Network Working Group
Internet-Draft
Expires: 15 October 1998

Jeffrey Mogul, DECWRL,
Arthur van Hoff, Marimba

15 April 1998

Duplicate Suppression in HTTP

draft-mogul-http-dupsup-00.txt

STATUS OF THIS MEMO

ABSTRACT

A significant fraction of Web content is often exactly duplicated under several different URIs. This duplication can lead to suboptimal use of network bandwidth, and unnecessary latency for users. Much of this duplication can be avoided through the use of a simple mechanism, described here, which allows a cache to efficiently substitute one byte-for-byte identical value for another. By doing so, the cache avoids some or all of the network costs associated with retrieving the duplicate value.

TABLE OF CONTENTS

1 Introduction 2
2 Terminology 3
3 Goals 4
4 Overview 5
4.1 Scenario 5
4.2 Requirements for a duplicate-suppression mechanism 6
4.3 How instance bodies may be uniquely indicated 7
4.3.1 Use of cryptographic digests 8
4.3.2 Use of Rabin fingerprints 8
4.3.3 Use of entity tags and uniqueness scopes 9
4.3.4 Other forms of indicia 10
4.4 Cache entry freshness 10
4.5 Message headers 11
4.6 Server hints 12
4.7 Other semantic issues 13
4.7.1 Is SubOK hop-by-hop? 13
4.7.2 Response caching in intermediate proxies 13
4.7.3 Conditional GETs 14
4.7.4 Interaction with Hit-metering 14
4.7.5 Interaction with compression and delta-encoding 16
4.7.6 Interaction with range retrievals 16
4.7.7 Other points 17
5 Specification 17
5.1 Protocol parameter specifications 17
5.1.1 Indicia schemes 17
5.1.2 Indicia values 17
5.2 Header specifications 17
5.2.1 SubOK 18
5.2.2 Subst 19
6 IANA Considerations 19
7 Security Considerations 19
7.1 Manipulation of instance-bodies 20
7.2 Manipulation of a uniqueness scope 20
7.3 False association of an indicia with a URI 22
8 Acknowledgements 22
9 References 22
10 Authors' addresses 24

1 Introduction

A significant fraction of Web content is often exactly duplicated under several different URIs. One trace-based study found that 18% of the non-empty message-bodies were identical to at least one other message-body for a different resource [4]. Another study showed that, of 30 million HTML and text documents found by a search-engine

crawler, more than 5.3 million (18%) were identical duplicates [3].

Content can be duplicated for many reasons. The observed rate of duplication of HTML and text documents suggests that many duplications are made instead of hyperlinking to a remote copy of a document. Many graphical elements, such as logos, bullets, backgrounds, bars, buttons, etc. are duplicated so that a particular page can be displayed without depending on the simultaneous availability of several Web servers. Push-based distribution mechanisms often send the same content to many users. The automatic distribution of software packages via the Web can lead to duplicated distribution of the same program or program components (such as a library).

This duplication can lead to suboptimal use of network bandwidth, and unnecessary latency for users. Each time a duplicate is loaded across the network instead of from a local or nearby cache, network bandwidth is wasted. Such retrievals may also increase the latency seen by the ultimate client, especially if the message is large or the available bandwidth is small.

This document describes a simple, optional, efficient, and compatible extension to HTTP/1.1 that can be used to avoid much of this duplication. In particular, it allows a client to inform a proxy cache that the client will accept either a response for the requested resource, or the substitution of the cached response from a different resource whose entity-body is, with extremely high probability, byte-for-byte identical to that of the requested resource. By performing the substitution, the cache avoids some or all of the network costs associated with retrieving the duplicate value.

2 Terminology

HTTP/1.1 [5] 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'' [11]. Unfortunately, the definition for ``entity'' in HTTP/1.1 is similar to that used in MIME [7], based on an entirely false analogy between MIME and HTTP.

In MIME, electronic mail messages do have distinct and separate existences, so the MIME definite ``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'' make sense.

In HTTP, however, a response message to a GET does not have a distinct and separate existence. Rather, it is describing the current state of a resource (or a variant, subject to a set of constraints). The HTTP/1.1 specification provides 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.

It is too late to fix the terminological failure in the HTTP/1.1 specification, so we instead 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, but without the application of any content-coding or transfer-coding.

One can think of an instance as a snapshot in the life of a resource.

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.

3 Goals

The goals of this proposal are:

1. Allow substitution of a cached response for a response of a requested resource, when the cached instance body is byte-for-byte identical to that of the requested instance, and when the URIs for the two resources do not match.

2. Interoperate with all HTTP/1.1-compliant implementations.

3. Add minimal overhead to HTTP messages.

4. Add minimal execution time for HTTP implementations.

5. Be entirely optional for any implementation and at any time.

The goals do not include:

- Compression of HTTP entity bodies; this is supported in the basic HTTP protocol.

- Delta encoding (or ``differential'' download) of an instance body that differs slightly from an entry in the client's own cache; this is supported in an independent extension [13].

