QUIC
Internet-Draft
Intended status: Standards Track
Expires: October 25, 2019

C. Krasic
Netflix
M. Bishop
Akamai Technologies
A. Frindell, Ed.
Facebook
April 23, 2019

QPACK: Header Compression for HTTP/3
draft-ietf-quic-qpack-08

Abstract

This specification defines QPACK, a compression format for efficiently representing HTTP header fields, to be used in HTTP/3. This is a variation of HPACK header compression that seeks to reduce head-of-line blocking.

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 [1].

Working Group information can be found at https://github.com/quicwg [2]; source code and issues list for this draft can be found at https://github.com/quicwg/base-drafts/labels/-qpack [3].

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 https://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 October 25, 2019.

Table of Contents

1. Introduction

The QUIC transport protocol was designed from the outset to support HTTP semantics, and its design subsumes many of the features of HTTP/2. HTTP/2 uses HPACK ([RFC7541]) for header compression, but QUIC's stream multiplexing comes into some conflict with HPACK. A key goal of the design of QUIC is to improve stream multiplexing relative to HTTP/2 by reducing head-of-line blocking. If HPACK were used for HTTP/3, it would induce head-of-line blocking due to built-in assumptions of a total ordering across frames on all streams.

QUIC is described in [QUIC-TRANSPORT]. The HTTP/3 mapping is described in [HTTP3]. For a full description of HTTP/2, see [RFC7540]. The description of HPACK is [RFC7541].

QPACK reuses core concepts from HPACK, but is redesigned to allow correctness in the presence of out-of-order delivery, with flexibility for implementations to balance between resilience against head-of-line blocking and optimal compression ratio. The design goals are to closely approach the compression ratio of HPACK with substantially less head-of-line blocking under the same loss conditions.

1.1. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Definitions of terms that are used in this document:

Header field: A name-value pair sent as part of an HTTP message.

Header list: An ordered collection of header fields associated with an HTTP message. A header list can contain multiple header fields with the same name. It can also contain duplicate header fields.

Header block: The compressed representation of a header list.

Encoder: An implementation which transforms a header list into a header block.

Decoder: An implementation which transforms a header block into a header list.

Absolute Index: A unique index for each entry in the dynamic table.

Base: A reference point for relative indicies. Dynamic references are made relative to a Base in header blocks.

Insert Count: The total number of entries inserted in the dynamic table.

QPACK is a name, not an acronym.

1.2. Notational Conventions

Diagrams use the format described in Section 3.1 of [RFC2360], with the following additional conventions:

x (A) Indicates that x is A bits long

x (A+) Indicates that x uses the prefixed integer encoding defined in Section 5.1 of [RFC7541], beginning with an A-bit prefix.

x ... Indicates that x is variable-length and extends to the end of the region.

2. Compression Process Overview

Like HPACK, QPACK uses two tables for associating header fields to indices. The static table (see Section 3.1) is predefined and contains common header fields (some of them with an empty value). The dynamic table (see Section 3.2) is built up over the course of the connection and can be used by the encoder to index header fields in the encoded header lists.

QPACK instructions appear in three different types of streams:

- The encoder uses a unidirectional stream to modify the state of the dynamic table without emitting header fields associated with any particular request.
- HEADERS and PUSH_PROMISE frames on request and push streams reference the table state without modifying it.
- The decoder sends feedback to the encoder on a unidirectional stream. This feedback enables the encoder to manage dynamic table state.

2.1. Encoder

An encoder compresses a header list by emitting either an indexed or a literal representation for each header field in the list. References to the static table and literal representations do not require any dynamic state and never risk head-of-line blocking. References to the dynamic table risk head-of-line blocking if the encoder has not received an acknowledgement indicating the entry is available at the decoder.

An encoder MAY insert any entry in the dynamic table it chooses; it is not limited to header fields it is compressing.

QPACK preserves the ordering of header fields within each header list. An encoder MUST emit header field representations in the order they appear in the input header list.

QPACK is designed to contain the more complex state tracking to the encoder, while the decoder is relatively simple.

2.1.1. Reference Tracking

An encoder MUST ensure that a header block which references a dynamic table entry is not received by the decoder after the referenced entry has been evicted. Hence the encoder needs to track information about each compressed header block that references the dynamic table until that header block is acknowledged by the decoder.

2.1.2. Blocked Dynamic Table Insertions

A dynamic table entry is considered blocking and cannot be evicted until its insertion has been acknowledged and there are no outstanding unacknowledged references to the entry. In particular, a dynamic table entry that has never been referenced can still be blocking.

Note: A blocking entry is unrelated to a blocked stream, which is a stream that a decoder cannot decode as a result of references to entries that are not yet available. Any encoder that uses the dynamic table has to keep track of blocked entries, whereas blocked streams are optional.

An encoder MUST NOT insert an entry into the dynamic table (or duplicate an existing entry) if doing so would evict a blocking entry. In this case, the encoder can send literal representations of header fields.

To ensure that the encoder is not prevented from adding new entries, the encoder can avoid referencing entries that are close to eviction. Rather than reference such an entry, the encoder can emit a Duplicate instruction (see Section 4.3.3), and reference the duplicate instead.

Determining which entries are too close to eviction to reference is an encoder preference. One heuristic is to target a fixed amount of available space in the dynamic table: either unused space or space that can be reclaimed by evicting non-blocking entries. To achieve this, the encoder can maintain a draining index, which is the smallest absolute index in the dynamic table that it will emit a reference for. As new entries are inserted, the encoder increases the draining index to maintain the section of the table that it will not reference. If the encoder does not create new references to

entries with an absolute index lower than the draining index, the number of unacknowledged references to those entries will eventually become zero, allowing them to be evicted.

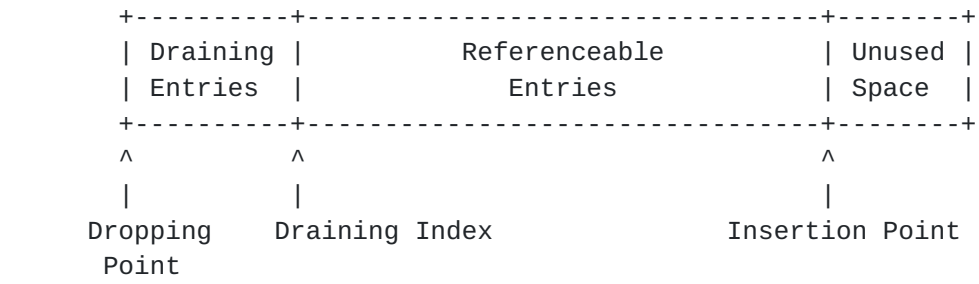


Figure 1: Draining Dynamic Table Entries

2.1.3. Avoiding Head-of-Line Blocking

Because QUIC does not guarantee order between data on different streams, a header block might reference an entry in the dynamic table that has not yet been received.

