

HTTP Working Group
Internet-Draft
Intended status: Experimental
Expires: January 3, 2019

K. Oku
Fastly
Y. Weiss
Akamai
July 2, 2018

Cache Digests for HTTP/2
draft-ietf-httpbis-cache-digest-05

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 January 3, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Notational Conventions	3
2. The CACHE_DIGEST Frame	3
2.1. Client Behavior	4
2.1.1. Creating a digest	4
2.1.2. Adding a URL to the Digest-Value	5
2.1.3. Removing a URL to the Digest-Value	7
2.1.4. Computing a fingerprint value	8
2.1.5. Computing the key	9
2.1.6. Computing a Hash Value	9
2.1.7. Computing an Alternative Hash Value	9
2.2. Server Behavior	10
2.2.1. Querying the Digest for a Value	10
3. The SETTINGS_SENDING_CACHE_DIGEST SETTINGS Parameter	11
4. The SETTINGS_ACCEPT_CACHE_DIGEST SETTINGS Parameter	12
5. IANA Considerations	12
6. Security Considerations	13
7. References	13
7.1. Normative References	13
7.2. Informative References	14
Appendix A. Encoding the CACHE_DIGEST frame as an HTTP Header	15
Appendix B. Changes	16
B.1. Since draft-ietf-httpbis-cache-digest-04	16
B.2. Since draft-ietf-httpbis-cache-digest-03	16
B.3. Since draft-ietf-httpbis-cache-digest-02	16
B.4. Since draft-ietf-httpbis-cache-digest-01	16
B.5. Since draft-ietf-httpbis-cache-digest-00	17
Appendix C. Acknowledgements	17
Authors' Addresses	17

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 freshly cached contents using a Cuckoo-filter [Cuckoo] based digest. 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).

```

+-----+-----+
|          Origin-Len (16)          | Origin? (\*)          ...
+-----+-----+
|                      Digest-Value? (\*)                      ...
+-----+-----+
```

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 and Section 2.1.2.

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.

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 stale-cached responses or 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.

When the **CACHE_DIGEST** frames sent represent the complete set of stored responses, the last such frame **SHOULD** have a **COMPLETE** flag set, to indicate to the server that it has all relevant state. 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 has no defined meaning when sent from servers, and **SHOULD** be ignored by clients.

2.1.1. Creating a digest

Given the following inputs:

- o "P", an integer smaller than 256, that indicates the probability of a false positive that is acceptable, expressed as "1/2**P".
- o "N", an integer that represents the number of entries - a prime number smaller than 2**32

1. Let "f" be the number of bits per fingerprint, calculated as $P + 3$
2. Let "b" be the bucket size, defined as 4.
3. Let "allocated" be the closest power of 2 that is larger than "N".
4. Let "bytes" be $f * \text{"allocated"} * b / 8$ rounded up to the nearest integer
5. Add 5 to "bytes"
6. Allocate memory of "bytes" and set it to zero. Assign it to "digest-value".
7. Set the first byte to "P"
8. Set the second till fifth bytes to "N" in big endian form
9. Return the "digest-value".

Note: "allocated" is necessary due to the nature of the way Cuckoo filters are creating the secondary hash, by XORing the initial hash and the fingerprint's hash. The XOR operation means that secondary hash can pick an entry beyond the initial number of entries, up to the next power of 2. In order to avoid issues there, we allocate the table appropriately. For increased space efficiency, it is recommended that implementations pick a number of entries that's close to the next power of 2.

2.1.2. Adding a URL to the Digest-Value

Given the following inputs:

- o "URL" a string corresponding to the Effective Request URI ([RFC7230], Section 5.5) of a cached response [RFC7234]
 - o "maxcount" - max number of cuckoo hops
 - o "digest-value"
1. Let "f" be the value of the first byte of "digest-value".
 2. Let "b" be the bucket size, defined as 4.
 3. Let "N" be the value of the second to fifth bytes of "digest-value" in big endian form.

