Network Working Group
Request for Comments: 3229
Category: Standards Track

J. Mogul
Compaq WRL
B. Krishnamurthy
F. Douglis
AT&T
A. Feldmann
Univ. of Saarbruecken
Y. Goland
A. van Hoff
Marimba
D. Hellerstein
ERS/USDA

January 2002

Delta encoding in HTTP

Status of this Memo

Copyright Notice

Abstract

This document describes how delta encoding can be supported as a compatible extension to HTTP/1.1.

Many HTTP (Hypertext Transport Protocol) requests cause the retrieval of slightly modified instances of resources for which the client already has a cache entry. Research has shown that such modifying updates are frequent, and that the modifications are typically much smaller than the actual entity. In such cases, HTTP would make more efficient use of network bandwidth if it could transfer a minimal description of the changes, rather than the entire new instance of the resource. This is called "delta encoding."

Table of Contents

1 Introduction

The World Wide Web is a distributed system, and so often benefits from caching to reduce retrieval delays. Retrieval of a Web resource (such as a document, image, icon, or applet) over the Internet or other wide-area networks usually takes enough time that the delay is over the human threshold of perception. Often, that delay is measured in seconds. Caching can often eliminate or significantly reduce retrieval delays.

Many Web resources change over time, so a practical caching approach must include a coherency mechanism, to avoid presenting stale information to the user. Originally, the Hypertext Transfer Protocol (HTTP) provided little support for caching, but under operational pressures, it quickly evolved to support a simple mechanism for maintaining cache coherency.

In HTTP/1.0 [2], the server may supply a "last-modified" timestamp with a response. If a client stores this response in a cache entry, and then later wishes to re-use the response, it may transmit a request message with an "If-modified-since" field containing that timestamp; this is known as a conditional retrieval. Upon receiving a conditional request, the server may either reply with a full response, or, if the resource has not changed, it may send an abbreviated reply, indicating that the client's cache entry is still valid. HTTP/1.0 also includes a means for the server to indicate, via an "Expires" timestamp, that a response will be valid until that time; if so, a client may use a cached copy of the response until that time, without first validating it using a conditional retrieval.

HTTP/1.1 [10] adds many new features to improve cache coherency and performance. However, it preserves the all-or-none model for responses to conditional retrievals: either the server indicates that the resource value has not changed at all, or it must transmit the entire current value.

Common sense suggests (and traces confirm), however, that even when a Web resource does change, the new instance is often substantially similar to the old one. If the difference, or "delta", between the two instances could be sent to the client instead of the entire new instance, a client holding a cached copy of the old instance could apply the delta to construct the new version. In a world of finite bandwidth, the reduction in response size and delay could be significant.

One can think of deltas as a way to squeeze as much benefit as possible from client and proxy caches. Rather than treating an entire response as the "cache line", with deltas we can treat arbitrary pieces of a cached response as the replaceable unit, and avoid transferring pieces that have not changed.

This document proposes a set of compatible extensions to HTTP/1.1 that allow clients and servers to use delta encoding with minimal overhead.

We assume that the reader is familiar with the HTTP/1.1 specification.

1.1 Related research and proposals

The idea of delta encoding to reduce communication or storage costs is not new. For example, the MPEG-1 video compression standard transmits occasional still-image frames, but most of the frames sent are encoded (to oversimplify) as changes from an adjacent frame. The SCCS and RCS [27] systems for software version control represent intermediate versions as deltas; SCCS starts with an original version and encodes subsequent ones with forward deltas, whereas RCS encodes previous versions as reverse deltas from their successors. Jacobson's technique for compressing IP and TCP headers over slow links [17] uses a clever, highly specialized form of delta encoding.

In spite of this history, it appears to have taken several years before anyone thought of applying delta encoding to HTTP, perhaps because the development of HTTP caching has been somewhat haphazard. The first published suggestion for delta encoding appears to have been by Williams et al. in a paper about HTTP cache removal policies [30], but these authors did not elaborate on their design until later [29].

The WebExpress project [15] appears to be the first published description of an implementation of delta encoding for HTTP (which they call "differencing"). WebExpress is aimed specifically at wireless environments, and includes a number of orthogonal optimizations. Also, the WebExpress design does not propose changing the HTTP protocol itself, but rather uses a pair of interposed proxies to convert the HTTP message stream into an optimized form. The results reported for WebExpress differencing are impressive, but are limited to a few selected benchmarks.

Banga et al. [1] describe the use of optimistic deltas, in which a layer of interposed proxies on either end of a slow link collaborate to reduce latency. If the client-side proxy has a cached copy of a resource, the server-side proxy can simply send a delta (or a 304

[Not Modified] response). If only the server-side proxy has a cached copy, it may optimistically send its (possibly stale) copy to the client-side proxy, followed (if necessary) by a delta once the server-side proxy has validated its own cache entry with the origin server. The use of optimistic deltas, unlike delta encoding, actually increases the number of bytes sent over the network, in an attempt to improve latency by anticipating a "Not Modified" response from the origin server. The optimistic delta paper, like the WebExpress paper, did not propose a change to the HTTP protocol itself, and reported results only for a small set of selected URLs.

Mogul et al. [23] collected lengthy traces, at two different sites, of the full contents of HTTP messages, to quantify the potential benefits of delta-encoded responses. They showed that delta encoding can provide remarkable improvements in response-size and response-delay for an important subset of HTTP content types. They proposed a set of HTTP extensions, but without the level of detail required for a specification. Douglis et al. [8] used the same sets of full-content traces to quantify the rate at which resources change in the Web.

The HTTP Distribution and Replication Protocol (DRP), proposed to W3C by Marimba, Netscape, Sun, Novell, and At Home, aims to provide a collection of new features for HTTP, to support "the efficient replication of data over HTTP" [13]. One aspect of the DRP proposal is the use of "differential downloading," which is essentially a form of delta encoding. The original DRP proposal uses a different approach than is described here, but a forthcoming revision of DRP will be revised to conform to the proposal in this document.

Tridgell and Mackerras [28] describe the "rsync" algorithm, which accomplishes something similar to delta encoding. In rsync, the client breaks a cache entry into a series of fixed-sized blocks, computes a digest value for each block, and sends the series of digest values to the server as part of its request. The origin server does the same block-based computation, and returns only those blocks whose digest values differ. We believe that it might be possible to support rsync using the "instance manipulation" framework described later in this document, but this has not been worked out in any detail.

2 Goals

The goals of this proposal are:

1. Reduce the mean size of HTTP responses, thereby improving latency and network utilization.

2. Avoid any extra network round trips.

3. Minimize the amount of per-request and per-response overheads.

4. Support a variety of encoding algorithms and formats.

5. Interoperate with HTTP/1.0 and HTTP/1.1.

6. Be fully optional for clients, proxies, and servers.

7. Allow moderately simple implementations.

The goals do not include:

- Reducing the number of HTTP requests sent to an origin server.

- Reducing the size of every HTTP message.

- Increasing the cache-hit ratio of HTTP caches.

- Allowing excessively simplistic implementations of delta encoding.

- Delta encoding of request messages, or of responses to methods other than GET.

  Nothing in this specification specifically precludes the use of a delta encoding for the body of a PUT request. However, no mechanism currently exists for the client to discover if the server can interpret such messages, and so we do not attempt to specify how they might be used.

3 Terminology

HTTP/1.1 [10] defines the following terms:

resource	A network data object or service that can be identified by a URI, as defined in section 3.2. Resources may be available in multiple representations (e.g. multiple languages, data formats, size, resolutions) or vary in other ways.
entity	The information transferred as the payload of a request or response. An entity consists of metainformation in the form of entity-header fields and content in the form of an entity-body, as described in section 7.

variant A resource may have one, or more than one, representation(s) associated with it at any given instant. Each of these representations is termed a ‘variant.’ Use of the term ‘variant’ does not necessarily imply that the resource is subject to content negotiation.

The dictionary definition for "entity" is "something that has separate and distinct existence and objective or conceptual reality" [21]. Unfortunately, the definition for "entity" in HTTP/1.1 is similar to that used in MIME [12], based on a false analogy between MIME and HTTP.

In MIME, electronic mail messages do have distinct and separate existences. MIME defines "entity" as something that "refers specifically to the MIME-defined header fields and contents of either a message or one of the parts in the body of a multipart entity."

In HTTP, however, a response message to a GET does not have a distinct and separate existence. Rather, it reflects the current state of a resource (or a variant, subject to a set of constraints). The HTTP/1.1 specification has no term to describe "the value that would be returned in response to a GET request at the current time for the selected variant of the specified resource." This leads to awkward wordings in the HTTP/1.1 specification in places where this concept is necessary.

