                        **Cache Digests for HTTP/2**
                     **draft-ietf-httpbis-cache-digest-00**

Abstract

   This specification defines a HTTP/2 frame type to allow clients to
   inform the server of their cache's contents.  Servers can then use
   this to inform their choices of what to push to clients.

Note to Readers

   Discussion of this draft takes place on the HTTP working group
   mailing list (ietf-http-wg@w3.org), which is archived at
   https://lists.w3.org/Archives/Public/ietf-http-wg/ .

   Working Group information can be found at http://httpwg.github.io/ ;
   source code and issues list for this draft can be found at
   https://github.com/httpwg/http-extensions/labels/cache-digest .

Status of This Memo

Copyright Notice

Table of Contents

## 1.  Introduction

HTTP/2 [RFC7540] allows a server to "push" synthetic request/response
pairs into a client's cache optimistically.  While there is strong
interest in using this facility to improve perceived Web browsing
performance, it is sometimes counterproductive because the client
might already have cached the "pushed" response.

When this is the case, the bandwidth used to "push" the response is
effectively wasted, and represents opportunity cost, because it could
be used by other, more relevant responses.  HTTP/2 allows a stream to
be cancelled by a client using a RST_STREAM frame in this situation,
but there is still at least one round trip of potentially wasted
capacity even then.

This specification defines a HTTP/2 frame type to allow clients to
inform the server of their cache's contents using a Golumb-Rice Coded
Set. Servers can then use this to inform their choices of what to
push to clients.

## 1.1.  Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in [RFC2119].

## 2.  The CACHE_DIGEST Frame

The CACHE_DIGEST frame type is 0xf1.  NOTE: This is an experimental
value; if standardised, a permanent value will be assigned.

```
+------------------------------------------------+
|                Digest-Value? (\*)          ...
+------------------------------------------------+
```

The CACHE_DIGEST frame payload has the following fields:

o  *Digest-Value*: A sequence of octets containing the digest as
   computed in Section 2.1.1.

The CACHE_DIGEST frame defines the following flags:

o  *RESET* (0x1): When set, indicates that any and all cache digests
   for the applicable origin held by the recipient MUST be considered
   invalid.

o  *COMPLETE* (0x2): When set, indicates that the currently valid set
   of cache digests held by the server constitutes a complete
   representation of the cache's state regarding that origin, for the
   type of cached response indicated by the "STALE" flag.

o  *VALIDATORS* (0x4): When set, indicates that the "validators"
   boolean in Section 2.1.1 is true.

o  *STALE* (0x8): When set, indicates that all cached responses
   represented in the digest-value are stale [RFC7234] at the point
   in them that the digest was generated; otherwise, all are fresh.

## 2.1.  Client Behavior

A CACHE_DIGEST frame can be sent from a client to a server on any
stream in the "open" state, and conveys a digest of the contents of
the client's cache for associated stream.

In typical use, a client will send one or more CACHE_DIGESTs
immediately after the first request on a connection for a given
origin, on the same stream, because there is usually a short period

of inactivity then, and servers can benefit most when they understand
the state of the cache before they begin pushing associated assets
(e.g., CSS, JavaScript and images).  Clients MAY send CACHE_DIGEST at
other times.

If the cache's state is cleared, lost, or the client otherwise wishes
the server to stop using previously sent CACHE_DIGESTs, it can send a
CACHE_DIGEST with the RESET flag set.

When generating CACHE_DIGEST, a client MUST NOT include cached
responses whose URLs do not share origins [RFC6454] with the request
of the stream that the frame is sent upon.

CACHE_DIGEST allows the client to indicate whether the set of URLs
used to compute the digest represent fresh or stale stored responses,
using the STALE flag.  Clients MAY decide whether to only sent
CACHE_DIGEST frames representing their fresh stored responses, their
stale stored responses, or both.

Clients can choose to only send a subset of the suitable stored
responses of each type (fresh or stale).  However, when the
CACHE_DIGEST frames sent represent the complete set of stored
responses of a given type, the last such frame SHOULD have a COMPLETE
flag set, to indicate to the server that it has all relevant state of
that type.  Note that for the purposes of COMPLETE, responses cached
since the beginning of the connection or the last RESET flag on a
CACHE_DIGEST frame need not be included.