Each header block contains a Required Insert Count, the lowest possible value for the Insert Count with which the header block can be decoded. For a header block with references to the dynamic table, the Required Insert Count is one larger than the largest Absolute Index of all referenced dynamic table entries. For a header block with no references to the dynamic table, the Required Insert Count is zero.

If the decoder encounters a header block with a Required Insert Count value larger than defined above, it MAY treat this as a stream error of type HTTP_QPACK_DECOMPRESSION_FAILED. If the decoder encounters a header block with a Required Insert Count value smaller than defined above, it MUST treat this as a stream error of type HTTP_QPACK_DECOMPRESSION_FAILED as prescribed in Section 3.2.7.

When the Required Insert Count is zero, the frame contains no references to the dynamic table and can always be processed immediately.

If the Required Insert Count is greater than the number of dynamic table entries received, the stream is considered "blocked." While blocked, header field data SHOULD remain in the blocked stream's flow control window. A stream becomes unblocked when the Insert Count becomes greater than or equal to the Required Insert Count for all header blocks the decoder has started reading from the stream.

The SETTINGS_QPACK_BLOCKED_STREAMS setting (see Section 5) specifies an upper bound on the number of streams which can be blocked. An encoder MUST limit the number of streams which could become blocked to the value of SETTINGS_QPACK_BLOCKED_STREAMS at all times. Note that the decoder might not actually become blocked on every stream which risks becoming blocked. If the decoder encounters more blocked streams than it promised to support, it MUST treat this as a stream error of type HTTP_QPACK_DECOMPRESSION_FAILED.

An encoder can decide whether to risk having a stream become blocked. If permitted by the value of SETTINGS_QPACK_BLOCKED_STREAMS, compression efficiency can often be improved by referencing dynamic table entries that are still in transit, but if there is loss or reordering the stream can become blocked at the decoder. An encoder avoids the risk of blocking by only referencing dynamic table entries which have been acknowledged, but this could mean using literals. Since literals make the header block larger, this can result in the encoder becoming blocked on congestion or flow control limits.

2.1.4. Known Received Count

In order to identify which dynamic table entries can be safely used without a stream becoming blocked, the encoder tracks the number of entries received by the decoder. The Known Received Count tracks the total number of acknowledged insertions.

When blocking references are permitted, the encoder uses header block acknowledgement to maintain the Known Received Count, as described in Section 4.4.2.

To acknowledge dynamic table entries which are not referenced by header blocks, for example because the encoder or the decoder have chosen not to risk blocked streams, the decoder sends an Insert Count Increment instruction (see Section 4.4.1).

2.2. Decoder

As in HPACK, the decoder processes header blocks and emits the corresponding header lists. It also processes dynamic table modifications from encoder instructions received on the encoder stream.

The decoder MUST emit header fields in the order their representations appear in the input header block.

2.2.1. State Synchronization

The decoder instructions (Section 4.4) signal key events at the decoder that permit the encoder to track the decoder's state. These events are:

- Complete processing of a header block
- Abandonment of a stream which might have remaining header blocks
- Receipt of new dynamic table entries

Knowledge that a header block with references to the dynamic table has been processed permits the encoder to evict entries to which no unacknowledged references remain (see Section 2.1.2). When a stream is reset or abandoned, the indication that these header blocks will never be processed serves a similar function (see Section 4.4.3).

The decoder chooses when to emit Insert Count Increment instructions (see Section 4.4.1). Emitting an instruction after adding each new dynamic table entry will provide the most timely feedback to the encoder, but could be redundant with other decoder feedback. By delaying an Insert Count Increment instruction, the decoder might be able to coalesce multiple Insert Count Increment instructions, or replace them entirely with Header Acknowledgements (see Section 4.4.2). However, delaying too long may lead to compression inefficiencies if the encoder waits for an entry to be acknowledged before using it.

2.2.2. Blocked Decoding

To track blocked streams, the Required Insert Count value for each stream can be used. Whenever the decoder processes a table update, it can begin decoding any blocked streams that now have their dependencies satisfied.

3. Header Tables

Unlike in HPACK, entries in the QPACK static and dynamic tables are addressed separately. The following sections describe how entries in each table are addressed.

3.1. Static Table

The static table consists of a predefined static list of header fields, each of which has a fixed index over time. Its entries are defined in Appendix A.

All entries in the static table have a name and a value. However, values can be empty (that is, have a length of 0).

Note the QPACK static table is indexed from 0, whereas the HPACK static table is indexed from 1.

When the decoder encounters an invalid static table index in a header block instruction it MUST treat this as a stream error of type "HTTP_QPACK_DECOMPRESSION_FAILED". If this index is received on the encoder stream, this MUST be treated as a connection error of type "HTTP_QPACK_ENCODER_STREAM_ERROR".

3.2. Dynamic Table

The dynamic table consists of a list of header fields maintained in first-in, first-out order. Each HTTP/3 endpoint holds a dynamic table that is initially empty. Entries are added by encoder instructions received on the encoder stream (see Section 4.3).

The dynamic table can contain duplicate entries (i.e., entries with the same name and same value). Therefore, duplicate entries MUST NOT be treated as an error by the decoder.

3.2.1. Dynamic Table Size

The size of the dynamic table is the sum of the size of its entries.

The size of an entry is the sum of its name's length in bytes (as defined in Section 4.1.2), its value's length in bytes, and 32.

The size of an entry is calculated using the length of its name and value without Huffman encoding applied.

3.2.2. Dynamic Table Capacity and Eviction

The encoder sets the capacity of the dynamic table, which serves as the upper limit on its size. The initial capcity of the dynamic table is zero.

Before a new entry is added to the dynamic table, entries are evicted from the end of the dynamic table until the size of the dynamic table is less than or equal to (table capacity - size of new entry) or until the table is empty. The encoder MUST NOT evict a blocking dynamic table entry (see Section 2.1.2).

