Network Working Group
Request for Comments: 2227
Category: Standards Track

J. Mogul
DECWRL
P. Leach
Microsoft
October 1997

Simple Hit-Metering and Usage-Limiting for HTTP

Status of this Memo

Copyright Notice

ABSTRACT

This document proposes a simple extension to HTTP, using a new "Meter" header, which permits a limited form of demographic information (colloquially called "hit-counts") to be reported by caches to origin servers, in a more efficient manner than the "cache-busting" techniques currently used. It also permits an origin server to control the number of times a cache uses a cached response, and outlines a technique that origin servers can use to capture referral information without "cache-busting."

TABLE OF CONTENTS

1 Introduction

For a variety of reasons, content providers want to be able to collect information on the frequency with which their content is accessed. This desire leads to some of the "cache-busting" done by existing servers. ("Cache-busting" is the use by servers of techniques intended to prevent caching of responses; it is unknown exactly how common this is.) This kind of cache-busting is done not for the purpose of maintaining transparency or security properties, but simply to collect demographic information. Some cache-busting is also done to provide different advertising images to appear on the same page (i.e., each retrieval of the page sees a different ad).

This proposal supports a model similar to that of publishers of hard-copy publications: such publishers (try to) report to their advertisers how many people read an issue of a publication at least once; they don't (try to) report how many times a reader re-reads an issue. They do this by counting copies published, and then try to estimate, for their publication, on average how many people read a

single copy at least once. The key point is that the results aren't exact, but are still useful. Another model is that of coding inquiries in such a way that the advertiser can tell which publication produced the inquiry.

1.1 Goals, non-goals, and limitations

HTTP/1.1 already allows origin servers to prevent caching of responses, and evidence exists [9] that at least some of the time, this is being done for the sole purpose of collecting counts of the number of accesses of specific pages. Some of this evidence is inferred from the study of proxy traces; some is based on explicit statements of the intention of the operators of Web servers. Information collected this way might or might not be of actual use to the people who collect it; the fact is that they want to collect it, or already do so.

The goal of this proposal is to provide an optional performance optimization for this use of HTTP/1.1.

This specification is:

- Optional: no server or proxy is required to implement it.
- Proxy-centered: there is no involvement on the part of end-client implementations.
- Solely a performance optimization: it provides no information or functionality that is not already available in HTTP/1.1. The intent is to improve performance overall, and reduce latency for almost all interactions; latency might be increased for a small fraction of HTTP interactions.
- Best-efforts: it does not guarantee the accuracy of the reported information, although it does provide accurate results in the absence of persistent network failures or host crashes.
- Neutral with respect to privacy: it reveals to servers no information about clients that is not already available through the existing features of HTTP/1.1.

The goals of this specification do not include:

- Solving the entire problem of efficiently obtaining extensive information about requests made via proxies.

- Improving the protection of user privacy (although our proposal may reduce the transfer of user-specific information to servers, it does not prevent it).

- Preventing or encouraging the use of log-exchange mechanisms.

- Avoiding all forms of "cache-busting", or even all cache-busting done for gathering counts.

This design has certain potential limitations:

- If it is not deployed widely in both proxies and servers, it will provide little benefit.

- It may, by partially solving the hit-counting problem, reduce the pressure to adopt more complete solutions, if any become available.

- Even if widely deployed, it might not be widely used, and so might not significantly improve performance.

These potential limitations might not be problems in actual practice.

1.2 Brief summary of the design

This section is included for people not wishing to read the entire document; it is not a specification for the proposed design, and over-simplifies many aspects of the design.

The goal of this design is to eliminate the need for origin servers to use "cache-busting" techniques, when this is done just for the purpose of counting the number of users of a resource. (Cache-busting includes techniques such as setting immediate Expiration dates, or sending "Cache-control: private" in each response.)

The design adds a new "Meter" header to HTTP; the header is always protected by the "Connection" header, and so is always hop-by-hop. This mechanism allows the construction of a "metering subtree", which is a connected subtree of proxies, rooted at an origin server. Only those proxies that explicitly volunteer to join in the metering subtree for a resource participate in hit-metering, but those proxies that do volunteer are required to make their best effort to provide accurate counts. When a hit-metered response is forwarded outside of the metering subtree, the forwarding proxy adds "Cache-control: s-maxage=0", so that other proxies (outside the metering subtree) are forced to forward all requests to a server in the metering subtree.

NOTE: the HTTP/1.1 specification does not currently define a "s-maxage" Cache-control directive. The HTTP working group has decided to add such a directive to the next revision of the HTTP/1.1 specification [7].

The Meter header carries zero or more directives, similar to the way that the Cache-control header carries directives. Proxies may use certain Meter directives to volunteer to do hit-metering for a resource. If a proxy does volunteer, the server may use certain directives to require that a response be hit-metered. Finally, proxies use a "count" Meter directive to report the accumulated hit counts.

The Meter mechanism can also be used by a server to limit the number of uses that a cache may make of a cached response, before revalidating it.

The full specification includes complete rules for counting "uses" of a response (e.g., non-conditional GETs) and "reuses" (conditional GETs). These rules ensure that the results are entirely consistent in all cases, except when systems or networks fail.

1.3 Terminology

This document uses terms defined and explained in the HTTP/1.1 specification [4], including "origin server," "resource," "hop-by-hop," "unconditional GET," and "conditional GET." The reader is expected to be familiar with the HTTP/1.1 specification and its terminology.

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1].