CACHE_DIGEST can be computed to include cached responses' ETags, as
indicated by the VALIDATORS flag.  This information can be used by
servers to decide what kinds of responses to push to clients; for
example, a stale response that hasn't changed could be refreshed with
a 304 (Not Modified) response; one that has changed can be replaced
with a 200 (OK) response, whether the cached response was fresh or
stale.

CACHE_DIGEST has no defined meaning when sent from servers, and
SHOULD be ignored by clients.

### 2.1.1.  Computing the Digest-Value

Given the following inputs:

o  "validators", a boolean indicating whether validators ([RFC7232])
   are to be included in the digest;

o  "URLs'", an array of (string "URL", string "ETag") tuples, each
   corresponding to the Effective Request URI ([RFC7230],

Section 5.5) of a cached response [RFC7234] and its entity-tag
[RFC7232] (if "validators" is true and if the ETag is available;
otherwise, null);

o  "P", an integer that MUST be a power of 2 smaller than 2**32, that
   indicates the probability of a false positive that is acceptable,
   expressed as "1/P".

"digest-value" can be computed using the following algorithm:

1.   Let N be the count of "URLs"' members, rounded to the nearest
     power of 2 smaller than 2**32.

2.   Let "hash-values" be an empty array of integers.

3.   For each ("URL", "ETag") in "URLs", compute a hash value
     (Section 2.1.2) and append the result to "hash-values".

4.   Sort "hash-values" in ascending order.

5.   Let "digest-value" be an empty array of bits.

6.   Write log base 2 of "N" to "digest-value" using 5 bits.

7.   Write log base 2 of "P" to "digest-value" using 5 bits.

8.   Let "C" be -1.

9.   For each "V" in "hash-values":

     1.  If "V" is equal to "C", continue to the next "V".

     2.  Let "D" be the result of "V - C - 1".

     3.  Let "Q" be the integer result of "D / P".

     4.  Let "R" be the result of "D modulo P".

     5.  Write "Q" '0' bits to "digest-value".

     6.  Write 1 '1' bit to "digest-value".

     7.  Write "R" to "digest-value" as binary, using log2("P") bits.

     8.  Let "C" be "V"

10.  If the length of "digest-value" is not a multiple of 8, pad it
     with 0s until it is.

**2.1.2**.  **Computing a Hash Value**

   Given:

   o  "URL", an array of characters

   o  "ETag", an array of characters

   o  "validators", a boolean

   o  "N", an integer

   o  "P", an integer

   "hash-value" can be computed using the following algorithm:

   1.  Let "key" be "URL" converted to an ASCII string by percent-
       encoding as appropriate [RFC3986].

   2.  If "validators" is true and "ETag" is not null:

       1.  Append "ETag" to "key" as an ASCII string, including both the
           "weak" indicator (if present) and double quotes, as per
           [RFC7232] Section 2.3.

   3.  Let "hash-value" be the SHA-256 message digest [RFC6234] of
       "key", expressed as an integer.

   4.  Truncate "hash-value" to log2( "N" * "P" ) bits.

**2.2**.  **Server Behavior**

   In typical use, a server will query (as per Section 2.2.1) the
   CACHE_DIGESTs received on a given connection to inform what it pushes
   to that client;

   o  If a given URL has a match in a current CACHE_DIGEST with the
      STALE flag unset, it need not be pushed, because it is fresh in
      cache;

   o  If a given URL and ETag combination has a match in a current
      CACHE_DIGEST with the STALE flag set, the client has a stale copy
      in cache, and a validating response can be pushed;

   o  If a given URL has no match in any current CACHE_DIGEST, the
      client does not have a cached copy, and a complete response can be
      pushed.

Servers MAY use all CACHE_DIGESTs received for a given origin as
current, as long as they do not have the RESET flag set; a
CACHE_DIGEST frame with the RESET flag set MUST clear any previously
stored CACHE_DIGESTs for its origin.  Servers MUST treat an empty
Digest-Value with a RESET flag set as effectively clearing all stored
digests for that origin.

Clients are not likely to send updates to CACHE_DIGEST over the
lifetime of a connection; it is expected that servers will separately
track what cacheable responses have been sent previously on the same
connection, using that knowledge in conjunction with that provided by
CACHE_DIGEST.

