QUIC Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 29, 2018

M. Bishop
Microsoft

September 25, 2017

Header Compression for HTTP/QUIC
draft-bishop-quic-http-and-qpack-04

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 decoder in the same order as generated by the encoder. This draft refines HPACK to loosen the ordering requirements for use over QUIC [I-D.ietf-quic-transport].

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 March 29, 2018.

Copyright Notice

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).

Further, streams in QUIC are lossy in the presence of stream resets. While HTTP/2 (via TCP) guarantees the delivery of all previously-sent data on a stream even if that stream is reset, QUIC does not retransmit lost frames if a stream has been reset, and may discard data which has not yet been delivered to the application.

Previous versions of QPACK were small deltas of HPACK to introduce order-resiliency. This version departs from HPACK more substantially to add resilience against reset message streams.

In the following sections, this document proposes a new version of HPACK which makes different trade-offs, enabling partial 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, [RFC2119] and indicate requirement levels for compliant implementations.

2. QPACK

2.1. Basic model

HPACK combines header table modification and message header emission in a single sequence of coded bytes. QPACK bifurcates these into two channels:

- Connection-wide sets of table update instructions sent on non-request streams
- Non-modifying instructions which use the current header table state to encode message headers on request streams

Because the per-message instructions introduce no changes to the header table state, no state is lost if these instructions are discarded due to a stream reset. Because the updates to the header table supply their own order controls (the delete logic), they can be

processed in any order and therefore delivered as messages using unidirectional QUIC streams.

2.2. 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 and less than 2^27. 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.

The dynamic table is still constrained to the size specified by the decoder. 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. To enable encoders to reclaim space, encoders can delete entries in the dynamic table, but can only reuse the index or the space after receiving confirmation of a successful deletion.

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 peers, implementations SHOULD impose a time limit and treat expiration of the timer as a decoding error.

2.2.1. Changes to Header Table Size

HTTP/QUIC prohibits mid-stream changes of settings. As a result, only one table size change is possible: From the value a client assumes during the 0-RTT flight to the actual value included in the server's SETTINGS frame. The assumed value is required to be either a server's previous value or zero. A server whose configuration has recently changed MAY overlook inadvertent violations of its maximum table size during the first round-trip.

In the case that the value has increased, either from zero to a non-zero value or from the cached value to a higher value, no action is required by the client. The encoder can simply begin using the additional space. In the case that the value has decreased, the encoder MUST immediately emit delete instructions which, upon completion, would bring the table within the required size.

Regardless of changes to header table size, the encoder MUST NOT add entries to the table which would result in a size greater than the maximum permitted. This can imply that no additions are permitted while waiting for these delete instructions to complete.

2.2.2. Dynamic Table State Synchronization

In order to ensure table consistency, all modifications of the header table occur as separate messages rather than on request streams. Request streams contain only indexed and literal header entries.

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

The encoder SHOULD track the following information about each entry in the table:

- The list of recently-active streams which reference the entry in a trailer block, if any

- The list of recently-active streams which reference the entry in a non-trailer block, if any

"Recently-active" streams are those which are still open or were closed less than a reasonable number of RTTs ago. An implementation MAY vary its definition of "recent" to trade off memory consumption and timely completion of deletes, and tracking no information is a functional (though potentially less performant) choice in this space.

The encoder MUST consider memory as committed beginning when the indexed entry is assigned.

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

1. *Delete requested.* The encoder emits a delete instruction indicating which streams might have referenced the entry. The encoder 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 decoder receives the delete instruction and checks the current state of its incoming streams (see Section 2.3.2.2). If more references might arrive, it stores the streams still needed and waits for them to complete.

3. *Delete acknowledged.* The decoder has received all QPACK frames which reference the deleted value, and can safely delete the entry. The decoder SHOULD promptly emit a Delete-Ack instruction on a header management stream.

4. *Delete completed.* When the encoder receives a Delete-Ack instruction 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.

