

QUIC  
Internet-Draft  
Intended status: Standards Track  
Expires: July 27, 2018

C. Krasic  
Google  
January 23, 2018

Header Compression for HTTP over QUIC  
draft-krasic-quic-qcram-04

Abstract

The design of the core QUIC transport subsumes many HTTP/2 features, prominent among them stream multiplexing. A key advantage of the QUIC transport is stream multiplexing free of head-of-line (HoL) blocking between streams. In HTTP/2, multiplexed streams can suffer HoL blocking due to TCP.

If HTTP/2's HPACK is used for header compression, HTTP/QUIC is still vulnerable to HoL blocking, because of HPACK's assumption of in-order delivery. This draft defines QCRAM, a variation of HPACK and mechanisms in the HTTP/QUIC mapping that allow the flexibility to avoid header-compression-induced HoL blocking.

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 July 27, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of

Internet-Draft

QCRAM

January 2018

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

<a href="#">1.</a>	Introduction . . . . .	<a href="#">2</a>
<a href="#">1.1.</a>	Head-of-Line Blocking in HPACK . . . . .	<a href="#">3</a>
<a href="#">1.2.</a>	Avoiding Head-of-Line Blocking in HTTP/QUIC . . . . .	<a href="#">3</a>
<a href="#">2.</a>	HTTP over QUIC mapping extensions . . . . .	<a href="#">4</a>
<a href="#">2.1.</a>	HEADERS and PUSH_PROMISE . . . . .	<a href="#">4</a>
<a href="#">2.2.</a>	HEADER_ACK . . . . .	<a href="#">4</a>
<a href="#">3.</a>	HPACK extensions . . . . .	<a href="#">5</a>
<a href="#">3.1.</a>	Allowed Instructions . . . . .	<a href="#">5</a>
<a href="#">3.2.</a>	Header Block Prefix . . . . .	<a href="#">5</a>
<a href="#">3.3.</a>	Hybrid absolute-relative indexing . . . . .	<a href="#">6</a>
<a href="#">3.4.</a>	Preventing Eviction Races . . . . .	<a href="#">6</a>
<a href="#">3.4.1.</a>	Blocked Evictions . . . . .	<a href="#">7</a>
<a href="#">3.5.</a>	Refreshing Entries with Duplication . . . . .	<a href="#">7</a>
<a href="#">4.</a>	Performance considerations . . . . .	<a href="#">7</a>
<a href="#">4.1.</a>	Speculative table updates . . . . .	<a href="#">7</a>
<a href="#">4.2.</a>	Fixed overhead. . . . .	<a href="#">8</a>
<a href="#">4.3.</a>	Co-ordinated Packetization . . . . .	<a href="#">8</a>
<a href="#">5.</a>	Security Considerations . . . . .	<a href="#">8</a>
<a href="#">6.</a>	IANA Considerations . . . . .	<a href="#">8</a>
<a href="#">7.</a>	Acknowledgments . . . . .	<a href="#">8</a>
<a href="#">8.</a>	References . . . . .	<a href="#">9</a>
<a href="#">8.1.</a>	Normative References . . . . .	<a href="#">9</a>
<a href="#">8.2.</a>	Informative References . . . . .	<a href="#">9</a>
	Author's Address . . . . .	<a href="#">9</a>

[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. QUIC's stream multiplexing comes into some conflict with header compression. A key goal of the design of QUIC is to improve stream multiplexing relative to HTTP/2 by eliminating HoL (head of line) blocking, which can occur in HTTP/2. HoL blocking can happen because all HTTP/2 streams are multiplexed onto a single TCP

connection with its in-order semantics. QUIC can maintain independence between streams because it implements core transport functionality in a fully stream-aware manner. However, the HTTP/QUIC mapping is still subject to HoL blocking if HPACK is used directly. HPACK exploits multiplexing for greater compression, shrinking the

representation of headers that have appeared earlier on the same connection. In the context of QUIC, this imposes a vulnerability to HoL blocking (see [Section 1.1](#)).

QUIC is described in [\[QUIC-TRANSPORT\]](#). The HTTP/QUIC mapping is described in [\[QUIC-HTTP\]](#). For a full description of HTTP/2, see [\[RFC7540\]](#). The description of HPACK is [\[RFC7541\]](#), with important terminology in [Section 1.3](#).

QCRAM modifies HPACK to allow correctness in the presence of out-of-order delivery, with flexibility for implementations to balance between resilience against HoL 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.

QCRAM is intended to be a relatively non-intrusive extension to HPACK; an implementation should be easily shared within stacks supporting both HTTP/2 over (TLS+)TCP and HTTP/QUIC.

