HTTP Working Group
Internet-Draft
Intended status: Experimental
Expires: December 1, 2017

K. Oku
DeNA Co, Ltd.
M. Nottingham

May 30, 2017

Cache Digests for HTTP/2
draft-ietf-httpbis-cache-digest-02

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

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 December 1, 2017.

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.

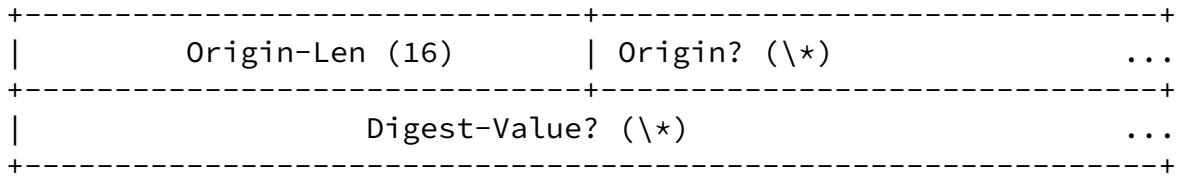
This specification defines a HTTP/2 frame type to allow clients to inform the server of their cache's contents using a Golomb-Rice Coded Set [Rice]. Servers can then use this to inform their choices of what to push to clients.

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 0xd (decimal 13).



The CACHE_DIGEST frame payload has the following fields:

Origin-Len: An unsigned, 16-bit integer indicating the length, in octets, of the Origin field.

Origin: A sequence of characters containing the ASCII serialization of an origin ([RFC6454], Section 6.2) that the Digest-Value applies to.

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.

- *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.

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

- *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 MUST be sent from a client to a server on stream 0, and conveys a digest of the contents of the client's cache for the indicated origin.

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 indicated origin. Clients MUST NOT send CACHE_DIGEST frames on connections that are not authoritative (as defined in [RFC7540], 10.1) for the indicated origin.

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:

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

- "URL", an array of characters
- "ETag", an array of characters
- "validators", a boolean
- "N", an integer
- "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;

- 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;

- 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;

- 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.

Servers MUST ignore CACHE_DIGEST frames sent on a stream other than 0.

2.2.1. Querying the Digest for a Value

Given:

- "digest-value", an array of bits

- "URL", an array of characters

- "ETag", an array of characters

- "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" be 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. The ACCEPT_CACHE_DIGEST SETTINGS Parameter

A server can notify its support for CACHE_DIGEST frame by sending the ACCEPT_CACHE_DIGEST (0x7) SETTINGS parameter. If the server is tempted to making optimizations based on CACHE_DIGEST frames, it SHOULD send the SETTINGS parameter immediately after the connection is established.

The value of the parameter is a bit-field of which the following bits are defined:

FRESH (0x1): When set, it indicates that the server is willing to make use of a digest of freshly-cached responses.

STALE (0x2): When set, it indicates that the server is willing to make use of a digest of stale-cached responses.

Rest of the bits MUST be ignored and MUST be left unset when sending.

The initial value of the parameter is zero (0x0) meaning that the server is not interested in seeing a CACHE_DIGEST frame.

Some underlying transports allow the server's first flight of application data to reach the client at around the same time when the client sends it's first flight data. When such transport (e.g., TLS 1.3 [I-D.ietf-tls-tls13] in full-handshake mode) is used, a client can postpone sending the CACHE_DIGEST frame until it receives a ACCEPT_CACHE_DIGEST settings value.

When the underlying transport does not have such property (e.g., TLS 1.3 in 0-RTT mode), a client can reuse the settings value found in previous connections to that origin [RFC6454] to make assumptions.

4. IANA Considerations

This document registers the following entry in the Permanent Message Headers Registry, as per [RFC3864]:

- Header field name: Cache-Digest
- Applicable protocol: http
- Status: experimental
- Author/Change controller: IESG
- Specification document(s): [this document]

This document registers the following entry in the HTTP/2 Frame Type Registry, as per [RFC7540]:

- Frame Type: CACHE_DIGEST
- Code: 0xd
- Specification: [this document]

This document registers the following entry in the HTTP/2 Settings Registry, as per [RFC7540]:

- Code: 0x7
- Name: ACCEPT_CACHE_DIGEST
- Initial Value: 0x0
- Reference: [this document]

5. 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."

6. References

6.1. 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>.

6.2. Informative References

[Fetch] "Fetch Standard", n.d., <https://fetch.spec.whatwg.org/>.

[I-D.ietf-tls-tls13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-20 (work in progress), April 2017.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004,

<http://www.rfc-editor.org/info/rfc3864>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

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

[Rice] Rice, R. and J. Plaunt, "Adaptive variable-length coding for efficient compression of spacecraft television data", IEEE Transactions on Communication Technology 19.6 , 1971.

[Service-Workers] Russell, A., Song, J., Archibald, J., and M. Kruisselbrink, "Service Workers 1", October 2016, <https://www.w3.org/TR/2016/WD-service-workers-1/>.

Appendix A. Encoding the CACHE_DIGEST frame as an HTTP Header

On some web browsers that support Service Workers [Service-Workers] but not Cache Digests (yet), it is possible to achieve the benefit of using Cache Digests by emulating the frame using HTTP Headers.

For the sake of interoperability with such clients, this appendix defines how a CACHE_DIGEST frame can be encoded as an HTTP header named "Cache-Digest".

The definition uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] with the list rule extension defined in [RFC7230], Appendix B.

```
Cache-Digest  = 1#digest-entity
digest-entity = digest-value *(OWS ";" OWS digest-flag)
```

```
digest-value  = <Digest-Value encoded using base64url>
digest-flag   = token
```

A Cache-Digest request header is defined as a list construct of cache-digest-entities. Each cache-digest-entity corresponds to a CACHE_DIGEST frame.

Digest-Value is encoded using base64url [RFC4648], Section 5. Flags that are set are encoded as digest-flags by their names that are compared case-insensitively.

Origin is omitted in the header form. The value is implied from the value of the ":authority" pseudo header. Client MUST only send Cache-Digest headers containing digests that belong to the origin specified by the HTTP request.

The example below contains one digest of fresh resource and has only the "COMPLETE" flag set.

```
Cache-Digest: AfdA; complete
```

Clients MUST associate Cache-Digest headers to every HTTP request, since Fetch [Fetch] - the HTTP API supported by Service Workers -

does not define the order in which the issued requests will be sent to the server nor guarantees that all the requests will be transmitted using a single HTTP/2 connection.

Also, due to the fact that any header that is supplied to Fetch is required to be end-to-end, there is an ambiguity in what a Cache-Digest header respresents when a request is transmitted through a proxy. The header may represent the cache state of a client or that of a proxy, depending on how the proxy handles the header.

Appendix B. Acknowledgements

Thanks to Adam Langley and Giovanni Bajo for their explorations of Golomb-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.

Appendix C. Changes

C.1. Since draft-ietf-httpbis-cache-digest-01

- Added definition of the Cache-Digest header.
- Introduce ACCEPT_CACHE_DIGEST SETTINGS parameter.
- Change intended status from Standard to Experimental.

C.2. Since draft-ietf-httpbis-cache-digest-00

- Make the scope of a digest frame explicit and shift to stream 0.

Authors' Addresses

Kazuho Oku
DeNA Co, Ltd.

Email: kazuhooku@gmail.com

Mark Nottingham

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