4. Let "key" be the return value of Section 2.1.5 with "URL" as input.
5. Let "h1" be the return value of Section 2.1.6 with "key" and "N" as inputs.
6. Let "dest_fingerprint" be the return value of Section 2.1.4 with "key" and "f" as inputs.
7. Let "h2" be the return value of Section 2.1.7 with "h1", "dest_fingerprint" and "N" as inputs.
8. Let "h" be either "h1" or "h2", picked in random.
9. While "maxcount" is larger than zero:
 1. Let "position_start" be $40 + "h" * "f" * "b"$.
 2. Let "position_end" be $"position_start" + "f" * "b"$.
 3. While "position_start" < "position_end":
 1. Let "bits" be "f" bits from "digest_value" starting at "position_start".
 2. If "bits" is all zeros, set "bits" to "dest_fingerprint" and terminate these steps.
 3. Add "f" to "position_start".
 4. Let "e" be a random number from 0 to "b".
 5. Subtract $"f" * ("b" - "e")$ from "position_start".
 6. Let "bits" be "f" bits from "digest_value" starting at "position_start".
 7. Let "fingerprint" be the value of bits, read as big endian.
 8. Set "bits" to "dest_fingerprint".
 9. Set "dest_fingerprint" to "fingerprint".
 10. Let "h" be Section 2.1.7 with "h", "dest_fingerprint" and "N" as inputs.
 11. Subtract 1 from "maxcount".

10. Subtract "f" from "position_start".
11. Let "fingerprint" be the "f" bits starting at "position_start".
12. Let "h1" be "h"
13. Subtract 1 from "maxcount".
14. If "maxcount" is zero, return an error.
15. Go to step 7.

2.1.3. Removing a URL to the Digest-Value

Given the following inputs:

- o "URL" a string corresponding to the Effective Request URI ([RFC7230], Section 5.5) of a cached response [RFC7234]
 - o "digest-value"
1. Let "f" be the value of the first byte of "digest-value".
 2. Let "b" be the bucket size, defined as 4.
 3. Let "N" be the value of the second to fifth bytes of "digest-value" in big endian form.
 4. Let "key" be the return value of Section 2.1.5 with "URL" as input.
 5. Let "h1" be the return value of Section 2.1.6 with "key" and "N" as inputs.
 6. Let "fingerprint" be the return value of Section 2.1.4 with "key" and "f" as inputs.
 7. Let "h2" be the return value of Section 2.1.7 with "h1", "fingerprint" and "N" as inputs.
 8. Let "hashes" be an array containing "h1" and "h2".
 9. For each "h" in "hashes":
 1. Let "position_start" be $40 + "h" * "f" * "b"$.
 2. Let "position_end" be $"position_start" + "f" * "b"$.

3. While "position_start" < "position_end":
 1. Let "bits" be "f" bits from "digest_value" starting at "position_start".
 2. If "bits" is "fingerprint", set "bits" to all zeros and terminate these steps.
 3. Add "f" to "position_start".

2.1.4. Computing a fingerprint value

Given the following inputs:

- o "key", an array of characters
 - o "f", an integer indicating the number of output bits
1. Let "hash-value" be the SHA-256 message digest [RFC6234] of "key", expressed as an integer.
 2. Let "h" be the number of bits in "hash-value"
 3. Let "fingerprint-value" be 0
 4. While "fingerprint-value" is 0 and "h" > "f":
 1. Let "fingerprint-value" be the "f" least significant bits of "hash-value".
 2. Let "hash-value" be the "h"- "f" most significant bits of "hash-value".
 3. Subtract "f" from "h".
 5. If "fingerprint-value" is 0, let "fingerprint-value" be 1.
 6. Return "fingerprint-value".

Note: Step 5 is to handle the extremely unlikely case where a SHA-256 digest of "key" is all zeros. The implications of it means that there's an infinitesimally larger probability of getting a "fingerprint-value" of 1 compared to all other values. This is not a problem for any practical purpose.

2.1.5. Computing the key

Given the following inputs:

- o "URL", an array of characters
- 1. Let "key" be "URL" converted to an ASCII string by percent-encoding as appropriate [RFC3986].
- 2. Return "key"

2.1.6. Computing a Hash Value

Given the following inputs:

- o "key", an array of characters.
- o "N", an integer

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

1. Let "hash-value" be the SHA-256 message digest [RFC6234] of "key", truncated to 32 bits, expressed as an integer.
2. Return "hash-value" modulo N.

2.1.7. Computing an Alternative Hash Value

Given the following inputs:

- o "hash1", an integer indicating the previous hash.
- o "fingerprint", an integer indicating the fingerprint value.
- o "N", an integer indicating the number of entries in the digest.
- 1. Let "fingerprint-string" be the value of "fingerprint" in base 10, expressed as a string.
- 2. Let "hash2" be the return value of Section 2.1.6 with "fingerprint-string" and "N" as inputs, XORed with "hash1".
- 3. Return "hash2".