2.3. Header Management Streams

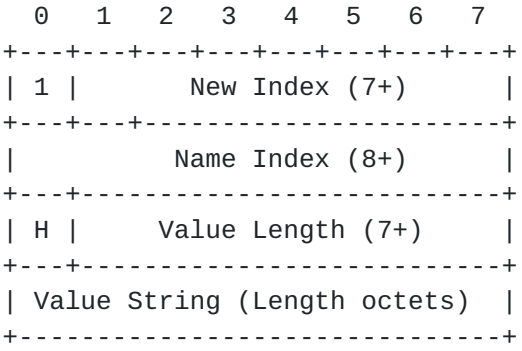
Header management streams are unidirectional streams in either direction which contain a series of QPACK instructions with no message boundaries. Data on these streams SHOULD be processed as soon as it arrives.

This section describes the instructions which are possible on header management streams.

2.3.1. Insert

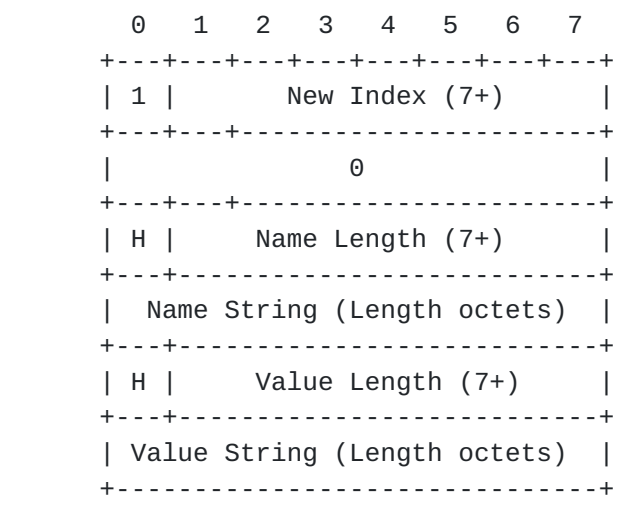
An addition to the header table starts with the '1' one-bit pattern, followed by the new index of the header represented as an integer with a 7-bit prefix. This value is always greater than the number of entries in the static table.

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 [RFC7541]). This value is always non-zero.



Insert Header Field -- Indexed Name

Otherwise, the header field name is represented as a string literal (see Section 5.2 of [RFC7541]). A value 0 is used in place of the 8-bit index, followed by the header field name.



Insert Header Field -- New Name

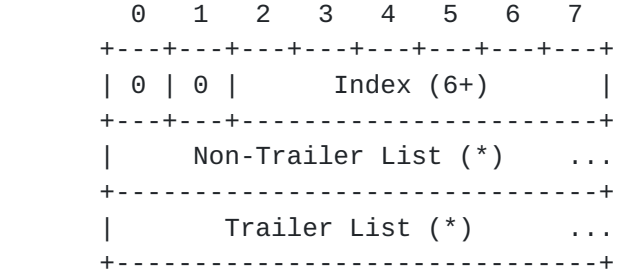
Either form of header field name representation is followed by the header field value represented as a string literal (see Section 5.2 of [RFC7541]).

An encoder MUST NOT attempt to place a value at an index not known to be vacant. A decoder MUST treat the attempt to insert into an occupied slot as a fatal error.

2.3.2. Delete

A deletion from the header table starts with the '00' two bit pattern, followed by the index of the affected entry represented as an integer with a 6-bit prefix. This value is always greater than the number of entries in the static table.

A delete instruction then encodes a series of stream IDs which might have contained references to the entry in question.



