Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 1, 2017

J. Butler

W. Lee

B. McQuade

K. Mixter

October 28, 2016

A Proposal for Shared Dictionary Compression over HTTP
draft-lee-sdch-spec-00

Abstract

This paper proposes an HTTP/1.1-compatible extension that supports inter-response data compression by means of a reference dictionary shared between user agent and server.

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 May 1, 2017.

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Introduction

In order to reduce payload size, HTTP/1.1 supports response compression via the Accept-Encoding and Content-Encoding headers. The most commonly used HTTP response compression encoding is gzip, which compresses data that is repeated within a given response. However, HTTP/1.1 does not provide a mechanism for compressing data that is repeated between responses. A different class of encoding technique, known as delta encoding, has proven effective at compressing inter-response data.

Previous efforts to extend HTTP/1.1 to support delta compression have focused on encoding an HTTP response as a delta of a previous version of that response. One such approach is discussed in RFC 3229 "Delta encoding in HTTP" [RFC3229]. While RFC 3229 is effective at reducing payload size for many types of resources, it may not be suitable for certain classes of responses.

Specifically, under RFC 3229, deltas can only be applied to responses originating from the same URL, and the means of identifying the instance to delta "from" is by a Last-Modified timestamp or entity-tag. This makes RFC 3229 unsuitable for compressing dynamically generated responses to a given URL with varying query parameters (e.g. a search results page), since these types of responses are difficult to identify uniquely using entity tags or last modified timestamps. Content hashes can be used, but false positives are possible. Also, storing all previous responses on the server may not be practical.

2. Proposal: Shared Dictionary Compression over HTTP

Existing techniques compress each response in isolation, and so cannot take advantage of cross-payload redundancy. For example, retrieving a set of HTML pages with the same header, footer, inlined JavaScript and CSS requires the retransmission of the same data multiple times. This paper proposes a compression technique that leverages this cross-payload redundancy.

In this proposal, a dictionary is a file downloaded by the user agent from the server that contains strings which are likely to appear in subsequent HTTP responses. In the case described above, if the header, footer, JavaScript and CSS are stored in a dictionary possessed by both user agent and server, the server can substitute these elements with references to the dictionary, and the user agent can reconstruct the original page from these references. By

substituting dictionary references for repeated elements in HTTP responses, the payload size can be reduced.

If either the user agent or the server does not support the extension, then ordinary HTTP responses are served.

If both the user agent and the server support the extension but the user agent does not have an applicable dictionary (as described in detail below), the server responds with an ordinary HTTP response that includes a header advertising the location of a relevant dictionary. This dictionary can be retrieved out-of-band by the user agent.

If both the user agent and the server support the extension and the user agent has an applicable dictionary, then each HTTP response includes references to strings in the dictionary, rather than repeating those strings in the response. The references require fewer bytes to encode than the strings themselves, reducing the payload size.

The HTTP header-based protocol for negotiating the presence of dictionaries on user agent and server is referred to in this proposal as the SDCH protocol. The compression scheme based on a particular dictionary shared between user agent and server is referred to as the SDCH encoding, and is built upon the VCDIFF compression data format [RFC3284].

3. Syntax

The grammar descriptions in the sections that follow depend on the following syntax: DIGIT (decimal digit), BASE64URLDIGIT (alphanumeric digit or "-" or "_"), PAYLOADBYTE (a byte), token (informally, a sequence of non-special, non-white space characters), rest-of-line (informally, a sequence of characters not including carriage return or line-feed). In the grammar below, HTTP_url, abs_path, and query are defined in RFC 7230 [RFC7230].

```
header = attr ":" value "\n"
attr = token
value = rest-of-line
dictionary-client-id = 1*BASE64URLDIGIT
dictionary-server-id = 1*BASE64URLDIGIT
payload = 1*PAYLOADBYTE
vcdiff-payload = 1*PAYLOADBYTE
partial-url = HTTP_url | abs_path [ "?" query ]
```

The attribute names (attr) are case-insensitive. White space is permitted between tokens.