### 2.2.1.  Querying the Digest for a Value

Given:

o  "digest-value", an array of bits

o  "URL", an array of characters

o  "ETag", an array of characters

o  "validators", a boolean

we can determine whether there is a match in the digest using the
following algorithm:

1.   Read the first 5 bits of "digest-value" as an integer; let "N"
     be two raised to the power of that value.

2.   Read the next 5 bits of "digest-value" as an integer; let "P" be
     two raised to the power of that value.

3.   Let "hash-value" be the result of computing a hash value
     (Section 2.1.2).

4.   Let "C" be -1.

5.   Read '0' bits from "digest-value" until a '1' bit is found; let
     "Q" bit the number of '0' bits.  Discard the '1'.

6.   Read log2("P") bits from "digest-value" after the '1' as an
     integer; let "R" be its value.

7.   Let "D" be "Q" * "P" + "R".

8.   Increment "C" by "D" + 1.

9.   If "C" is equal to "hash-value", return 'true'.

10.  Otherwise, return to step 5 and continue processing; if no match
     is found before "digest-value" is exhausted, return 'false'.

## 3.  IANA Considerations

This draft currently has no requirements for IANA.  If the
CACHE_DIGEST frame is standardised, it will need to be assigned a
frame type.

## 4.  Security Considerations

The contents of a User Agent's cache can be used to re-identify or
"fingerprint" the user over time, even when other identifiers (e.g.,
Cookies [RFC6265]) are cleared.

CACHE_DIGEST allows such cache-based fingerprinting to become
passive, since it allows the server to discover the state of the
client's cache without any visible change in server behaviour.

As a result, clients MUST mitigate for this threat when the user
attempts to remove identifiers (e.g., "clearing cookies").  This
could be achieved in a number of ways; for example: by clearing the
cache, by changing one or both of N and P, or by adding new,
synthetic entries to the digest to change its contents.

TODO: discuss how effective the suggested mitigations actually would
be.

Additionally, User Agents SHOULD NOT send CACHE_DIGEST when in
"privacy mode."

## 5.  References

## 5.1.  Normative References

[RFC2119]   Bradner, S., "Key words for use in RFCs to Indicate
            Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/
            RFC2119, March 1997,
            <http://www.rfc-editor.org/info/rfc2119>.

[RFC3986]   Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
            Resource Identifier (URI): Generic Syntax", STD 66, RFC
            3986, DOI 10.17487/RFC3986, January 2005,
            <http://www.rfc-editor.org/info/rfc3986>.

   [RFC6234]  Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms
              (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI
              10.17487/RFC6234, May 2011,
              <http://www.rfc-editor.org/info/rfc6234>.

   [RFC6454]  Barth, A., "The Web Origin Concept", RFC 6454, DOI
              10.17487/RFC6454, December 2011,
              <http://www.rfc-editor.org/info/rfc6454>.

   [RFC7230]  Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
              Protocol (HTTP/1.1): Message Syntax and Routing", RFC
              7230, DOI 10.17487/RFC7230, June 2014,
              <http://www.rfc-editor.org/info/rfc7230>.

   [RFC7232]  Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer
              Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI
              10.17487/RFC7232, June 2014,
              <http://www.rfc-editor.org/info/rfc7232>.

   [RFC7234]  Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke,
              Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching",
              RFC 7234, DOI 10.17487/RFC7234, June 2014,
              <http://www.rfc-editor.org/info/rfc7234>.

   [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>.

## 5.2.  Informative References

   [RFC6265]  Barth, A., "HTTP State Management Mechanism", RFC 6265,
              DOI 10.17487/RFC6265, April 2011,
              <http://www.rfc-editor.org/info/rfc6265>.

## Appendix A.  Acknowledgements

   Thanks to Adam Langley and Giovanni Bajo for their explorations of
   Golumb-coded sets.  In particular, see
   http://giovanni.bajo.it/post/47119962313/golomb-coded-sets-smaller-
   than-bloom-filters , which refers to sample code.

   Thanks to Stefan Eissing for his suggestions.

Authors' Addresses

    Kazuho Oku
    DeNA Co, Ltd.

    Email: kazuhooku@gmail.com


    Mark Nottingham

    Email: mnot@mnot.net
    URI:    https://www.mnot.net/