- Providing a mechanism for clients to discover the entity tag, checksum, digest, or other instance-specific information that allows the accurate specification of substitutable instance bodies. This information can be provided by numerous techniques; the extension described in this document assumes the existence of one or more such techniques.

4 Overview

4.1 Scenario

We start by describing part of a scenario in which duplicate suppression might be used. Imagine that an HTTP client has obtained, by some means (we will address this later), the MD5 digest for the body of a resource instance that the client wishes to retrieve. Imagine further that the client is using a caching proxy, and that the client has some reason to believe that the resource may be duplicated in the Web.

To be specific, assume that the resource is http://foo.com/logo.gif, and that the known MD5 digest is "HUXZLQLMuI/KZ5KDcJPcOA==". The client sends this request to the proxy:

```
GET http://foo.com/logo.gif HTTP/1.1
Host: foo.com
SubOK: md5="HUXZLQLMuI/KZ5KDcJPcOA==", inform
```

The meaning of this request is ``I want the value of http://foo.com/logo.gif, but you can substitute any response whose MD5 instance digest is HUXZLQLMuI/KZ5KDcJPcOA==. Please inform me of any substitution, though.'' (Instance digests are described in

another document [15].)

If the cache has a fresh copy of the requested resource, it simply returns that to the client, and ignores the SubOK header field. Otherwise, the proxy may then check its cache to see if any cached instance has the right MD5 digest. If not, it simply forwards the request as it normally would.

However, if the proxy does find a cached instance of another resource with the specified MD5 digest, it may return the cached resource to the client.

For example, the proxy might return:

```
HTTP/1.1 200 OK
Date: Thu, 29 Jan 98 17:47:55 GMT
Age: 37
Etag: "xyzzy"
Content-Type: image/gif
Subst: http://bar.com/foo_logo.gif
```

The Subst header field is added because the client used the ``inform'' directive; it tells the client where the response actually originated.

Note that there is an assumption behind this scenario, that if the client receives the ``right'' instance body, it does not care if it receives the ``wrong'' headers. In most cases, this is not an issue. Either the header is pretty much guaranteed to be the same for the original and for the substitute (e.g., it is unlikely that the Content-Type would differ if the instance body is the same), or it is not consequential to the use of the response (e.g., the Date header field). However, if the client does want the actual headers from the requested URI, then we provide a optional mechanism, to be described later, that make this possible.

4.2 Requirements for a duplicate-suppression mechanism

To generalize from this scenario, a mechanism for duplicate suppression needs:

1. A way for the client to indicate that it is willing to accept substitutions.

2. A way for the client to concisely and reliably indicate what instance body value it wants to receive.

Both requirements are met by the SubOK header field, which allows a client to indicate the circumstances under which it will accept substitutions. It also allows the client to give various directives regarding the behavior of the proxy cache. The full specification of the SubOK header field is given in section 5.2.1.

It is important to understand that the client will be using a mechanism not described in this document to obtain a value, such as an instance digest, that defines the specific instance body being retrieved. We will refer to such a value as an ``indicia'', ``an identifying marking ... used to single out one thing from another'' [12] (the plural can be either ``indicia'' or ``indicias'').

Several such mechanisms are available:

- 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'' [9]. DRP includes an ``index'' data structure, to ``describe the exact state of a set of data files.'' A DRP index can include attributes, such as an entity tag, instance digest, or URN, associated with a given URI.

- Similarly, the proposed ``Extensions for Distributed Authoring on the World Wide Web'' (WEBDAV) [8] includes a method for obtaining arbitrary properties of a resource, including its entity tag, or for a collection of resources. The WEBDAV property mechanism appears to be easily extended to other types of information about resources.

- Several researchers have proposed that Web servers can provide, in a response about one URI, some ``hint'' information pertaining to other resources. For example, in a ``predictive prefetching'' mechanism [18], the server can suggest other URIs that the client application might beneficially prefetch. Or a server might indicate which cached responses are still valid or invalid [10]. In either case, the ``hint'' information could convey an entity tag or digest for the current instance(s) of the relevant resource(s). This information would not only prevent the client from attempting to retrieve something it already has a fresh copy of; it could also be used in the duplicate suppression mechanism.

4.3 How instance bodies may be uniquely indicated

The second requirement for the SubOK header is that the client can use some form of indicia to ``concisely and reliably indicate'' to a proxy cache what instance body it wants to receive. Generally, a URL is unacceptable as an indicia, because the binding between a URL and an instance may vary both with time (as the resource is modified) and with arbitrary parameters of the request, such as the ``Accept-Language'' header. (While in principle the client could

discover which request fields the desired response varies on, in practice this is likely to be too complicated.)

The two obvious candidates for indicia are cryptographic instance digests and strong entity tags.

4.3.1 Use of cryptographic digests

A cryptographic digest algorithm, such as MD5 or SHA, takes a sequence of bytes (i.e., an instance body) as its input, and produces a short bit-string as its output. Such algorithms have two attractive properties:

- The output form is concise (24 octets for MD5, 28 octets for SHA, using the somewhat inefficient base64 encoding).

- The probability that two inputs will generate the same output (``collide'') is extremely small.

