QUIC
Internet-Draft
Intended status: Standards Track
Expires: January 18, 2018

C. Krasic
Google

July 17, 2017

# Header Compression for HTTP over QUIC
## draft-krasic-quic-qcram-01

Abstract

The design of the core QUIC transport and the mapping of HTTP semantics over it subsume many HTTP/2 features, prominent among them stream multiplexing and HTTP header compression. A key advantage of the QUIC transport is that provides stream multiplexing free of HoL blocking between streams, while in HTTP/2 multiplexed streams can suffer HoL blocking primarily due to HTTP/2's layering above TCP. However, assuming HPACK is used for header compression, HTTP over QUIC is still vulnerable to HoL blocking, because of how HPACK exploits header redundancies between multiplexed HTTP transactions. This draft defines QCRAM, a variation of HPACK and mechanisms in the QUIC HTTP mapping that allow QUIC implementations 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 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 January 18, 2018.

Table of Contents



## 1. Introduction

The QUIC transport protocol was designed from the outset to support HTTP semantics, and its design subsumes most of the features of HTTP/2. Two of those features, stream multiplexing and header compression come into some conflict in QUIC. A key goal of the design of QUIC is to improve stream multiplexing relative to HTTP/2, by eliminating HoL (head of line) blocking that can occur in HTTP/2.

HoL blocking can happen because 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 over QUIC mapping is still subject to HoL blocking if HPACK is used directly as in HTTP/2. 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 as will be described more below (Section 2.1).

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

## 2. QCRAM overview

Readers may wish to refer to [RFC7540] Section 1.4 to review HPACK terminology, and [QUIC-HTTP], Sections 4 on "HTTP over QUIC stream mapping" and 4.2.1 on "Header Compression".

This draft extends HPACK and the HTTP over QUIC mapping with the option to avoid HoL blocking. QCRAM is intended to be a relatively non-intrusive extension to HPACK, an implementation should be easily shared within stacks supporting both HTTP/2 and HTTP over QUIC.

QCRAM strives to solve HoL blocking in the simplest way possible. To that end, the mechanisms QCRAM defines are largely at the granularity of header blocks, as opposed to individual header field representations. QCRAM also employs HPACK fallback modes that simplify certain edge cases and offer flexibilty.

For greatest performance, QCRAM requires QUIC specific mechanisms that leverage tight integration between transport and HTTP layers, as will be described in Section 2.2.3.

### 2.1. Example of HoL blocking

The following is an example of how HPACK can induce HoL blocking in QUIC. Assume two HTTP message exchange streams "A" and "B", and corresponding header blocks "HA" and "HB". Stream "B" experiences HoL blocking due to "A" as follows:

1. HPACK encodes header field "HB[i]" using an index that refers to a table entry that resulted from header field "HA[j]".

2. "HA" and "HB" are delivered via distinct packets that are inflight in the same round trip.

3. "HB"'s packet is delivered but "HA"'s is dropped. HPACK can not decode "HB" until "HA"'s packet is successfully retransmitted.

## 2.2. How QCRAM avoids HoL blocking

Continuing the example, QCRAM's approach is as follows.

1. "HB[i]" can refer to "HA[j]" if "HA[j]" was delivered in a prior round trip.

2. "HB[i]" can refer to "HA[j]" if "HA" and "HB" are to be delivered in the same packet.

3. Otherwise, HB is vulnerable to HoL blocking due to HA. "HB[i]" should be represented using an HPACK literal. Alternatively, HB should use HPACK fallback (Section 2.2.2).

Some degree of coordination between the HTTP mapping and core QUIC transport is required to distinguish the above cases. The first case can be supported if the transport has some method to notify as previously written data are acknowledged Section 4.2. The second case can be approximate or precise, depending on whether the transport interface allows packet granularity coordination Section 2.2.3.

