QUIC Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 19, 2017

M. Bishop
Microsoft

November 15, 2016

HTTP over QUIC - Mapping and Header Compression
draft-bishop-quic-http-and-qpack-00

Abstract

HTTP/2 [RFC7540] uses HPACK [RFC7541] for header compression. However, HPACK relies on the in-order message-based semantics of the HTTP/2 framing layer in order to function. Messages can only be successfully decoded if processed by the receiver in the same order as generated by the sender. This draft refines HPACK to loosen the ordering requirements for use over QUIC [I-D.hamilton-quic-transport-protocol] and describes changes to [I-D.shade-quic-http2-mapping] to leverage the new compression.

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 May 19, 2017.

Table of Contents

1. Introduction

HPACK has a number of features that were intended to provide performance advantages to HTTP/2, but which don't live well in an out-of-order environment such as that provided by QUIC.

The largest challenge is the fact that elements are referenced by a very fluid index. Not only is the index implicit when an item is added to the header table, the index will change without notice as other items are added to the header table. Static entries occupy the first 61 values, followed by dynamic entries. A newly-added dynamic entry would cause older dynamic entries to be evicted, and the retained items are then renumbered beginning with 62. This means that, without processing all preceding header sets, no index into the

dynamic table can be interpreted, and the index of a given entry cannot be predicted.

Any solution to the above will almost certainly fall afoul of the memory constraints the decompressor imposes. The automatic eviction of entries is done based on the compressor's declared dynamic table size, which MUST be less than the maximum permitted by the decompressor (and relayed using an HTTP/2 SETTINGS value).

In the following sections, this document proposes a new version of HPACK which makes different trade-offs, enabling out-of-order interpretation and bounded memory consumption with minimal head-of-line blocking. None of the proposed improvements to HPACK (strongly-typed fields, binary compression of common header syntax) are currently included, but certainly could be.

1.1. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119] and indicate requirement levels for compliant STuPiD implementations.

2. QPACK

2.1. Changes to Static and Dynamic Tables

QPACK uses two tables for associating header fields to indexes. The static table is unchanged from [RFC7541].

The dynamic table is a map from index to header field. Indices are arbitrary numbers greater than the last index of the static table. Each insert instruction will specify the index being modified. While any index MAY be chosen for a new entry, smaller numbers will yield better compression performance. Once an index has been assigned, its value is immutable for the lifetime of that dynamic table.

In order to improve resiliency to reordering, an encoder MAY send multiple insert instructions for the same value to the same index. However, any attempt to insert a different value to an occupied index is a fatal error.

The dynamic table is still constrained to the size specified by the receiver. An attempt to add a header to the dynamic table which causes it to exceed the maximum size MUST be treated as an error by a decoder.

Because it is possible for QPACK frames to arrive which reference indices which have not yet been defined, such frames MUST wait until another frame has arrived and defined the index. In order to guard against malicious senders, implementations SHOULD impose a time limit and treat expiration of the timer as a decoding error. However, if the implementation chooses not to abort the connection, the remainder of the header block MUST be decoded and the output discarded.

2.1.1. Dynamic table state synchronization

No entries are evicted from the dynamic table. Size management is purely the responsibility of the sender, which MUST NOT exceed the declared memory size of the receiver.

Both sender and receiver will maintain a count of references to the indexed entry. This count includes:

- Insertions to the field, both the initial and any redundant indexed literal emissions.
- Literal values which use the indexed entry to provide the header name
- Explicit emissions of the indexed value

The sender MUST consider memory as committed beginning with the first time the indexed entry is assigned. An encoder MAY repeat the insertion instruction in other frames rather than leveraging the index while it waits for the frame to arrive.

When the sender wishes to delete an inserted value, it flows through the following set of states:

1. *Delete requested.* The sender emits a delete instruction including the terminal value of the reference counter. The sender MUST NOT reference the entry in any subsequent frame until this state machine has completed and MUST continue to include the entry in its calculation of consumed memory.
2. *Delete pending.* The receiver receives the delete instruction and compares the sender's counter with its own. If the receiver's counter is less than the sender's, it stores the sender's counter and waits for other QPACK frames to arrive.
3. *Delete acknowledged.* The receiver has received all QPACK frames which reference the deleted value, and can safely delete the entry. The receiver SHOULD promptly emit a QPACK-ACK frame, but MAY delay briefly waiting for other pending deletes as well.

4. *Delete completed.* When the sender receives a QPACK-ACK frame acknowledging the delete, it no longer counts the size of the deleted entry against the table size and MAY emit insert instructions for the field with a new value.