2 Overview

The design described in this document introduces several new features to HTTP:

- Hit-metering: allows an origin server to obtain reasonably accurate counts of the number of clients using a resource instance via a proxy cache, or a hierarchy of proxy caches.

- Usage-limiting: allows an origin server to control the number of times a cached response may be used by a proxy cache, or a hierarchy of proxy caches, before revalidation with the origin server.

These new non-mandatory features require minimal new protocol support, no change in protocol version, relatively little overhead in message headers. The design adds no additional network round-trips in any critical path that directly affects user-perceived latency (see section 4.3 for an analysis).

The primary goal of hit-metering and usage-limiting is to obviate the need for an origin server to send "Cache-control: s-maxage=0" with responses for resources whose value is not likely to change immediately. In other words, in cases where the only reason for contacting the origin server on every request that might otherwise be satisfied by a proxy cache entry is to allow the server to collect demographic information or to control the number of times a cache entry is used, the extension proposed here will avoid a significant amount of unnecessary network traffic and latency.

This design introduces one new "Meter" header, which is used both in HTTP request messages and HTTP response messages. The Meter header is used to transmit a number of directives and reports. In particular, all negotiation of the use of hit-metering and usage limits is done using this header. No other changes to the existing HTTP/1.1 specification [4] are proposed in this document.

This design also introduces several new concepts:

1. The concepts of a "use" of a cache entry, which is when a proxy returns its entity-body in response to a conditional or non-conditional request, and the "reuse" of a cache entry, which is when a proxy returns a 304 (Not Modified) response to a conditional request which is satisfied by that cache entry.

2. The concept of a hit-metered resource, for which proxy caches make a best-effort attempt to report accurate counts of uses and/or reuses to the origin server.

3. The concept of a usage-limited resource, for which the origin server expects proxy caches to limit the number of uses and/or reuses.

The new Meter directives and reports interact to allow proxy caches and servers to cooperate in the collection of demographic data. The goal is a best-efforts approximation of the true number of uses and/or reuses, not a guaranteed exact count.

The new Meter directives also allow a server to bound the inaccuracy of a particular hit-count, by bounding the number of uses between reports. It can also, for example, bound the number of times the same ad is shown because of caching.

Section 7.1 describes a way to use server-driven content negotiation (the Vary header) that allows an HTTP origin server to flexibly separate requests into categories and count requests by category. Implementation of such a categorized hit counting is likely to be a very small modification to most implementations of Vary; some implementations may not require any modification at all.

2.1 Discussion

Mapping this onto the publishing model, a proxy cache would increment the use-count for a cache entry once for each unconditional GET done for the entry, and once for each conditional GET that results in sending a copy of the entry to update a client's invalid cached copy. Conditional GETs that result in 304 (Not Modified) are not included in the use-count, because they do not result in a new user seeing the page, but instead signify a repeat view by a user that had seen it before. However, 304 responses are counted in the reuse-count. HEADs are not counted at all, because their responses do not contain an entity-body.

The Meter directives apply only to shared proxy caches, not to end-client (or other single-user) caches. Single user caches should not use Meter, because their hits will be automatically counted as a result of the unconditional GET with which they first fetch the page, from either the origin-server or from a proxy cache. Their subsequent conditional GETs do not result in a new user seeing the page.

The mechanism specified here counts GETs; other methods either do not result in a page for the user to read, aren't cached, or are "written-through" and so can be directly counted by the origin server. (If, in the future, a "cachable POST" came into existence, whereby the entity-body in the POST request was used to select a cached response, then such POSTs would have to be treated just like GETs.) The applicability of hit-metering to any new HTTP methods that might be defined in the future is currently unspecifiable.

In the case of multiple caches along a path, a proxy cache does the obvious summation when it receives a use-count or reuse-count in a request from another cache.

3 Design concepts

In order to allow the introduction of hit-metering and usage-limiting without requiring a protocol revision, and to ensure a reasonably close approximation of accurate counts, the negotiation of metering and usage-limiting is done hop-by-hop, not end-to-end. If one considers the "tree" of proxies that receive, store, and forward a specific response, the intent of this design is that within some (possibly null) "metering subtree", rooted at the origin server, all proxies are using the hit-metering and/or usage-limiting requested by the origin server.

Proxies at the leaves of this subtree will insert a "Cache-control: s-maxage=0" directive, which forces all other proxies (below this subtree) to check with a leaf of the metering subtree on every request. However, it does not prevent them from storing and using the response, if the revalidation succeeds.

No proxy is required to implement hit-metering or usage-limiting. However, any proxy that transmits the Meter header in a request MUST implement every unconditional requirement of this specification, without exception or amendment.

This is a conservative design, which may sometimes fail to take advantage of hit-metering support in proxies outside the metering subtree. However, it is likely that without the reliability offered by a conservative design, managers of origin servers with requirements for accurate approximations will not take advantage of any hit-metering proposal.

The hit-metering/usage-limiting mechanism is designed to avoid any extra network round-trips in the critical path of any client request, and (as much as possible) to avoid excessively lengthening HTTP messages.

The Meter header is used to transmit both negotiation information and numeric information.

A formal specification for the Meter header appears in section 5; the following discussion uses an informal approach to improve clarity.

3.1 Implementation of the "metering subtree"

The "metering subtree" approach is implemented in a simple, straightforward way by defining the new "Meter" header as one that MUST always be protected by a Connection header in every request or response. I.e., if the Meter header is present in an HTTP message, that message:

1. MUST contain "Connection: meter", and MUST be handled according to the HTTP/1.1 specification of the Connection header.