### [1.1](#). Head-of-Line Blocking in HPACK

HPACK enables several types of header representations, one of which also adds the header to a dynamic table of header values. These values are then available for reuse in subsequent header blocks simply by referencing the entry number in the table.

If the packet containing a header is lost, that stream cannot complete header processing until the packet is retransmitted. This is unavoidable. However, other streams which rely on the state created by that packet *also* cannot make progress. This is the problem which QUIC solves in general, but which is reintroduced by HPACK when the loss includes a HEADERS frame.

### [1.2](#). Avoiding Head-of-Line Blocking in HTTP/QUIC

In the example above, the second stream contained a reference to data which might not yet have been processed by the recipient. Such references are called "vulnerable," because the loss of a different packet can keep the reference from being usable.

The encoder can choose on a per-header-block basis whether to favor higher compression ratio (by permitting vulnerable references) or HoL resilience (by avoiding them). This is signaled by the BLOCKING flag in HEADERS and PUSH\_PROMISE frames (see [Section 2](#)).

If a header block contains no vulnerable header fields, BLOCKING MUST be 0. This implies that the header fields are represented either as

references to dynamic table entries which are known to have been received, or as Literal header fields (see [\[RFC7541\] Section 6.2](#)).

If a header block contains any header field which references dynamic table state which the peer might not have received yet, the BLOCKING flag MUST be set. If the peer does not yet have the appropriate state, such blocks might not be processed on arrival.

The header block contains a prefix ([Section 3.2](#)). This prefix contains table offset information that establishes total ordering among all headers, regardless of reordering in the transport (see [Section 3.3](#)). In blocking mode, the prefix additionally identifies the minimum state required to process any vulnerable references in the header block (see "Depends" in [Section 3.3](#)). When the necessary state has arrived, the header block can be processed. Notice that while blocked, HB's header field data remains in stream B's flow control window.

## [2](#). HTTP over QUIC mapping extensions

### [2.1](#). HEADERS and PUSH\_PROMISE

HEADERS and PUSH\_PROMISE frames define a new flag.

BLOCKING (0x01): Indicates the stream might need to wait for dependent headers before processing. If 0, the frame can be processed immediately upon receipt.

HEADERS frames can be sent on the Connection Control Stream as well

as on request / push streams.

## [2.2.](#) HEADER\_ACK

The HEADER\_ACK frame (type=0x8) is sent from the decoder to the encoder on the Control Stream when the decoder has fully processed a header block. It is used by the encoder to determine whether subsequent indexed representations that might reference that block are vulnerable to HoL blocking.

The HEADER\_ACK frame indicates the stream on which the header block was processed by encoding the Stream ID as a variable-length integer. The same Stream ID can be identified multiple times, as multiple header-containing blocks can be sent on a single stream in the case of intermediate responses, trailers, pushed requests, etc. as well as on the Control Streams.

Since header 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. This information can then be used to correctly track outstanding stream references to checkpoints.

```
  0  1  2  3  4  5  6  7
+---+---+---+---+---+---+---+---+
|           Stream ID [i]           |
+---+-----+-----+-----+-----+
```

HEADER\_ACK frame

The HEADER\_ACK frame does not define any flags.

## [3.](#) HPACK extensions

### [3.1.](#) Allowed Instructions

HEADERS frames on the Control Streams SHOULD contain only Literal with Incremental Indexing representations. Frames on this stream modify the dynamic table state without generating output to any particular request.

HEADERS and PUSH\_PROMISE frames on request and push streams MUST NOT contain Literal with Incremental Indexing representations. Frames on these streams reference the dynamic table in a particular state without modifying it, but emit the headers for an HTTP request or response.

### 3.2. Header Block Prefix

In HEADERS and PUSH\_PROMISE frames, HPACK Header data is prefixed by an integer: "Base Index". "Base index" is the cumulative number of entries added to the table prior to encoding the current block, it is encoded as a single 8-bit prefix integer:

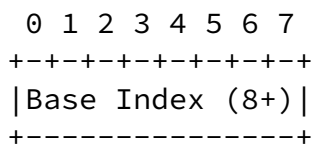


Figure 1: Absolute indexing (BLOCKING=0x0)

[Section 3.3](#) describes the role of "Base Index".

When the BLOCKING flag is 0x1, a the prefix additionally contains a second HPACK integer (8-bit prefix) 'Depends':

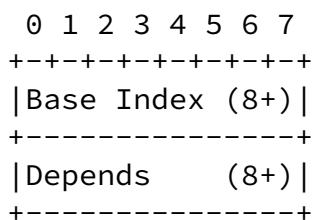


Figure 2: Absolute indexing (BLOCKING=0x1)

Depends is used to identify header dependencies, namely the largest table entry referred to by indexed representations within the following header block. Its usage is described in [Section 1.2](#). The largest index referenced is "Base Index - Depends".