### 2.2.1. Splitting writes from reads

_Note: this draft assumes the HTTP mapping uses a single QUIC stream per HTTP message exchange._

HPACK indexed entries refer to an entry by its current position in the dynamic table. As Figure 1 of [RFC7541] illustrates, newest entries have smallest indices, and oldest 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). The approach is acceptable for HTTP/2 because TCP is totally ordered, but it is is problematic in the out-of-order context of QUIC.

QCRAM partitions HPACK Header Field Representations (refer to [RFC7540] Section 6.2) data into _writes_ and _reads_. The writes are made up of Literal Header Fields with Incremental Indexing (refer to [RFC7540] Section 6.2.1), all others are reads.

1. Writes are delivered on the on the Connection Control Stream (refer to [QUIC-HTTP] Section 1). This primary purpose of this separation is to ensure that stream resets can not drop HPACK data that is required to keep the encoder and decoder versions of the dynamic table synchronized. Since all writes are on the same

stream, they have the same ordering properties as in HTTP/2, and HPACK's implicit positioning can remain unchanged for writes.

2. Reads are delivered on the HTTP Message Exchange Streams. QCRAM uses a hybrid absolute-relative indexing approach for reads. The purpose is to ensure that indexed header field representations remain coherent under out-of-order delivery. QCRAM header block fragments for reads start with an integer that conveys an absolute base index (defined in Section Section 3.2.1). The format of individual indexed representations does not change (from HPACK), but their semantics become absolute in combination with the base index. Processing of reads MUST block if the corresponding entry has not been added to the table yet. To protect against buggy or malicious implementations, a timer should be used to set an upper bound on such blocking and in the event of a timeout SHOULD reset the stream with HTTP_HPACK_DECOMPRESSION_TIMEOUT.

### 2.2.2. HPACK Fallback

There are two modes of HPACK fallback for a given header block HX:

1. HX is delivered on the HTTP request-response stream, but may only contain indexed entries that reference the static HPACK table. This mode is HoL free, but looses the compression benefit of the HPACK dynamic table.

2. HX is delivered on the Connection Control Stream. In this case, the implemenation must take care to ensure that headers and body data are surfaced to the application in the correct sequence. This mode favors compression, accepting vulnerability to HoL blocking.

Both modes are immune to reference-after-eviction races, discussed in Section 3.2.3.

### 2.2.3. Header Blocks, Fragments, Frames, Packets...

_Note: this section describes mechanisms for optimal coordination compression with packetization. It remains an open issue whether such coordination adds more complexity than is worthwhile._

As with other aspects of QUIC, QCRAM aims to leverage opportunities for tighter integration between layers, in ways that may not have been practical in HTTP/2 due to various forms of ossification. The two specific instance of this are coordination of framing with packet generation, as described in the following paragraph, and use of

transport acknowledgments to reason about encoder-decoder state synchronization, which will be described in Section 4.

QCRAM header compression framing differs slightly from HTTP/2. Section 4.3 of [RFC7540] declares that:

> Header lists are collections of zero or more header fields. When transmitted over a connection, a header list is serialized into a header block using HTTP header compression [RFC7541]. The serialized header block is then divided into one or more octet sequences, called header block fragments, and transmitted within the payload of HEADERS (Section 6.2), PUSH_PROMISE (Section 6.6), or CONTINUATION (Section 6.10) frames.

Where RFC 7540 suggests that HPACK serialize a complete header list into a single header block, QCRAM header encoding MAY be progressive: compression of a Header List happens iteratively, where each iteration produces a single Header Block Fragment constrained to fit within the space available in the current transport packet. _Each iteration informs the progressive HPACK encoder of available space and the encoder generates only as many HPACK representations as fit_. The resulting header block fragment is encapsulated by an HTTP mapping headers frame (HEADERS or PUSH_PROMISE or CONTINUATION), and the headers frame will be encapsulated by a QUIC transport-level STREAM frame. This combined with the logic of Section 3.2.2, favors compression within a packet and avoids vulnerability to HoL blocking between packets concurrently in flight.

## 3. HPACK extensions

### 3.1. HPACK fallback

In QCRAM, the HPACK encoder interface needs to support the HoL-free HPACK fallback, causing it to operate in mode that disables references to the dynamic table.