2. MUST NOT be sent in response to a request from a client whose version number is less than HTTP/1.1.

3. MUST NOT be accepted from a client whose version number is less than HTTP/1.1.

The reason for the latter two restrictions is to protect against proxies that might not properly implement the Connection header. Otherwise, a subtree that includes an HTTP/1.0 proxy might erroneously appear to be a metering subtree.

> Note: It appears that for the Connection header mechanism to function correctly, a system receiving an HTTP/1.0 (or lower-version) message that includes a Connection header must act as if this header, and all of the headers it protects, ought to have been removed from the message by an intermediate proxy.
>
> Although RFC2068 does not specifically require this behavior, it appears to be implied. Otherwise, one could not depend on the stated property (section 14.10) that the protected options "MUST NOT be communicated by proxies over further connections." This should probably be clarified in a subsequent draft of the HTTP/1.1 specification.
>
> This specification does not, in any way, propose a modification of the specification of the Connection header.

From the point of view of an origin server, the proxies in a metering subtree work together to obey usage limits and to maintain accurate usage counts. When an origin server specifies a usage limit, a proxy in the metering subtree may subdivide this limit among its children in the subtree as it sees fit. Similarly, when a proxy in the subtree receives a usage report, it ensures that the hits represented by this report are summed properly and reported to the origin server.

When a proxy forwards a hit-metered or usage-limited response to a client (proxy or end-client) not in the metering subtree, it MUST omit the Meter header, and it MUST add "Cache-control: s-maxage=0" to the response.

3.2 Format of the Meter header

The Meter header is used to carry zero or more directives. Multiple Meter headers may occur in an HTTP message, but according to the rules in section 4.2 of the HTTP/1.1 specification [4], they may be combined into a single header (and should be so combined, to reduce overhead).

For example, the following sequence of Meter headers

```
Meter: max-uses=3
Meter: max-reuses=10
Meter: do-report
```

may be expressed as

```
Meter: max-uses=3, max-reuses=10, do-report
```

3.3 Negotiation of hit-metering and usage-limiting

An origin server that wants to collect hit counts for a resource, by simply forcing all requests to bypass any proxy caches, would respond to requests on the resource with "Cache-control: s-maxage=0". (An origin server wishing to prevent HTTP/1.0 proxies from improperly caching the response could also send both "Expires: <now>", to prevent such caching, and "Cache-control: max-age=NNNN", to allow newer proxies to cache the response).

The purpose of the Meter header is to obviate the need for "Cache-control: s-maxage=0" within a metering subtree. Thus, any proxy may negotiate the use of hit-metering and/or usage-limiting with the next-hop server. If this server is the origin server, or is already part of a metering subtree (rooted at the origin server), then it may complete the negotiation, thereby extending the metering subtree to include the new proxy.

To start the negotiation, a proxy sends its request with one of the following Meter directives:

`will-report-and-limit` indicates that the proxy is willing and able to return usage reports and will obey any usage-limits.

`wont-report` indicates that the proxy will obey usage-limits but will not send usage reports.

`wont-limit` indicates that the proxy will not obey usage-limits but will send usage reports.

A proxy willing to neither obey usage-limits nor send usage reports MUST NOT transmit a Meter header in the request.

By definition, an empty Meter header:

```
Meter:
```

is equivalent to "Meter: will-report-and-limit", and so, by the definition of the Connection header (see section 14.10 of the HTTP/1.1 specification [4]), a request that contains

```
Connection: Meter
```

and no explicit Meter header is equivalent to a request that contains

```
Connection: Meter
Meter: will-report-and-limit
```

This makes the default case more efficient.

An origin server that is not interested in metering or usage-limiting the requested resource simply ignores the Meter header.

If the server wants the proxy to do hit-metering and/or usage-limiting, its response should include one or more of the following Meter directives:

For hit-metering:

do-report	specifies that the proxy MUST send usage reports to the server.
dont-report	specifies that the proxy SHOULD NOT send usage reports to the server.
timeout=NNN	sets a metering timeout of NNN minutes, from the time that this response was originated, for the reporting of a hit-count. If the proxy has a non-zero hit count for this response when the timeout expires, it MUST send a report to the server at or before that time. Implies "do-report".

By definition, an empty Meter header in a response, or any Meter header that does not contain "dont-report", means "Meter: do-report"; this makes a common case more efficient.

Note: an origin server using the metering timeout mechanism to bound the collection period over which hit-counts are obtained should adjust the timeout values in the responses it sends so that all responses generated within that period reach their metering timeouts at or before the end of that period.

If the origin server simply sends a constant metering timeout T with each response for a resource, the reports that it receives will reflect activity over a period whose duration is between T and N*T (in the worst case), where N is the maximum depth of the metering subtree.

For usage-limiting

max-uses=NNN sets an upper limit of NNN "uses" of the response, not counting its immediate forwarding to the requesting end-client, for all proxies in the following subtree taken together.

max-reuses=NNN sets an upper limit of NNN "reuses" of the response for all proxies in the following subtree taken together.

When a proxy has exhausted its allocation of "uses" or "reuses" for a cache entry, it MUST revalidate the cache entry (using a conditional request) before returning it in a response. (The proxy SHOULD use this revalidation message to send a usage report, if one was requested and it is time to send it. See sections 3.4 and 3.5.)

These Meter response-directives apply only to the specific response that they are attached to.