If the size of the new entry is less than or equal to the dynamic table capacity, then that entry is added to the table. It is an error if the encoder attempts to add an entry that is larger than the

dynamic table capacity; the decoder MUST treat this as a connection error of type "HTTP_QPACK_ENCODER_STREAM_ERROR".

A new entry can reference an entry in the dynamic table that will be evicted when adding this new entry into the dynamic table. Implementations are cautioned to avoid deleting the referenced name or value if the referenced entry is evicted from the dynamic table prior to inserting the new entry.

Whenever the dynamic table capacity is reduced by the encoder, entries are evicted from the end of the dynamic table until the size of the dynamic table is less than or equal to the new table capacity. This mechanism can be used to completely clear entries from the dynamic table by setting a capacity of 0, which can subsequently be restored.

3.2.3. Maximum Dynamic Table Capacity

To bound the memory requirements of the decoder, the decoder limits the maximum value the encoder is permitted to set for the dynamic table capacity. In HTTP/3, this limit is determined by the value of SETTINGS_QPACK_MAX_TABLE_CAPACITY sent by the decoder (see Section 5). The encoder MUST not set a dynamic table capacity that exceeds this maximum, but it can choose to use a lower dynamic table capacity (see Section 4.3.4).

For clients using 0-RTT data in HTTP/3, the server's maximum table capacity is the remembered value of the setting, or zero if the value was not previously sent. When the client's 0-RTT value of the SETTING is 0, the server MAY set it to a non-zero value in its SETTINGS frame. If the remembered value is non-zero, the server MUST send the same non-zero value in its SETTINGS frame. If it specifies any other value, or omits SETTINGS_QPACK_MAX_TABLE_CAPACITY from SETTINGS, the encoder must treat this as a connection error of type "HTTP_QPACK_DECODER_STREAM_ERROR".

For HTTP/3 servers and HTTP/3 clients when 0-RTT is not attempted or is rejected, the maximum table capacity is 0 until the encoder processes a SETTINGS frame with a non-zero value of SETTINGS_QPACK_MAX_TABLE_CAPACITY.

When the maximum table capacity is 0, the encoder MUST NOT insert entries into the dynamic table, and MUST NOT send any encoder instructions on the encoder stream.

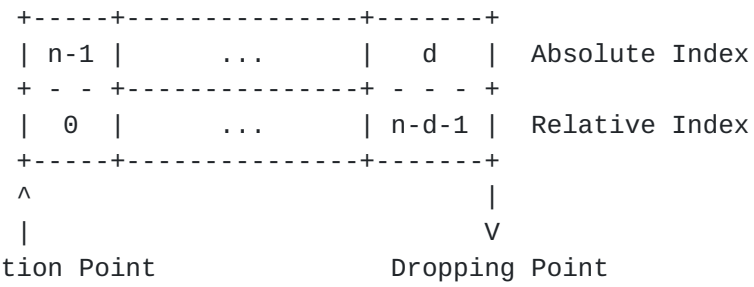
3.2.4. Absolute Indexing

Each entry possesses both an absolute index which is fixed for the lifetime of that entry and a relative index which changes based on the context of the reference. The first entry inserted has an absolute index of "0"; indices increase by one with each insertion.

3.2.5. Relative Indexing

The relative index begins at zero and increases in the opposite direction from the absolute index. Determining which entry has a relative index of "0" depends on the context of the reference.

In encoder instructions, a relative index of "0" always refers to the most recently inserted value in the dynamic table. Note that this means the entry referenced by a given relative index will change while interpreting instructions on the encoder stream.



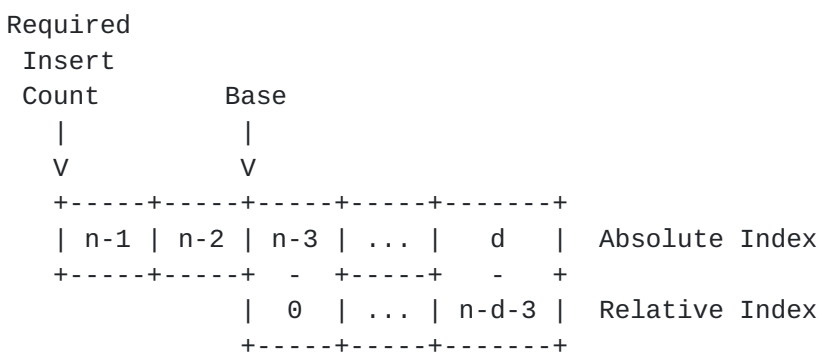
n = count of entries inserted
d = count of entries dropped

Example Dynamic Table Indexing - Control Stream

Unlike encoder instructions, relative indices in header block instructions are relative to the Base at the beginning of the header block (see Section 4.5.1). This ensures that references are stable even if the dynamic table is updated while decoding a header block.

The Base is encoded as a value relative to the Required Insert Count. The Base identifies which dynamic table entries can be referenced using relative indexing, starting with 0 at the last entry added.

Post-Base references are used for entries inserted after base, starting at 0 for the first entry added after the Base, see Section 3.2.6.

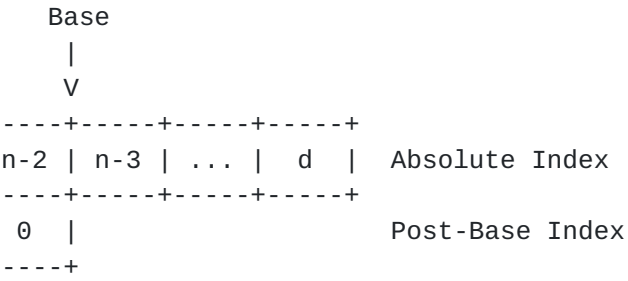


n = count of entries inserted
d = count of entries dropped

Example Dynamic Table Indexing - Relative Index in Header Block

3.2.6. Post-Base Indexing

A header block can reference entries added after the entry identified by the Base. This allows an encoder to process a header block in a single pass and include references to entries added while processing this (or other) header blocks. Newly added entries are referenced using Post-Base instructions. Indices for Post-Base instructions increase in the same direction as absolute indices, with the zero value being the first entry inserted after the Base.



n = count of entries inserted
d = count of entries dropped

Example Dynamic Table Indexing - Post-Base Index in Header Block

3.2.7. Invalid References

If the decoder encounters a reference in a header block instruction to a dynamic table entry which has already been evicted or which has an absolute index greater than or equal to the declared Required Insert Count (see Section 4.5.1), it MUST treat this as a stream error of type "HTTP_QPACK_DECOMPRESSION_FAILED".