Although the probability of a collision is non-zero, a carefully constructed digest algorithm should reduce this probability to a point far below the probability of undetected TCP data corruption, for example. For MD5, for example, ``[it] is conjectured that the difficulty of coming up with two messages having the same message digest is on the order of 2^64 operations'' [20]. It is widely assumed that the output of MD5 is ``close to random,'' so the probability of an MD5 collision between two distinct instance bodies picked at random should be quite small.

4.3.2 Use of Rabin fingerprints

Another similar way to produce appropriate indicia values is to use the Rabin ``fingerprint'' method [19]. A Rabin fingerprint is a polynomial checksum with the property that if two files are different, the probability that their fingerprints are equal (a ``collision'') is extremely low. More precisely, in a universe with N documents of mean length L bits, using k-bit fingerprints, the expected number of collisions is (N^2)*L/(2^k).

For example, if one assumes that the entire web contains 1 quadrillion (10^15) files, and that the mean length is 1 megabyte (about 2^23 bits), and one uses a 128-bit fingerprint, then the probability of even one collision is

((10^15)^2)*(20^23)/(2^128) = (10^30)/(2^105) = 0.025

Note that any given proxy cache would presumably contain a much smaller number of cache entries at any given time, and so the actual probability of a collision in the duplicate-suppression algorithm would be infinitesimal.

This probably guarantee depends, however, on the random choice of a parameter of the Rabin algorithm, so that this parameter is independent of any of the possible input files. This implies, in

turn, that an adversary knowing the value of this parameter can easily construct a pair of different files with identical fingerprints.

Since we cannot standardize on an indicia function with a secret parameter, this means that the Rabin fingerprint is not a full substitute for a secure message digest, such as MD5 or SHA. However, simple experiments [2] suggest that the cost of computing a fingerprint is significantly less than computing an MD5 or SHA digest, and so for applications (such as within an intranet) where security is not necessary, fingerprints might be the most appropriate way to generate indicia for duplicate suppression.

4.3.3 Use of entity tags and uniqueness scopes

The HTTP/1.1 specifies that a strong entity tag ``may be shared by two entities of a resource only if they are equivalent by octet equality.'' This requirement is stated only with respect to a specific resource, so under this rule, instances of two different resources could have identical strong entity tags. Therefore, entity tags as specified in HTTP/1.1 are not sufficient for detecting duplication among different resources.

However, HTTP/1.1 does not forbid the use of entity tags that are unique across a broader scope. The proposal for support of delta encoding in HTTP [13] introduced the concept of a ``uniqueness scope'' of an entity tag: the set of resources across which an entity tag is unique for all time, and specified a DCluster header, which allows an origin server to describe a uniqueness scope for the entity tag carried by a response. A uniqueness scope is described as the set of URLs that share a set of prefixes.

This approach can also be applied to the problem of duplicate suppression. Assume that the proxy cache has a cached response for an instance of resource http://bar.com/foo_logo.gif, with an entity tag of "xyzzy", and a uniqueness scope that includes the prefix ``http://foo.com/''. If the client then sends the proxy this request;

```
GET http://foo.com/logo.gif HTTP/1.1
Host: foo.com
SubOK: etag="xyzzy"
```

then the proxy can determine that (1) the requested resource instance and the cached instance are in the same uniqueness scope, and (2) the entity tags do match. This means, by the definition of a ``uniqueness scope'', and because ``xyzzy'' is a strong entity tag, that the cached instance body is exactly identical to the requested instance body.

Note that the requesting client need not know the uniqueness scope of the entity tag used in the SubOK header field. It is the proxy cache that has to determine whether the requested resource is in the

uniqueness scope of any of its cached responses. This implies the use of a somewhat sophisticated data structure by the proxy, but not one with great cost or complexity.

The DCluster header field proposed in [13] has one drawback for this application: because it was specified for use with delta encoding, rather than duplicate suppression, the intention is that ``the DCluster header does not necessarily describe the entire uniqueness scope of an entity tag. Rather, it describes a subset of the uniqueness scope whose members are likely to differ by small deltas.'' When doing duplicate suppression, on the other hand, one would like to know the entire uniqueness scope, so as to maximize the likelihood of finding a duplicate instance body. This implies either the use of a distinct header name for purposes of duplicate suppression, or a modification of the DCluster proposal to allow a distinction to be made.

In this document, we propose a distinct header field name, to allow independent discussion of the duplicate suppression proposal and the delta encoding proposal, but at some point it might prove reasonable to merge the two headers into one.

4.3.4 Other forms of indicia

It is possible that some form of URN could be used to indicate the correct instance body, especially if the URN embodies some indication of version number. Immutability is not enough, however. For example, although the namespace of IETF RFCs describes a set of immutable documents, each RFC can appear in several formats. Thus, two resources that each represent the same RFC may not have byte-for-byte identical instance bodies.

The DRP proposal [9] defined a variety of URN that embeds an MD5 or SHA digest within the URN. This kind of URN has the necessary properties for duplicate suppression; however, it might not be convenient for other purposes. The mechanism described in this document allows the use of URNs in the SubOK header field, but does not fully specify the constraints on URNs used in this way.