### 3.2. QCRAM

The encoder interface should allow writes and reads returned in separate destinations, to be written to the Connection Control Stream and the HTTP message exchanges streams respectively. For the writes, the encoder should emit only Literal Header Fields with Incremental Indexing, and skip other entries. For the reads, the complete header field set should be represented. For each of the write entries, there will be a corresponding indexed entry in the reads.

The interfaces should also specify whether HPACK fallback is desired. If so, then if an entry is found to be vulnerable to HoL blocking

(see Section 3.2.2), the QCRAM encode should be aborted, and the block re-encoded using an HPACK fallback mode.

Finally, if progressive encoding is supported, the encoder interface should allow iteration, with the number of bytes available in the current packet specified at each iteration.

### 3.2.1. Indexing

The read data should be prefixed by the absolute index to allow out of order processing. The prefix is a single HPACK integer (8-bit prefix) that encodes the value of the base index, defined as the total number of entries that had been inserted to the dynamic table thus far.

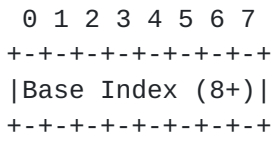


Figure 1: Base Index

### 3.2.2. HoL blocking logic

QCRAM adds three integer _epochs_ to HPACK state, provided by the HTTP mapping layer:

1. "encode_epoch": the count of header block fragments encoded thus far. When entries are added to they dynamic table, the current encode epoch is stored with the entry.

2. "packet_epoch": the first encode epoch in the current QUIC packet.

3. "commit_epoch": the highest in-order encode epoch acknowledged to the encoder side.

The following must hold: "encode_epoch >= packet_epoch > commit_epoch". Section 3.2 describes how the epoch values are computed.

QCRAM encode conceptually works by first generating the writes, and then processing the header list again to generate the reads. For reads, the encoder will emit an indexed representation only if it is not vulnerable to HoL blocking, that is if there is a matching entry in the dynamic table such that: "entry.encode_epoch <= commit_epoch or entry.encode_epoch >= packet_epoch". If not, it either generate a incremental indexed literal, or abort the encode if HPACK fallback is

enabled. For aborted encodes, the HTTP mapping should re-encode using one of the HPACK fallback modes.

### 3.2.3. Table evictions

Since QCRAM allows headers to be processed out of order, it is possible that a header block fragment may contain references to entries that have been evicted by the time it arrives. The decoder can detect this because the absolute index of the most recent eviction is known. If the decoder does not have the referenced entry it MUST reset the stream with "HTTP_HPACK_EVICTION_TOO_LATE". See Section 5.2 for performance considerations.

### 3.2.4. Manditory De-duplication for equivilant entries

HPACK allows duplicate table entries, that is entries that have the same name and value. QCRAM's HoL blocking logic could lead to a large number of duplicates. For example, if many headers are sent in the same round trip that all contain the same new header field (potentially large), QCRAM's logic may result in many duplicate table insertions leading to a form of table explosion. Contrast with HPACK where all but the first entry would have been (non-incremental) indexed representations. To help mitigate such cases, HPACK for QCRAM is required to de-duplicate strings in the dynamic table. The table insertion logic should check if the new entry matches any existing entries (name and value), and if so, table accounting MUST charge only the overhead portion ([RFC7541] Section 4.1) to the new entry. De-duplication is left as an exercise of the implementation, but using reference counted pointers to strings in table entries would be typical. De-duplication MUST be used for QCRAM and HPACK fallback modes.

## 4. HTTP Mapping changes

This draft assumes the HTTP mapping uses a single QUIC stream per HTTP message exchange. As described above, some header data will be sent on the Connection Control Stream, and other on the HTTP message exchange streams.