4. Dictionary Description

4.1. General

In the proposed protocol, a dictionary can only be used with a limited set of URLs and for a limited duration of time, referred to as its scope and lifetime, respectively. A dictionary is composed of the data used by the compression algorithm, known as the payload, as well as metadata describing its scope and lifetime. The scope is specified by a domain attribute and path attribute that are patterned after the same named attributes from the HTTP State Management Specification [RFC2965].

4.2. Syntax of Dictionary Metadata

The syntax of dictionary metadata is as follows:

```
dictionary-metadata = 1*dictionary-header "\n"
dictionary-header = "domain" ":" value "\n"
| "path" ":" value "\n"
| "path-equals" ":" value "\n"
| "format-version" ":" value "\n"
| "max-age" ":" value "\n"
| "port" ":" <"> portlist <"> "\n"
portlist = 1#portnum
portnum = 1*DIGIT
```

A complete dictionary definition then has this format: n dictionary-definition = dictionary-metadata payload

Informally, the metadata for a dictionary is a series of headers, similar in form to HTTP headers, terminated by an empty line. The dictionary payload begins immediately after this blank line.

The valid dictionary header identifiers are described below:

o Domain: domain.

Required. Indicates the domain to which the dictionary applies. The domain specification must explicitly start with a dot. For example, a dictionary with the domain specification ".google.com" may be used to compress a response served from the host name www.google.com, but not used to compress a response served from the host name www.gmail.com. Only printable ASCII characters are permitted in the domain value. International Domain Names must be specified using IDNA.

o Path: path.

Optional. Indicates the set of URL paths for which this dictionary is valid. If unspecified, the dictionary applies to all paths within the given domain.

o Path-equals: path.

Optional. Indicates the exact URL path for which this dictionary is valid. If both "path" and "path-equals" are specified, the dictionary applies only to those URLs which satisfy both criteria.

o Format-version: version.

Optional. Indicates the version of the dictionary payload. If unspecified, the format version defaults to "1.0". Currently, the only acceptable value is "1.0".

o Max-age: delta-seconds.

Optional. Indicates the amount of time that a dictionary can be advertised to the server by the user agent, relative to the time it was downloaded. If unspecified, the default is 30 days from the time the dictionary was downloaded by the user agent. * Port: port list. Optional. Indicates the comma-separated list of ports to which this dictionary applies. If unspecified, the dictionary applies to all ports.

Like HTTP headers, dictionary header identifiers are case-insensitive. Unknown headers will be ignored by the user agent, allowing other headers to be added in the future.

4.3. Dictionary Scope

The specific rules of when a dictionary can be applied to a URL, i.e. that define its scope, are modeled after the rules for cookie scoping. The term "domain-match" is defined in RFC 2965. We define path-matching as follows For two strings that represent paths, P1 and P2, P1 path-matches P2 if either:

1. P2 is equal to P1

2. P2 is a prefix of P1 and either the final character in P2 is "/" or the character following P2 in P1 is "/".

For example, "/tec/waldo" path-matches "/tec", "/tec/", and "/tec/waldo", but does not path-match "/tec/wal".

Given these definitions of domain-match and path-match, a request URL falls within a dictionary's scope exactly when all of the following are true:

1. The request URL's host name domain-matches the Domain attribute of the dictionary.

2. If the dictionary has a Port attribute, the request port is one of the ports listed in the Port attribute.

3. The request URL path-matches the path attribute of the dictionary.

4. The request URL's scheme matches the scheme of the dictionary.

If a URL falls within a dictionary's scope, the dictionary is said to "apply" to the URL.

4.4. Dictionary Identifier

In communications between user agent and server, a dictionary is identified by the first 96 bits of the SHA-256 digest [RFC6234] of a dictionary's metadata and payload (see dictionary-definition above) exactly as it is received by the user agent from the server. Both user agent and server compute this identifier independently, based on the metadata and the payload of the dictionary. This digest should be unique within a dictionary's scope (domain and path) in order to prevent dictionary identifier collisions.

