Abstract

This document describes an improved HTTP caching method which can be applied in addition to the standard caching behavior described in [Part6]. It defines the associated header field that controls this improved caching mechanism and a modified caching operation which is slightly different to one in [Part6].

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 November 17, 2014.

Copyright Notice

Copyright (c) 2014 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.
1. Introduction

HTTP caching has a significant potential for reducing Interdomain traffic, especially when shared caches are used within operator networks. Recent studies have shown very promising results regarding the cacheability of HTTP traffic (see [Ager], [Erman]).

Unfortunately this potential can not be fully used by the standard caching behavior described in [Part6]. The following two reasons mainly limit the benefit of caching today:

1. Different URLs for one specific resource:

   For cache systems which follow the instructions in [Part6] the URL mainly serves as a identifier for the cached content. Unfortunately due to mechanisms like load balancing and/or the
use of CDNs the URL for one specific resource can vary. From the point of the cache system two different URLs mean two different cache items notwithstanding that the cache items can be identical in their bit-representation. Therefore caching systems usually store one specific content several times and use storage capacity which could potentially be used for caching of other contents.

2. Personalization of HTTP messages in the header:

When HTTP messages carry personal information like cookies, session IDs in the query string (this affects also point 1) or other header attributes for the purpose of personalization (or managing state) then shared caches cannot reuse these responses for following requests. In this context content producers allow caching only in the browser of the user (e.g. via Cache-control: private) or deny caching at all. If a specific representation is requested several times by different clients then this would result in HTTP messages which differ in the headers while the bodies are equal. According to [Ager] personalization is also one of the main reasons for the unused potential of caching.

The goal of this proposal is to address these challenges and come up with caching, varying URLs and personalization.

1.1. Requirements Language

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 RFC 2119 [RFC2119].

2. Specification

The approach for an improved HTTP caching in this proposal is twofold.

Section 2.1 introduces a new header field with a hash value. This is used for precisely identifying the transferred content in the body of HTTP messages and to signal the permission for caching and reusing of the body in intermediate cache systems.

The modified caching operation described in Section 2.2 uses the above-mentioned header field and ensures that all headers (of HTTP request and response messages) are exchanged between client and server even if the body of a response message is coming from an intermediate cache systems.
2.1. HTTP header field extension

For precisely identifying the transferred content independent of the used URL and independent of additional header fields in the context of content negotiation the following header field is used:

```
Cache-NT: sha-256 "=" <base64 encoded sha256 output>
```

The new header field carries an SHA-256 value (algorithm as in [SHS]) which is computed and encoded the following way:

1. When a client wants to retrieve a specific content it uses a HTTP GET request with a URL to address the resource. Additionally the client can use further header fields to negotiate that representation of the resource which fits best for the client (this mechanisms is called content negotiation in [RFC2616]). The SHA-256 value MUST be computed over that representation of the resource which would be send by the server to the client in case of a successful response with status code 200 OK.

2. The SHA-256 hash value MUST be computed before the modifications of the possibly present header fields Content-Encoding, Content-Range and Transfer-Encoding are applied.

3. The SHA-256 hash value MUST always be computed over the full representation even if only parts of it are transferred to the client (e.g. partial content, delta encoding). The hash value serves as an unique identifier for intermediate cache systems to identify also parts of the full representation.

4. The SHA-256 hash value MUST be computed by the origin server. It SHOULD be computed only once (when the resource is made available on the server or when the resource has changed). It SHOULD not be computed in the moment when the server receives the request due to not delaying the response.

5. After computing the SHA-256 hash value the output of it MUST be base64 encoded without line wrapping.

The Cache-NT header field is sent by the server in successful responses with status codes 200 or 206. If the header field is present then the server signals that the body of the response can be used for caching by intermediate cache systems for subsequent requests in compliance with the cache operation described in Section 2.2.

In the following some examples are given:
Example header field:

Cache-NT: sha-256=ZDJhODRmNGI4YjY1M ... DgyMjlkYTgwNGEyNiAgLQo=

Example for computation of the hash value under UNIX:

sha256sum PopularVideo.mp4 | base64 -w0

Several examples of request-response pairs:

a)  
+---------------------------------------------+  
| GET /videos/PopularVideo.webm HTTP/1.1      |  
| Host: example.com                           |  
+---------------------------------------------+  
  
| HTTP/1.1 200 OK                             |  
| Content-Type: video/webm                    |  
| Cache-NT: sha-256=AAAAAAAAAA...AAAAAAAAAA   |  
| ...                                         |  
+---------------------------------------------+  

b)  
+---------------------------------------------+  
| GET /videos/PopularVideo.webm HTTP/1.1      |  
| Host: example.com                           |  
| Range: bytes=0-499                          |  
+---------------------------------------------+  
  
| HTTP/1.1 206 Partial Content                |  
| Content-Type: video/webm                    |  
| Content-Range: bytes 0-499/1000             |  
| Cache-NT: sha-256=AAAAAAAAAA...AAAAAAAAAA   |  
| ...                                         |  
+---------------------------------------------+  

=> same hash value as in a) because only a part of the representation is requested

c)  
+---------------------------------------------+  
| GET /videos/PopularVideo HTTP/1.1           |  
| Host: example.com                           |  
| Accept: video/mp4                           |  
+---------------------------------------------+  

2.2. Modified cache operation

The modified cache operation is slightly different to the one in [Part6]. It uses the header field described in Section 2.1 and ensures that all headers (of HTTP request and response messages) are exchanged between client and server even if the body of a response message is coming from an intermediate cache systems. Client requests will never terminate at intermediate cache systems as in [Part6].

2.2.1. Incoming Request Messages

Incoming request messages MUST always be forwarded to the origin server by the intermediate cache system.