Header frames will be sent on the Connection Control Stream for:

1. Writes of QCRAM headers.

2. Speculative writes to the table (see Section 5.3).

3. The HTTP fallback mode that favors compression ratio. In this case, the header must indicate the stream-id to which the header

applies. _TBD: Describe wire format changes to add optional stream-id to headers frames._

Header frames will be sent on the HTTP message exchange stream for:

1. Reads of QCRAM headers.

2. The HTTP fallback mode that favors resilience to HoL blocking.

The HPACK encoder interface is extended for QCRAM. When encoding headers, the HTTP mapping indicates QCRAM or HPACK fallback. The HTTP mapping also provides the commit, packet, and encoding epoch. For QCRAM encodes, the interface includes separate outputs for the writes and reads, and a return value that indicates when a header should be re-encoded with HPACK fallback (see Section 2.2).

### 4.1. Encode epoch

"encode_epoch" increments for every new header block fragment encoded.

### 4.2. Commit epoch

"commit_epoch", the highest in-order acknowledged encode epoch. An encode epoch is considered acknowledged when all the bytes of the corresponding header frame have been acknowledged. The mapping layer keeps a _commit queue_, to track of header frames by their encode epochs, and monitors transport acknowledgments to determine when to advance "commit_epoch". This monitoring should only apply to the header frames sent on the Connection Control Stream. Header frames on the HTTP message exchange streams are read only with respect to the dynamic table, hence do not result in commits. The calculation of "commit_epoch" piggybacks on existing QUIC transport mechanisms, no corresponding wire format changes are needed.

### 4.3. Packet epoch

"packet_epoch" is the first encode epoch in the current QUIC packet. If packet coordination is not employed, this may be approximated. A simple approximation would be to set "packet_epoch = encode_epoch" for each header frame written to the Connection Control Stream.

## 5. Performance considerations

### 5.1. Memory footprint

The progressive compression regime described in Section 2.2.3 would change the memory footprint associated with header processing, potentially for better and worse:

- Delaying compression until there is space on the wire might necessitate some form of additional copying and buffering.
- Delaying compression might mean that uncompressed headers are retained longer and consume more memory.
- Progressive compression might save memory if accumulating the full header in the first place takes significant time, as might be the case with larger headers transiting proxy or front end servers.

### 5.2. Table evictions

HTTP_HPACK_EVICTION_TOO_LATE resets should be rare in practice, nevertheless there are strategies the might further reduce their likelihood. Encoder implementations could uses heuristics to assess vulnerabiltiy to such errors, and employ HPACK fallback to headers deemed at high risk. Decoder implemenations may choose to reserve extra table space, delaying evictions relative to the encoder.

### 5.3. Speculative table updates

Implementations can _speculatively_ send header frames on the HTTP Connection Control Stream. Such headers would not be associated with any HTTP transaction, but could be used strategically to improve performance. For instance, the encoder might decide to resend entries for the most popular header fields, to ensure they have the small indices and hence minimal size on the wire.

### 5.4. Eliminating sequence numbers

Due to the hybrid indexing scheme, and the HPACK fallbacks, the sequence numbers currently defined by the HTTP Mapping in the wire format of "HEADERS" and "PUSH_PROMISE" frames are unnecessary with QCRAM.

### 5.5. Fixed overhead.

HPACK defines overhead as 32 bytes ([RFC7541] Section 4.1). QCRAM adds the encode epoch per table entry, and requires mechanisms to de-duplicate strings. A larger value than 32 might be more accurate for QCRAM.

## 6. Security Considerations

TBD.

## 7. IANA Considerations

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

## 8. Acknowledgments

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

- Mike Bishop
- Patrick McManus
- Biren Roy
- Alan Frindell
- Ian Swett

## 9. Normative References

[QUIC-HTTP] Bishop, M., Ed., "Hypertext Transfer Protocol (HTTP) over QUIC", July 2017.

[QUIC-TRANSPORT] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", July 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, <http://www.rfc-editor.org/info/rfc7540>.

[RFC7541] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <http://www.rfc-editor.org/info/rfc7541>.

## Author's Address

Charles 'Buck' Krasic
Google

Email: ckrasic@google.com