Note that the limit on "uses" set by the max-uses directive does not include the use of the response to satisfy the end-client request that caused the proxy's request to the server. This counting rule supports the notion of a cache-initiated prefetch: a cache may issue a prefetch request, receive a max-uses=0 response, store that response, and then return that response (without revalidation) when a client makes an actual request for the resource. However, each such response may be used at most once in this way, so the origin server maintains precise control over the number of actual uses.

A server MUST NOT send a Meter header that would require a proxy to do something that it has not yet offered to do. A proxy receiving a Meter response-directive asking the proxy to do something it did not volunteer to do SHOULD ignore that directive.

A proxy receiving a Meter header in a response MUST either obey it, or it MUST revalidate the corresponding cache entry on every access. (I.e., if it chooses not to obey the Meter header in a response, it MUST act as if the response included "Cache-control: s-maxage=0".)

Note: a proxy that has not sent the Meter header in a request for the given resource, and which has therefore not volunteered to honor Meter directives in a response, is not required to honor them. If, in this situation, the server does send a Meter header in a response, this is a protocol error. However, based on the robustness principle, the proxy may choose to interpret the Meter header as an implicit request to include "Cache-control: s-maxage=0" when it forwards the response, since this preserves the apparent intention of the server.

A proxy that receives the Meter header in a request may ignore it only to the extent that this is consistent with its own duty to the next-hop server. If the received Meter request header is inconsistent with that duty, or if no Meter request header is received and the response from the next-hop server requests any form of metering or limiting, then the proxy MUST add "Cache-control: s-maxage=0" to any response it forwards for that request. (A proxy SHOULD NOT add or change the Expires header or max-age Cache-control directive.)

For example, if proxy A receives a GET request from proxy B for URL X with "Connection: Meter", but proxy A's cached response for URL does not include any Meter directives, then proxy A may ignore the metering offer from proxy B.

However, if proxy A has previously told the origin server "Meter: wont-limit" (implying will-report), and the cached response contains "Meter: do-report", and proxy B's request includes "Meter: wont-report", then proxy B's offer is inconsistent with proxy A's duty to the origin server. Therefore, in this case proxy A must add "Cache-control: s-maxage=0" when it returns the cached response to proxy B, and must not include a Meter header in this response.

If a server does not want to use the Meter mechanism, and will not want to use it any time soon, it may send this directive:

wont-ask recommends that the proxy SHOULD NOT send any Meter directives to this server.

The proxy SHOULD remember this fact for up to 24 hours. This avoids virtually all unnecessary overheads for servers that do not wish to use or support the Meter header. (This directive also implies "dont-report".)

3.4 Transmission of usage reports

To transmit a usage report, a proxy sends the following Meter header in a request on the appropriate resource:

```
Meter: count=NNN/MMM
```

The first integer indicates the count of uses of the cache entry since the last report; the second integer indicates the count of reuses of the entry (see section 5.3 for rules on counting uses and reuses). The transmission of a "count" directive in a request with no other Meter directive is also defined as an implicit transmission of a "will-report-and-limit" directive, to optimize the common case. (A proxy not willing to honor usage-limits would send "Meter: count=NNN/MMM, wont-limit" for its reports.)

Note that when a proxy forwards a client's request and receives a response, the response that the proxy sends immediately to the requesting client is not counted as a "use". I.e., the reported count is the number of times the cache entry was used, and not the number of times that the response was used.

A proxy SHOULD NOT transmit "Meter: count=0/0", since this conveys no useful information.

Usage reports MUST always be transmitted as part of a conditional request (such as a GET or HEAD), since the information in the conditional header (e.g., If-Modified-Since or If-None-Match) is required for the origin server to know which instance of a resource is being counted. Proxys forwarding usage reports up the metering subtree MUST NOT change the contents of the conditional header, since otherwise this would result in incorrect counting.

A usage report MUST NOT be transmitted as part of a forwarded request that includes multiple entity tags in an If-None-Match or If-Match header.

> Note: a proxy that offers its willingness to do hit-metering (report usage) must count both uses and reuses. It is not possible to negotiate the reporting of one but not the other.

3.5 When to send usage reports

A proxy that has offered to send usage reports to its parent in the metering subtree MUST send a usage report in each of these situations:

1. When it forwards a conditional GET on the resource instance on behalf of one of its clients (if the GET is conditional on at most one entity-tag).

2. When it forwards a conditional HEAD on the resource instance on behalf of one of its clients.

3. When it must generate a conditional GET to satisfy a client request because the max-uses limit has been exceeded.

4. Upon expiration of a metering timeout associated with a cache entry that has a non-zero hit-count.

5. When it removes the corresponding non-zero hit-count entry from its storage for any reason including:

   - the proxy needs the storage space for another hit-count entry.

   - the proxy is not able to store more than one response per resource, and a request forwarded on behalf of a client has resulted in the receipt of a new response (one with a different entity-tag or last-modified time).

   Note that a cache might continue to store hit-count information even after having deleted the body of the response, so it is not necessary to report the hit-count when deleting the body; it is only necessary to report it if the proxy is about to "forget" a non-zero value.

(Section 5.3 explains how hit-counts become zero or non-zero.)

If the usage report is being sent because the proxy is about to remove the hit-count entry from its storage, or because of an expired metering timeout:

- The proxy MUST send the report as part of a conditional HEAD request on the resource instance.

- The proxy is not required to retry the HEAD request if it fails (this is a best-efforts design). To improve accuracy, however, the proxy SHOULD retry failed HEAD requests, subject to resource constraints.

- The proxy is not required to serialize any other operation on the completion of this request.