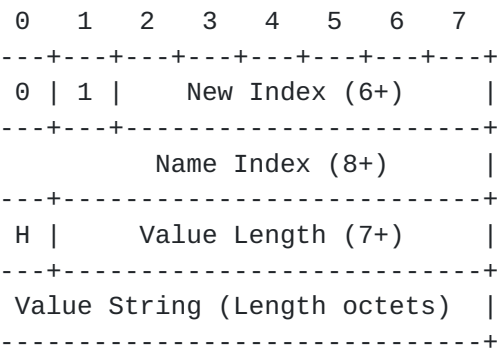
The decoder can receive a delete instruction for a vacant table entry. A decoder MUST NOT consider this to be an error, but MUST handle the delete as usual even while waiting for the definition to arrive.

2.2. Changes to Binary Format

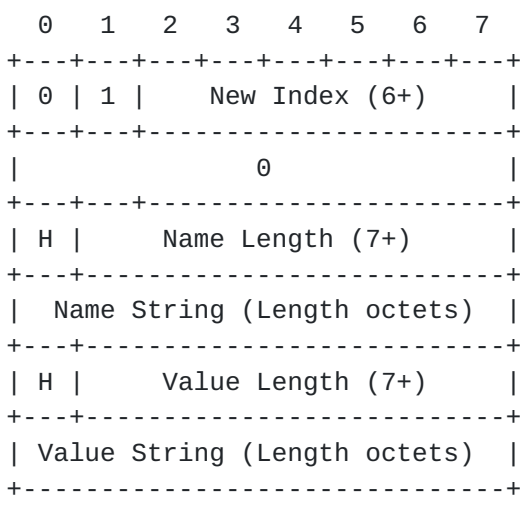
2.2.1. Literal Header Field Representation

(This section replaces [RFC7541], Section 6.2.1.)

A literal header field with indexing representation results in inserting a header field to the decoded header list and inserting it as a new entry into the dynamic table.



Literal Header Field with Indexing -- Indexed Name



Literal Header Field with Indexing -- New Name

A literal header field with incremental indexing representation starts with the '01' 2-bit pattern, followed by the new index of the header represented as an integer with a 6-bit prefix. This value is always greater than the number of entries in the static table.

If the header field name matches the header field name of an entry stored in the static table or the dynamic table, the header field name can be represented using the index of that entry. In this case, the index of the entry is represented as an integer with an 8-bit prefix (see Section 5.1 of [RFC7231]). This value is always non-zero.

Otherwise, the header field name is represented as a string literal (see Section 5.2 of [RFC7231]). A value 0 is used in place of the 8-bit index, followed by the header field name.

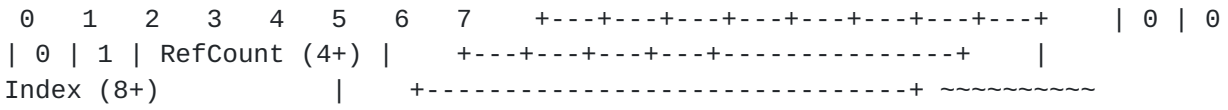
Either form of header field name representation is followed by the header field value represented as a string literal (see Section 5.2).

An encoder MUST NOT attempt to place a value at an index not known to be vacant. An encoder MAY insert the same value to the same vacant slot multiple times in different frames, to reduce the risk of blocking from out-of-order frame interpretation. However, a decoder MUST treat the attempt to insert a different header field into an occupied slot as a fatal error.

2.2.2. Deletion

(This section replaces [RFC7541], Section 6.2.3.)

DISCUSS: I stole the never-indexed instruction code to avoid renumbering _all_ the instructions to fit a new one. If we think we still need this in QUIC, we'll have to do the renumbering later.



Header Field Deletion

The sender may delete an entry in the dynamic header table at any time in order to stay below the receiver's declared memory boundary. This instruction tells the receiver that they should prepare to delete the specified entry after all preceding frames referencing it have been received. The delete instruction includes the count of such frames to facilitate the receiver's garbage collection process.

2.2.3. The QPACK-ACK frame

Each peer MUST periodically emit a QPACK-ACK frame (0xTBD) on QUIC stream 3 to reflect the current state of its header table. A peer MAY omit sending a new QPACK-ACK frame if the dynamic table has not changed since the last frame.

The QPACK-ACK frame defines no flags and consists of a bitmap. The first bit in the bitmap reflects the first index after the static table (currently 62), and each successive bit indicates the next integer value. Each bit MUST be set if the indexed entry has had a delete complete since the preceding QUIC frame and MUST be unset otherwise. Indices beyond the end of the QPACK-ACK frame are assumed to be unset.

Upon receipt, an encoder uses the table to confirm which items have been deleted. At this point, the space can be recovered by the encoder and the encoder can safely reuse the index for future insertions.