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`, a complete response need not be pushed; The server MAY push a 304 response for that resource, indicating the client that it hasn't changed.
- 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`.

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

2.2.1. Querying the Digest for a Value

Given the following inputs:

- o "URL" a string corresponding to the Effective Request URI ([RFC7230], Section 5.5) of a cached response [RFC7234].
 - o "digest-value", an array of bits.
1. Let "f" be the value of the first byte of "digest-value".
 2. Let "b" be the bucket size, defined as 4.
 3. Let "N" be the value of the second to fifth bytes of "digest-value" in big endian form.
 4. Let "key" be the return value of Section 2.1.5 with "URL" as input.

5. Let "h1" be the return value of Section 2.1.6 with "key" and "N" as inputs.
 6. Let "fingerprint" be the return value of Section 2.1.4 with "key" and "f" as inputs.
 7. Let "h2" be the return value of Section 2.1.7 with "h1", "fingerprint" and "N" as inputs.
 8. Let "hashes" be an array containing "h1" and "h2".
 9. For each "h" in "hashes":
 1. Let "position_start" be $40 + "h" * "f" * "b"$.
 2. Let "position_end" be $"position_start" + "f" * "b"$.
 3. While "position_start" < "position_end":
 1. Let "bits" be "f" bits from "digest_value" starting at "position_start".
 2. If "bits" is "fingerprint", return true
 3. Add "f" to "position_start".
 10. Return false.
3. The SETTINGS_SENDING_CACHE_DIGEST SETTINGS Parameter

A Client SHOULD notify its support for CACHE_DIGEST frames by sending the SETTINGS_SENDING_CACHE_DIGEST (0xXXX) SETTINGS parameter.

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

DIGEST_PENDING (0x1): When set it indicates that the client has a digest to send, and the server may choose to wait for a digest in order to make server push decisions.

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 client has no digest to send the server.

4. The SETTINGS_ACCEPT_CACHE_DIGEST SETTINGS Parameter

A server can notify its support for CACHE_DIGEST frame by sending the SETTINGS_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:

ACCEPT (0x1): When set, it indicates that the server is willing to make use of a digest of 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 its 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 SETTINGS_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.

5. IANA Considerations

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

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

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

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

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

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

6. 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."

7. References

7.1. 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, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/rfc6234>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://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, <<https://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, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/rfc7540>>.

7.2. Informative References

- [Cuckoo] "Cuckoo Filter: Practically Better Than Bloom", n.d., <<https://www.cs.cmu.edu/~dga/papers/cuckoo-conext2014.pdf>>.
- [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-tls13-28 (work in progress), March 2018.

- [RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <<https://www.rfc-editor.org/info/rfc3864>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://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, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<https://www.rfc-editor.org/info/rfc6265>>.
- [Service-Workers]
Russell, A., Song, J., Archibald, J., and M. Kruisselbrink, "Service Workers 1", W3C Working Draft WD-service-workers-1-20161011, October 2016, <<https://www.w3.org/TR/2016/WD-service-workers-1-20161011/>>.

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], Section 7.

```
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 a digest of one 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 represents 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. Changes

B.1. Since draft-ietf-httpbis-cache-digest-04

- o Remove ETag from the digest key calculations.
- o Add SETTINGS_ prefix to parameter names.

B.2. Since draft-ietf-httpbis-cache-digest-03

- o Yoav becomes an author; Mark steps down.

B.3. Since draft-ietf-httpbis-cache-digest-02

- o Switch to Cuckoo Filter.

B.4. Since draft-ietf-httpbis-cache-digest-01

- o Added definition of the Cache-Digest header.
- o Introduce ACCEPT_CACHE_DIGEST SETTINGS parameter.

- o Change intended status from Standard to Experimental.

B.5. Since draft-ietf-httpbis-cache-digest-00

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

Appendix C. Acknowledgements

+{:numbered="false"}

Thanks to Stefan Eissing for his suggestions.

Authors' Addresses

Kazuho Oku
Fastly

Email: kazuhooku@gmail.com

Yoav Weiss
Akamai

Email: yoav@yoav.ws
URI: <https://blog.yoav.ws/>