Delete Instruction

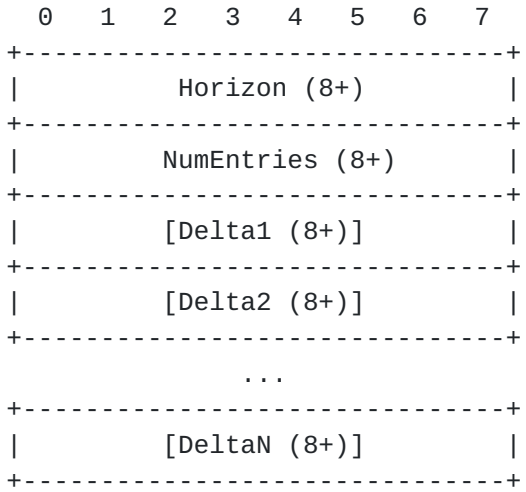
Both the Non-Trailer List and Trailer List are Stream ID Lists (see below) encoding a list of streams which might have referenced the entry either in non-trailer or trailer blocks.

2.3.2.1. Stream ID List

A Stream ID List encodes a sequence of stream IDs in two parts: First, a Horizon value indicates the first non-occurrence about which data is maintained. If data is maintained from the beginning of the connection, the Horizon is zero. This allows senders to succinctly express both old state which has been discarded and large regions where many or all streams contain references.

Following the horizon, a sequence of deltas indicates all streams since the Horizon on which a value has been used.

This structure permits either side to adjust the amount of tracking complexity it is willing to devote to ensure timely deletions. In the simplest case, a Stream ID List might be a horizon value followed by one zero byte. This indicates an absolute cut-off after which the entry is guaranteed not to be referenced, and requires the receiver to wait until all prior requests have been completed. Similarly, the receiver can create equivalent-but-less-complex forms of a Stream ID list by increasing the Horizon value and discarding all explicit stream entries less than the new value.



Stream ID List

The field are as follows:

Horizon: The ID of the first stream for which the sender retains state which does not reference the deleted entry in the indicated block

NumEntries: The number of streams greater than the Horizon which might reference the entry and are listed in the remainder of the instruction

Delta1..N: A sequence of streams greater than the Horizon which might reference the entry, encoded as the difference in stream number from the previously-listed stream. This field is repeated NumEntries times.

2.3.2.2. Delete Validation

In order to safely delete an entry, a decoder MUST ensure that all outstanding references have arrived and been processed. Because no data is available about stream IDs less than the Horizon, a decoder MUST assume that any earlier stream ID might have contained a reference to the value in question.

A decoder can ensure all outstanding references have been processed by verifying that the following statements are true:

- In the Non-Trailer Block, all streams less than the Horizon and all streams explicitly listed are in one of two states:
  - closed
  - headers completely processed
- In the Trailer Block, all streams less than the Horizon and all streams explicitly listed are in one of three states:
  - closed
  - headers completely processed AND no trailers are expected
  - trailers completely processed

An implementation MAY omit the "trailers completely processed" case, since the stream is expected to close immediately after receipt of the trailers block.

If these conditions are not met upon receipt of a Delete instruction, a decoder MUST wait to emit a Delete-Ack instruction until the outstanding streams have reached an appropriate state.

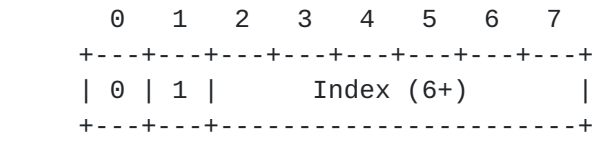
Note that a decoder MAY condense the list of specified streams by increasing the Horizon value and discarding those explicitly-listed stream IDs which are less than the new Horizon it has chosen. This delays delete completion, but reduces the amount of state to be

tracked by the decoder without changing the correctness of the requirements above.

2.3.3. Delete-Ack

Confirmation that a delete has completed is expressed by an instruction which starts with the '01' two-bit pattern, followed by the index of the affected entry represented as an integer with a 6-bit prefix. This value is always greater than the number of entries in the static table.

Note that unlike all other instructions, this instruction refers to the receiver's dynamic table, not the sender's.