3. HTTP over QUIC Mapping

[I-D.shade-quic-http2-mapping] refers to QUIC Stream 3 as carrying "headers," but more accurately, it carries a nearly-complete HTTP/2 session, complete with framing and multiplexing. The mapping deletes certain elements of HTTP/2's framing layer which can be delegated to the QUIC transport layer.

This was done in large part for expediency, reusing HTTP/2 code in place anywhere no QUIC-specific approach had yet been added. A primary driver of this is the need for in-order reliable delivery of frames carrying HPACK data (HEADERS, CONTINUATION, PUSH_PROMISE).

While the ability to reuse HTTP/2 framing is useful, the double-mux layer is unwieldy and has proved unpopular in the working group. This section presents an alternate mapping preserving some HTTP/2 code, but delegating all multiplexing to the QUIC layer.

QPACK would permit header data to be on-stream with the request/response bodies, but some framing is still required. It would be possible (and perhaps desirable) to introduce a simplified version of HTTP/2's framing on each QUIC stream.

3.1. Stream usage

In both QUIC and HTTP/2, odd-numbered streams are client-initiated, while even-numbered streams are server-initiated. A single HTTP transaction spans two streams, differentiated by the next stream bit. 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. This amounts to the second least-significant bit differentiating the two streams in a request.

The payload of each frame type is unmodified from HTTP/2 unless otherwise noted. Frames which would be sent on stream zero in HTTP/2 are sent on QUIC stream 3.

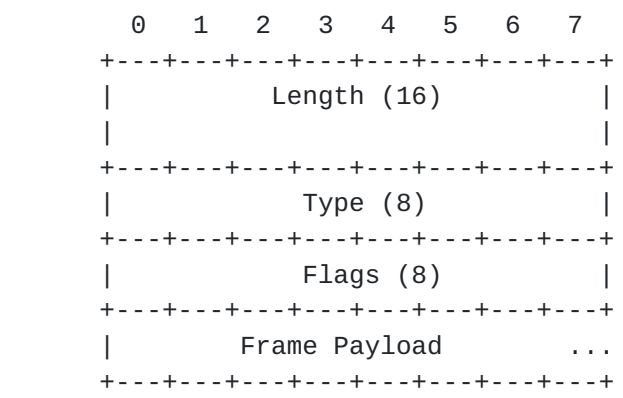
Because stream creation does not depend on particular frames, the requirement that a stream begin only with HEADERS is omitted.

The second stream is used to carry any message payload, eliminating the DATA frame. The first stream is the request control stream and is used to carry all other frames which would have been on-stream in HTTP/2.

3.2. On-Stream Framing Definition

Many framing concepts from HTTP/2 can be elided away on QUIC, because the transport deals with them. Because these frames would already be on a stream, they can omit the stream number. Because the 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.

On QUIC streams other than Stream 1, the general frame format is as follows:



HTTP/QUIC frame format

The fields are defined as in [RFC7540]. The frames currently defined are described in this section:

3.2.1. DATA

DATA frames (type=0x0) do not exist.

3.2.2. HEADERS

The HEADERS frame (type=0x1) is used to carry part of a header set, compressed using QPACK ({QPACK}). The PRIORITY-equivalent regions have been removed, since a stream MAY begin with a PRIORITY frame and the size of the QUIC stream format requires changes to how these fields are handled.

Padding MUST NOT be used. The flags defined are:

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

HEADERS frames from various streams may be processed by the QPACK decoder in any order, completely or partially. It is not necessary to withhold decoding results until the end of the header block has arrived. However, depending on the contents, the processing of one

frame might depend on other QPACK frames. The results of decoding MUST be emitted in the same order as the HEADERS frames were placed on the stream.

3.2.3. PUSH_PROMISE

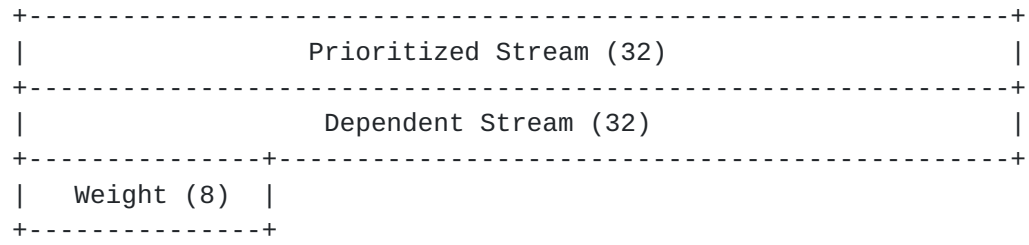
The PUSH_PROMISE frame (type=0x02) is used to carry a request header set from server to client, as in HTTP/2. It contains the same flags as HEADERS.