To express this concept, we define a new term, for use in this document:

instance The entity that would be returned in a status-200 response to a GET request, at the current time, for the selected variant of the specified resource, with the application of zero or more content-codings, but without the application of any instance manipulations (see below) or transfer-codings.

It is convenient to think of an entity tag, in HTTP/1.1, as being associated with an instance, rather than an entity. That is, for a given resource, two different response messages might include the same entity tag, but two different instances of the resource should never be associated with the same (strong) entity tag.

We will informally use the term "delta," in this document, to mean an HTTP response encoded as the difference between two instances.

More formally, delta encodings are members of a potentially larger class of transformations on instances, leading to this new term:

instance manipulation
: An operation on one or more instances which may result in an instance being conveyed from server to client in parts, or in more than one response message. For example, a range selection or a delta encoding. Instance manipulations are end-to-end, and often involve the use of a cache at the client.

For reasons that will become clear later on, it is convenient to think about subrange selection as a form of instance manipulation. In some contexts, compression might also be treated as an instance manipulation, rather than as a content-coding or transfer-coding.

4 The HTTP message-generation sequence

HTTP/1.1 supports a number of different transformations on the body of a value:

Content-coding
: According to the specification, "Content coding values indicate an encoding transformation that has been or can be applied to an entity. Content codings are primarily used to allow a document to be compressed or otherwise usefully transformed without losing the identity of its underlying media type and without loss of information. Frequently, the entity is stored in coded form, transmitted directly, and only decoded by the recipient." Content-codings are normally end-to-end transformations; i.e., once applied at the sender, they are not removed except at the ultimate recipient. An intermediate server may apply a content-coding, in appropriate circumstances.

Transfer-coding
: According to the specification, "Transfer coding values are used to indicate an encoding transformation that has been, can be, or may need to be applied to an entity-body in order to ensure "safe transport" through the network. This differs from a content coding in that the transfer coding is a property of the message, not of the original entity." Transfer-codings are explicitly hop-by-hop transformations (although, as an optimization, an intermediate proxy may store the transfer-coded version of a message if this behavior is not inconsistent with its externally visible function.)

Ranges An HTTP client, using the Range header, may request that the server return one or more subranges of the instance, rather than the entire instance value. HTTP/1.1 only supports byte-ranges, although there is some possibility that future extensions will allow for other kinds of range-specifiers (such as chapters of a document).

A client signals its willingness to receive a content-coding by sending an "Accept-Encoding" header, listing the set of content-codings that it understands. It may optionally include information about which content-codings it prefers. If a server uses any non-identity content-coding(s), it includes a "Content-Encoding" header field in the response, listing these content-codings in their order of application.

RFC 2068 [9] did not include an analogous mechanism for negotiating the use of transfer-codings, although it does include an analogous "Transfer-Encoding" header for marking the response. A new "TE" header has since been added to HTTP/1.1 [10], analogous to the "Accept-Encoding" header.

In this document, we add new, optional message headers to support the use of instance manipulations. A client signals its willingness to receive an instance-manipulation by sending an "A-IM" header (short for "Accept-Instance-Manipulation", which is far too long to spell out), analogous to the "Accept-Encoding" header. Similarly, a server lists the set of instance-manipulations it has applied using an "IM" header.

One must understand the relationship between these transformations in order to see how delta encoding applies to HTTP responses.

Conceptually, the various transformations are applied in the following sequence:

1. Upon receiving a GET request, the server uses the URI in the request to identify the requested resource.

2. Optionally, it uses information from the request (and perhaps additional information) to select a variant of that resource.

3. At this point, the server may apply a non-identity content-coding to the instance, or one might have been inherent in its generation. This also results in a Content-Encoding header.

4. The result of the first three steps, at the time when the request is processed, is an instance. The instance includes a body (possibly empty) and possibly some instance headers. The entity tag, if any, is assigned at this point. That is, an entity tag is associated with an instance, NOT an entity.

5. The server may then apply an instance-manipulation. For example, if the request included a Range header, the server may optionally produce a range response, consisting of the original set of headers, a Content-Range header, and the appropriate range(s) from the (possibly encoded) body. Delta encodings are instance-manipulations, and are computed at this stage.

6. The result of the fifth step becomes the entity, consisting of entity headers and an entity body.

7. The server may then apply a non-identity transfer-coding; on-the-fly compression could be done in this step. If so, a Transfer-Encoding header is added to the message.

8. The results of the seventh step is the message, consisting of a message body (the transfer-coded version of the entity body), the entity headers, and additional response and general headers.

Note: Section 14.13 of the HTTP/1.1 specification [10] says "The Content-Length entity-header field indicates the size of the entity-body." In other words, Content-Length measures the length of an entity, not of an instance or of a variant. For example, if the message is a delta encoding, Content-Length gives the length of the delta encoding, not the length of the current instance.

Diagrammatically, the sequence is:

```
    datatype        operation leading to next datatype
    ========        ==================================
    resource
                |   choose acceptable variant, if needed
                v
    variant
                |   apply content-coding, if any
                v

                |   compute/assign entity tag
                v
    instance
                |   apply instance manipulation, if any
                v      (delta encoding, range selection, etc.)
    entity-body
                |   apply transfer-coding, if any
                v
    message-body
```

This formalization of the HTTP message generation sequence has not previously been described. However, it is clear that Range selection needs to be done after the entity tag has been assigned and after any content-coding has been applied, and before any transfer-coding is applied. Therefore, this formalization is fully consistent with previous practice and specification.

4.1 Relationship between deltas and ranges

If both Ranges and delta encodings are forms of instance manipulation, which should be applied first? This depends on how the Range is being used.

Ranges are used for two main purposes, at the discretion of the requesting client:

1. to complete a partial response after a premature termination of a message transmission.

2. to obtain just selected sections of an instance.

In the first use of Range, it would have to be applied after any delta encoding, since the intended use is to recover an intact copy of the delta-encoded instance. In the second use of Range, it would have to be applied before any delta encoding, because otherwise the

offsets specified in the Range request would be meaningless (the client generally cannot know how a server's delta encoding maps instance byte offsets to entity byte offsets).

Therefore, we need a mechanism to allow the client to specify the order in which two or more instance-manipulations should be applied. This is easily provided as part of the specification of the "A-IM" header (see section 10.5.3), where we require that the server apply instance-manipulations in the order that they are listed in the "A-IM" header. We also include a "range" literal in the set of registered instance-manipulations, to allow the client to specify (by its ordering with respect to other instance-manipulations) whether range selection is done before or after delta encoding.

We also need a mechanism for the server to indicate in which order two or more instance-manipulations have been applied; this is part of the specification of the "IM" header (see section 10.5.2), where we follow the same practice used for the "Content-Encoding" header: the "IM" header lists the instance-manipulations in the order that were applied (including, perhaps, the special "range" literal).

A similar issue arises when Ranges are combined with compression. If the client is using a Range to complete a partial response after a premature termination of a compressed message, then the Range would have to be applied after the compression. This is feasible in unmodified HTTP/1.1, because the compression can be done as a content-coding. However, if the client is using a Range to obtain selected sections of an instance, it would normally be able to specify offsets only in terms of the uncompressed variant. If the selected portion was large enough to warrant compression, the client could request a compressed transfer-coding, but this is a hop-by-hop transformation and is not the most efficient approach (especially if an HTTP/1.0 proxy is in the path).

We can resolve this issue by supporting the use of compression as an instance-manipulation (as well as as a content-coding or transfer-coding), and by using the new mechanism that allows the client to specify that the compression instance-manipulation is done after the Range instance-manipulation.

This also allows the client to control whether compression is done before or after delta encoding, since some simple differencing algorithms (such as the UNIX "diff" command) require post-compression of their output to yield the best results.

5 Basic mechanisms

In this section, we explain the concepts behind delta encoding. This is not meant as a formal specification of the proposed extensions; see section 10 for that.

5.1 Background: an overview of HTTP cache validation

When a client has a response in its cache, and wishes to ensure that this cache entry is current, HTTP/1.1 allows the client to do a "conditional GET", using one of two forms of "cache validators." In the traditional form, available in both HTTP/1.0 and in HTTP/1.1, the client may use the "If-Modified-Since" request-header to present to the server the "Last-Modified" timestamp (if any) that the server provided with the response. If the server's timestamp for the resource has not changed, it may send a response with a status code of 304 (Not Modified), which does not transmit the body of the resource. If the timestamp has changed, the server would normally send a response with a status code of 200 (OK), which carries a complete copy of the resource, and a new Last-Modified timestamp.