Delete-Ack Instruction

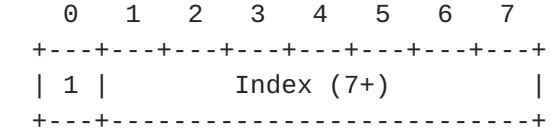
This instruction MUST NOT be sent before the conditions described in Section 2.3.2.2 have been satisfied, and SHOULD be sent as soon as possible once they are.

2.4. Format of Encoded Headers on Message Streams

Frames which carry HTTP message headers encode them using the following instructions:

2.4.1. Indexed Header Field Representation

An indexed header field representation identifies an entry in either the static table or the dynamic table and causes that header field to be added to the decoded header list, as described in Section 3.2 of [RFC7541].



Indexed Header Field

An indexed header field starts with the '1' 1-bit pattern, followed by the index of the matching header field, represented as an integer with a 7-bit prefix (see Section 5.1 of [RFC7541]).

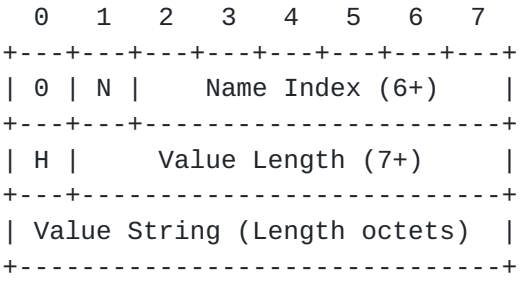
The index value of 0 is not used. It MUST be treated as a decoding error if found in an indexed header field representation.

2.4.2. Literal Header Field Representation

A literal header field representation starts with the '0' 1-bit pattern and causes a header field to be added the decoded header list.

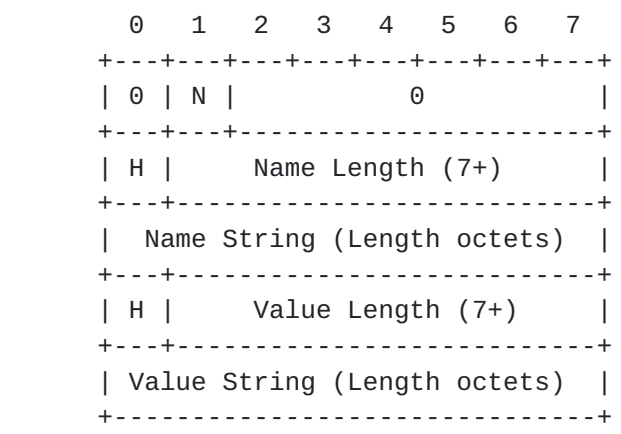
The second bit, 'N', indicates whether an intermediary is permitted to add this header to the dynamic header table on subsequent hops. When the 'N' bit is set, the encoded header MUST always be encoded with this specific literal representation. In particular, when a peer sends a header field that it received represented as a literal header field with the 'N' bit set, it MUST use the same representation to forward this header field. This bit is intended for protecting header field values that are not to be put at risk by compressing them (see Section 7.1 of [RFC7541] for more details).

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 a 6-bit prefix (see Section 5.1 of [RFC7541]). This value is always non-zero.



Literal Header Field -- Indexed Name

Otherwise, the header field name is represented as a string literal (see Section 5.2 of [RFC7541]). A value 0 is used in place of the 6-bit index, followed by the header field name.



Literal Header Field -- Literal Name

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

3. Use in HTTP/QUIC

HTTP/QUIC [I-D.ietf-quic-http] currently retains the HPACK encoder/decoder from HTTP/2, using a Sequence number to enforce ordering. Using QPACK instead would entail the following changes:

- The Sequence field is removed from HEADERS frames (Section 5.2.2) and PUSH_PROMISE frames (Section 5.2.6).
- Header Block Fragments consist of QPACK data instead of HPACK data.
- Just as unidirectional push streams have a stream header identifying their type and Push ID, a header will need to be added to differentiate header table update streams from requests and pushes.