The digest serves not only as an identifier but also as a safeguard against attempts to maliciously intercept or otherwise modify dictionary contents, since a compromised dictionary will hash to a different identifier and the server will not recognize it. The user agent identifier for a dictionary is defined as the URL-safe base64 encoding (as described in RFC 3548, section 4 [RFC3548] of the first 48 bits (bits 0..47) of the dictionary's SHA-256 digest. The server identifier for a dictionary is the URL-safe base64 encoding of the second 48 bits (bits 48..95). When identifying a dictionary to the server, the user agent uses the user agent identifier, and similarly, when identifying a dictionary to the user agent, the server uses the server identifier. Note that both user agent and server have the entire dictionary and can thus compute both identifiers for the dictionary.

As a consequence of this scheme, dictionaries do not need to be explicitly named by site maintainers, as the protocol avoids identifying them in any way other than the above digest-generated identifiers.

4.5. Differences between Dictionaries and Cookies

Dictionaries are similar to cookies in that they allow sharing of state over HTTP. Thus, we have modeled dictionaries after cookies, as described in RFC 2965. However, because dictionaries are typically larger than cookies, embedding a dictionary in the response would increase latency of the response. Thus a dictionary is always sent as a separate HTTP response (unlike a cookie which is included in a Set-Cookie header of any HTTP response). The Get-Dictionary HTTP response header is used to tell the user agent that it should fetch a dictionary separately for use in future requests.

Likewise, rather than including the dictionary contents in the HTTP request headers (like a cookie in the Cookie header), dictionary identifiers (described above) are used to advertise available dictionaries in HTTP requests from the user agent to the server.

5. User Agent / Server Interaction Description

5.1. User Agent Role in HTTP Request Generation

The user agent:

1. Advertises support for the proposed protocol by adding the "sdch" token to the Accept-Encoding header of HTTP requests.

2. Advertises any dictionaries it possesses that apply to the URL being requested (per the scoping rules above) in the Avail-Dictionary request header.

The Avail-Dictionary header syntax is as follows: avail-dictionary-header = "Avail-Dictionary" ":" 1#dictionary-client where dictionary-client-id is the user agent identifier part for the dictionary based on the SHA-256 digest as described above. The value of this header is informally a comma separated list of user agent dictionary identifiers.

The user agent must advertise every dictionary it has cached that applies to the requested URL. It is only the presence of the dictionary identifier in this header that indicates to the server that the user agent possesses and therefore does not need to download the dictionary. Since the user agent must advertise every dictionary it has, it is the site maintainer's responsibility to avoid making too many dictionaries available at a given time. Advertising many dictionaries in this header can counteract the benefits of compression.

Note that for each individual request the user agent has discretion over whether or not to add "sdch" Accept-Encoding token and the Avail-Dictionary header. Since some responses, such as image data, are unlikely to benefit from dictionary compression, the user agent can reduce the size of its requests by not sending this token and header. The user agent may decide whether or not to add these headers based on file extensions in URLs or the context of the request. For instance, the user agent may choose to not advertise SDCH for URLs referenced in IMG elements.

5.2. Server Role in HTTP Response Generation

When a server that supports the extension receives a request that indicates that the user agent supports the protocol (e.g. the "sdch" token is present in the Accept-Encoding request header), two independent decisions must be made. The server must decide: 1. if it wants to send an encoded response. 2. if it wants to inform the user agent about additional dictionaries it can download and use in the future.

The server may return an encoded response only if all of the following are true: 1. The Accept-Encoding request header contains the "sdch" token. 2. The server can send a response compressed with a dictionary whose dictionary-client-id is in the Avail-Dictionary request header.

A server may return a response that is not encoded even if it recognizes a dictionary advertised by the user agent. If the server decides to not use SDCH encoding when a Avail-Dictionary header is present, it must include a specific HTTP header X-SDCH-Encoding with value "0" in the response. The syntax of the X-SDCH-Encoding header is:

sdch-not-used-header = "X-SDCH-Encoding" ":" "0"

The server indicates that an HTTP response is encoded by inserting the token "sdch" into the Content-Encoding header of the HTTP response.

