QUIC
Internet-Draft
Intended status: Standards Track
Expires: July 7, 2018

C. Krasic
Google

January 3, 2018

Header Compression for HTTP over QUIC
draft-krasic-quic-qcram-03

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 it 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 if 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 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 7, 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 [RFC7541] Section 1.3 to review HPACK terminology, and [QUIC-HTTP], Sections 4 on "HTTP over QUIC stream mapping" and 4.2.1 on "Header Compression". QCRAM extensions to HPACK allow correctness in the presence of out-of-order delivery, with flexibility to balance between resilience against HoL blocking and compression ratio.

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 over QUIC.

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 minimizes HoL blocking

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

1. "HB[i]" will not introduce HoL blocking if "HA" has been acknowledged, otherwise it is vulnerable. A new HQ frame type HEADERS_ACK is defined (see Section 3). When the decoder has

processed a header block, HEADERS_ACK is sent from the decoder back to the encoder.

The encoder can choose on a per header block basis whether to favor higher compression ratio or HoL resilience, signaled by the BLOCKING flag in HEADERS and PUSH_PROMISE frames (see Section 3).

If HB contains no vulnerable header fields, BLOCKING MUST be 0.

If BLOCKING is not set, then for each "HB[i]" that is vulnerable:

1. "HB[i]" is represented with one of the Literal variants (see [RFC7541] Section 6.2), trading lower compression ratio for HoL resilience.

If BLOCKING is set then HB is encoded in blocking mode:

1. "HB[i]" is represented with an Indexed Representation. This favors compression ratio.

In blocking mode, after reading HB's prefix stream B might block. Stream B proceeds with reading and processing the rest of HB only once all HB's dependencies are satisfied. The header prefix contains table offset information that establishes total ordering among all headers, regardless of reordering in the transport (see Section 4.1). In blocking mode, the prefix additionally identifies the largest (absolute) index I that HB depends on (see "Depends" in Section Section 4.2). HB's dependencies are satisfied when all entries less than or equal to I have been inserted into the table. Notice that while blocked, HB's header field data remains in stream B's flow control window.

3. HTTP over QUIC mapping extensions

3.1. HEADERS and PUSH_PROMISE

HEADER 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 header can always be processed immediately upon receipt.

3.2. HEADERS_ACK

The HEADERS_ACK frame (type=0x8) is sent by the decoder side to the encoder when a 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 HEADERS_ACK frame does not define any flags, and has no payload.

4. HPACK extensions

4.1. Header Block Prefix

In HEADERS and PUSH_PROMISE frames, HPACK Header data are prefixed by a pair of integers pair of integers: "Fill" and the "Evictions". "Fill" is the number of entries in the table, and "Evictions" is the cumulative number entries that have been evicted from the table. Their sum is the cumulative number of entries inserted before the following header block was encoded. Each is encoded as a single HPACK integer (8-bit prefix):

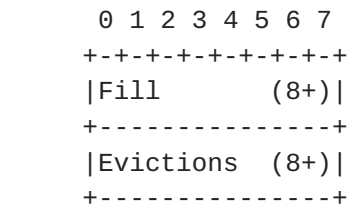

Figure 1: Absolute indexing (BLOCKING=0x0)

Section 4.2 describes the role of "Fill" and Section 4.3 covers the role of "Evictions".

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

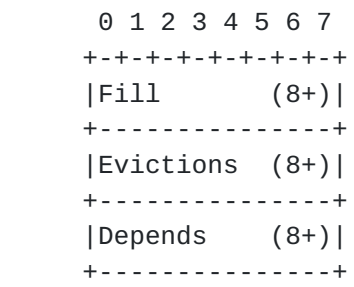

Figure 2: Absolute indexing (BLOCKING=0x1)

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 2.2. The largest entry index is "Evictions + Fill - Depends".