A HEADERS or PUSH_PROMISE frame MAY contain an arbitrary number of QPACK instructions, but QPACK instructions SHOULD NOT cross a boundary between successive HEADERS frames. A partial HEADERS or PUSH_PROMISE frame MAY be processed upon arrival and the resulting partial header set emitted or buffered according to implementation requirements.

4. Implementation trade-offs

This document specifies a means for the encoder to express the choices it made while encoding, but intentionally does not mandate what those choices should be. In this section, potential areas for implementation tuning are explored.

4.1. Compression Efficiency versus Blocking Avoidance

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 using literal instructions rather than referencing the dynamic table until the insertion is believed to be complete.

The most efficient compression algorithm will reference a table entry whenever it exists in the table, but risks blocking when subject to packet loss or reordering. The most conservative algorithm will always emit literals to guarantee that no blocking will ever occur. Most implementations will choose a balance between these two extremes.

Better efficiency while being similarly conservative can be achieved by permitting references to table entries only once these entries are confirmed to be present in the table. More optimization can be achieved when the reference is known to be in the same packet as the definition.

Increases in efficiency can be achieved by assuming greater risk of blocking - implementations might choose a particular balance, or adjust their aggressiveness based on observed network characteristics.

Since it is possible to insert header values without emitting them on a stream, an encoder MAY also proactively insert header values which it believes will be needed on future requests, at the cost of reduced compression efficiency for incorrect predictions.

The ability to split updates to the header table into discrete messages reduces the possibility for head-of-line blocking within the table update streams. Implementations SHOULD limit the size of table update messages to avoid head-of-line blocking within these messages.

4.2. Timely Delete Completion versus State Commitment

Anything 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 Delete-Ack instructions to enable the encoder to recover the table space.

The encoder can enable deletes to complete more quickly by maintaining a complete history of which streams have referenced any

given table entry and providing this list as part of the delete instruction. The encoder can also choose to maintain less state by advancing the Horizon value (see Section 2.3.2.1). This value indicates the starting point of the provided history, and can be advanced arbitrarily far to discard history. This comes at the potential cost of a decoder taking longer to acknowledge that entries have been removed, but this cost is zero if all previous requests are known to have completed. Therefore, this history can be pruned without performance impact by removing entries where all data is known to have been successfully delivered and interpreted, if some transport coordination is employed.

An encoder which chooses to maintain no history would simply supply a Horizon value of a stream which has not yet been used, meaning that deletes cannot complete until all currently-active requests have completed.

A decoder can perform the same trade-off in the event the encoder's supplied history is more state than it wishes to track.

5. Security Considerations

A malicious encoder might attempt to consume a large amount of space on the decoder by opening the maximum number of streams, adding entries to the table, then sending delete instructions enumerating many streams in a Stream ID List.

To guard against such attacks, a decoder SHOULD bound its state tracking by generalizing the list of streams to be tracked. This is most easily achieved by advancing the Horizon to a later value and discarding explicit Stream IDs to track, but can also be accomplished by eliding explicit streams in ranges. This does not cause any loss of consistency for deletes, but could delay completion and reduce performance if done aggressively.

6. IANA Considerations

This document currently makes no request of IANA, and might not need to.

7. Acknowledgements

This draft draws heavily on the text of [RFC7541]. 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. Normative References

[I-D.ietf-quic-http]
Bishop, M., "Hypertext Transfer Protocol (HTTP) over QUIC", draft-ietf-quic-http-06 (work in progress), September 2017.

[I-D.ietf-quic-transport]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-06 (work in progress), September 2017.

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

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://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, <https://www.rfc-editor.org/info/rfc7541>.

Author's Address

Mike Bishop
Microsoft

Email: michael.bishop@microsoft.com