A compatible server may instruct a compatible user agent to download one or more new dictionaries by including the Get-Dictionary header in the HTTP response. The server may advertise a Get-Dictionary header even if the response is not encoded. The syntax of the Get-Dictionary header is: get-dictionary-header = "Get-Dictionary" ":" 1#partial-url where partial-url is either a complete URL, or just the absolute URL path (in which case the scheme, host, and port of the originating server would be used when requesting the dictionary). If a complete URL is provided, it must have the same scheme, host, and

port as the originating server. The Content-Type header of dictionary responses must be application/x-sdch-dictionary. The value in the get dictionary header is a comma-separated list of partial-url elements.

The server must not advertise a dictionary with a dictionary-client-id that the user agent has listed in the Avail-Dictionary header.

The server may use SDCH compression with a dictionary that the user agent has advertised and also include a Get-Dictionary header for a different dictionary that the user agent has not advertised.

The server must prevent SDCH-encoded responses from being cached by intermediate proxies. See the section below on proxy caching for additional details.

The server should limit the number of active dictionaries at any one time, by using well-scoped dictionaries. A server that has many active dictionaries with overlapping scope will cause user agents to generate a very long Avail-Dictionary header, the overhead of which can counteract the benefits of SDCH compression.

The server may decide to precompute and cache SDCH-encoded responses if a given SDCH-encoded response will be served multiple times (e.g. for static content).

The server may apply multiple Content-Encodings to the response, (e.g. sdch and gzip) in which case subsequent encoding tokens are appended to the Content-Encoding header, per the HTTP/1.1 RFC section 14.11.

5.3. User Agent Role in HTTP Response Handling

An SDCH-compatible user agent must inspect the Content-Encoding HTTP response header to determine if the response is SDCH-encoded. If the Content-Encoding includes the "sdch" token, the user agent must perform SDCH decompression on the response.

If the HTTP response includes a Get-Dictionary header, the user agent must verify that the partial-url specified refers to the same server that generated the response. If so, the user agent may download the dictionary at the given URL.

There are two different URLs to consider when downloading and storing a dictionary. The referer URL is the URL of the request that resulted in the server responding with a Get-Dictionary header.

The dictionary URL is defined as follows:

1. If the partial-url is a complete URL, the dictionary URL is the partial-url.

2. If the partial-url is just a path URL, the dictionary URL is generated from the scheme and host name of the referrer URL and the path in the partial-url.

The user agent may retrieve a dictionary if the origin of the dictionary matches the origin of the referrer. HTTP redirects may only be followed if the origin matches as well.

Upon retrieving the dictionary, the user agent must validate the dictionary. Here again, the validation rules are modeled after the rules for when a user agent can accept an HTTP cookie. A dictionary is invalid and must not be stored if any of the following are true:

1. The dictionary has no Domain attribute.

2. The effective host name that derives from the referrer URL host name does not domain-match the Domain attribute.

3. The Domain attribute is a top level domain.

4. The referrer URL host is a host domain name (not IP address) and has the form HD, where D is the value of the Domain attribute, and H is a string that contains one or more dots.

5. If the dictionary has a Port attribute and the referrer URL's port was not in the list.

If the dictionary is valid and user agent decides to store the dictionary, the scheme of the dictionary URL should also be stored along with dictionary.

5.4. SDCH-Encoded Response Body

An SDCH-encoded response starts with the dictionary-server-id used to compress the response. The syntax of the SDCH-encoded response is: dictionary-compression-response = dictionary-server-id "\0" vcdiff-payload

6. Examples

For the purpose of these examples, assume the following dictionaries exist on the server and can be downloaded from the following URLs:

"Search results" dictionary

- domain: .google.com
- path: /search
- user agent ID: TWFuIGlz
- server ID: JOWk0d2N
- download location: /dictionaries/search_dict

"Help pages" dictionary

- domain: .google.com
- path: /
- user agent ID: GVhc3V48
- server ID: O9d2_m3-
- download location: /dictionaries/help_dict

Note that the dictionary identifier consists of two parts: user agent ID and the server ID. Most of the detail of the request and response headers has been omitted.