### 3.3. Hybrid absolute-relative indexing

HPACK indexed entries refer to an entry by its current position in the dynamic table. As Figure 1 of [[RFC7541](#)] illustrates, newer entries have smaller indices, and older entries are evicted first if the table is full. Under this scheme, each insertion to the table causes the index of all existing entries to change (implicitly). Implicit index updates are acceptable for HTTP/2 because TCP is totally ordered, but are problematic in the out-of-order context of QUIC.

QCRAM uses a hybrid absolute-relative indexing approach. The prefix defined in [Section 3.2](#) is used by the decoder to interpret all subsequent HPACK instructions at absolute positions for indexed lookups and insertions.

Since QCRAM handles blocking at the header block level, it is an error if the HPACK decoder encounters an indexed representation that refers to an entry missing from the table, and the connection MUST be closed with the "HTTP\_HPACk\_DECOMPRESSION\_FAILED" error code.

#### [3.4.](#) Preventing Eviction Races

Due to out-of-order arrival, QCRAM's eviction algorithm requires changes (relative to HPACK) to avoid the possibility that an indexed representation is decoded after the referenced entry has already been evicted. QCRAM employs a two-phase eviction algorithm, in which the encoder will not evict entries that have outstanding (unacknowledged) references.

##### [3.4.1.](#) Blocked Evictions

The decoder MUST NOT permit an entry to be evicted while a reference to that entry remains unacknowledged. If a new header to be inserted into the dynamic table would cause the eviction of such an entry, the encoder MUST NOT emit the insert instruction until the reference has been processed by the decoder and acknowledged.

The encoder can emit a literal representation for the new header in order to avoid encoding delays, and MAY insert the header into the table later if desired.

To ensure that the blocked eviction case is rare, references to the oldest entries in the dynamic table SHOULD be avoided. When one of the oldest entries in the table is still actively used for references, the encoder SHOULD emit an Indexed-Duplicate representation instead (see [Section 3.5](#)).

### [3.5](#). Refreshing Entries with Duplication

```
  0 1 2 3 4 5 6 7
+--+--+--+--+--+--+
|0|0|1|Index(5+)|
+--+--+-----+
```

Figure 3: Indexed Header Field with Duplication

`_Indexed-Duplicates_` insert a new entry into the dynamic table which duplicates an existing entry. [\[RFC7541\]](#) allows duplicate HPACK table entries, that is entries that have the same name and value.

This replaces the HPACK instruction for Dynamic Table Size Update (see [Section 6.3 of \[RFC7541\]](#), which is not supported by HTTP over QUIC).

## [4](#). Performance considerations

### [4.1](#). Speculative table updates

Implementations can `_speculatively_` send header frames on the HTTP Control Streams which are not needed for any current HTTP request or response. Such headers could be used strategically to improve performance. For instance, the encoder might decide to `_refresh_` by sending Indexed-Duplicate representations for popular header fields ([Section 3.2](#)), ensuring they have small indices and hence minimal size on the wire.

### [4.2](#). Fixed overhead.



HPACK defines overhead as 32 bytes ([\[RFC7541\] Section 4.1](#)). QCRAM adds some per-entry state, to track acknowledgment status and eviction reference count. A larger value than 32 might be more accurate for QCRAM.

#### [4.3.](#) Co-ordinated Packetization

When a dynamic table entry is both defined and referenced by header blocks within the same packet, there is no risk of HoL blocking and using an indexed representation is strictly better than using a literal. An implementation could attempt to exploit this exception by employing co-ordination between QCRAM compression and QUIC transport packetization. However, if the packet is lost, the transport might choose a different packetization when retransmitting the missing data.

#### [5.](#) Security Considerations

TBD.

#### [6.](#) IANA Considerations

This document registers a new frame type, HEADER\_ACK, for HTTP/QUIC. This will need to be added to the IANA Considerations of [\[QUIC-HTTP\]](#).

#### [7.](#) Acknowledgments

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

- o Mike Bishop
- o Patrick McManus
- o Biren Roy
- o Alan Frindell
- o Ian Swett
- o Ryan Hamilton

## [8.](#) References

### [8.1.](#) Normative References

#### [QUIC-HTTP]

Bishop, M., "Hypertext Transfer Protocol (HTTP) over QUIC", [draft-ietf-quic-http-08](#) (work in progress), December 2017.

[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>>.

### [8.2.](#) Informative References

#### [QUIC-TRANSPORT]

Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", [draft-ietf-quic-transport-08](#) (work in progress), December 2017.

[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>>.

## Author's Address

Charles 'Buck' Krasic  
Google

Email: [ckrasic@google.com](mailto:ckrasic@google.com)

Krasic

Expires July 27, 2018

[Page 9]