Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 10, 2016

K. Oku
DeNA Co, Ltd.
M. Nottingham

January 7, 2016

Cache Digests for HTTP/2
draft-kazuho-h2-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

The issues list for this draft can be found at https://github.com/mnot/I-D/labels/h2-cache-digest .

The most recent (often, unpublished) draft is at https://mnot.github.io/I-D/h2-cache-digest/ .

Recent changes are listed at https://github.com/mnot/I-D/commits/gh-pages/h2-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.

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 cache associated with that stream, as explained in Section 2.1.

In typical use, a client will send CACHE_DIGEST 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, but servers ought not expect frequent updates; instead, if they wish to continue to utilise the digest, they will need update it with responses sent to that client on the connection.

Servers MUST NOT use any but the most recent CACHE_DIGEST for a given origin as current, and MUST treat an empty Digest-Value as effectively clearing all stored digests for that origin.

CACHE_DIGEST has no defined meaning when sent from servers to clients, and MAY be ignored.

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

The CACHE_DIGEST frame payload has the following fields:

- o Digest-Value: An optional sequence of octets containing the digest as computed in Section 2.1.

2.1. Computing the Digest-Value

The set of URLs that is used to compute Digest-Value MUST only include URLs that share origins [RFC6454] with the stream that CACHE_DIGEST is sent on, and they MUST be fresh [RFC7234].

A client MAY choose a subset of the available stored responses to include in the set. Additionally, it MUST choose a parameter, "P", that indicates the probability of a false positive it is willing to tolerate, expressed as "1/P".

"P" MUST be a power of 2.

To compute a digest-value for the set "URLs" and "P":

1. Let N be the count of "URLs"' members, rounded up to power of 2.

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

3. Append 0 to "hash-values".

4. For each "URL" in URLs, follow these steps:

    1. Convert "URL" to an ASCII string by percent-encoding as appropriate [RFC3986].

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

    3. Append "key" modulo ("N" * "P") to "hash-values".

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

6. Let "digest" be an empty array of bits.

7. Write log base 2 of "N" and "P" to "digest" as octets.

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

    1. Let "W" be the value following "V" in "hash-values".

    2. If "W" and "V" are equal, continue to the next "V".

    3. Let "D" be the result of "W - V - 1".

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

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

    6. Write "Q" '1' bits to "digest".

    7. Write 1 '0' bit to "digest".

    8. Write "R" to "digest" as binary, using log2(P) bits.

9. If "V" is the second-to-last member of "hash-values", stop iterating through "hash-values" and continue to the next step.

9. If the length of "digest" is not a multiple of 8, pad it with 1s until it is.

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>.

[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.

Authors' Addresses

Kazuho Oku
DeNA Co, Ltd.

Email: kazuhooku@gmail.com

Mark Nottingham

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