If the decoder encounters a reference in an encoder instruction to a dynamic table entry which has already been dropped, it MUST treat this as a connection error of type "HTTP_QPACK_ENCODER_STREAM_ERROR".

4. Wire Format

4.1. Primitives

4.1.1. Prefixed Integers

The prefixed integer from Section 5.1 of [RFC7541] is used heavily throughout this document. The format from [RFC7541] is used unmodified. QPACK implementations MUST be able to decode integers up to 62 bits long.

4.1.2. String Literals

The string literal defined by Section 5.2 of [RFC7541] is also used throughout. This string format includes optional Huffman encoding.

HPACK defines string literals to begin on a byte boundary. They begin with a single flag (indicating whether the string is Huffman-coded), followed by the Length encoded as a 7-bit prefix integer, and finally Length bytes of data. When Huffman encoding is enabled, the Huffman table from Appendix B of [RFC7541] is used without modification.

This document expands the definition of string literals and permits them to begin other than on a byte boundary. An "N-bit prefix string literal" begins with the same Huffman flag, followed by the length encoded as an (N-1)-bit prefix integer. The remainder of the string literal is unmodified.

A string literal without a prefix length noted is an 8-bit prefix string literal and follows the definitions in [RFC7541] without modification.

4.2. Instructions

There are three separate QPACK instruction spaces. Encoder instructions (Section 4.3) carry table updates, decoder instructions (Section 4.4) carry acknowledgments of table modifications and header processing, and header block instructions (Section 4.5) convey an encoded representation of a header list by referring to the QPACK table state.

Encoder and decoder instructions appear on the unidirectional stream types described in this section. Header block instructions are

contained in HEADERS and PUSH_PROMISE frames, which are conveyed on request or push streams as described in [HTTP3].

4.2.1. Encoder and Decoder Streams

QPACK defines two unidirectional stream types:

- An encoder stream is a unidirectional stream of type "0x02". It carries an unframed sequence of encoder instructions from encoder to decoder.

- A decoder stream is a unidirectional stream of type "0x03". It carries an unframed sequence of decoder instructions from decoder to encoder.

HTTP/3 endpoints contain a QPACK encoder and decoder. Each endpoint MUST initiate a single encoder stream and decoder stream. Receipt of a second instance of either stream type be MUST treated as a connection error of type HTTP_WRONG_STREAM_COUNT. These streams MUST NOT be closed. Closure of either unidirectional stream type MUST be treated as a connection error of type HTTP_CLOSED_CRITICAL_STREAM.

4.3. Encoder Instructions

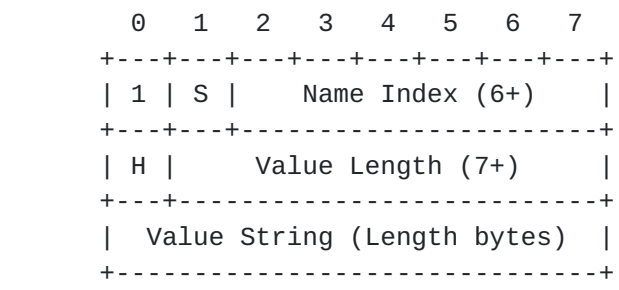
Table updates can add a table entry, possibly using existing entries to avoid transmitting redundant information. The name can be transmitted as a reference to an existing entry in the static or the dynamic table or as a string literal. For entries which already exist in the dynamic table, the full entry can also be used by reference, creating a duplicate entry.

This section specifies the following encoder instructions.

4.3.1. Insert With Name Reference

An addition to the header table where the header field name matches the header field name of an entry stored in the static table or the dynamic table starts with the '1' one-bit pattern. The "S" bit indicates whether the reference is to the static (S=1) or dynamic (S=0) table. The 6-bit prefix integer (see Section 5.1 of [RFC7541]) that follows is used to locate the table entry for the header name. When S=1, the number represents the static table index; when S=0, the number is the relative index of the entry in the dynamic table.

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

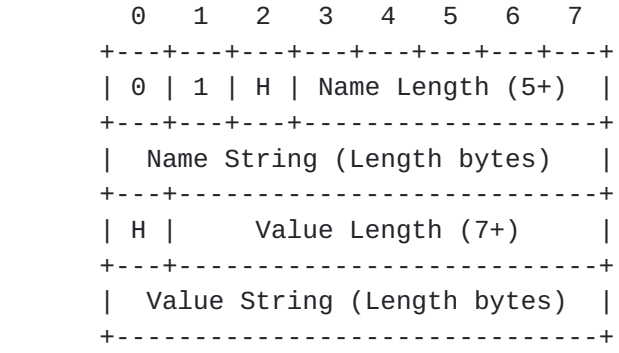


Insert Header Field -- Indexed Name

4.3.2. Insert Without Name Reference

An addition to the header table where both the header field name and the header field value are represented as string literals (see Section 4.1) starts with the '01' two-bit pattern.

The name is represented as a 6-bit prefix string literal, while the value is represented as an 8-bit prefix string literal.



Insert Header Field -- New Name

4.3.3. Duplicate

Duplication of an existing entry in the dynamic table starts with the '000' three-bit pattern. The relative index of the existing entry is represented as an integer with a 5-bit prefix.

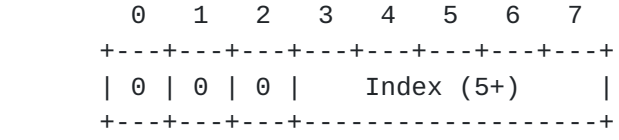


Figure 2: Duplicate

The existing entry is re-inserted into the dynamic table without resending either the name or the value. This is useful to mitigate

the eviction of older entries which are frequently referenced, both to avoid the need to resend the header and to avoid the entry in the table blocking the ability to insert new headers.

4.3.4. Set Dynamic Table Capacity

An encoder informs the decoder of a change to the dynamic table capacity using an instruction which begins with the '001' three-bit pattern. The new dynamic table capacity is represented as an integer with a 5-bit prefix (see Section 5.1 of [RFC7541]).