6.1. Example 1: Initial Interaction, User Agent has No Dictionaries

1. user agent's request

```
GET /search?q=sprouts HTTP/1.1
Host: www.google.com
Accept-Encoding: sdch, gzip
```

1. server's response

```
HTTP/1.1 200 OK
Content-type: text/html
Content-Encoding: gzip
Get-Dictionary: /dictionaries/search_dict, /dictionaries/help_dict
Cache-Control: private
```

Note that the response returned by the server does NOT use SDCH encoding, since the user agent does not have a dictionary. The server simply provides the locations of the dictionaries for future use. The user agent may choose to retrieve one or both dictionaries separately.

6.2. Example 2: User Agent Requests the Dictionary

1. user agent's request

```
GET /dictionaries/search_dict HTTP/1.1
Host: www.google.com
Accept-Encoding: sdch, gzip
```

1. server's response

```
HTTP/1.1 200 OK
Content-type: application/x-sdch-dictionary
Content-Encoding: gzip

Domain: .google.com
Path: /search
Format-version: 1.0

...dictionary contents...
```

Upon receiving this response, the user agent computes the digest of the dictionary and determines the user agent ID is TWFuIGlz and the server ID is JOWk0d2N.

6.3. Example 3: User Requests Page AND User Agent Has Already Downloaded

the Dictionary

1. user agent's request

```
GET /search&q=brussel+sprouts HTTP/1.1
Host: www.google.com
Accept-Encoding: sdch, gzip
Avail-Dictionary: TWFuIGlz
```

1. server's response

```
HTTP/1.1 200 OK
Content-type: text/html
Content-Encoding: sdch, gzip
Get-Dictionary: /dictionaries/help_dict
Cache-Control: private

JOWk0d2N<NUL>...VCDIFFed response...
```

(note that the response shown to the left the result of gzip decompression)

The server has properly identified the dictionary using its server ID and the user agent can confirm that the second 48 bits of the SHA-256 digest of the dictionary match its computation. It can then decompress the VCDIFF response using this dictionary. Even though the "search results" dictionary was used to decompress the response, the server has chosen to indicate another dictionary could be requested by the user agent from http://www.google.com/dictionaries/help_dict. This dictionary must be different than the "search results" dictionary as the server must never request the user agent download a dictionary it knows the user agent already has. Let's assume the user agent decides to download this dictionary.

6.4. Example 4: User Requests with Multiple Dictionaries

1. user agent's request

```
GET /search&q=brussels HTTP/1.1
Host: www.google.com
Accept-Encoding: sdch, gzip
Avail-Dictionary: GVhc3V48,TWFuIGlz
```

1. server's response

```
HTTP/1.1 200 OK
Content-type: text/html
Content-Encoding: sdch, gzip
Cache-Control: private

JOWk0d2N<NUL>...VCDIFFed response...  (note that the response shown
to the left the result of gzip decompression)
```

The user agent advertises that it has already downloaded two dictionaries that apply. The server may compress the response with either dictionary. As the server has no other dictionaries that apply to the request, it does not advertise any dictionaries in its response.

7. Implementation Considerations

7.1. Implementation Limits

There are practical limitations to the number and size of the dictionaries a user agent can store. It is suggested that general use, non-mobile user agents should have the following minimum capabilities:

- At least 300 dictionaries stored

- At least 100KB of payload per dictionary
- At least 10MB of total dictionary contents
- At least 20 dictionaries stored per domain

7.2. Dictionary Downloading

The user agent always has the choice of whether or not to download a dictionary. It is recommended that the user agent be implemented with sufficient state to avoid downloading too many dictionaries from the same server. A malfunctioning server may also request the user agent continually download the same dictionary. One simple method to avoid both of these possibilities is for the user agent to rate-limit downloading dictionaries from the same domain.

When the user agent receives a response with a Get-Dictionary header with dictionary download URLs that it may fetch, it should perform the dictionary downloads in the background. This is possible as the dictionary to be downloaded is guaranteed to not be needed to decompress the response with the Get-Dictionary header. The user agent should be careful to abort background dictionary downloads that do not complete in a reasonable amount of time.