This timestamp-based approach is prone to error because of the lack of timestamp resolution: if a resource changes twice during one second, the change might not be detectable. Therefore, HTTP/1.1 also allows the server to provide an entity tag with a response. An entity tag is an opaque string, constructed by the server according to its own needs; the protocol specification imposes a bare minimum of requirements on entity tags. (In particular, a "strong" entity tag must change if the value of the resource changes.) In this case, the client may validate its cache entry by sending its conditional request using the "If-None-Match" request-header, presenting the entity tag associated with the cached response. (The protocol defines several other ways to transmit entity tags, such as the "If-Range" header, used for short-circuiting an otherwise necessary round trip.) If the presented entity tag matches the server's current tag for the resource, the server should send a 304 (Not Modified) response. Otherwise, the server should send a 200 (OK) response, along with a complete copy of the resource.

In the existing HTTP protocol (HTTP/1.0 or HTTP/1.1), a client sending a conditional request can expect either of two responses:

- status = 200 (OK), with a full copy of the resource, because the server's copy of the resource is presumably different from the client's cached copy.

- status = 304 (Not Modified), with no body, because the server's copy of the resource is presumably the same as the client's cached copy.

Informally, one could think of these as "deltas" of 100% and 0% of the resource, respectively. Note that these deltas are relative to a specific cached response. That is, a client cannot request a delta without specifying, somehow, which two instances of a resource are being differenced. The "new" instance is implicitly the current instance that the server would return for an unconditional request, and the "old" instance is the one that is currently in the client's cache. The cache validator (last-modified time or entity tag) is what is used to communicate to the server the identity of the old instance.

5.2 Requesting the transmission of deltas

In order to support the transmission of actual deltas, an extension to HTTP/1.1 needs to provide these features:

1. A way to mark a request as conditional.

2. A way to specify the old instance, to which the delta will be applied by the client.

3. A way to indicate that the client is able to apply one or more specific forms of delta encoding.

4. A way to mark a response as being delta-encoded in a particular format.

The first two features are already provided by HTTP/1.1: the presence of a conditional request-header (such as "If-Modified-Since" or "If-None-Match") marks a request as conditional, and the value of that header uniquely specifies the old instance (ignoring the problem of last-modified timestamp granularity).

We defer discussion of the fourth feature, until section 5.6.

The third feature, a way for the client to indicate that it is able to apply deltas (aside from the trivial 0% and 100% deltas), can be accomplished by transmitting a list of acceptable delta-encoding formats in a request-header field; specifically, the "A-IM" header. The presence of this list in a conditional request indicates that the client is able to apply delta-encoded cache updates.

For example, a client might send this request:

```
GET /foo.html HTTP/1.1
Host: bar.example.net
If-None-Match: "123xyz"
A-IM: vcdiff, diffe, gzip
```

The meaning of this request is that:

- The client wants to obtain the current value of /foo.html.
- It already has a cached response (instance) for that resource, whose entity tag is "123xyz".
- It is willing to accept delta-encoded updates using either of two formats, "diffe" (i.e., output from the UNIX "diff -e" command), and "vcdiff". (Encoding algorithms and formats, such as "vcdiff", are described in section 6.)
- It is willing to accept responses that have been compressed using "gzip," whether or not these are delta-encoded. (It might be useful to compress the output of "diff -e".) However, based on the mandatory ordering constraint specified in section 10.5.3, if both delta encoding and compression are applied, then this "A-IM" request header specifies that compression should be done last.

If, in this example, the server's current entity tag for the resource is still "123xyz", then it should simply return a 304 (Not Modified) response, as would a traditional server.

If the entity tag has changed, presumably but not necessarily because of a modification of the resource, the server could instead compute the delta between the instance whose entity tag was "123xyz" and the current instance.

We defer discussion of what the server needs to store, in order to compute deltas, until section 7.

We note that if a client indicates it is willing to accept deltas, but the server does not support this form of instance-manipulation, the server will simply ignore this aspect of the request. (HTTP always allows an implementation to ignore a header that is not required by a specification that the implementation complies with, and the specification of "A-IM" allows the server to ignore an instance-manipulation it does not understand.) So if a server either does not implement the A-IM header at all, or does not implement any

of the instance manipulations listed in the A-IM header, it acts as if the client had not requested a delta-encoded response: the server generates a status-200 response.

5.3 Choice of delta algorithm and format

The server is not required to transmit a delta-encoded response. For example, the result might be larger than the current size of the resource. The server might not be able to compute a delta for this type of resource (e.g., a compressed binary format); the server might not have sufficient CPU cycles for the delta computation; the server might not support any of the delta formats supported by the client; or, the network bandwidth might be high enough that the delay involved in computing the delta is not worth the delay avoided by sending a smaller response.

However, if the server does want to compute a delta, and the set of encodings it supports has more than one encoding in common with the set offered by the client, which encoding should it use? This is mostly at the option of the server, although the client can express preferences using "Quality Values" (or "qvalues") in the "A-IM" header. The HTTP/1.1 specification [10] describes qvalues in more detail. (Clients may prefer one delta encoding format over another that generates a smaller encoding, if the decoding costs for the first format are lower and the client is resource-constrained.)

Server implementations have a number of possible approaches. For example, if CPU cycles are plentiful and network bandwidth is scarce, the server might compute each of the possible encodings and then send the smallest result. Or the server might use heuristics to choose an encoding format, based on things such as the content-type of the resource, the current size of the resource, and the expected amount of change between instances of the resource.

Note that it might pay to cache the deltas internally to the server, if a resource is typically requested by several different delta-capable clients between modifications. In this case, the cost of computing a delta may be amortized over many responses, and so the server might use a more expensive computation.

5.4 Identification of delta-encoded responses

A response using delta encoding must be identified as such. This is done using the "IM" response-header, specified in section 10.5.2.

However, a simplistic application of this approach would cause serious problems if a delta-encoded response flows through an intermediate (proxy) cache that is not cognizant of the delta

mechanism. Because the Internet still includes a significant number of HTTP/1.0 caches, which might never be entirely replaced, and because the HTTP specifications insist that message recipients ignore any header field that they do not understand, a non-delta-capable proxy cache that receives a delta-encoded response might store that response, and might later return it to a non-delta-capable client that has made a request for the same resource. This naive client would believe that it has received a valid copy of the entire resource, with predictably unpleasant results.

To solve this problem, we propose that delta-encoded responses (actually, all instance-manipulated responses) be identified as such using a new HTTP status code. For specificity in the discussion that follows, we will use the (currently unassigned) code of 226, with a reason phrase of "IM Used". (We see no benefit in spelling out the words "Instance Manipulation Used," since this requires the transmission of unnecessary bytes, and this Reason-phrase should not normally be seen by human users.) There is some precedent for this approach: the HTTP/1.1 specification introduces the 206 (Partial Content) status code, for the transmission of sub-ranges of a resource. Existing proxies apparently forward responses with unknown status codes, and do not attempt to cache them.

An alternative to using a new status code would be to use the "Expires" header to prevent HTTP/1.0 caches from storing the response, then use "Cache-Control: max-age" (defined in HTTP/1.1) to allow more modern caches to store delta-encoded responses. This adds many bytes to the response headers, and so would reduce the effectiveness of delta encoding. It is also not entirely clear that this approach suppresses all caching by all HTTP/1.0 proxies.

> We were reluctant to define an additional status code as part of the support for delta encoding. However, we see no other efficient way to remain compatible with the deployed base of HTTP/1.0 cache implementations.

5.5 Guaranteeing cache safety

Although we are not aware of any HTTP/1.1 proxy implementations that would attempt to cache a response with an unknown 2xx status code, the HTTP/1.1 specification does allow this behavior if the response carries an Expires or Cache-Control header field that explicitly allows caching. This would present a problem when a 226 (IM Used) response carries such headers.

The solution in that case is to exploit the Cache Control Extensions mechanism from the HTTP/1.1 specification. We define a new cache-directive, "im", which indicates that the "no-store" cache-directive may be ignored by implementations that conform to the specification for the IM and A-IM headers.

For example, this response:

```
HTTP/1.1 226 IM Used
ETag: "489uhw"
IM: vcdiff
Date: Tue, 25 Nov 1997 18:30:05 GMT
Cache-Control: no-store, im, max-age=30

...
```

"MUST NOT" be stored by a cache that complies with the HTTP/1.1 specification (which states that the max-age cache-directive "implies that the response is cacheable [...] unless some other, more restrictive cache directive is also present."). However, a cache that does comply with the specification for the im cache-directive (i.e., a cache that complies with the specification for the A-IM and IM header fields, and the 226 status code) ignores the no-store directive, and therefore sees the max-age directive as allowing caching.

> We are not entirely sure that all HTTP/1.1 caches obey the rule that the max-age directive is overridden by the no-store directive. If operational testing reveals this to be a problem, more elaborate solutions are possible.

Warning to origin server implementors: it does not suffice to send

```
Vary: If-None-Match, A-IM
```

in status-226 responses. We have discovered at least one scenario where this does not prevent a proxy cache that does not implement IM and A-IM from incorrectly "validating" a cached 226 response.