4.4 Cache entry freshness

Suppose a client requests a resource

```
GET http://foo.com/logo.gif HTTP/1.1
Host: foo.com
SubOK: md5="HUXZLQLMuI/KZ5KDcJPcOA==", inform
```

and the proxy cache responds by substituting an instance of another resource:

```
HTTP/1.1 200 OK
Date: Thu, 29 Jan 98 17:47:55 GMT
Age: 37
Cache-control: max-age=3600
Etag: "xyzzy"
```

```
      Content-Type: image/gif
```

Subst: http://bar.com/foo_logo.gif

The response sent by the proxy indicates, through its Age and Cache-control headers, that it is still fresh for most of an hour.

However, the Age and Cache-Control headers were drawn from a response for http://bar.com/foo_logo.gif, not a response for http://foo.com/logo.gif. How does the client know how long it can safely cache this response as a representation of http://foo.com/logo.gif?

Recall that we have already assumed the existence of a mechanism, outside the scope of this document, that the origin server has used to provide an indicia to the client, presumably for an instance that is current at the time the indicia is sent. The same mechanism could easily be used to transfer freshness-lifetime information about that instance: whenever the origin server sends the indicia, it also sends an expiration time or maximum-age value.

For example, if the indicia for an instance is sent in a DRP index data structure, the freshness lifetime could either be provided within the index, as an attribute of the instance, or it could be the freshness lifetime of the response containing the index (i.e., the Expires or max-age value from that response's HTTP headers).

4.5 Message headers

In section 4.1, we noted that the simple mechanism described so far does not necessarily provide to the client the actual headers that would have been obtained if an instance of the requested resource had been returned, rather than a cached instance of a different resource with the same instance body. We also noted that this might not, in general, be a problem.

However, if it is important that the client get the actual headers for the requested resource, the duplicate-suppression mechanism can be augmented by several simple extensions.

The first extension is outside the scope of this document, and would be part of the mechanism(s) used by the origin server to provide the indicia to the client (as described in section 4.2). Along with the indicia value that describes the instance body, the origin server would provide a directive telling the client to obtain the actual resource headers even if the instance body is obtained from a duplicate. For example, this might be done using an additional attribute in a DRP index data structure (as with the freshness-lifetime information; see section 4.4).

Once the client knows that it ought to obtain the actual headers of the requested instance, it needs to indicate this requirement to the

proxy cache (if any) performing the substitution. It does so using a directive in the SubOK header field, e.g.:

```
GET http://foo.com/logo.gif HTTP/1.1
Host: foo.com
SubOK: md5="HUXZLQLMuI/KZ5KDcJPcOA==", hdrs
```

The ``hdrs'' directive tells the proxy cache that the client will accept substitution for the instance body of http://foo.com/logo.gif (as long as the MD5 digest matches), but the proxy MUST provide the actual instance headers of the resource given by the Request-URI.

If the proxy has a fresh cached copy of the headers for the Request-URI (that is, the Age of the cache entry with these headers has not reached the max-age value, or the cached Expires deadline has not been reached), then the proxy MAY simply return these cached headers, along with the substituted instance body. However, if the proxy does not have a fresh cache entry containing these headers, then the ``hdrs'' directive means that the proxy MUST contact the origin server to obtain a fresh copy of the headers.

Note that while this does require the proxy to send a request to the origin server, it does not require the proxy to retrieve the instance-body from the origin server. Therefore, even in the case where the client's request causes a message to be sent to the origin server, the duplicate suppression mechanism might still avoid retrieving the instance body from the origin server. This could significantly reduce bandwidth utilization and latency, if the body is large.

The best means for the proxy to obtain fresh headers for the Request-URI is to use a HEAD request. The HEAD method, by its specification, returns exactly what a corresponding GET request would return, except for a message-body.

A conditional GET request will not work, because the HTTP/1.1 specification for the 304 (Not Modified) response requires that the origin server SHOULD NOT provide many of the entity-headers for the instance. For example, if the Content-Encoding for the substituted instance really did differ from that of the requested instance, a 304 response would not reliably reveal this difference.

Note that the unlikelihood of this scenario (two byte-for-byte identical instance bodies with different Content-Encodings) implies that the ``SubOK: hdrs'' mechanism probably will not be used very often (and so the attendant overhead will not be incurred). On the other hand, the possibility of such a scenario (perhaps involving some other, as yet undefined, entity-header) implies the need for such a mechanism.

4.6 Server hints

The mechanism by which the client obtains indicia information from the server is, as stated earlier, outside the scope of this document. We have explained elsewhere that certain other useful information could be supplied by the server at the same time:

- Freshness lifetime information for the instance described by the indicia (section 4.4).

- Indication of the need for the client to obtain the actual HTTP headers for the Request-URI (section 4.5).

The server might also provide other ``hint'' information along with the indicia. This could include

- An indication of likelihood of duplication; the client would only expend request bytes, and proxy processing time, on the SubOK header in cases where there the likelihood of duplication is above some threshold. (Note that the mere fact that the server provides an indicia value might not, in itself, be an indication of likely duplication. For example, an MD5 digest might be provided simply for security reasons.)