7.3. Data Integrity

If the dictionaries are tied to individual users or specific user actions, HTTP may leak this information to passive attacker by allowing the Get-Dictionary info to be seen. When using HTTPS, the same risk is prevented in the design document since Get-Dictionary URLs are required to be same-origin as the response.

However, Downloading dictionaries over HTTPS or advertising dictionaries over HTTPS might introduce new security risks.

TODO: add some examples. For example, SDCH-over-HTTPS subject to compression oracle attacks similar to CRIME/BREACH with the difference that the compression context is not supplied by the attacker. If an attacker had the contents of a dictionary, there is a theoretical possibility where a server sends a static response XOR'ed with user-provided data. The Attacker can provide data which reduced the size of the response when XOR'ed with the static response, the attacker may then be able to determine the contents of the static response.

The protocol needs to ensure that the content as decompressed by the user agent with a given dictionary is identical to the server's

originally intended content. The three areas that can cause a data integrity problem are discussed below.

7.3.1. Data tampered by Proxy

We have found incorrectly implemented proxies which tamper with an SDCH response and make the response unable to be decompressed to the server's originally intended content. The tampering may not be detected in the SDCH encoding itself if the proxy makes SDCH content look like non-SDCH content, for instance, by stripping the 'sdch' token from the content-encoding header of the response or by adding additional encodings (like gzip) on top of the SDCH and gzipped response without making the Content-Encoding header match. In order to detect when this occurs, the HTTP header X-SDCH-Encoding must be added to the response by the server to inform the client that the response was originally not SDCH encoded by the server. Should the user agent advertise SDCH capability in the request but receive a non-SDCH encoded response without the X-SDCH-Encoding header, it suggests that the response was tampered by a proxy. The user agent may then take action to avoid using SDCH in the future.

7.3.2. Dictionary mismatch

When a dictionary information is exchanged between user agent and server, it is necessary to ensure that the dictionary identifiers are completely unambiguous, or the decompressed result may differ from the original content. To address this issue, SDCH uses the first 96 bits of the SHA-256 digest of a dictionary's metadata and payload to create the dictionary identifiers used by the user agent and server to avoid ambiguity. (Please refer to the section "Dictionaries description" above for details.)

7.3.3. Data corruption / malicious attacks

While this issue is not specific to SDCH, it can be exacerbated due to the nature of the stateful compression. For example, if the dictionary is corrupted or maliciously modified in a persistent on-disk cache, all subsequent responses decoded by using this dictionary will be corrupt. For this reason, the user agent and server should revalidate the dictionaries' integrity when they are loaded from non-volatile storage.

Other issues like data corruption during transmission in the encoded payload could have much bigger adverse effect than that in the plain text. TCP provides a checksum, but it cannot detect some errors like swapped bytes. To address this issue, SDCH includes an Adler32 checksum [RFC1950] in the encoded data shards. (Please refer to appendix "VCDIFF Encoding Format and SDCH" for details.)

8. Response Caching

8.1. User Agent Cache

The user agent should honor HTTP caching directives (Cache-Control, Expires,...) for caching responses, whether or not the responses are SDCH-encoded. When caching the SDCH-encoded responses, the SDCH-encoded responses should be decoded before being written to the cache. If this is not possible, the user agent may cache SDCH-encoded responses, unless the HTTP response headers indicate that the response is not cacheable. In this case, an SDCH-encoded cache entry should be invalidated when (1) the dictionary used to encode that response is deleted from the dictionary store, (2) the SDCH decompression user agent is uninstalled (if it is implemented as a browser add-on), or (3) the SDCH capable user agent is disabled.

Intermediate Caches

The server should use HTTP cache headers that prevent non-SDCH-aware intermediate cache servers from storing the encoded contents. The cache directive "Cache-Control: private" can be used for this purpose.

If the compressed response can be cached by proxy caches, the server must include the HTTP header "Vary: Accept-Encoding, Avail-Dictionary" to alert proxies about sending the cached content only to the user agents who can decode it. Note that some proxies may not respect the Vary header, in which case non-SDCH-capable user agents would end up downloading SDCH-encoded responses. Thus, we recommend that SDCH-encoded responses not be cacheable by intermediate proxies unless there is a very compelling reason. Further, "Vary: Accept-Encoding, Avail-Dictionary" will not match requests unless these headers match exactly.