5.6 Transmission of delta-encoded responses

A delta-encoded response differs from a standard response in four ways:

1. It carries a status code of 226 (IM Used).

2. It carries an "IM" response-header field, indicating which delta encoding is used in this response.

3. Its message-body is a delta encoding of the current instance, rather than a full copy of the instance.

4. It might carry several other new headers, as described later in this document.

For example, a response to the request given in section 5.2 might look like:

```
HTTP/1.1 226 IM Used
ETag: "489uhw"
IM: vcdiff
Date: Tue, 25 Nov 1997 18:30:05 GMT

...
```

(We do not show the actual contents of the response body, since this is a binary format.)

Note: the Etag header in a 226 response with a delta encoding provides the entity tag of the current instance of the resource variant. It is not meaningful to associate an entity tag with the delta value, which is not an instance.

5.7 Examples of requests combining Range and delta encoding

In the example used in section 5.2, the client sends:

```
GET /foo.html HTTP/1.1
Host: bar.example.net
If-None-Match: "123xyz"
A-IM: vcdiff, diffe, gzip
```

and the server either responds with a 304 (Not Modified) response, or with the appropriate delta encoding.

Here are a few more examples, to clarify how the client request should be interpreted.

If the client sends

```
GET /foo.html HTTP/1.1
Host: bar.example.net
If-None-Match: "123xyz"
A-IM: vcdiff, diffe, gzip, range
Range: bytes=0-99
```

then the meaning is the same as in the example above, except that after the delta encoding (and compression, if any) is computed, the server then returns only the first 100 bytes of the output of the delta encoding. (If it is shorter than 100 bytes, the entire delta encoding is returned.) Because the "range" token appears last in the "A-IM" header, this tells the origin server to apply any range selection after the other instance-manipulations.

The interaction between the If-Range mechanism and delta encoding is somewhat complex. (If-Range means, informally, "if the entity is unchanged, send me the part(s) that I am missing; otherwise, send me the entire new entity.") Here is an example that should clarify the use of this combination.

Suppose that the client wants to have the complete current instance of http://bar.example.net/foo.html. It already has a (complete) cache entry for this URI, with entity tag "A", so it issues this request:

```
GET /foo.html HTTP/1.1
host: bar.example.net
If-None-Match: "A"
A-IM: vcdiff
```

Suppose that the server's current instance has entity tag "B", and that the server also has retained a copy of the instance with entity tag "A". Then, the server could compute the difference between "B" and "A", and respond with:

```
HTTP/1.1 226 IM Used
Etag: "B"
IM: vcdiff
Date: Tue, 25 Nov 1997 18:30:05 GMT
Content-Length: 1000

...
```

but the network connection is terminated after the client has received exactly 900 bytes of the message body for the delta-encoded content.

The client wants to retrieve the remaining 100 bytes of the delta encoding that was being sent in the interrupted response. It therefore should send:

```
GET /foo.html HTTP/1.1
host: bar.example.net
If-None-Match: "A"
If-Range: "B"
A-IM: vcdiff,range
Range: bytes=900-
```

This rather elaborate request has a well-defined meaning, which depends on the current entity tag Tcur of the instance when the server receives the request:

Tcur = "A" (i.e., for some reason, the instance has reverted to the value already in the client's cache). The server should return a 304 (Not Modified) response, as required by the HTTP/1.1 specification for "If-None-Match".

Tcur = "B" (i.e., the instance has not changed again). The HTTP/1.1 specification for "If-None-Match", in this case, is that the header field is ignored (by a server that does not understand delta encoding). Therefore, this is equivalent to the client's previous request, except that the Range selection is applied after the vcdiff instance manipulation (if both are to be applied). So the (delta-aware) server again computes the delta between the "A" instance and the "B" instance (or uses a cached computation of the delta), then applies the Range selection, and returns a 226 (IM Used) response, with an message-body containing bytes 900 to 999 of the result of the vcdiff encoding, with an "IM:vcdiff,range" response header.

Tcur = "C" (i.e., the instance has changed again). In this case, the HTTP/1.1 specification for "If-None-Match" again means that this is equivalent to an unconditional request for the current instance. The specification for "If-Range" requires the server to return the entire current instance. However, a delta-aware server can construct the delta between the "A" instance described by the "If-None-Match" field and the current ("C") instance, and return a 226 (IM Used) response, with an "IM:vcdiff" response header.

If the client's request had not included the "If-None-Match: "A"" header field, the server could not have computed a delta, since it would not have known which entire instance was already available to

the client. If the request had not included the "If-Range: "B"" header field, the server could not have distinguished between the latter two cases (Tcur = "B" or Tcur = "C") and would not have been able to apply the Range selection to the result of delta encoding.

On the other hand, suppose that the client has a cache entry for the "A" instance of http://bar.example.net/foo.html, and it has already received the first 900 bytes of a new instance "B" (perhaps as the result of an aborted transfer). Now the client wants to receive the entire current instance, so it could send this request:

```
GET /foo.html HTTP/1.1
host: bar.example.net
If-None-Match: "A"
If-Range: "B"
A-IM: range,vcdiff
Range: bytes=900-
```

In this example, as in the previous example, if Tcur = "A" then the server should send 304 (Not Modified), and if Tcur = "C", then the server should send the entire new instance, either as a 200 response or as a delta encoding against instance "A".

However, if Tcur = "B", in this case the server should first select the specified range (bytes 900 through the end) from both instances "A" and "B", then compute the delta encoding between these ranges (using vcdiff), and then transmit the result using a 226 (IM Used) response with an "IM:range,vcdiff" response header.

6 Encoding algorithms and formats

A number of delta encoding algorithms and formats have been described in the literature:

diff -e — The UNIX "diff" program is ubiquitously available, and is relatively fast for both encoding and decoding (decoding is actually done using the "ed" program). However, the size of the resulting deltas is relatively large. This algorithm can only be used on text-format files.

diff -e | gzip — Running the output of "diff" through a compression algorithm such as "gzip" [5] (or, perhaps better, "deflate" [7, 6]) yields a more compact encoding, but the costs of encoding and decoding are much higher than for "diff" by itself. This algorithm can only be used on text-format files.

vcdiff (vdelta) The algorithm that generates the "vcdiff" format [19, 20] inherently compresses its output, and generally produces smaller results than the combination of "diff" and "gzip". The algorithm also runs much faster, and can be applied to binary-format input. The "vcdiff" format is based on previous work on an algorithm named "vdelta." (Note that the "vcdiff" format can be used either for delta encoding or as a compressed format, so two different instance-manipulation values would have to be registered in order to distinguish these two uses, should its use as a compressed format be adopted.) The most recent published study suggests that "vdelta" is the best overall delta algorithm [16].

gdiff The gdiff format [14] was specified as a generic, algorithm-independent format for expressing deltas. Because it is more generic it is easy to implement, but it may not be the most compact encoding format.

Our proposal does not recommend any specific algorithm or format, but rather encourages client and server implementors to choose the most appropriate one(s). However, to avoid the possibility of excessively long "A-IM" headers, we suggest that, after some period of experimentation, it might be reasonable to specify a "recommended" set of delta formats for general-purpose HTTP implementations.

We suspect that it should be possible to devise a delta encoding algorithm appropriate for use on typical image encodings, such as GIF and JPEG. Although experiments with vdelta have not shown much potential [23], this may simply be because these experiments used vdelta directly on the already-compressed forms of these encodings. However, it might be necessary to devise a delta encoding algorithm that is aware of the two-dimensional nature of images. We have some expectation that this is possible, since MPEG compression relies on computing deltas between successive frames of a video stream.

7 Management of base instances

If the time between modifications of a resource is less than the typical eviction time for responses in client caches, this means that the "old instance" indicated in a client’s conditional request might not refer to the most recent prior instance. This raises the question of how many old instances of a resource should be maintained by the server, if any. We call these old instances "base instances."

There are many possible options for server implementors. For example:

- The server might not store any old instances, and so would never respond with a delta.

- The server might only store the most recent prior instance; requests attempting to validate this instance could be answered with a delta, but requests attempting to validate older instances would be answered with a full copy of the resource.

- The server might store all prior instances, allowing it to provide a delta response for any client request.

- The server might store only a subset of the prior instances. The use of a Least Recently Used (LRU) algorithm to determine this kind of subset has proved effective in some similar circumstances, such as cache replacement.

The server might not have to store prior instances explicitly. It might, instead, store just the deltas between specific base instances and subsequent instances (or the inverse deltas between base instances and prior instances). This approach might be integrated with a cache of computed deltas.

None of these approaches necessarily requires additional protocol support. However, if a server administrator wants to store only a subset of the prior instances, but would like the server to be able to respond using deltas as often as possible, then the client needs some additional information. Otherwise, the client's "If-None-Match" header might specify a base instance not stored at the server, even though an appropriate base instance is held in the client's cache.