- [Other hints?]

4.7 Other semantic issues

There are a number of issues that have not yet been adequately addressed in this design, or that need to be considered when evaluating the design.

Right now, some of the text here is just a placeholder for a full discussion.

4.7.1 Is SubOK hop-by-hop?

Should the SubOK header be sent hop-by-hop (i.e., protected by a Connection header)? If not, then it could be acted upon by a "distant" cache. But this might not be a huge benefit. It might be reasonable to make it forwardable by an implementation without requiring the implementation to fully implement the header.

4.7.2 Response caching in intermediate proxies

If client C1 sends a substitutable request for resource R1 via proxy P1 and then via proxy P2, and then P2 substitutes R2 and then responds, should P1 cache the response as if for R2 or as if for R1?

If the SubOK header isn't hop-by-hop, and P1 is naive about SubOK/Subst, then it would seem risky to allow P1 to cache the response. Another naive client (C2) might make a request for R1 via P1, and P1 would then naively return the substituted cached response. C2 would thus receive a substitute without realizing it.

Is there a good mechanism to prevent this? Perhaps some application of "Cache-Control: s-maxage=0" would work, but at a cost of increased complexity. Alternatively, this could be a good argument in favor of

making SubOK a hop-by-hop header.

However, it's not entirely clear that C2, in the example above, would actually suffer from its naivete. The cached response for R2 should be effectively equivalent to a cached response for R1, and C2 shouldn't notice the difference.

4.7.3 Conditional GETs

In principle, a conditional GET request (e.g., one including an If-Modified-Since or If-None-Match header field) can include the SubOK header field. The server's interpretation of such a combination is straightforward:

- Evaluate the conditional(s). If this would result in a response status of 304 (Not Modified) or 412 (Precondition failed), ignore the SubOK field.

- If the conditionals have no effect, then the proxy may evaluate the SubOK header to see if a substitution can be made.

Informally, the meaning is "if my cache entry is valid, tell me so. Otherwise, send me a full response but feel free to substitute an equivalent instance body."

In practice, it may not make much sense for a client to include a SubOK header field in a conditional GET. Any mechanism that the client uses to obtain indicia values would probably also give it direct information on the validity of its cache entries. If so, the client could make the validity determination locally, rather than asking the proxy (via an If-Modified-Since or If-None-Match header field) to make this determination.

4.7.4 Interaction with Hit-metering

Hit-metering [14] is a proposed extension to HTTP/1.1 that allows an origin server to negotiate with proxy caches to receive reports about the number of times a cached response is used. This allows the origin server to obtain accurate use-counts for a resource, without defeating caching.

The hit-metering design depends on the existence of a connected path between the requesting client and the origin server, since negotiations are made with respect to hops along this path, and reports are forwarded hop-by-hop. However, if a client using the duplicate suppression mechanism makes a request for resource R1 via a proxy cache, and the proxy cache substitutes an instance of resource R2, there may never be a complete request path connecting the proxy to the origin server for resource R1. In such a case, the proxy cannot negotiate the use of hit-metering for resource R1. In other words, if the origin server is expecting to count uses of R1, and the proxy substitutes an instance of R2 for a potential use of R1, the

origin server's expectation will be violated.

There are several ways to resolve the contradiction between hit-metering and duplicate suppression:

1. The duplicate suppression mechanism requires that the client know an indicia value for the requested resource. Although this document does not specify how a client obtains an indicia value for a resource instance, it presumably comes from the origin server for that resource. Therefore, if an origin server wishes to reliably hit-meter a resource, it need only refrain from providing indicia values. This would prevent any substitution for the resource in question.

2. If the origin server desires only to know the sum of the use-counts for a collection of mutually substitutable instances, rather than the individual counts, then the existing hit-metering mechanism should suffice. (This assumes that the origin server is the only source for instances with the given indicia value.) The substitution counts as a use of the cache entry (by the ``counting rules'' of the hit-metering specification [14]) and so it will be reported to the origin server, albeit for a resource other than the client's Request-URI.

3. If the client's request includes the ``hdrs'' subok-directive, this forces the substituting proxy to complete the path to the origin server for the Request-URI, which gives the origin server information about the initial reference. It also allows the origin server to inform that the response, if cached, should be hit-metered. The proxy will then count (and report) further uses of the cache entry for the Request-URI (if one adopts a slightly broad view of the requirements of the ``counting rules'' of the hit-metering specification [14]: in this case, the proxy should increment the count for the Request-URI, and not for the substituted cache entry). Therefore, if the mechanism by which the client obtains an indicia value from the origin server also has a means to require that the client use the ``hdrs'' subok-directive if it sends a SubOK header field, the origin server will obtain accurate use-counts.

The second alternative may not be suitable for many applications of hit-metering (for example, counting uses of an ad banner may be less interesting than knowing which sites lead to its display). The third alternative requires additional mechanism and minor changes to the hit-metering specification, and only works if the user agents can be trusted to honor the origin server's demand to use the ``hdrs''

subok-directive. Therefore, in practice the first alternative (provide no indicia for hit-metered resources) may be the best.