A proxy cache may provide one of three levels of support for caching SDCH-encoded objects.

1. No support - Never cache any response if the header Vary is present.

2. Basic support - The proxy cache only serves cached SDCH-encoded content if all cache serving conditions are satisfied and the values of the HTTP headers specified in the Vary header of the cached content exactly match the corresponding headers in the HTTP request.

3. Full support - The proxy should understand the SDCH protocol, should know what dictionary is used to encode/decode the

response, and should be able to download advertised dictionaries. The cache needs to have both SDCH user agent and server logic in it. The server should store the SDCH decoded responses in its cache.

Dictionary Caching User Agent Cache

As dictionary payloads may be large compared to the size of individual HTTP responses, in order to maximize latency improvements and minimize the bandwidth overhead of downloading dictionaries, it is recommended that the user agent persistently store dictionaries in a dictionary cache (e.g. on disk). It is suggested that the user agent implement a maximum limit on number of dictionaries stored per domain in order to avoid allowing one domain to force dictionaries for other domains out of the user agent's dictionary cache. To implement a fixed maximum size cache it is recommended that the cache manager first evict the dictionaries that were least recently used for decoding.

Ideally dictionaries will be stored in the same cache as HTTP responses and may be inspected and cleared by the user using existing user interfaces. However, new support may be created to fulfill the need for the user agent to be able to quickly determine which dictionaries should be advertised for a given request.

The user agent should be careful to validate that a dictionary matches its original identifier before being used for decompression to prevent malicious attacks on the dictionary cache. The user agent may implicitly handle this by always recomputing the hash before advertising the dictionary. However, to improve efficiency, the user agent may cache the original digest of the dictionary, advertise the dictionary with that digest, and then only for the dictionary selected by the server to encode the response, verify that the cached dictionary digest still matches the digest computed from the cached dictionary.

The user agent must not evict dictionaries from its dictionary store that have been advertised in the Avail-Dictionary header of a HTTP request for which a response has not yet been returned.

If a user agent downloads a dictionary which has the same identifier as another previously downloaded dictionary which are applicable to the same hosts, the user agent must be careful to either ignore the new dictionary or evict the old dictionary. If the two dictionaries with the same identifier have exactly the same contents the choice is not important, however this indicates a server error as a server must never instruct the user agent to download a dictionary that was advertised by the user agent. The user agent may want to avoid

downloading dictionaries from this server in the future as they may not be new and downloading unnecessary dictionaries can increase latency.

Intermediate Caches

The dictionary should be treated as a regular HTTP response by intermediate proxies. Thus, the normal HTTP caching consideration for intermediate proxies should apply to the dictionary as well.

9. Future Directions

==========

As currently proposed, SDCH is not applicable to another case where differential compression would be beneficial: large files that change infrequently and in small ways, such as JavaScript and CSS files referenced by other HTML documents.

TODO: Re-evaluate dictionary scoping rules, current approach that patterned after the same named attributes from the HTTP State Management Specification [RFC2965] may not be the best choice.

10. Current Status and Updates

For current information about the status of this proposal: https://groups.google.com/group/SDCH

11. IANA Considerations

This document makes no requests of IANA.

12. Security Considerations

Some security considerations are discussed in the data integrity section above, but the author anticipates further work to describe these.

13. Acknowledgements

The authors would like to acknowledge the support of Google, Inc. for the development of this work. Technical editor: Harriett Hardman. Feedback and comments: Greg Badros, Chandra Chereddi, Darren Fisher, Ted Hardie, Ashu Jain, Ian Hickson, Othman Laraki, Jim Roskind, Ryan Sleevi, Lincoln Smith, Randy Smith, and Linus Upson.

14. Appendix: VCDIFF Encoding Format and SDCH

Although the SDCH protocol is proposed so that it could be adapted for use with any differential-encoding format, it currently uses the VCDIFF encoding format. This format was chosen because its definition is publicly available as the RFC 3284 draft standard. The VCDIFF format is independent of the method used for finding the longest possible matches between the dictionary (source) data and the payload (target) data.