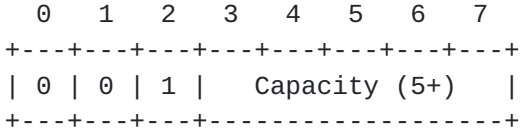


Figure 3: Set Dynamic Table Capacity

The new capacity MUST be lower than or equal to the limit described in Section 3.2.3. In HTTP/3, this limit is the value of the SETTINGS_QPACK_MAX_TABLE_CAPACITY parameter (see Section 5) received from the decoder. The decoder MUST treat a new dynamic table capacity value that exceeds this limit as a connection error of type "HTTP_QPACK_ENCODER_STREAM_ERROR".

Reducing the dynamic table capacity can cause entries to be evicted (see Section 3.2.2). This MUST NOT cause the eviction of blocking entries (see Section 2.1.2). Changing the capacity of the dynamic table is not acknowledged as this instruction does not insert an entry.

4.4. Decoder Instructions

Decoder instructions provide information used to ensure consistency of the dynamic table. They are sent from the decoder to the encoder on a decoder stream; that is, the server informs the client about the processing of the client's header blocks and table updates, and the client informs the server about the processing of the server's header blocks and table updates.

This section specifies the following decoder instructions.

4.4.1. Insert Count Increment

The Insert Count Increment instruction begins with the '00' two-bit pattern. The instruction specifies the total number of dynamic table inserts and duplications since the last Insert Count Increment or Header Acknowledgement that increased the Known Received Count for

the dynamic table (see Section 2.1.4). The Increment field is encoded as a 6-bit prefix integer. The encoder uses this value to determine which table entries might cause a stream to become blocked, as described in Section 2.2.1.

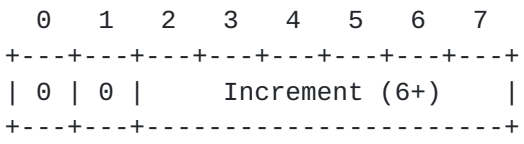


Figure 4: Insert Count Increment

An encoder that receives an Increment field equal to zero or one that increases the Known Received Count beyond what the encoder has sent MUST treat this as a connection error of type "HTTP_QPACK_DECODER_STREAM_ERROR".

4.4.2. Header Acknowledgement

After processing a header block whose declared Required Insert Count is not zero, the decoder emits a Header Acknowledgement instruction on the decoder stream. The instruction begins with the '1' one-bit pattern and includes the header block's associated stream ID, encoded as a 7-bit prefix integer. It is used by the peer's encoder to know when it is safe to evict an entry, and possibly update the Known Received Count.

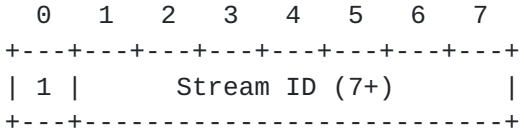


Figure 5: Header Acknowledgement

The same Stream ID can be identified multiple times, as multiple header blocks can be sent on a single stream in the case of intermediate responses, trailers, and pushed requests. Since HEADERS and PUSH_PROMISE frames on each stream are received and processed in order, this gives the encoder precise feedback on which header blocks within a stream have been fully processed.

If an encoder receives a Header Acknowledgement instruction referring to a stream on which every header block with a non-zero Required Insert Count has already been acknowledged, that MUST be treated as a connection error of type "HTTP_QPACK_DECODER_STREAM_ERROR".

When blocking references are permitted, the encoder uses acknowledgement of header blocks to update the Known Received Count.

If a header block was potentially blocking, the acknowledgement implies that the decoder has received all dynamic table state necessary to process the header block. If the Required Insert Count of an acknowledged header block was greater than the encoder's current Known Received Count, the block's Required Insert Count becomes the new Known Received Count.

4.4.3. Stream Cancellation

The instruction begins with the '01' two-bit pattern. The instruction includes the stream ID of the affected stream - a request or push stream - encoded as a 6-bit prefix integer.

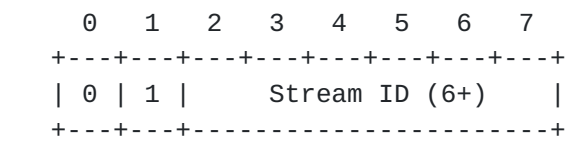


Figure 6: Stream Cancellation

A stream that is reset might have multiple outstanding header blocks with dynamic table references. When an endpoint receives a stream reset before the end of a stream, it generates a Stream Cancellation instruction on the decoder stream. Similarly, when an endpoint abandons reading of a stream it needs to signal this using the Stream Cancellation instruction. This signals to the encoder that all references to the dynamic table on that stream are no longer outstanding. A decoder with a maximum dynamic table capacity equal to zero (see Section 3.2.3) MAY omit sending Stream Cancellations, because the encoder cannot have any dynamic table references.

An encoder cannot infer from this instruction that any updates to the dynamic table have been received.

4.5. Header Block Instructions

HTTP/3 endpoints convert header lists to headers blocks and exchange them inside HEADERS and PUSH_PROMISE frames. A decoder interprets header block instructions in order to construct a header list. These instructions reference the static table, or dynamic table in a particular state without modifying it.

This section specifies the following header block instructions.

4.5.1. Header Block Prefix

Each header block is prefixed with two integers. The Required Insert Count is encoded as an integer with an 8-bit prefix after the encoding described in Section 4.5.1.1). The Base is encoded as sign-and-modulus integer, using a single sign bit and a value with a 7-bit prefix (see Section 4.5.1.2).

These two values are followed by instructions for compressed headers. The entire block is expected to be framed by the using protocol.

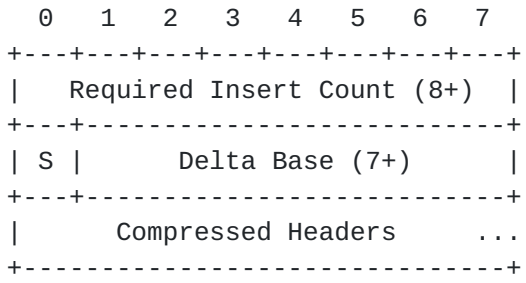


Figure 7: Frame Payload

4.5.1.1. Required Insert Count

Required Insert Count identifies the state of the dynamic table needed to process the header block. Blocking decoders use the Required Insert Count to determine when it is safe to process the rest of the block.