Note: proxy implementors are strongly encouraged to batch several HEAD-based reports to the same server, when possible, over a single persistent connection, to reduce network overhead as much as possible. This may involve a non-naive algorithm for scheduling the deletion of hit-count entries.

If the usage count is sent because of an arriving request that also carries a "count" directive, the proxy MUST combine its own (possibly zero) use and reuse counts with the arriving counts, and then attempt to forward the request.

However, the proxy is not required to forward an arriving request with a "count" directive, provided that:

- it can reply to the request using a cached response, in compliance with other requirements of the HTTP specification.

- such a response does not exceed a max-uses limit.

- it is not required to forward the request because of an expired metering timeout.

If an arriving request carries a "count" directive, and the proxy no longer has a cache entry for the resource, the proxy MUST forward the "count" directive. (This is, in any case, what a proxy without a suitable cache entry would normally do for any valid request it receives.)

3.6 Subdivision of usage-limits

When an origin server specifies a usage limit, a proxy in the metering subtree may subdivide this limit among its children in the subtree as it sees fit.

For example, consider the situation with two proxies P1 and P2, each of which uses proxy P3 as a way to reach origin server S. Imagine that S sends P3 a response with

Meter: max-uses=10

The proxies use that response to satisfy the current requesting end-client. The max-uses directive in this example allows the combination of P1, P2, and P3 together to satisfy 10 additional end-client uses (unconditional GETs) for the resource.

This specification does not constrain how P3 divides up that allocation among itself and the other proxies. For example, P3 could retain all of max-use allocation for itself. In that case, it would forward the response to P1 and/or P2 with

Meter: max-uses=0

P3 might also divide the allocation equally among P1 and P2, retaining none for itself (which may be the right choice if P3 has few or no other clients). In this case, it could send

Meter: max-uses=5

to the proxy (P1 or P2) that made the initial request, and then record in some internal data structure that it "owes" the other proxy the rest of the allocation.

Note that this freedom to choose the max-uses value applies to the origin server, as well. There is no requirement that an origin server send the same max-uses value to all caches. For example, it might make sense to send "max-uses=2" the first time one hears from a cache, and then double the value (up to some maximum limit) each time one gets a "use-count" from that cache. The idea is that the faster a cache is using up its max-use quota, the more likely it will be to report a use-count value before removing the cache entry. Also, high and frequent use-counts imply a corresponding high efficiency benefit from allowing caching.

Again, the details of such heuristics would be outside the scope of this specification.

4 Analysis

This section includes informal analyses of several aspects of hit-metering:

1. the accuracy of results when applied to counting users (section 4.1).

2. the problem of counting users whose browsers do not include caches, such as Network Computers (section 4.2).

3. delays imposed on "critical paths" for HTTP operations (section 4.3).

4.1 Approximation accuracy for counting users

For many (but not all) service operators, the single most important aspect of the request stream is the number of distinct users who have retrieved a particular entity within a given period (e.g., during a given day). The hit-metering mechanism is designed to provide an origin server with an approximation of the number of users that reference a given resource. The intent of the design is that the precision of this approximation is consistent with the goals of simplicity and optional implementation.

Almost all Web users use client software that maintains local caches, and the state of the art of local-caching technology is quite effective. (Section 4.2 discusses the case where end-client caches are small or non-existent.) Therefore, assuming an effective and persistent end-client cache, each individual user who retrieves an entity does exactly one GET request that results in a 200 or 203 response, or a 206 response that includes the first byte of the entity. If a proxy cache maintains and reports an accurate use-count of such retrievals, then its reported use-count will closely approximate the number of distinct users who have retrieved the entity.

There are some circumstances under which this approximation can break down. For example, if an entity stays in a proxy cache for much longer than it persists in the typical client cache, and users often re-reference the entity, then this scheme will tend to over-count the number of users. Or, if the cache-management policy implemented in typical client caches is biased against retaining certain kinds of frequently re-referenced entities (such as very large images), the use-counts reported will tend to overestimate the user-counts for such entities.

Browser log analysis has shown that when a user revisits a resource, this is almost always done very soon after the previous visit, almost always with fewer than eight intervening references [11]. Although this result might not apply universally, it implies that almost all reuses will hit in the end-client cache, and will not be seen as unconditional GETs by a proxy cache.

The existing (HTTP/1.0) "cache-busting" mechanisms for counting distinct users will certainly overestimate the number of users behind a proxy, since it provides no reliable way to distinguish between a user's initial request and subsequent repeat requests that might have been conditional GETs, had not cache-busting been employed. The

"Cache-control: s-maxage=0" feature of HTTP/1.1 does allow the separation of use-counts and reuse-counts, provided that no HTTP/1.0 proxy caches intervene.

Note that if there is doubt about the validity of the results of hit-metering a given set of resources, the server can employ cache-busting techniques for short periods, to establish a baseline for validating the hit-metering results. Various approaches to this problem are discussed in a paper by James Pitkow [9].

4.2 What about "Network Computers"?

The analysis in section 4.1 assumed that "almost all Web users" have client caches. If the Network Computers (NC) model becomes popular, however, then this assumption may be faulty: most proposed NCs have no disk storage, and relatively little RAM. Many Personal Digital Assistants (PDAs), which sometimes have network access, have similar constraints. Such client systems may do little or no caching of HTTP responses. This means that a single user might well generate many unconditional GETs that yield the same response from a proxy cache.

