | 0x2  | Section 5.2.5.2 |
|                      |      |                |
| Reserved             | 0x3  | N/A            |
|                      |      |                |
| Reserved             | 0x4  | N/A            |
|                      |      |                |
| Reserved             | 0x5  | N/A            |
|                      |      |                |
| MAX_HEADER_LIST_SIZE | 0x6  | Section 5.2.5.2 |
+----------------------+------+----------------+
```

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	0x0	Peer	Section 6.1

	7	generating excessive load		
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_MALFORMED_DATA	0x0D	Invalid DATA frame	Section 6.1	
HTTP_INTERRUPTED_HEADERS	0x0E	Incomplete HEADERS block	Section 6.1	
HTTP_WRONG_STREAM	0x0F	A frame was sent on the wrong stream	Section 6.1	
HTTP_MULTIPLE_SETTINGS	0x10	Multiple SETTINGS frames	Section 6.1	
HTTP_DUPLICATE_PUSH	0x11	Duplicate server push	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", draft-ietf-quic-
              transport (work in progress), August 2017.

   [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>.

   [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax
              Specifications: ABNF", STD 68, RFC 5234,
              DOI 10.17487/RFC5234, January 2008,
              <http://www.rfc-editor.org/info/rfc5234>.

   [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", 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-04

   o Cite RFC 5234 (#404)

   o Return to a single stream per request (#245,#557)

   o Use separate frame type and settings registries from HTTP/2 (#81)

   o SETTINGS_ENABLE_PUSH instead of SETTINGS_DISABLE_PUSH (#477)

   o Restored GOAWAY (#696)

   o Identify server push using Push ID rather than a stream ID
      (#702,#281)

   o DATA frames cannot be empty (#700)

B.2. Since draft-ietf-quic-http-03

   None.

B.3. Since draft-ietf-quic-http-02

   o Track changes in transport draft

B.4. Since **draft-ietf-quic-http-01**

   o 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

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

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

   o HPACK Sequence counter can wrap (#173)

   o 0-RTT guidance added

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

B.5. Since **draft-ietf-quic-http-00**

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

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

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

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

   o Described CONNECT pseudo-method (#95)

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

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

B.6. **Since draft-shade-quic-http2-mapping-00**

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

   o Updated authors/editors list

Author's Address

   Mike Bishop (editor)
   Microsoft

   Email: Michael.Bishop@microsoft.com