4.2. Hybrid absolute-relative indexing

HPACK indexed entries refer to an entry by its current position in the dynamic table. As Figure 1 of RFC7541 [1] 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). Implicit index updates are 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. The prefix defined in Section 4.1 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 stream 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.

4.3. 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 is already evicted. QCRAM employs a two-phase eviction algorithm, in which the encoder will not evict entries that have outstanding (unacknowledged) references. The QCRAM encoder maintains a counter as entries are evicted, which is the cumulative number of evictions so far, "Evictions" (Section 4.1). On arrival at the decoder, if "Evictions" is higher than previously seen, the decoder MUST evict all entries at or below. Unlike HPACK where the decoder follows the same logic as the encoder to perform evictions, in QCRAM the decoder evicts exclusively based on the encoder's explicit guidance.

4.3.1. Blocked Evictions

In some cases, the encoder must forgo eviction by selecting a literal representation (blocked eviction), namely in the event that the entry subject to eviction _is_ referenced by one or more unacknowledged header frames. To assure that the blocked eviction case is rare, a form of thresholding MAY be applied that constrains selection of Indexed representations, such that the oldest entries in the dynamic table will largely be evictable. The constraint is applied when encoding header fields: comparing the cumulative position (in bytes) of the matching entry to a threshold, categorizing oldest entries (past threshold) as at-risk. Avoiding references to at-risk entries,

the encoder SHOULD use an Indexed-Duplicate representation instead (see Section 4.5).

4.4. Handling Stream Resets

The QCRAM encoder has the option to select representations that might require blocking (Section 2.2 case 3), but the decoder must be prevented from becoming hung if the stream associated with the referenced entry is reset. On stream reset, the QCRAM encoder MUST check if the stream has unacknowledged headers, and if so resend them on the Control Stream ([QUIC-HTTP] Section 4.1). If header blocks are resent on the control stream, duplicate arrivals are possible due to reset-acknowledgment races. The decoder MUST ignore duplicate header block arrivals, which is straightforward because of unambiguous indexing (see Section 4.2).

4.5. Refreshing Entries with Duplication

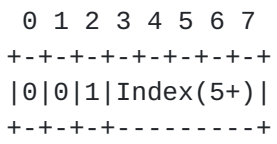

Figure 3: Indexed Header Field with Duplication

Indexed-Duplicates are treated as an Indexed Header Field Representation (see [RFC7541] Section 6.1), additionally inserting a new duplicate entry. [RFC7541] allows duplicate HPACK table entries, that is entries that have the same name and value.

Figure 2 annexes the representation for HPACK Dynamic Table Size Update (see Section 6.3 of RFC7541), which is not supported by HTTP over QUIC.

4.5.1. Mandatory Entry De-duplication

To help mitigate memory consumption due to duplicate entries, 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.

Specific de-duplication mechanisms are left to implementations, but using a map in conjunction with reference counted pointers to strings would be typical.

5. Performance considerations

5.1. 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 _refresh_ by sending Indexed-Duplicate representations for popular header fields (Section 4.1), ensuring they have small indices and hence minimal size on the wire.

5.2. 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, and requires mechanisms to de-duplicate strings. A larger value than 32 might be more accurate for QCRAM.

5.3. Co-ordinated Packetization

In Section 2.2, an exception exists when the representation of "HA[i]" and "HB[j]" are delivered within the same transport packet. If so, there is no risk of HoL blocking and using an indexed representation is strictly better than using a literal. An implementation could exploit this exception by employing co-ordination between QCRAM compression and QUIC transport packetization.

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
- Ryan Hamilton

9. References

9.1. Normative References

[QUIC-HTTP] Bishop, M., Ed., "Hypertext Transfer Protocol (HTTP) over QUIC", January 2018.

[QUIC-TRANSPORT] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", January 2018.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

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

9.2. URIs

[1] https://tools.ietf.org/html/rfc7541#section-2.3.3

Author's Address

Charles 'Buck' Krasic
Google

Email: ckrasic@google.com