First note that the hit-metering design in this document, even with such clients, provides an approximation no worse than available with unmodified HTTP/1.1: the counts that a proxy would return to an origin server would represent exactly the number of requests that the proxy would forward to the server, if the server simply specifies "Cache-control: s-maxage=0".

However, it may be possible to improve the accuracy of these hit-counts by use of some heuristics at the proxy. For example, the proxy might note the IP address of the client, and count only one GET per client address per response. This is not perfect: for example, it fails to distinguish between NCs and certain other kinds of hosts. The proxy might also use the heuristic that only those clients that never send a conditional GET should be treated this way, although we are not at all certain that NCs will never send conditional GETs.

Since the solution to this problem appears to require heuristics based on the actual behavior of NCs (or perhaps a new HTTP protocol feature that allows unambiguous detection of cacheless clients), it appears to be premature to specify a solution.

4.3 Critical-path delay analysis

In systems (such as the Web) where latency is at issue, there is usually a tree of steps which depend on one another, in such a way that the final result cannot be accomplished until all of its predecessors have been. Since the tree structure admits some

parallelism, it is not necessary to add up the timings for each step to discover the latency for the entire process. But any single path through this dependency tree cannot be parallelized, and the longest such path is the one whose length (in units of seconds) determines the overall latency. This is the "critical path", because no matter how much shorter one makes any other path, that cannot change the overall latency for the final result.

If one views the final result, for a Web request, as rendering a page at a browser, or otherwise acting on the result of a request, clearly some network round trips (e.g., exchanging TCP SYN packets if the connection doesn't already exist) are on the critical path. This hit-metering design does add some round-trips for reporting non-zero counts when a cache entry is removed, but, by design, these are off any critical path: they may be done in parallel with any other operation, and require only "best efforts", so a proxy does not have to serialize other operations with their success or failure.

Clearly, anything that changes network utilization (either increasing or decreasing it) can indirectly affect user-perceived latency. Our expectation is that hit-metering, on average, will reduce loading and so even its indirect effects should not add network round-trips in any critical path. But there might be a few specific instances where the added non-critical-path operations (specifically, usage reports upon cache-entry removal) delay an operation on a critical path. This is an unavoidable problem in datagram networks.

5 Specification

5.1 Specification of Meter header and directives

The Meter general-header field is used to:

- Negotiate the use of hit-metering and usage-limiting among origin servers and proxy caches.

- Report use counts and reuse counts.

Implementation of the Meter header is optional for both proxies and origin servers. However, any proxy that transmits the Meter header in a request MUST implement every requirement of this specification, without exception or amendment.

The Meter header MUST always be protected by a Connection header. A proxy that does not implement the Meter header MUST NOT pass it through to another system (see section 5.5 for how a non-caching proxy may comply with this specification). If a Meter header is

received in a message whose version is less than HTTP/1.1, it MUST be ignored (because it has clearly flowed through a proxy that does not implement Meter).

A proxy that has received a response with a version less than HTTP/1.1, and therefore from a server (or another proxy) that does not implement the Meter header, SHOULD NOT send Meter request directives to that server, because these would simply waste bandwidth. This recommendation does not apply if the proxy is currently hit-metering or usage-limiting any responses from that server. If the proxy receives a HTTP/1.1 or higher response from such a server, it should cease its suppression of the Meter directives.

All proxies sending the Meter header MUST adhere to the "metering subtree" design described in section 3.

```
    Meter = "Meter" ":" 0#meter-directive

    meter-directive = meter-request-directive
                    | meter-response-directive
                    | meter-report-directive

    meter-request-directive =
                      "will-report-and-limit"
                    | "wont-report"
                    | "wont-limit"

    meter-report-directive =
                    | "count" "=" 1*DIGIT "/" 1*DIGIT

    meter-response-directive =
                      "max-uses" "=" 1*DIGIT
                    | "max-reuses" "=" 1*DIGIT
                    | "do-report"
                    | "dont-report"
                    | "timeout" "=" 1*DIGIT
                    | "wont-ask"
```

A meter-request-directive or meter-report-directive may only appear in an HTTP request message. A meter-response-directive may only appear in an HTTP response directive.

An empty Meter header in a request means "Meter: will-report-and-limit". An empty Meter header in a response, or any other response including one or more Meter headers without the "dont-report" or "wont-ask" directive, implies "Meter: do-report".

The meaning of the meter-request-directives are as follows:

will-report-and-limit
: indicates that the proxy is willing and able to return usage reports and will obey any usage-limits.

wont-report
: indicates that the proxy will obey usage-limits but will not send usage reports.

wont-limit
: indicates that the proxy will not obey usage-limits but will send usage reports.

A proxy willing neither to obey usage-limits nor to send usage reports MUST NOT transmit a Meter header in the request.

The meaning of the meter-report-directives are as follows:

count "=" 1*DIGIT "/" 1*DIGIT
: Both digit strings encode decimal integers. The first integer indicates the count of uses of the cache entry since the last report; the second integer indicates the count of reuses of the entry.

Section 5.3 specifies the counting rules.

The meaning of the meter-response-directives are as follows:

max-uses "=" 1*DIGIT
: sets an upper limit on the number of "uses" of the response, not counting its immediate forwarding to the requesting end-client, for all proxies in the following subtree taken together.

max-reuses "=" 1*DIGIT
: sets an upper limit on the number of "reuses" of the response for all proxies in the following subtree taken together.

do-report
: specifies that the proxy MUST send usage reports to the server.

dont-report
: specifies that the proxy SHOULD NOT send usage reports to the server.

timeout "=" 1*DIGIT
: sets a metering timeout of the specified number of minutes (not seconds) after the origination of this response (as indicated by its "Date" header). If the

proxy has a non-zero hit count for this response when the timeout expires, it MUST send a report to the server at or before that time. Timeouts should be implemented with an accuracy of plus or minus one minute. Implies "do-report".

wont-ask specifies that the proxy SHOULD NOT send any Meter headers to the server. The proxy should forget this advice after a period of no more than 24 hours.