An encoder and decoder for the VCDIFF format, intended for use with SDCH, has been released as open-source under the Apache license. This package is called "open-vcdiff". It uses the Bentley/McIlroy technique for finding matches between the dictionary and target data. It conforms to the VCDIFF draft standard, with the following exceptions:

Interleaved format

The VCDIFF draft standard format divides each encoded delta window into three sections (data, instructions, and addresses), with the aim of improving compressibility of the encoded file using a secondary compressor such as gzip. The drawback to this approach is that none of the target data can be reconstructed unless the entire delta window is available. The delta window is received in packets over the network and it is desirable to be able to process its contents as they arrive. In order to facilitate decoding a stream of packets from the network, we have modified the VCDIFF format so that it interleaves the data, instructions, and addresses instead of placing them in three separate sections. Each instruction is followed by its size and then by an address or literal data.

Adler32 checksum

The format can be modified to include an Adler32 checksum [RFC1950] of the target window data. If the checksum format is used, then bit 2 (0x04, defined as VCD_CHECKSUM) of the Win_Indicator byte will be set, and the checksum will appear just after the "Length of addresses for COPYs" field and before the "Data section for ADDs and RUNs" section in the encoding.

Version header byte (Header4)

If either of the two enhancements described above is used, then the resulting format will not conform to the VCDIFF draft standard as described in RFC 3284. In order to indicate this deviation from the standard, the fourth byte in the encoding (Header4, reserved for the VCDIFF version code) will be set to 0x53 (a capital "S" character in

ASCII.) If neither enhancement is used, the fourth byte may be 0x00 (a null character), the default value described in the standard.

VCD_TARGET flag and target COPY instructions not allowed for SDCH

The SDCH protocol is intended to produce a delta between static dictionary data and target data. Secondary compression with gzip will be used to eliminate redundancy within the target data. For this reason, when using VCDIFF for SDCH, the Win_Indicator flag should always include the VCD_SOURCE flag, never the VCD_TARGET flag. COPY instructions should only reference addresses within the source data, never within the previously decoded target.

The Xdelta package (http://xdelta.org) produces a format based on VCDIFF, though not 100% compatible with the RFC draft standard. That package has been released under the GNU General Public License.

15. References

15.1. 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>.

[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>.

15.2. Informative References

[RFC3284] Korn, D., MacDonald, J., Mogul, J., and K. Vo, "The VCDIFF Generic Differencing and Compression Data Format", RFC 3284, DOI 10.17487/RFC3284, June 2002, <http://www.rfc-editor.org/info/rfc3284>.

[RFC3229] Mogul, J., Krishnamurthy, B., Douglis, F., Feldmann, A., Goland, Y., van Hoff, A., and D. Hellerstein, "Delta encoding in HTTP", RFC 3229, DOI 10.17487/RFC3229, January 2002, <http://www.rfc-editor.org/info/rfc3229>.

[RFC3929] Hardie, T., "Alternative Decision Making Processes for Consensus-Blocked Decisions in the IETF", RFC 3929, DOI 10.17487/RFC3929, October 2004, <http://www.rfc-editor.org/info/rfc3929>.

[RFC3548] Josefsson, S., Ed., "The Base16, Base32, and Base64 Data Encodings", RFC 3548, DOI 10.17487/RFC3548, July 2003, <http://www.rfc-editor.org/info/rfc3548>.

[RFC2965] Kristol, D. and L. Montulli, "HTTP State Management Mechanism", RFC 2965, DOI 10.17487/RFC2965, October 2000, <http://www.rfc-editor.org/info/rfc2965>.

[RFC1950] Deutsch, P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, DOI 10.17487/RFC1950, May 1996, <http://www.rfc-editor.org/info/rfc1950>.

[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>.

Authors' Addresses

Jon Butler

Email: jkbutler@google.com

Wei-Hsin Lee

Email: weihsinl@google.com

Bryan McQuade

Email: mcquade@google.com

Kenneth Mixter

Email: kmixter@google.com