The encoder transforms the Required Insert Count as follows before encoding:

```
if ReqInsertCount == 0:
   EncInsertCount = 0
else:
   EncInsertCount = (ReqInsertCount mod (2 * MaxEntries)) + 1
```

Here "MaxEntries" is the maximum number of entries that the dynamic table can have. The smallest entry has empty name and value strings and has the size of 32. Hence "MaxEntries" is calculated as

```
MaxEntries = floor( MaxTableCapacity / 32 )
```

"MaxTableCapacity" is the maximum capacity of the dynamic table as specified by the decoder (see Section 3.2.3).

This encoding limits the length of the prefix on long-lived connections.

The decoder can reconstruct the Required Insert Count using an algorithm such as the following. If the decoder encounters a value of EncodedInsertCount that could not have been produced by a conformant encoder, it MUST treat this as a stream error of type "HTTP_QPACK_DECOMPRESSION_FAILED".

TotalNumberOfInserts is the total number of inserts into the decoder's dynamic table.

```
FullRange = 2 * MaxEntries
if EncodedInsertCount == 0:
   ReqInsertCount = 0
else:
   if EncodedInsertCount > FullRange:
      Error
   MaxValue = TotalNumberOfInserts + MaxEntries

   # MaxWrapped is the largest possible value of
   # ReqInsertCount that is 0 mod 2*MaxEntries
   MaxWrapped = floor(MaxValue / FullRange) * FullRange
   ReqInsertCount = MaxWrapped + EncodedInsertCount - 1

   # If ReqInsertCount exceeds MaxValue, the Encoder's value
   # must have wrapped one fewer time
   if ReqInsertCount > MaxValue:
      if ReqInsertCount < FullRange:
         Error
      ReqInsertCount -= FullRange
```

For example, if the dynamic table is 100 bytes, then the Required Insert Count will be encoded modulo 6. If a decoder has received 10 inserts, then an encoded value of 3 indicates that the Required Insert Count is 9 for the header block.

4.5.1.2. Base

The "Base" is used to resolve references in the dynamic table as described in Section 3.2.5.

To save space, the Base is encoded relative to the Insert Count using a one-bit sign and the "Delta Base" value. A sign bit of 0 indicates that the Base is greater than or equal to the value of the Insert Count; the value of Delta Base is added to the Insert Count to determine the value of the Base. A sign bit of 1 indicates that the Base is less than the Insert Count. That is:

```
   if S == 0:
      Base = ReqInsertCount + DeltaBase
   else:
      Base = ReqInsertCount - DeltaBase - 1
```

A single-pass encoder determines the Base before encoding a header block. If the encoder inserted entries in the dynamic table while encoding the header block, Required Insert Count will be greater than the Base, so the encoded difference is negative and the sign bit is set to 1. If the header block did not reference the most recent entry in the table and did not insert any new entries, the Base will be greater than the Required Insert Count, so the delta will be positive and the sign bit is set to 0.

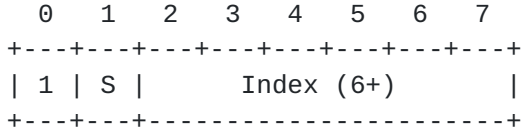
An encoder that produces table updates before encoding a header block might set Required Insert Count and the Base to the same value. In such case, both the sign bit and the Delta Base will be set to zero.

A header block that does not reference the dynamic table can use any value for the Base; setting Delta Base to zero is the most efficient encoding.

For example, with an Required Insert Count of 9, a decoder receives a S bit of 1 and a Delta Base of 2. This sets the Base to 6 and enables post-base indexing for three entries. In this example, a regular index of 1 refers to the 5th entry that was added to the table; a post-base index of 1 refers to the 8th entry.

4.5.2. Indexed Header Field

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].

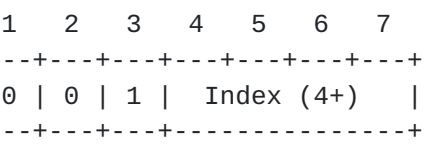


Indexed Header Field

If the entry is in the static table, or in the dynamic table with an absolute index less than the Base, this representation starts with the '1' 1-bit pattern, followed by the "S" bit indicating whether the reference is into the static (S=1) or dynamic (S=0) table. Finally, the relative index of the matching header field is represented as an integer with a 6-bit prefix (see Section 5.1 of [RFC7541]).

4.5.3. Indexed Header Field With Post-Base Index

If the entry is in the dynamic table with an absolute index greater than or equal to the Base, the representation starts with the '0001' 4-bit pattern, followed by the post-base index (see Section 3.2.6) of the matching header field, represented as an integer with a 4-bit prefix (see Section 5.1 of [RFC7541]).



Indexed Header Field with Post-Base Index

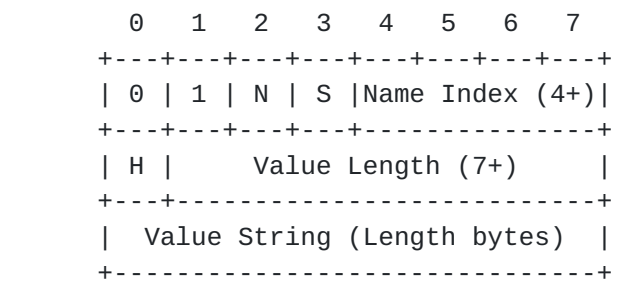
4.5.4. Literal Header Field With Name Reference

A literal header field with a name reference represents a header where the header field name matches the header field name of an entry stored in the static table or the dynamic table.

If the entry is in the static table, or in the dynamic table with an absolute index less than the Base, this representation starts with the '01' two-bit pattern. If the entry is in the dynamic table with an absolute index greater than or equal to the Base, the representation starts with the '0000' four-bit pattern.

Only the header field name stored in the static or dynamic table is used. Any header field value MUST be ignored.

The following 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 a 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 a literal 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).

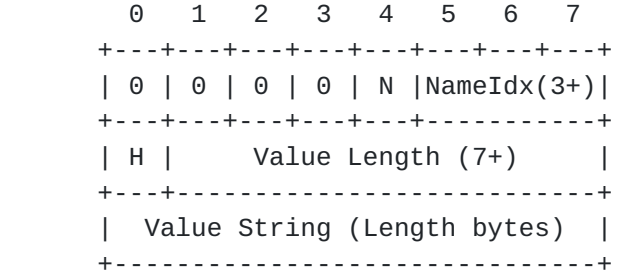


Literal Header Field With Name Reference

For entries in the static table or in the dynamic table with an absolute index less than the Base, the header field name is represented using the relative index of that entry, which is represented as an integer with a 4-bit prefix (see Section 5.1 of [RFC7541]). The "S" bit indicates whether the reference is to the static (S=1) or dynamic (S=0) table.