Section 5.3 specifies the counting rules, and in particular specifies a somewhat non-obvious interpretation of the max-uses value.

5.2 Abbreviations for Meter directives

To allow for the most efficient possible encoding of Meter headers, we define abbreviated forms of all Meter directives. These are exactly semantically equivalent to their non-abbreviated counterparts. All systems implementing the Meter header MUST implement both the abbreviated and non-abbreviated forms. Implementations SHOULD use the abbreviated forms in normal use.

The abbreviated forms of Meter directive are shown below, with the corresponding non-abbreviated literals in the comments:

```
    Abb-Meter = "Meter" ":" 0#abb-meter-directive

    abb-meter-directive = abb-meter-request-directive
                        | abb-meter-response-directive
                        | abb-meter-report-directive

    abb-meter-request-directive =
                          "w"           ; "will-report-and-limit"
                        | "x"           ; "wont-report"
                        | "y"           ; "wont-limit"

    abb-meter-report-directive =
                        | "c" "=" 1*DIGIT "/" 1*DIGIT   ; "count"

    abb-meter-response-directive =
                          "u" "=" 1*DIGIT       ; "max-uses"
                        | "r" "=" 1*DIGIT       ; "max-reuses"
                        | "d"                   ; "do-report"
                        | "e"                   ; "dont-report"
                        | "t" "=" 1*DIGIT       ; "timeout"
                        | "n"                   ; "wont-ask"
```

Note: although the Abb-Meter BNF rule is defined separately from the Meter rule, one may freely mix abbreviated and non-abbreviated Meter directives in the same header.

5.3 Counting rules

Note: please remember that hit-counts and usage-counts are associated with individual responses, not with resources. A cache entry that, over its lifetime, holds more than one response is also not a "response", in this particular sense.

Let R be a cached response, and V be the value of the Request-URI and selecting request-headers (if any, see section 14.43 of the HTTP/1.1 specification [4]) that would select R if contained in a request. We define a "use" of R as occurring when the proxy returns its stored copy of R in a response with any of the following status codes: a 200 (OK) status; a 203 (Non-Authoritative Information) status; or a 206 (Partial Content) status when the response contains byte #0 of the entity (see section 5.4 for a discussion of Range requests).

Note: when a proxy forwards a client's request and receives a response, the response that the proxy sends immediately to the requesting client is not counted as a "use". I.e., the reported count is the number of times the cache entry was used, and not the number of times that the response was used.

We define a "reuse" of R as as occurring when the proxy responds to a request selecting R with a 304 (Not Modified) status, unless that request is a Range request that does not specify byte #0 of the entity.

5.3.1 Counting rules for hit-metering

A proxy participating in hit-metering for a cache response R maintains two counters, CU and CR, associated with R. When a proxy first stores R in its cache, it sets both CU and CR to 0 (zero). When a subsequent client request results in a "use" of R, the proxy increments CU. When a subsequent client request results in a "reuse" of R, the proxy increments CR. When a subsequent client request selecting R (i.e., including V) includes a "count" Meter directive, the proxy increments CU and CR using the corresponding values in the directive.

When the proxy sends a request selecting R (i.e., including V) to the inbound server, it includes a "count" Meter directive with the current CU and CR as the parameter values. If this request was caused by the proxy's receipt of a request from a client, upon receipt of the server's response, the proxy sets CU and CR to the

number of uses and reuses, respectively, that may have occurred while the request was in progress. (These numbers are likely, but not certain, to be zero.) If the proxy's request was a final HEAD-based report, it need no longer maintain the CU and CR values, but it may also set them to the number of intervening uses and reuses and retain them.

5.3.2 Counting rules for usage-limiting

A proxy participating in usage-limiting for a response R maintains either or both of two counters TU and TR, as appropriate, for that resource. TU and TR are incremented in just the same way as CU and CR, respectively. However, TU is zeroed only upon receipt of a "max-uses" Meter directive for that response (including the initial receipt). Similarly, TR is zeroed only upon receipt of a "max-reuses" Meter directive for that response.

A proxy participating in usage-limiting for a response R also stores values MU and/or MR associated with R. When it receives a response including only a max-uses value, it sets MU to that value and MR to infinity. When it receives a response including only a max-reuses value, it sets MR to that value and MU to infinity. When it receives a response including both max-reuses and max-reuses values, it sets MU and MR to those values, respectively. When it receives a subsequent response including neither max-reuses nor max-reuses values, it sets both MU and MR to infinity.

If a proxy participating in usage-limiting for a response R receives a request that would cause a "use" of R, and TU >= MU, it MUST forward the request to the server. If it receives a request that would cause a "reuse" of R, and TR >= MR, it MUST forward the request to the server. If (in either case) the proxy has already forwarded a previous request to the server and is waiting for the response, it should delay further handling of the new request until the response arrives (or times out); it SHOULD NOT have two revalidation requests pending at once that select the same response, unless these are Range requests selecting different subranges.