4.7.5 Interaction with compression and delta-encoding

HTTP/1.1 supports the use of compression both as a content-coding and as a transfer-coding. A more recent proposal describes the use of delta encoding both as a content-coding and as a transfer-coding [13]. In delta encoding, the server sends the differences between two instances of a resource, instead of sending the newer instance in its entirety.

The duplicate suppression mechanism described in this specification operates on instances. By the definition in section 2, an instance body does not include any content-coding or transfer-coding. Therefore, when a proxy performing duplicate suppression decides that a particular cache entry is a suitable substitute, this decision is independent of whether the cached response was received with a content-coding. Even if the substitution is appropriate according to the indicia, the proxy cannot return the cache entry if its content-coding is incompatible with the request (unless the proxy is able to undo the incompatible content-coding).

Transfer-codings are, by definition, hop-by-hop. Therefore, at least in concept, a cache entry does not have a transfer-coding. (In practice, a cache implementation may choose to store a transfer-coded version of a response, as a performance optimization, if this behavior has no external visibility.)

Therefore, duplicate suppression is essentially orthogonal to compression and delta-encoding.

4.7.6 Interaction with range retrievals

HTTP/1.1 supports the retrieval of sub-ranges of a resource value. The range selection is done after the application of any content-coding, and before the application of any transfer-coding. As the result of receiving a Partial Content response, a proxy cache might create an entry that contains only a partial instance (or it might update an existing full or partial cache entry, resulting in a full instance).

It is certainly possible for a substitution to be returned in response to a request with a Range header. The subsitution, since it is done on an instance-by-instance basis, is performed before any range selection.

Because the duplicate suppression mechanism operates on instances, it is not possible to use a partial cache entry as a substitution, unless the request uses a Range header that specifies a subset of the partial cache entry. In any case, the content-coding of the cache entry must also be compatible with the Accept-Encoding header of the request.

4.7.7 Other points

T. B. S.

5 Specification

5.1 Protocol parameter specifications

5.1.1 Indicia schemes

An indicia scheme is an algorithm used to generate an indicia from an instance body.

```
indicia-scheme = token
```

The Internet Assigned Numbers Authority (IANA) acts as a registry for indicia-scheme tokens. Initially, the registry contains the following tokens:

MD5	The MD5 algorithm, as specified in RFC 1321 [20]. The output of this algorithm is encoded using the base64 encoding [1].
SHA	The SHA-1 algorithm [16]. The output of this algorithm is encoded using the base64 encoding [1].
UNIXcksum	The algorithm computed by the UNIX ``cksum'' command, as defined by the Single UNIX Specification, Version 2 [17]. The output of this algorithm is an ASCII digit string representing the 32-bit CRC, which is the first word of the output of the UNIX ``cksum'' command.

5.1.2 Indicia values

An indicia value is the results of applying an indicia scheme to an instance body.

```
indicia-value = quoted-string
```

Syntax issues:

1. Should we limit indicia-value to quoted-string, or could we also also token (e.g., a base64 encoding without quotes?)

5.2 Header specifications

The following headers are defined.

5.2.1 SubOK

The SubOK request header field is used to provide directives from an end-client to a proxy cache regarding the possible substitution of an instance body from a cached response for one resource instance for the instance body of the resource instance specified by the client's request. A proxy MAY ignore the SubOK request header field on any request.

```
SubOK = "SubOK" ":" #subok-directive
subok-directive  = subok-mandatory-directive
                 | subok-indicia-directive
                 | subok-extension-directive
subok-mandatory-directive =
                  "inform"
                | "hdrs"
subok-indicia-directive = indicia-scheme "=" indicia-value
subok-extension-directive =
                token [ "=" (quoted-string | token) ]
```

All comparisons of subok-directive tokens are case-insensitive. Comparisons of quoted-strings in subok-directive values are case-sensitive.

If a proxy does not ignore the entire SubOK field, it MUST ignore any subok-indicia-directive or subok-extension-directive that it does not understand, but it MUST implement all elements of the subok-mandatory-directive set.

A proxy MAY substitute an appropriately fresh cache entry from a resource other than the Request-URI, if an indicia-value in the SubOK header field of the request matches the corresponding value for that cache entry.

If the proxy's cache includes a usable entry for the Request-URI, it SHOULD NOT substitute an entry from a different resource.

Syntax issues:

1. Should we allow a token as the value of a subok-extension-directive, or should we restrict this to a quoted-string?
2. This syntax does not have an extensible way to mark directives as mandatory. This means either that we have to specify all of the subok-mandatory-directive values now, or we need to devise a marking mechanism.

The meaning of the subok-mandatory-directive values is

inform If a substitution is made, the response MUST include a valid Subst header field.

hdrs If a substitution is made, the HTTP headers in the response MUST be a fresh superset of the headers that would have been returned for a HEAD method on the Request-URI. If the proxy has a cache entry containing such headers that has not exceeded its freshness lifetime, the proxy may return these cached headers. Otherwise, it MUST forward a HEAD request (or a GET) request towards the origin server for the Request-URI, and then forward the response. (The HEAD request may be satisfied by an intervening proxy cache, subject to any Cache-Control directives in the original client's request.)

