QUIC
Internet-Draft
Intended status: Standards Track
Expires: September 28, 2017

C. Krasic
Google

March 27, 2017

Header Compression for HTTP over QUIC
draft-krasic-quic-qcram-00

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 September 28, 2017.

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 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.

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. For full performance, QCRAM requires QUIC specific mechanisms that leverage tight integration between transport and HTTP layers, as will be described in Section 2.2.1.

2.1. Example of HoL blocking

The following is an example of how HPACK can induce HoL blocking in QUIC. Assume two message control 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. If QCRAM is enabled, "HB[i]" will be represented using an HPACK literal. Otherwise an indexed representation may be used, but HB must processed in-order, after HA.

It is worth noting that rules 1. and 2. are situations where "HB" is not at risk of HoL blocking, even without QCRAM. Only in rule 3 does QCRAM come into play giving the encoder the choice between HoL avoidance or better compression.

2.2.1. Header Blocks, Fragments, Frames, Packets...

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 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.

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 3.2.

QCRAM header compression SHOULD 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), and the headers frame will be encapsulated by a QUIC transport-level STREAM frame. An implementation that can not support such coordination MUST forego references allowed by rule 2 of the previous section.

2.2.2. Absolute Indexing

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 uses a hybrid absolute-relative indexing approach. Every QCRAM header block fragment starts with an integer that conveys an absolute base index. The format of individual indexed representations does not change, but their semantics become absolute in combination with the base index. Similarly, the base index is used to perform table insertions at unambiguous positions.

3. Changes to HPACK and HTTP over QUIC

QCRAM is optional on a per header frame basis. QCRAM enabled header frames can be decoded on receipt, otherwise the header frame should be processed in strict order as per Section 4.2.1 of the HTTP mapping.

3.1. HPACK changes

QCRAM adds three integer _epochs_ to HPACK state, all derived from

the sequence numbers of HTTP Mapping (refer to [QUIC-HTTP] Sections 5.2.2 and 5.2.4.), and provided to the HPACK layer by the HTTP mapping:

1. "encode_epoch": the sequence number of the header frame enclosing the header block fragment, as per the HTTP Mapping. 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. When multiple header frames are packed into a single QUIC packet, they should be ordered.

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 ho the epoch values are computed.

3.1.1. Indexed representations

As each header block fragment is processed, HPACK is informed whether QCRAM is enabled. If so, 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". Otherwise a literal must be used.

3.1.2. Indexing

Every QCRAM header block fragment must start with a single HPACK integer that encodes the value of the base index, defined as the total number of entries that had been inserted to the dynamic table before encoding the current header block. As described above, the decoder will use this as the starting point for insertions, and for interpreting indexed representations.

3.1.3. Table evictions

Since QCRAM allows headers to be processed out of order, it might be

possible that a header block fragment may contain references to entries that have been evicted by the time it arrives. For example, suppose HB was encoded after HA, and HB evicts an entry referenced by HA. If due to network drops HB is decoded first, the reference in HA will become invalid.

To handle this with minimal complexity, QCRAM takes the following approach: if "packet_epoch > commit_epoch + 1", and if while encoding the current header block fragment an eviction becomes necessary, then QCRAM must be disabled for the current header frame. The first condition might be paraphrased as: are there any header block packets still in flight before the current one?

In the above example, HB would not be QCRAM enabled, hence the decoder must ensure to process HB strictly after HA.

*Compared to other QUIC state such as receive buffers, the default table size of 4,096 octets (see [RFC7540] Section 6.5.2.) is very modest. Deployment data suggests it is rarely increased in practice, and experiments to increase it did not yield significant gains. Consequently, I think it's best to avoid any heroic measures to deal with performance under full tables. *

3.2. HTTP Mapping changes

An additional flag is added to HEADERS and PUSH_PROMISE (refer to Sections 5.2.1. and 5.2.4. of [QUIC-HTTP]):

QCRAM (0x8): This header block fragment can be decoded upon receipt.

When encoding headers, the HTTP mapping layer notifies the HPACK layer whether QCRAM is set, and provides the commit, packet, and encoding epochs:

- the encoding epoch increments for every new header block fragment encoded.
- an encode epoch is considered acknowledged when all the bytes of the corresponding header frame have been acknowledged. The mapping layer keeps track of header frames by their encode epochs, and monitors transport acknowledgments to determine "commit_epoch", the highest in-order acknowledged encode epoch.

This piggybacks on existing QUIC transport mechanisms, no additional wire format changes are needed.

- the mapping layer coordinates with packet writing to manage space available for header frames, and advances the packet epoch at packet boundaries. Implementations that forgo coordinated packetization MUST set "packet_epoch" equal to "encode_epoch".

4. Performance considerations

Beyond sequence numbers already defined in Section 5.2.1 and 5.2.4, the only additional overhead of QCRAM is the base index added to header blocks. For a typical connection with fewer than 256 requests, the index would consume approximately 1 byte per header block.

It might be advantageous to allow implementations to send header frames on the HTTP control stream (QUIC stream 3). Such headers would not be associated with any HTTP transaction, but could be used strategically to improve performance. For instance, as a means to avoid disabling QCRAM due to table eviction, or to ensure most frequently used entries have the smallest indices.

For QCRAM header frames, the base index is sufficient to decode correctly. If QCRAM were made mandatory rather than optional, then it would be feasible to remove sequence number from wire format of "HEADERS" and "PUSH_PROMISE" frames, as well as the QCRAM flag. However, this would imply that once the table became full, insertions could only occur during during periods with a single header block in flight.

Alternatively, if it were desirable to support a middle ground between totally ordered HPACK and the present draft, one way might be to extend the concept of packet epoch to denote a sequence of one _or more_ packets. A pair of new flags would be added to header frames to signal the start and end of such packet sequences. The decoder would have to have buffering based logic to ensure header blocks within a packet sequence are processed in order, similar to the logic used in totally ordered HPACK.

5. Security Considerations

TBD.

6. IANA Considerations

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

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:

- Mike Bishop
- Patrick McManus
- Biren Roy

8. Normative References

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

[QUIC-TRANSPORT] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport".

[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