4.5.5. Literal Header Field With Post-Base Name Reference

For entries in the dynamic table with an absolute index greater than or equal to the Base, the header field name is represented using the post-base index of that entry (see Section 3.2.6) encoded as an integer with a 3-bit prefix.



Literal Header Field With Post-Base Name Reference

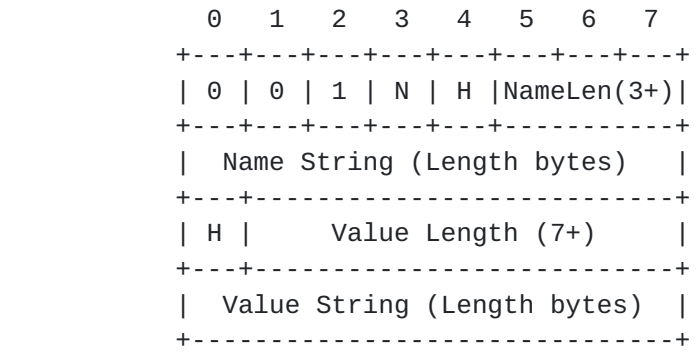
4.5.6. Literal Header Field Without Name Reference

An addition to the header table where both the header field name and the header field value are represented as string literals (see Section 4.1) starts with the '001' three-bit pattern.

The fourth 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 a 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 a literal 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).

The name is represented as a 4-bit prefix string literal, while the value is represented as an 8-bit prefix string literal.



Literal Header Field Without Name Reference

5. Configuration

QPACK defines two settings which are included in the HTTP/3 SETTINGS frame.

SETTINGS_QPACK_MAX_TABLE_CAPACITY (0x1): An integer with a maximum value of 2^30 - 1. The default value is zero bytes. See Section 3.2 for usage. This is the equivalent of the SETTINGS_HEADER_TABLE_SIZE from HTTP/2.

SETTINGS_QPACK_BLOCKED_STREAMS (0x7): An integer with a maximum value of 2^16 - 1. The default value is zero. See Section 2.1.3.

6. Error Handling

The following error codes are defined for HTTP/3 to indicate failures of QPACK which prevent the stream or connection from continuing:

HTTP_QPACK_DECOMPRESSION_FAILED (0x200): The decoder failed to interpret a header block instruction and is not able to continue decoding that header block.

HTTP_QPACK_ENCODER_STREAM_ERROR (0x201): The decoder failed to interpret an encoder instruction received on the encoder stream.

HTTP_QPACK_DECODER_STREAM_ERROR (0x202): The encoder failed to interpret a decoder instruction received on the decoder stream.

Upon encountering an error, an implementation MAY elect to treat it as a connection error even if this document prescribes that it MUST be treated as a stream error.

7. Security Considerations

TBD.

8. IANA Considerations

8.1. Settings Registration

This document specifies two settings. The entries in the following table are registered in the "HTTP/3 Settings" registry established in [HTTP3].

Setting Name	Code	Specification
QPACK_MAX_TABLE_CAPACITY	0x1	Section 5
QPACK_BLOCKED_STREAMS	0x7	Section 5

8.2. Stream Type Registration

This document specifies two stream types. The entries in the following table are registered in the "HTTP/3 Stream Type" registry established in [HTTP3].

Stream Type	Code	Specification	Sender
QPACK Encoder Stream	0x02	Section 4.2.1	Both
QPACK Decoder Stream	0x03	Section 4.2.1	Both

8.3. Error Code Registration

This document specifies three error codes. The entries in the following table are registered in the "HTTP/3 Error Code" registry established in [HTTP3].

Name	Code	Description	Specification
HTTP_QPACK_DECOMPRESSION_FAILED	0x200	Decompression of a header block failed	Section 6
HTTP_QPACK_ENCODER_STREAM_ERROR	0x201	Error on the encoder stream	Section 6
HTTP_QPACK_DECODER_STREAM_ERROR	0x202	Error on the decoder stream	Section 6

9. References

9.1. Normative References

[HTTP3] Bishop, M., Ed., "Hypertext Transfer Protocol Version 3 (HTTP/3)", draft-ietf-quic-http-20 (work in progress), April 2019.

[QUIC-TRANSPORT] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-18 (work in progress), April 2019.

[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>.

[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>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

9.2. Informative References

[RFC2360] Scott, G., "Guide for Internet Standards Writers", BCP 22, RFC 2360, DOI 10.17487/RFC2360, June 1998, <https://www.rfc-editor.org/info/rfc2360>.

[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>.

9.3. URIs

[1] https://mailarchive.ietf.org/arch/search/?email_list=quic

[2] https://github.com/quicwg

[3] https://github.com/quicwg/base-drafts/labels/-qpack

Appendix A. Static Table

Index	Name	Value
0	:authority	
1	:path	/
2	age	0
3	content-disposition	
4	content-length	0
5	cookie	
6	date	
7	etag	
8	if-modified-since	
9	if-none-match	
10	last-modified	
11	link	

12	location		
13	referer		
14	set-cookie		
15	:method	CONNECT	
16	:method	DELETE	
17	:method	GET	
18	:method	HEAD	
19	:method	OPTIONS	
20	:method	POST	
21	:method	PUT	
22	:scheme	http	
23	:scheme	https	
24	:status	103	
25	:status	200	
26	:status	304	
27	:status	404	
28	:status	503	
29	accept	*/*	
30	accept	application/dns-message	
31	accept-encoding	gzip, deflate, br	
32	accept-ranges	bytes	
33	access-control-allow-headers	cache-control	
34	access-control-allow-headers	content-type	

35	access-control-allow-origin	*	
36	cache-control	max-age=0	
37	cache-control	max-age=2592000	
38	cache-control	max-age=604800	
39	cache-control	no-cache	
40	cache-control	no-store	
41	cache-control	public, max-age=31536000	
42	content-encoding	br	
43	content-encoding	gzip	
44	content-type	application/dns-message	
45	content-type	application/javascript	
46	content-type	application/json	
47	content-type	application/x-www-form-urlencoded	
48	content-type	image/gif	
49	content-type	image/jpeg	
50	content-type	image/png	
51	content-type	text/css	
52	content-type	text/html; charset=utf-8	
53	content-type	text/plain	
54	content-type	text/plain;charset=utf-8	
55	range	bytes=0-	
56	strict-transport-security	max-age=31536000	
57	strict-transport-security	max-age=31536000; includesubdomains	