5.2.2 Subst

The Subst response-header field MUST be used by a proxy to supply the URI of the original source of an entity-body, if the source is different from the client's Request-URI, and if the client's request included the ``inform'' directive in a SubOK request header field. Otherwise, a proxy MAY send a Subst response-header field, if it makes a substitution based on the information in a SubOK request header field.

```
Subst = "Subst" ":" absoluteURI
```

Is it possible to use Content-Location instead of defining a new header, or would this be a conflict with the existing definition of Content-Location?
---

6 IANA Considerations

This section will provide the necessary IANA considerations information, for the entering of indicia-scheme algorithms into the registries for HTTP indicia-schemes. (Will reflect rules in draft-iesg-iana-considerations-01.)
---

7 Security Considerations

The duplicate suppression is vulnerable to a number of spoofing attacks

There are three kinds of spoofing attack possible:

1. Manipulation of instance-bodies so as to cause an incorrect substitution by an unwitting proxy cache.

2. Manipulation of the ``uniqueness scope'' of an entity tag so as to cause an incorrect substitution by an unwitting proxy cache.

3. False association of an indicia with a URI.

7.1 Manipulation of instance-bodies

In the first kind of attack, assume that the attacker can predict that a client C will request, via proxy P, a resource R1 and give a particular indicia value I in SubOK header field of the request. If the attacker can cause proxy P to load an instance of a different resource R2, with the same indicia I, into P's cache before client C makes its request, then P might unwittingly send C the instance of R2, instead of the requested instance of R1. The attacker can do this by constructing the appropriate value of R2, and then sending a request for R2 via proxy P.

This is only really a problem if the instances of R1 and R2 are different, but in a way not easily detectable by the user. For example, the attacker might cause the user to see a somewhat modified version of a document that the user wanted to see.

This attack depends on the kind of indicia in use. If a secure message digest, such as MD5 or SHA, is used for the indicia, there is no obvious vulnerability to spoofing (provided that there is no feasible attack on the digest algorithm). This is because it is infeasible for an attacker to construct a bogus instance body that has the same secure message digest as a known instance body.

However, if the Rabin fingerprinting method is used to generate the indicia, it is quite feasible for an attacker to construct a bogus instance body with a specific fingerprint value. This means that the Rabin fingerprinting method is not an appropriate method to use in an insecure environment, although it may be useful within an Intranet.

The Rabin fingerprinting method could be protected by using a secure end-to-end instance digest, but this would obviate much of the performance benefit of using the Rabin method. However, there might still be a benefit in cases where the server supplying the indicia and the client are both able to efficiently compute a digest, but duplicate values could come from other servers not willing to attach a secure instance digest to their responses.

7.2 Manipulation of a uniqueness scope

A similar problem arises when using entity tags as indicia. Supposing the attacker can predict that the client C will soon make

this request via proxy P:

```
GET http://foo.com/logo.gif HTTP/1.1
Host: foo.com
SubOK: etag="xyzzy"
```

Suppose that the attacker first requests, via proxy P, this response for http://evilhacker.com/blob.gif (from a server controlled by the attacker):

```
HTTP/1.1 200 OK
Date: Thu, 29 Jan 98 18:47:55 GMT
Etag: "xyzzy"
DCluster: "//foo.com/"
```

Then, when client C does make its request for http://foo.com/logo.gif, proxy P will believe that it can substitute its cached instance body from the response for http://evilhacker.com/blob.gif (because the entity tags match, and the client's Request-URI matches the prefix in the DCluster header field of the cached response).

A similar spoofing attack, via the DCluster header field, is possible when using delta encoding. Therefore, the specification for delta encoding [13] includes a set of recommendations for preventing this attack; we provide an analogous set here. Note, however, that the circumstances are sufficiently different that the defenses are also somewhat different.

One possible protection against this attack would be for the server to provide a secure message digest along with the entity-tag based indicia value. Under the assumption that the attacker cannot construct a bogus resource instance with the same message-digest value, this should protect against the spoofing attack. However, if the server does provide a secure message digest, it seems preferable to use this directly as the indicia value, rather than to use it only as prevention against spoofing.

If the responses in the proxy's cache are signed by the origin server, this would allow the client to detect spoofing (provided that the client has some means of discovering the true identity of the server). It might be possible to use the proposed Digest Authentication scheme [6] for this purpose, but we have not done the necessary analysis. Also, restriction of duplicate suppression to properly signed responses greatly restricts the potential benefits of duplication suppression, by eliminating the possibility of using a duplicate response from any of a large set of servers.

Another defense against such an attack is for the proxy to ignore a ``DCluster'' header that specifies a different server from that in

the Request-URI. However, this defense is ineffective if a server is shared among multiple, possibly mutually untrustworthy, content providers. As with signature-based defenses, it also greatly reduces the potential effectiveness of duplicate suppression.