For HTTP/1.0 or HTTP/1.1 requests the cache system SHOULD keep track of the desired connection state by evaluating the Connection header field.

For HTTP/1.1 requests the cache system MUST keep track of all pipelined requests.

2.2.2. Incoming Response Messages

The cache system analyzes the header of incoming response messages. If the status code IS NOT 200 or 206 then the response is forwarded to the client without modifications. If the status code IS 200 or 206 then the cache system looks for the Cache-NT header field (described in Section 2.1). Two situations can arise:

a. The Cache-NT header field IS NOT present:

   Then the response message is forwarded to the client without modifications.

b. The Cache-NT header field IS present:

   Then the cache system analyzes the hash value in the Cache-NT header field. Two situations can arise:
1. The cache system has NO cache entry which fits to the hash value in the Cache-NT header field (cache miss):

   Then the response message is forwarded to the client without modifications. To prevent cache poisoning the cache system computes the hash value over the transferred representation in the body (as it is described in Section 2.1) and if it does match to the hash value in the Cache-NT header field of the response from the server then a copy of the message body is stored in the cache system. Figure 2 visualizes this cache operation in case of a cache miss.

2. The cache system has an cache entry which fits to the hash value in the Cache-NT header field (cache hit):

   After receiving of the whole message header the cache system aborts the transfer of the message body from the server:

   o HTTP/2: Via sending RST_STREAM to the server. As each HTTP request-response exchange is assigned to a single stream no side effects will arise.

   o HTTP/1.0: Via closing the TCP connection to the server (and sending TCP_RST). If the TCP connection was intended to stay open (signaling via the Connection header field) then the cache system SHOULD reopen the TCP connection to the server immediately for following requests by the client.

   o HTTP/1.1: Via closing the TCP connection (and sending TCP_RST). If the TCP connection was intended to stay open (signaling via the Connection header field) then the cache system SHOULD reopen the TCP connection to the server immediately for following requests by the client. If pipelining was used then the cache system MUST retrieve all requests after the current request once again.

   After that the cache system uses the already received message header from the server and concatenates it with the locally stored body. In this process the cache systems MUST follow the possibly present header fields

   o Content-Encoding

   o Content-Range
and MUST transform the body in the right way. This means that the client will receive exactly the same HTTP response message which was originally send out by the server. Figure 1 visualizes this cache operation in case of a cache hit.
Cache operation in case of cache hit.

Figure 1
Cache operation in case of cache miss.

Figure 2
2.3. Suggestions

In case of a cache hit the cache system aborts the transfer of the response body from the server after the whole header has been received (see Section 2.2). As the transfer of the body cannot be aborted immediately the server will still send some parts of the body. How many Kilobytes are transferred depends mainly on the congestion window of the underlying TCP connection. If the congestion window is small then only a few Kilobytes of the response will go over the wire.

Evaluations at Technische Universitaet Chemnitz have shown that at least around 20 Kilobytes are transferred between origin server and cache system in case of a cache hit (this is for a HTTP/1.0 or HTTP/1.1 request right after opening a TCP connection). Therefore including the Cache-NT header field for small resources does not make much sense from the point of caching as the whole body is being transferred before the cache system can abort it.

3. IANA Considerations

3.1. Header Field Registration

HTTP header fields are registered within the Message Header Field Registry maintained at [1].

This document defines the following HTTP header fields, so their associated registry entries shall be updated according to the permanent registrations below (see [BCP90]):

| Header Field Name | Protocol | Status            | Reference |
|-------------------+----------+-------------------+----------|
| Cache-NT          | http     | proposed standard | Section 2.1 |

The change controller is: "IETF (iesg@ietf.org) - Internet Engineering Task Force".

3.2. Cache Directive Registration

This document defines the following HTTP header field directives:

<table>
<thead>
<tr>
<th>Cache Directive</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>sha-256</td>
<td>Section 2.1</td>
</tr>
</tbody>
</table>

Drechsler Expires November 17, 2014
4. Security Considerations

This section is meant to inform developers, information providers, and users of known security concerns specific to the caching mechanism described in this proposal. In addition more general security considerations of HTTP caching are discussed in Section 8 of [Part6].

The cache operation in Section 2.2 uses the Cache-NT header field (see Section 2.1) in incoming response messages. If the hash value in the Cache-NT header field of the (server) response does not correspond to the representation in the body of that response then a wrong body is maybe concatenated to the header of the server and send to the client (this occurs when the cache system has an cache entry which fits to the hash value in the response of the server). Origin server SHOULD always include the correct hash value in the Cache-NT header field which fits to the representation in the body. Intermediaries MUST NOT change the hash value in the Cache-NT. In addition the client can compute the hash value over the full representation (in case of responses with 200 OK) itself and can re-validate it with the value in the Cache-NT header field.

If a cache system does not have a cache entry which fits to the hash value in the Cache-NT header field then it forwards the response to the client and stores a local copy of the body (see Section 2.2). To prevent cache poisoning the cache system SHOULD compute the hash value over the full representation in the body (in case of responses with 200 OK) itself and SHOULD re-validate it with the value in the Cache-NT header field.

Another security concern will arise if significant security flaws in the used hash algorithm (currently SHA-256) are detected. Then the cache can easily be poisoned. In this case origin servers and intermediate cache systems MUST switch to another hash algorithm (e.g. SHA-512 or the upcoming SHA-3 family).

5. References

5.1. Normative References


5.2. Informative References


5.3. URIs


Author’s Address

Chris Drechsler (editor)
Technische Universitaet Chemnitz
09107 Chemnitz
Germany

Email: chris.drechsler.etit@tu-chemnitz.de