The payload contains a QPACK headers block encoding the request whose response is promised, preceded by 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 3.1.)

TODO: QUIC stream space may be enlarged; would need to redefine Promised Stream field in this case.

3.2.4. PRIORITY

The PRIORITY frame (type=0x2) specifies the sender-advised priority of a stream and is sent on Stream 3. It can refer to a stream in any state, including idle or closed streams.



PRIORITY Frame Payload

The payload of a PRIORITY frame contains the following fields:

Prioritized Stream: The 32-bit stream identifier for the stream whose priority is being modified.

Stream Dependency:
:The 32-bit stream identifier for the stream that this stream depends on.

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

The PRIORITY frame defines one flag:

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

If a PRIORITY frame is received which attempts to modify a stream which is not a request control scheme, the recipient MUST respond with a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

The PRIORITY frame can affect a stream in any state. Note that this frame could arrive after processing or frame sending has completed, which would cause it to have no effect on the identified stream. For a stream that is in the "half-closed (remote)" or "closed" state, this frame can only affect processing of the identified stream and its dependent streams; it does not affect frame transmission on that stream.

The PRIORITY frame can create a dependency on a stream in the "idle" or "closed" state. This allows for the reprioritization of a group of dependent streams by altering the priority of an unused or closed parent stream.

A PRIORITY frame with a length other than 9 octets MUST be treated as a connection error of type FRAME_SIZE_ERROR.

3.2.5. SETTINGS

The EXTENDED_SETTINGS frame as defined in [I-D.bishop-httpbis-extended-settings] will be renamed SETTINGS and will replace the HTTP/2 SETTINGS frame.

TODO: SETTINGS_ACK and stream state. Do we need to emit the ACK on every active stream? What about idle streams and in-flight data?

3.2.6. Other frames not mentioned

QUIC stream 3 is equivalent to HTTP/2's stream 0, and the same framing is used as for other streams. SETTINGS frames remain on stream 3, as do any other HTTP/2 stream-zero frames. This enables HTTP/2 extension frames which do not have a hard cross-stream ordering requirement to continue to function.

3.3. HTTP Message Exchanges

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

An HTTP message (request or response) consists of:

1. for a response only, zero or more header blocks (a sequence of HEADERS frames with End Header Block set on the last) on the control stream containing the message headers of informational (1xx) HTTP responses (see [RFC7230], Section 3.2 and [RFC7231], Section 6.2),

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

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

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

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 a header block following the body. Such a header block is a sequence of HEADERS frames with End Header Block set on the last frame. Header blocks after the first but before the end of the stream are invalid. These MUST be decoded to maintain QPACK 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. Performance Considerations

While QPACK is designed to minimize head-of-line blocking between streams on header decoding, there are some situations in which lost or delayed packets can still impact the performance of header compression.

References to indexed entries will block if the frame containing the entry definition is lost or delayed. Encoders MAY choose to trade off compression efficiency and avoid blocking by repeating the literal-with-indexing instruction rather than referencing the dynamic table until the insertion is known to be complete.

Delayed frames which prevent deletes from completing can prevent the encoder from adding any new entries due to the maximum table size. This does not block the encoder from continuing to make requests, but could sharply limit compression performance. Encoders would be well-served to delete entries in advance of encountering the table maximum. Decoders SHOULD be prompt about emitting QPACK-ACK frames to enable the sender to recover the table space.

5. Security Considerations

The security considerations for QPACK are believed to be the same as for HPACK.

6. IANA Considerations

This document currently makes no request of IANA, but probably should.

7. Acknowledgements

This draft draws heavily on the text of [RFC7540] and [RFC7541], as well as the ideas of [I-D.shade-quic-http2-mapping]. The indirect input of those authors is gratefully acknowledged, as well as ideas gleefully stolen from:

- Jana Iyengar
- Patrick McManus
- Martin Thomson
- Charles 'Buck' Krasic
- Kyle Rose

8. References

8.1. Normative References

[I-D.bishop-httpbis-extended-settings]
Bishop, M., "HTTP/2 Extended SETTINGS Extension", draft-bishop-httpbis-extended-settings-00 (work in progress), June 2016.

[I-D.hamilton-quic-transport-protocol]
Hamilton, R., Iyengar, J., Swett, I., and A. Wilk, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-hamilton-quic-transport-protocol-01 (work in progress), October 2016.

[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>.

8.2. Informative References

[I-D.shade-quic-http2-mapping]
Shade, R. and M. Warres, "HTTP/2 Semantics Using The QUIC Transport Protocol", draft-shade-quic-http2-mapping-00 (work in progress), July 2016.

Author's Address

Mike Bishop
Microsoft

Email: michael.bishop@microsoft.com