We recommend, therefore, that the use of entity tags as indicia values be restricted to environments, such as well-protected intranets, where the threat of spoofing attacks is prevented by other means (such as a trustworthy community of employees).

7.3 False association of an indicia with a URI

In the third kind of attack, a malicious server tells a client that the current instance of a resource R1 has a given indicia value I, and thereby induces the client to make a request for that resource with a SubOK header field carrying indicia value I. If the attacker can predict that the request will be sent via a proxy P which has a cache entry for an instance of a resource R2 that also has an indicia value of I, then the client may receive an instance of R2 in its response. This would be bad if the true value of R1 was actually quite different.

This attack has two potential consequences. It might be that the attacker really is the origin server for resource R1, but for some reason wants to be able to convince the client that the value of R1 is something different. It is not clear if this kind of attack is a serious threat.

The other consequence is more significant. If the attacker can provide the client with bogus indicia for a resource R1 that it does not actually control, then the attacker can easily cause the client to see the wrong apparent value for R1, with the unwitting assistance of the proxy. The provision of a secure message digest with, or as, the indicia value does not help in this case, since the attacker could easily provide the message digest for R2.

This implies that clients should not accept indicia information for a resource from a server that cannot be trusted to provide honest information about a resource. For example, if a DRP index data structure sent by server evilhacker.com provides an indicia for a resource at server victimserver.com, the client should probably ignore the indicia, unless it has additional information implying that evilhacker.com reliably speaks on behalf of victimserver.com. (This defense breaks down if a server hostname is shared among multiple, possibly mutually untrustworthy, content providers.)

8 Acknowledgements

Andrei Broder helped improve our understanding of digest and fingerprint functions.

9 References

NOTE TO RFC EDITOR: many of the references here might be out of date. Please verify these with the primary author of this Internet-Draft before issuing this document as an RFC.

1. N. Borenstein and N. Freed. MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. RFC 1521, IETF, September, 1993.

2. Andrei Broder. Some applications of Rabin's fingerprinting method. In R. Capocelli, A. De Santis, and U. Vaccaro, Ed., Sequences II: Methods in Communications, Security, and Computer Science, Springer-Verlag, 1993, pp. 143-152.

3. Andrei Z. Broder, Steven C. Glassman, and Mark S. Manasse. Syntactic Clustering of the Web. Proc. 6th International World Wide Web Conf., Santa Clara, CA, April, 1997, pp. 391-404. http://www6.nttlabs.com/HyperNews/get/PAPER205.html.

4. 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. To appear.

5. Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, and Tim Berners-Lee. Hypertext Transfer Protocol -- HTTP/1.1. RFC 2068, HTTP Working Group, January, 1997.

6. J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, L. Stewart. An Extension to HTTP: Digest Access Authentication. RFC 2069, HTTP Working Group, January, 1997.

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

8. Y. Y. Goland, E. J. Whitehead, Jr., A. Faizi, S. R. Carter, and D. Jensen. Extensions for Distributed Authoring on the World Wide Web -- WEBDAV. Internet-Draft draft-ietf-webdav-protocol-06, HTTP Working Group, January, 1998. This is a work in progress.

9. 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. URL http://www.w3.org/TR/NOTE-drp-19970825.html.

10. Balachander Krishnamurthy and Craig E. Willis. Piggyback server invalidation for proxy cache coherency. Proc. 7th International World Wide Web Conf., Australia, April, 1998, pp. ??-??. To appear.

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

12. Merriam-Webster. Webster's Third New International Dictionary.

Merriam-Webster Inc., Springfield, MA, 1986.

13. Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, Yaron Goland, and Arthur van Hoff. Delta encoding in HTTP. Internet-Draft draft-mogul-http-delta-00, HTTP Working Group, January, 1998. This is a work in progress.

14. Jeffrey C. Mogul and Paul Leach. Simple Hit-Metering and Usage-Limiting for HTTP. RFC 2227, Internet Engineering Task Force, October, 1997.

15. Jeffrey C. Mogul and Arthur Van Hoff. Instance Digests in HTTP. Internet-Draft draft-mogul-http-digest-00, HTTP Working Group, January, 1998. This is a work in progress.

16. National Institute of Standards and Technology. Secure Hash Standard. FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION 180-1, U.S. Department of Commerce, April, 1995. http://csrc.nist.gov/fips/fip180-1.txt.

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

18. Venkata N. Padmanabhan and Jeffrey C. Mogul. "Using Predictive Prefetching to Improve World Wide Web Latency". Computer Communication Review 26, 3 (1996), 22-36.

19. M. O. Rabin. Fingerprinting by random polynomials. Report TR-15-81, Department of Computer Science, Harvard University, 1981.

20. R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321, Network Working Group, April, 1992.

10 Authors' addresses

Jeffrey C. Mogul
Western Research Laboratory
Digital Equipment Corporation
250 University Avenue
Palo Alto, California, 94305, U.S.A.
Email: mogul@wrl.dec.com
Phone: 1 650 617 3304 (email preferred)

Arthur van Hoff
Marimba, Inc.
440 Clyde Avenue
Mountain View, CA 94043
1 (650) 930 5283
avh@marimba.com