We identify two additional protocol changes to help solve this problem.

7.1 Multiple entity tags in the If-None-Match header

Although the examples we have given so far show only one entity tag in an "If-None-Match" header, the HTTP/1.1 specification allows the header to carry more than one entity-tag. This feature was included in HTTP/1.1 to support efficient caching of multiple variants of a resource, but it is not restricted to that use.

Suppose that a client has kept more than one instance of a resource in its cache. That is, not only does it keep the most recent instance, but it also holds onto copies of one or more prior, invalid instances. (Alternatively, it might retain sufficient delta or

inverse-delta information to reconstruct older instances.) In this case, it could use its conditional request to tell the server about all of the instances it could apply a delta to. For example, the client might send:

```
GET /foo.html HTTP/1.1
host: bar.example.net
If-None-Match: "123xyz", "337pey", "489uhw"
A-IM: vcdiff
```

to indicate that it has three instances of this resource in its cache. If the server is able to generate a delta from any of these prior instances, it can select the appropriate base instance, compute the delta, and return the result to the client.

In this case, however, the server must also tell the client which base instance to use, and so we need to define a response header, named "Delta-Base", for this purpose. For example, the server might reply:

```
HTTP/1.1 226 IM Used
ETag: "1acl059"
IM: vcdiff
Delta-Base: "337pey"
Date: Tue, 25 Nov 1997 18:30:05 GMT
```

This response tells the client to apply the delta to the cached response with entity tag "337pey", and to associate the entity tag "1acl059" with the result.

Of course, if the server has retained more than one of the prior instances identified by the client, this could complicate the problem of choosing the optimal delta to return, since now the server has a choice not only of the delta format, but also of the base instance to use.

7.2 Hints for managing the client cache

Support for multiple entity tags in choosing the base instance implies that a client might benefit from storing multiple old instances of a resource in its cache. A client with finite space would not want to keep all old instances, so it must manage its cache for maximal effectiveness by saving those instances most likely to be useful for future deltas. Although this could be accomplished using information purely local to the client (e.g., an LRU algorithm), certain "hint" information from the server could improve the client's ability to manage its cache. The use of hints for improving Web cache performance has been described previously [4, 22].

If the server intends to retain certain instances and not others, it can label the responses that transmit the retained instances. This would help the client manage its cache, since it would not have to retain all prior instances on the possibility that only some of them might be useful later. The label is a hint to the client, not a promise that the server will indefinitely retain an instance.

We propose adding a new directive to the existing "Cache-Control" header for this purpose, named "retain". For example, in response to an unconditional request, the server might send:

```
HTTP/1.1 200 OK
ETag: "337pey"
Date: Tue, 25 Nov 1997 18:30:05 GMT
Cache-Control: retain
```

to suggest that a delta-capable client should retain this instance. The "retain" directive could also appear in a delta response, referring to the current instance:

```
HTTP/1.1 226 IM Used
ETag: "1acl059"
Date: Tue, 25 Nov 1997 18:30:05 GMT
Cache-Control: retain
IM: vcdiff
Delta-Base: "337pey"
```

The "retain" directive includes an optional timeout parameter, which the server can use if it expects to delete an old base instance at a particular time. For example,

```
HTTP/1.1 200 OK
ETag: "337pey"
Date: Tue, 25 Nov 1997 18:30:05 GMT
Cache-Control: retain=3600
```

means that the server intends to retain this base instance for one hour.

Another situation where a server can provide a hint to a client is where the server supports the delta mechanism in general, but does not intend to provide delta-encoded responses for a particular resource. By sending a "retain=0" directive, it indicates that the client should not waste request-header bytes attempting to obtain a delta-encoded response using this base instance (and, by implication, for this resource). It also indicates that the client ought not waste cache space on this instance after it has become stale. To

avoid wasting response-header bytes, a server ought not send "retain=0", except in reply to a request that attempts to obtain a delta-encoded response.

Note that the "retain" directive is orthogonal to the "max-age" directive. The "max-age" directive indicates how long a cache entry remains fresh (i.e.,can be used without contacting the origin server for revalidation); the "retain" directive is of interest to a client AFTER the cache entry has become stale.

In practice, the "Cache-Control" response-header field might already be present, so the cost (in bytes) of sending this directive might be smaller than these examples implies.

8 Deltas and intermediate caches

Although we have designed the delta-encoded responses so that they will not be stored by naive proxy caches, if a proxy does understand the delta mechanism, it might be beneficial for it to participate in sending and receiving deltas.

A proxy could participate in several independent ways:

- In addition to forwarding a delta-encoded response, the proxy might store it, and then use it to reply to a subsequent request with a compatible "If-None-Match" field (i.e., one that is either a superset of the corresponding field of the request that first elicited the response, or one that includes the "Delta-Base" value in the cached response), and with a compatible "IM" response-header field (one that includes the actual delta-encoding format used in the response.) Of course, such uses are subject to all of the other HTTP rules concerning the validity of cache entries.

- In addition to forwarding a delta-encoded response, the proxy might apply the delta to the appropriate entry in its own cache, which could then be used for later responses (even from non-delta-capable clients).

- When the proxy receives a conditional request from a delta-capable client, and the proxy has a complete copy of an up-to-date ("fresh," in HTTP/1.1 terminology) response in its cache, it could generate a delta locally and return it to the requesting client.

- When the proxy receives a request from a non-delta-capable client, it might convert this into a delta request before forwarding it to the server, and then (after applying a

resulting delta response to one of its own cache entries) it would return a full-body response to the client (or a response with status code 206 or 304, as appropriate).

All of these optional techniques increase proxy software complexity, and might increase proxy storage or CPU requirements. However, if applied carefully, they should help to reduce the latencies seen by end users, and load on the network. Generally, CPU speed and disk costs are improving faster than network latencies, so we expect to see increasing value available from complex proxy implementations.

9 Digests for data integrity

When a recipient reassembles a complete HTTP response from several individual messages, it might be necessary to check the integrity of the complete response. For example, the client's cache might be corrupt, or the implementation of delta encoding (either at client or server) might have a bug.

HTTP/1.1 includes mechanisms for ensuring the integrity of individual messages. A message may include a "Content-MD5" response header, which provides an MD5 message digest of the body of the message (but not the headers). The Digest Authentication mechanism [11] provides a similar message-digest function, except that it includes certain header fields. Neither of these mechanisms makes any provision for covering a set of data transmitted over several messages, as would be the case for the result of applying a delta-encoded response (or, for that matter, a Range response).

Data integrity for reassembled messages requires the introduction of a new message header. Such a mechanism is proposed in a separate document [24]. One might still want to use the Digest Authentication mechanism, or something stronger, to protect delta messages against tampering.

10 Specification

In this specification, the key words "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY" are to be interpreted as described in RFC 2119 [3].

10.1 Protocol parameter specifications

This specification defines a new HTTP parameter type, an instance-manipulation:

```
instance-manipulation = token [imparams]

imparams = ";" imparam-name [ "=" ( token | quoted-string ) ]
imparam-name = token
```

Note that the imparam-name MUST NOT be "q", to avoid ambiguity with the use of qvalues (see [10]).

The set of instance-manipulation values is initially:

- vcdiff
  A delta using the "vcdiff" encoding format [19, 20].

- diffe
  The output of the UNIX "diff -e" command [26].

- gdiff
  The GDIFF encoding format [14].

- gzip
  Same definition as the HTTP "gzip" content-coding.

- deflate
  Same definition as the HTTP "deflate" content-coding.

- range
  A token indicating that the result is partial content, as the result of a range selection.

- identity
  A token used only in the A-IM header (not in the IM header), to indicate whether or not the identity instance-manipulation is acceptable.

For convenience in the rest of this specification, we define a subset of instance-manipulation values as delta-coding values:

```
delta-coding = "vcdiff" | "diffe" | "gdiff" | token
```

Future instance-manipulation values might also be included in this list.

10.2 IANA Considerations

The Internet Assigned Numbers Authority (IANA) administers the name space for instance-manipulation values. Values and their meaning must be documented in an RFC or other peer-reviewed, permanent, and readily available reference, in sufficient detail so that interoperability between independent implementations is possible. Subject to these constraints, name assignments are First Come, First Served (see RFC 2434 [25]).

This specification also inserts a new value in the IANA HTTP Status Code Registry (see RFC 2817 [18]). See section 10.4.1 for the specification of this code.

10.3 Basic requirements for delta-encoded responses

A server MAY send a delta-encoded response if all of these conditions are true:

1. The server would be able to send a 200 (OK) response for the request.

2. The client’s request includes an A-IM header field listing at least one delta-coding.

3. The client’s request includes an If-None-Match header field listing at least one valid entity tag for an instance of the Request-URI (a "base instance").