58	strict-transport-security	max-age=31536000; includesubdomains; preload
59	vary	accept-encoding
60	vary	origin
61	x-content-type-options	nosniff
62	x-xss-protection	1; mode=block
63	:status	100
64	:status	204
65	:status	206
66	:status	302
67	:status	400
68	:status	403
69	:status	421
70	:status	425
71	:status	500
72	accept-language	
73	access-control-allow-credentials	FALSE
74	access-control-allow-credentials	TRUE
75	access-control-allow-headers	*
76	access-control-allow-methods	get
77	access-control-allow-methods	get, post, options
78	access-control-allow-	options

	methods		
79	access-control-expose-headers	content-length	
80	access-control-request-headers	content-type	
81	access-control-request-method	get	
82	access-control-request-method	post	
83	alt-svc	clear	
84	authorization		
85	content-security-policy	script-src 'none'; object-src 'none'; base-uri 'none'	
86	early-data	1	
87	expect-ct		
88	forwarded		
89	if-range		
90	origin		
91	purpose	prefetch	
92	server		
93	timing-allow-origin	*	
94	upgrade-insecure-requests	1	
95	user-agent		
96	x-forwarded-for		
97	x-frame-options	deny	
98	x-frame-options	sameorigin	

Appendix B. Sample One Pass Encoding Algorithm

Pseudo-code for single pass encoding, excluding handling of duplicates, non-blocking mode, and reference tracking.

```
   baseIndex = dynamicTable.baseIndex
   largestReference = 0
   for header in headers:
     staticIdx = staticTable.getIndex(header)
     if staticIdx:
       encodeIndexReference(streamBuffer, staticIdx)
       continue

     dynamicIdx = dynamicTable.getIndex(header)
     if !dynamicIdx:
       # No matching entry.  Either insert+index or encode literal
       nameIdx = getNameIndex(header)
       if shouldIndex(header) and dynamicTable.canIndex(header):
         encodeLiteralWithIncrementalIndex(controlBuffer, nameIdx,
                                           header)
         dynamicTable.add(header)
         dynamicIdx = dynamicTable.baseIndex

     if !dynamicIdx:
       # Couldn't index it, literal
       if nameIdx <= staticTable.size:
         encodeLiteral(streamBuffer, nameIndex, header)
       else:
         # encode literal, possibly with nameIdx above baseIndex
         encodeDynamicLiteral(streamBuffer, nameIndex, baseIndex,
                              header)
         largestReference = max(largestReference,
                                dynamicTable.toAbsolute(nameIdx))
     else:
       # Dynamic index reference
       assert(dynamicIdx)
       largestReference = max(largestReference, dynamicIdx)
       # Encode dynamicIdx, possibly with dynamicIdx above baseIndex
       encodeDynamicIndexReference(streamBuffer, dynamicIdx,
                                   baseIndex)

   # encode the prefix
   encodeInteger(prefixBuffer, 0x00, largestReference, 8)
   if baseIndex >= largestReference:
     encodeInteger(prefixBuffer, 0, baseIndex - largestReference, 7)
   else:
     encodeInteger(prefixBuffer, 0x80,
                   largestReference  - baseIndex, 7)

   return controlBuffer, prefixBuffer + streamBuffer
```

Appendix C. Change Log

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

C.1. Since draft-ietf-quic-qpack-06

- Clarify initial dynamic table capacity maximums (#2276, #2330, #2330)

C.2. Since draft-ietf-quic-qpack-05

- Introduced the terms dynamic table capacity and maximum dynamic table capacity.
- Renamed SETTINGS_HEADER_TABLE_SIZE to SETTINGS_QPACK_MAX_TABLE_CAPACITY.

C.3. Since draft-ietf-quic-qpack-04

- Changed calculation of Delta Base Index to avoid an illegal value (#2002, #2005)

C.4. Since draft-ietf-quic-qpack-03

- Change HTTP settings defaults (#2038)
- Substantial editorial reorganization

C.5. Since draft-ietf-quic-qpack-02

- Largest Reference encoded modulo MaxEntries (#1763)
- New Static Table (#1355)
- Table Size Update with Insert Count=0 is a connection error (#1762)
- Stream Cancellations are optional when SETTINGS_HEADER_TABLE_SIZE=0 (#1761)
- Implementations must handle 62 bit integers (#1760)
- Different error types for each QPACK stream, other changes to error handling (#1726)
- Preserve header field order (#1725)

- Initial table size is the maximum permitted when table is first usable (#1642)

C.6. Since draft-ietf-quic-qpack-01

- Only header blocks that reference the dynamic table are acknowledged (#1603, #1605)

C.7. Since draft-ietf-quic-qpack-00

- Renumbered instructions for consistency (#1471, #1472)
- Decoder is allowed to validate largest reference (#1404, #1469)
- Header block acknowledgments also acknowledge the associated largest reference (#1370, #1400)
- Added an acknowledgment for unread streams (#1371, #1400)
- Removed framing from encoder stream (#1361,#1467)
- Control streams use typed unidirectional streams rather than fixed stream IDs (#910,#1359)

C.8. Since draft-ietf-quic-qcram-00

- Separate instruction sets for table updates and header blocks (#1235, #1142, #1141)
- Reworked indexing scheme (#1176, #1145, #1136, #1130, #1125, #1314)
- Added mechanisms that support one-pass encoding (#1138, #1320)
- Added a setting to control the number of blocked decoders (#238, #1140, #1143)
- Moved table updates and acknowledgments to dedicated streams (#1121, #1122, #1238)

Acknowledgments

This draft draws heavily on the text of [RFC7541]. The indirect input of those authors is gratefully acknowledged, as well as ideas from:

- Ryan Hamilton

- Patrick McManus
- Kazuho Oku
- Biren Roy
- Ian Swett
- Dmitri Tikhonov

Buck's contribution was supported by Google during his employment there.

A substantial portion of Mike's contribution was supported by Microsoft during his employment there.

Authors' Addresses

Charles 'Buck' Krasic
Netflix

Email: ckrasic@netflix.com

Mike Bishop
Akamai Technologies

Email: mbishop@evequefou.be

Alan Frindell (editor)
Facebook

Email: afrind@fb.com