A delta-encoded response:

- MUST carry a status code of 226 (IM Used).

- MUST include an IM header field listing, at least, the delta-coding employed.

- MAY include a Delta-Base header field listing the entity tag of the base-instance.

10.4 Status code specifications

The following new status code is defined for HTTP.

10.4.1 226 IM Used

The server has fulfilled a GET request for the resource, and the response is a representation of the result of one or more instance-manipulations applied to the current instance. The actual current instance might not be available except by combining this response with other previous or future responses, as appropriate for the specific instance-manipulation(s). If so, the headers of the resulting instance are the result of combining the headers from the status-226 response and the other instances, following the rules in section 13.5.3 of the HTTP/1.1 specification [10].

The request MUST have included an A-IM header field listing at least one instance-manipulation. The response MUST include an Etag header field giving the entity tag of the current instance.

A response received with a status code of 226 MAY be stored by a cache and used in reply to a subsequent request, subject to the HTTP expiration mechanism and any Cache-Control headers, and to the requirements in section 10.6.

A response received with a status code of 226 MAY be used by a cache, in conjunction with a cache entry for the base instance, to create a cache entry for the current instance.

10.5 Header specifications

The following headers are defined, for use as entity-headers. (Due to the terminological confusion discussed in section 3, some entity-headers are more properly associated with instances than with entities.)

10.5.1 Delta-Base

The Delta-Base entity-header field is used in a delta-encoded response to specify the entity tag of the base instance.

```
Delta-Base = "Delta-Base" ":" entity-tag
```

A Delta-Base header field MUST be included in a response with an IM header that includes a delta-coding, if the request included more than one entity tag in its If-None-Match header field.

Any response with an IM header that includes a delta-coding MAY include a Delta-Base header.

> We are not aware of other cases where a delta-encoded response MUST or SHOULD include a Delta-Base header, but we have not done an exhaustive or formal analysis. Implementors might be wise to include a Delta-Base header in every delta-encoded response.

A cache or proxy that receives a delta-encoded response that lacks a Delta-base header MAY add a Delta-Base header whose value is the entity tag given in the If-None-Match field of the request (but only if that field lists exactly one entity tag).

10.5.2 IM

The IM response-header field is used to indicate the instance-manipulations, if any, that have been applied to the instance represented by the response. Typical instance manipulations include delta encoding and compression.

```
IM = "IM" ":" #(instance-manipulation)
```

Instance-manipulations are defined in section 10.1.

As a special case, if the instance-manipulations include both range selection and at least one other non-identity instance-manipulation, the IM header field MUST be used to indicate the order in which all of these instance-manipulations, including range selection, were applied. If the IM header lists the "range" instance-manipulation, the response MUST include either a Content-Range header or a multipart/byteranges Content-Type in which each part contains a Content-Range header. (See section 10.10 for specific discussion of combining delta encoding and multipart/byteranges.)

Responses that include an IM header MUST carry a response status code of 226 (IM Used), as specified in section 10.4.1.

The server SHOULD omit the IM header if it would list only the "range" instance-manipulation. Such responses would normally be sent with response status code 206 (Partial Content), as specified by HTTP/1.1 [10].

Examples of the use of the IM header include:

```
IM: vcdiff
```

This example indicates that the entity-body is a delta encoding of the instance, using the vcdiff encoding.

```
IM: diffe, deflate, range
```

This example indicates that the instance has first been delta-encoded using the diffe encoding, then the result of that has been compressed using deflate, and finally one or more ranges of that compressed encoding have been selected.

```
IM: range, vcdiff
```

This example indicates that one or more ranges of the instance have been selected, and the result has then been delta encoded against identical ranges of a previous base instance.

A cache using a response received in reply to one request to reply to a subsequent request MUST follow the rules in section 10.6 if the cached response includes an IM header field.

10.5.3 A-IM

The A-IM request-header field is similar to Accept, but restricts the instance-manipulations (section 10.1) that are acceptable in the response. As specified in section 10.5.2, a response may be the result of applying multiple instance-manipulations.

```
A-IM = "A-IM" ":" #( instance-manipulation
                          [ ";" "q" "=" qvalue ] )
```

When an A-IM request-header field includes one or more delta-coding values, the request MUST contain an If-None-Match header field, listing one or more entity tags from prior responses for the request-URI.

A server tests whether an instance-manipulation (among the ones it is capable of employing) is acceptable, according to a given A-IM header field, using these rules:

1. If the instance-manipulation is listed in the A-IM field, then it is acceptable, unless it is accompanied by a qvalue of 0. (As defined in section 3.9 of the HTTP/1.1 specification [10], a qvalue of 0 means "not acceptable.") A server MUST NOT use a non-identity instance-manipulation for a response unless the instance-manipulation is listed in an A-IM header in the request.

2. If multiple but incompatible instance-manipulations are acceptable, then the acceptable instance-manipulation with the highest non-zero qvalue is preferred.

3. The "identity" instance-manipulation is always acceptable, unless specifically refused because the A-IM field includes "identity;q=0".

If an A-IM field is present in a request, and if the server cannot send a response which is acceptable according to the A-IM header, then the server SHOULD send an error response with the 406 (Not Acceptable) status code.

If a response uses more than one instance-manipulation, the instance-manipulations MUST be applied in the order in which they appear in the A-IM request-header field.

The server's choice about whether to apply an instance-manipulation SHOULD be independent of its choice to apply any subsequent two-input instance-manipulations to the response. (Two-input instance-manipulations include delta-codings, because they take two different values as input. Compression and "range" instance-manipulations take only one input. Other instance-manipulations may be defined in the future.)

Note: the intent of this requirement is to prevent the server from generating a delta-encoded response that the client can only decode by first applying an instance-manipulation encoding to its cached base instance. A server implementor might wish to consider what the client would logically have in its cache, when deciding which instance-manipulations to apply prior to a delta-coding.

Examples:

A-IM: vcdiff, gdiff

This example means that the client will accept a delta encoding in either vcdiff or gdiff format.

A-IM: vcdiff, gdiff;q=0.3

This example means that the client will accept a delta encoding in either vcdiff or gdiff format, but prefers the vcdiff format.

A-IM: vcdiff, diffe, gzip

This example means that the client will accept a delta encoding in either vcdiff or diffe format, and will accept the output of the delta encoding compressed with gzip. It also means that the client will accept a gzip compression of the instance, without any delta encoding, because A-IM provides no way to insist that gzip be used only if diffe is used.

It is left to the server implementor to choose useful combinations of acceptable instance-manipulations (for example, following diffe by gzip is useful, but following vcdiff by gzip probably is not useful).

10.6 Caching rules for 226 responses

When a client or proxy receives a 226 (IM Used) response, it MAY use this response to create a cache entry in three ways:

1. It MAY decode all of the instance-manipulations to recover the original instance, and store that instance in the cache. In this case, the recovered instance is stored as a status-200 response, and MUST be used in accordance with the normal HTTP caching rules.

2. It MAY decode all of the instance-manipulations except for range selection(s), and store the result in the cache. In this case, the result is stored as a status-206 response, and MUST be used in accordance with the normal HTTP caching rules for Partial Content.

3. It MAY store the status-226 (IM Used) response as a cache entry.

A status-226 cache entry MUST NOT be used in response to a subsequent request under any of these conditions (a cache that never stores status-226 responses may ignore these tests):

1. If any of the instance-manipulation values from the IM header field in the cached response do not appear in the subsequent request's A-IM header field. The comparison between the headers is done using an exact match on each instance-manipulation value including any associated imparams values (see section 10.1).

2. If the order of instance-manipulation values appearing in the cached IM header field differs from the order of that set of instance-manipulations in the A-IM header field of the subsequent request.

3. If the cache implementation is not aware of, or is not at least conditionally compliant with, the specification of any of the instance-manipulation values in the cached IM header field.

Note: This rule allows for extending the set of instance-manipulations without causing deployed cache implementations to commit errors. The specification of new instance-manipulations may include additional caching rules to improve cache-hit rates in cognizant implementations.

4. If any of the instance-manipulation values in the cached IM header field is a delta-coding, and the cache entry includes a Delta-Base header field, and that Delta-Base entity tag is not one of the entity tags listed in an If-None-Match header field of the subsequent request.

5. If any of the instance-manipulation values in the cached IM header field is a delta-coding, the cache entry does not include a Delta-Base header field, and the If-None-Match header field of the request that led to that cache entry does not match the If-None-Match header field of the subsequent request.

If the IM header field of the cached response includes the "range" instance-manipulation, then a status-226 cache entry MUST NOT be used in response to a subsequent request if the cached response is inconsistent with the Range header field value(s) in the request, as would be the case for a cached 206 (Partial Content) response.

Note: we know of no existing, published formal specification for deciding if a cached status-206 response is consistent with a subsequent request. We believe that either of these conditions is sufficient:

1. The ranges specified in the headers of the request that led to the cached response are the same as specified in the headers of the subsequent request.

2. The ranges specified in the cached response are the same as specified in the headers of the subsequent request.

Further analysis might be necessary.

10.7 Rules for deltas in the presence of content-codings

The use of delta encoding with content-encoded instances adds some slight complexity. When a client (perhaps a proxy) has received a delta encoded response, either or both of that new response and a cached previous response may have non-identity content-codings. We specify rules for the server and client, to prevent situations where the client is unable to make sense of the server’s response.

10.7.1 Rules for generating deltas in the presence of content-codings

When a server generates a delta-encoded response, the list of content-codings the server uses (i.e., the value of the response's Content-Encoding header field) SHOULD be a prefix of the list of content-codings the server would have used had it not generated a delta encoding.

This requirement allows a client receiving a delta-encoded response to apply the delta to a cached base instance without having to apply any content-codings during the process (although the client might, of course, be required to decode some content-codings).

10.7.2 Rules for applying deltas in the presence of content-codings

When a client receives a delta response with one or more non-identity content codings:

1. If both the new (delta) response and the cached response (instance) have exactly the same set of content-codings, the client applies the delta response to the cached response without removing the content-codings from either response.

2. If the new (delta) response and the cached response have a different set of content-codings, before applying the delta the client decodes one or more content-codings from the cached response, until the result has the same set of content-codings as the delta response.

3. If a proxy or cache is forwarding the result of applying the delta response to a cached base instance response, or later forwards this result from a cache entry, the forwarded response MUST carry the same Content-Encoding header field as the new (delta) response (and so it must be content-encoded as indicated by that header field).

The intent of these rules (and in particular, rule #3) is that the results are always consistent with the rule that the entity tag is associated with the result of the content-coding, and that any recipient after the application of the delta-coding receives exactly the same response it would have received as a status-200 response from the origin server (without any delta-coding).

10.7.3 Examples for using A-IM, IM, and content-codings

Suppose a client, with an empty cache, sends this request:

```
GET /foo.html HTTP/1.1
Host: example.com
Accept-encoding: gzip
```

and the origin server responds with:

```
HTTP/1.1 200 OK
Date: Wed, 24 Dec 1997 14:00:00 GMT
Etag: "abc"
Content-encoding: gzip
```

We will use the notation URI;entity-tag to denote specific instances, so this response would cause the client to store in its cache the entity GZIP(foo.html;"abc").

Then suppose that the client, a minute later, issues this conditional request:

```
GET /foo.html HTTP/1.1
Host: example.com
If-none-match: "abc"
Accept-encoding: gzip
A-IM: vcdiff
```

If the server is able to generate a delta-encoded response, it might choose one of two alternatives. The first is to compute the delta from the compressed instances (although this might not yield the most efficient coding):

```
HTTP/1.1 226 IM Used
Date: Wed, 24 Dec 1997 14:01:00 GMT
Etag: "def"
Delta-base: "abc"
Content-encoding: gzip
IM: vcdiff
```

The body of this response would be the result of VCDIFF_DELTA(GZIP(foo.html;"abc"), GZIP(foo.html;"def")). The client would store as a new cache entry the entity GZIP(foo.html;"def"), after recovering that entity by applying the delta to its previous cache entry.

The server's other alternative would be to compute the delta from the uncompressed values, returning:

```
HTTP/1.1 226 IM Used
Date: Wed, 24 Dec 1997 14:01:00 GMT
Delta-base: "abc"
Etag: "ghi"
IM: vcdiff
```

The body of this response would be the result of VCDIFF_DELTA(GUNZIP(GZIP(foo.html;"abc")), foo.html;"ghi"), or more simply VCDIFF_DELTA(foo.html;"abc", foo.html;"ghi"). The client would store as a new cache entry the entity foo.html;"ghi" (i.e., without any content-coding), after recovering that entity by applying the delta to its previous cache entry.

Note that the new value of foo.html (at 14:01:00 GMT) without the gzip content-coding must have a different entity tag from the compressed instance of the same underlying file.

The client's second request might have been:

```
GET /foo.html HTTP/1.1
Host: example.com
If-none-match: "abc"
Accept-encoding: gzip
A-IM: diffe, gzip
```

The client lists gzip in both the Accept-Encoding and A-IM headers, because if the server does not support delta encoding, the client would at least like to achieve the benefits of compression (as a content-coding). However, if the server does support the diffe delta-coding, the client would like the result to be compressed, and this must be done as an instance-manipulation.

A server that does support diffe might reply:

```
HTTP/1.1 226 IM Used
Date: Wed, 24 Dec 1997 14:01:00 GMT
Delta-base: "abc"
Etag: "ghi"
IM: diffe, gzip
```

The body of this response would be the result of GZIP(DIFFE_DELTA(GUNZIP(GZIP(foo.html;"abc")), foo.html;"ghi")), or more simply GZIP(DIFFE_DELTA(foo.html;"abc", foo.html;"ghi")). Because the gzip compression is, in this case, an instance-manipulation and not a content-coding, it is not retained when the reassembled response is stored or forwarded, so the client would store as a new cache entry the entity foo.html;"ghi" (without any content-coding or compression).

10.8 New Cache-Control directives

We define two new cache-directives (see section 14.9 of RFC 2616 [10] for the specification of cache-directive).

10.8.1 Retain directive

The set of cache-response-directive values is augmented to include the retain directive.

```
cache-response-directive = ...
        | "retain" [ "=" delta-seconds ]
```

A retain directive is always a "hint" from a server to a client; it never specifies a mandatory action for the recipient.

The presence of a retain directive indicates that a delta-capable client ought to retain the instance in the response in its cache, space permitting, and ought to use the corresponding entity tag in a future request for a delta-encoded response. I.e., the server is likely to provide delta-encoded responses using the corresponding instance as a base instance. By implication, if a client has retrieved and cached several instances of a resource, some of which are marked with "retain" and some not, then there is no point in caching the instances not marked with "retain".

If the retain directive includes a delta-seconds value, then the server is likely to stop using the corresponding instance as a base instance after the specified number of seconds. A client ought not use the corresponding entity tag in a future request for a delta-encoded response after that interval ends. The interval is measured from the time that the response is generated, so a client ought to include the response's Age in its calculations.

If the retain directive includes a delta-seconds value of zero, a client SHOULD NOT use the corresponding entity tag in a future request for a delta-encoded response.

> Note: We recommend that server implementors consider the bandwidth implications of sending the "retain=0" directive to clients or proxies that might not have the ability to make use of it.

10.8.2 IM directive

The set of cache-response-directive values is augmented to include the im directive.

```
cache-response-directive = ...
        | "im"
```

A cache that complies with the specification for the IM header, the A-IM header, and the 226 response-status code SHOULD ignore a no-store cache-directive if an im directive is present in the same response. All other implementations MUST ignore the im directive (i.e., MUST observe a no-store directive, if present).

10.9 Use of compression with delta encoding

The application of data compression to the diffe and gdiff delta codings has been shown to greatly reduce the size of the resulting message bodies, in many cases. (The vcdiff coding, on the other hand, is inherently compressed and does not benefit from further compression.) Therefore, it is strongly recommended that implementations that support the diffe and/or gdiff delta codings also support the gzip and/or deflate compression codings. (The deflate coding provides a more compact result.) However, this is not a requirement for the use of delta encoding, primarily because the CPU-time costs associated with compression and decompression may be excessive in some environments.

A client that supports both delta encoding and compression as instance-manipulations signals this by, for example

```
A-IM: diffe, deflate
```

The ordering rule stated in section 10.5.3 requires, if the server uses both instance-manipulations in the response, that compression be applied to the result of the delta encoding, rather than vice versa. I.e., the response in this case would include

```
IM: diffe, deflate
```

Note that a client might accept compression either as a content-coding or as an instance-manipulation. For example:

```
Accept-Encoding: gzip
A-IM: gzip, gdiff
```

In this example, the server may apply the gzip compression, either as a content-coding or as an instance-manipulation, before delta encoding. Remember that the entity tag is assigned after content-coding but before instance-manipulation, so this choice does affect the semantics of delta encoding.

10.10 Delta encoding and multipart/byteranges

A client may request multiple, non-contiguous byte ranges in a single request. The server's response uses the "multipart/byteranges" media type (section 19.2 of [10]) to convey multiple ranges in a response. If a multipart/byteranges response is delta encoded (i.e, uses a delta-coding as an instance-manipulation), the delta-related headers are associated with the entire response, not with the individual parts. (This is because there is only one base instance and one current instance involved.) A delta-encoded response with multiple ranges MUST use the same delta-coding for all of the ranges.

If a server chooses to use a delta encoding for a multipart/byteranges response, it MUST generate a response in accordance with the following rules.

When a multipart/byteranges response uses a delta-coding prior to a range selection, the A-IM and IM header fields list the delta-coding before the "range" literal. (Recall that this is the approach taken to obtain a partial response after a premature termination of a message transmission.) The server firsts generates a sequence of bytes representing the difference (delta) between the base instance and the current instance, then selects the specified ranges of bytes, and transmits each such range in a part of the multipart/byteranges media type.

When a multipart/byteranges response uses a delta-coding after a range selection, the A-IM and IM header fields list the delta-coding after the "range" literal. (Recall that this is the approach taken to obtain an updated version just of selected sections of an instance.) The server first selects the specified ranges from the current instance, and also selects the same specified ranges from the base instance. (Some of these selected ranges might be the empty sequence, if the instance is not long enough.) The server then generates the individual differences (deltas) between the pairs of ranges, and transmits each such difference in a part of the multipart/byteranges media type.

11 Quantifying the protocol overhead

The proposed protocol changes increase the size of the HTTP message headers slightly. In the simplest case, a conditional request (i.e., one for a URI for which the client already has a cache entry) would include one more header, e.g.:

```
A-IM:vcdiff
```

This is about 13 extra bytes. A recent study [23] reports mean request sizes from two different traces of 281 and 306 bytes, so the net increase in request size would be between 4% and 5%.

Because a client must have an existing cache entry to use as a base for a delta-encoded response, it would never send "A-IM: vcdiff" (or listing other delta encoding formats) for its unconditional requests. The same study showed that at least 46% of the requests in lengthy traces were for URLs not seen previously in the trace; this means that no more than about half of typical client requests could be conditional (and the actual fraction is likely to be smaller, given the finite size of real caches).

The study also showed that 64% of the responses in a lengthy trace were for image content-types (GIF and JPEG). As noted in section 6, we do not currently know of a delta-encoding format suitable for such image types. Unless a client did support such a delta-encoding format, it would presumably not ask for a delta when making a conditional request for image content-types.

Taken together, these factors suggest that the mean increase in request header size would be much less than 5%, and probably below 1%.

Delta-encoded responses carry slightly longer headers. In the simplest case, a response carries one more header, e.g.:

```
IM:vcdiff
```

This is about 11 bytes. Other headers (such as "Delta-Base") might also be included. However, none of these extra headers would be included except in cases where a delta encoding is actually employed, and the sender of the response can avoid sending a delta encoding if this results in a net increase in response size. Thus, a delta-encoded response should never be larger than a regular response for the same request.

Simulations suggest that, when delta encoding pays off at all, it saves several thousand bytes [23]. Thus, adding a few dozen bytes to the response headers should almost never obviate the savings in the message-body size.

Finally, the use of the "retain" Cache-Control directive might cause some additional overhead. Some server heuristics might be successful in limiting the use of these headers to situations where they would probably optimize future responses. Neither of these headers is necessary for the simpler uses of delta encoding.

12 Security Considerations

We are not aware of any aspects of the basic delta encoding mechanism that affect the existing security considerations for the HTTP/1.1 protocol.

13 Acknowledgements

Phong Vo has provided a great deal of guidance in the choice of delta encoding algorithms and formats. Issac Goldstand and Mike Dahlin provided a number of useful comments on the specification. Dave Kristol suggested many textual corrections.

14 Intellectual Property Rights

The IETF has been notified of intellectual property rights claimed in regard to some or all of the specification contained in this document. For more information consult the online list of claimed rights, at <http://www.ietf.org/ipr.html>.

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP 11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

15 References

1. Gaurav Banga, Fred Douglis, and Michael Rabinovich. Optimistic Deltas for WWW Latency Reduction. Proc. 1997 USENIX Technical Conference, Anaheim, CA, January, 1997, pp. 289-303.

2. Berners-Lee, T., Fielding, R. and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.

3. Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

4. Edith Cohen, Balachander Krishnamurthy, and Jennifer Rexford. Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters. Proc. SIGCOMM '98, September, 1998, pp. 241-253.

5. Deutsch, P., "GZIP file format specification version 4.3", RFC 1952, May 1996.

6. Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

7. Deutsch, P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996.

8. Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey Mogul. Rate of Change and Other Metrics: a Live Study of the World Wide Web. Proc. Symposium on Internet Technologies and Systems, USENIX, Monterey, CA, December, 1997, pp. 147-158.

9. Fielding, R., Gettys, J., Mogul, J., Nielsen, H. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997.

10. Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

11. Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P., Luotonen, A., Luotonen, L. and L. Stewart, "HTTP Authentication: Basic and Digest Access Authnetication", RFC 2617, June 1999.

12. Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

13. Arthur van Hoff, John Giannandrea, Mark Hapner, Steve Carter, and Milo Medin. The HTTP Distribution and Replication Protocol. Technical Report NOTE-DRP, World Wide Web Consortium, August, 1997.

14. Arthur van Hoff and Jonathan Payne. Generic Diff Format Specification. Technical Report NOTE-GDIFF, World Wide Web Consortium, August, 1997.

15. Barron C. Housel and David B. Lindquist. WebExpress: A System for Optimizing Web Browsing in a Wireless Environment. Proc. 2nd Annual Intl. Conf. on Mobile Computing and Networking, ACM, Rye, New York, November, 1996, pp. 108-116.

16. James J. Hunt, Kiem-Phong Vo, and Walter F. Tichy. An Empirical Study of Delta Algorithms. IEEE Soft. Config. and Maint. Workshop, 1996.

17. Jacobson, V., "Compressing TCP/IP Headers for Low-Speed Serial Links", RFC 1144, February 1990.

18. Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, May 2000.

19. David G. Korn and Kiem-Phong Vo. A Generic Differencing and Compression Data Format. Technical Report HA1630000-021899-02TM, AT&T Labs - Research, February, 1999.

20. Korn, D. and K. Vo, "The VCDIFF Generic Differencing and Compression Data Format", Work in Progress.

21. Merriam-Webster. Webster’s Seventh New Collegiate Dictionary. G. & C. Merriam Co., Springfield, MA, 1963.

22. Jeffrey C. Mogul. Hinted caching in the Web. Proc. Seventh ACM SIGOPS European Workshop, Connemara, Ireland, September, 1996, pp. 103-108.

23. Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. Research Report 97/4, DECWRL, July, 1997.

24. Mogul, J. and A. Van Hoff, "Instance Digests in HTTP", RFC 3230, January 2002.

25. Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

26. The Open Group. The Single UNIX Specification, Version 2 - 6 Vol Set for UNIX 98. Document number T912, The Open Group, February, 1997.

27. W. Tichy. "RCS - A System For Version Control". Software - Practice and Experience 15, 7 (July 1985), 637-654.

28. Andrew Tridgell and Paul Mackerras. The rsync algorithm. Technical Report TR-CS-96-05, Department of Computer Science, Australian National University, June, 1996.

29. Stephen Williams. Personal communication. http://ei.cs.vt.edu/~williams/DIFF/prelim.html.

30. Stephen Williams, Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, and Edward A. Fox. Removal Policies in Network Caches for World-Wide Web Documents. Proc. SIGCOMM '96, Stanford, CA, August, 1996, pp. 293-305.

16 Authors' addresses

Jeffrey C. Mogul
Western Research Laboratory
Compaq Computer Corporation
250 University Avenue
Palo Alto, California, 94305, U.S.A.

Phone: 1 650 617 3304 (email preferred)
EMail: JeffMogul@acm.org

Balachander Krishnamurthy
AT&T Labs - Research
180 Park Ave, Room D-229
Florham Park, NJ 07932-0971, U.S.A.

EMail: bala@research.att.com

Fred Douglis
AT&T Labs - Research
180 Park Ave, Room B-137
Florham Park, NJ 07932-0971, U.S.A.

Phone: 1 973 360-8775
EMail: douglis@research.att.com

Anja Feldmann
University of Saarbruecken, Germany,
Computer Science Department
Im Stadtwald, Geb. 36.1, Zimmer 310
D-66123 Saarbruecken, Germany

EMail: anja@cs.uni-sb.de

Yaron Y. Goland

Email: yaron@goland.org

Arthur van Hoff
Marimba, Inc.
440 Clyde Avenue
Mountain View, CA 94043, U.S.A.

Phone: 1 650 930 5283
EMail: avh@marimba.com

Daniel M. Hellerstein
Economic Research Service, USDA
1909 Franwall Ave, Wheaton MD 20902

Phone: 1 202 694-5613 or 1 301 649-4728
EMail: danielh@crosslink.net or webmaster@srehttp.org

17 Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.