There is a special case of this rule for the "max-uses" directive: if the proxy receives a response with "max-uses=0" and does not forward it to a requesting client, the proxy should set a flag PF associated with R. If R is true, then when a request arrives while if TU >= MU, if the PF flag is set, then the request need not be forwarded to the server (provided that this is not required by other caching rules). However, the PF flag MUST be cleared on any use of the response.

Note: the "PF" flag is so named because this feature is useful only for caches that could issue a "prefetch" request before an actual client request for the response. A proxy not implementing prefetching need not implement the PF flag.

5.3.3 Equivalent algorithms are allowed

Any other algorithm that exhibits the same external behavior (i.e., generates exactly the same requests from the proxy to the server) as the one in this section is explicitly allowed.

Note: in most cases, TU will be equal to CU, and TR will be equal to CR. The only two cases where they could differ are:

1. The proxy issues a non-conditional request for the resource using V, while TU and/or TR are non-zero, and the server's response includes a new "max-uses" and/or "max-reuses" directive (thus zeroing TU and/or TR, but not CU and CR).

2. The proxy issues a conditional request reporting the hit-counts (and thus zeroing CU and CR, but not TU or TR), but the server's response does not include a new "max-uses" and/or "max-reuses" directive.

To solve the first case, the proxy has several implementation options

- Always store TU and TR separately from CU and CR.

- Create "shadow" copies of TU and TR when this situation arises (analogous to "copy on write").

- Generate a HEAD-based usage report when the non-conditional request is sent (or when the "max-uses=0" is received), causing CU and CR to be zeroed (analogous in some ways to a "memory barrier" instruction).

In the second case, the server implicitly has removed the usage-limit(s) on the response (by setting MU and/or MR to infinity), and so the fact that, say, TU is different from CU is not significant.

Note: It may also be possible to eliminate the PF flag by sending extra HEAD-based usage-report requests, but we recommend against this; it is better to allocate an extra bit per entry than to transmit extra requests.

5.4 Counting rules: interaction with Range requests

HTTP/1.1 allows a client to request sub-ranges of a resource. A client might end up issuing several requests with the net effect of receiving one copy of the resource. For uniformity of the results seen by origin servers, proxies need to observe a rule for counting these references, although it is not clear that one rule generates accurate results in every case.

The rule established in this specification is that proxies count as a "use" or "reuse" only those Range requests that result in the return of byte #0 of the resource. The rationale for this rule is that in almost every case, an end-client will retrieve the beginning of any resource that it references at all, and that it will seldom retrieve any portion more than once. Therefore, this rule appears to meet the goal of a "best-efforts" approximation.

5.5 Implementation by non-caching proxies

A non-caching proxy may participate in the metering subtree; this is strongly recommended.

A non-caching proxy (HTTP/1.1 or higher) that participates in the metering subtree SHOULD forward Meter headers on both requests and responses, with the appropriate Connection headers.

If a non-caching proxy forwards Meter headers, it MUST comply with these restrictions:

1. If the proxy forwards Meter headers in responses, such a response MUST NOT be returned to any request except the one that elicited it.

2. Once a non-caching proxy starts forwarding Meter headers, it should not arbitrarily stop forwarding them (or else reports may be lost).

A proxy that caches some responses and not others, for whatever reason, may choose to implement the Meter header as a caching proxy for the responses that it caches, and as a non-caching proxy for the responses that it does not cache, as long as its external behavior with respect to any particularly response is fully consistent with this specification.

5.6 Implementation by cooperating caches

Several HTTP cache implementations, most notably the Harvest/Squid cache [2], create cooperative arrangements between several caches. If such caches use a protocol other than HTTP to communicate between themselves, such as the Internet Cache Protocol (ICP) [12], and if they implement the Meter header, then they MUST act to ensure that their cooperation does not violate the intention of this specification.

In particular, if one member of a group of cooperating caches agrees with a server to hit-meter a particular response, and then passes this response via a non-HTTP protocol to a second cache in the group, the caches MUST ensure that the server which requested the metering receives reports that appropriately account for any uses or resues made by the second cache. Similarly, if the first cache agreed to usage-limit the response, the total number of uses by the group of caches MUST be limited to the agreed-upon number.

6 Examples

6.1 Example of a complete set of exchanges

This example shows how the protocol is intended to be used most of the time: for hit-metering without usage-limiting. Entity bodies are omitted.

A client sends request to a proxy:

```
GET http://foo.com/bar.html HTTP/1.1
```

The proxy forwards request to the origin server:

```
GET /bar.html HTTP/1.1
Host: foo.com
Connection: Meter
```

thus offering (implicitly) "will-report-and-limit".

The server responds to the proxy:

```
HTTP/1.1 200 OK
Date: Fri, 06 Dec 1996 18:44:29 GMT
Cache-control: max-age=3600
Connection: meter
Etag: "abcde"
```

thus (implicitly) requiring "do-report" (but not requiring usage-limiting).

The proxy responds to the client:

```
HTTP/1.1 200 OK
Date: Fri, 06 Dec 1996 18:44:29 GMT
Etag: "abcde"
Cache-control: max-age=3600, proxy-mustcheck
Age: 1
```

Since the proxy does not know if its client is an end-system, or a proxy that doesn't do metering, it adds the "proxy-mustcheck" directive.

Another client soon asks for the resource:

```
GET http://foo.com/bar.html HTTP/1.1
```

and the proxy sends the same response as it sent to the other client, except (perhaps) for the Age value.

After an hour has passed, a third client asks for the response:

```
GET http://foo.com/bar.html HTTP/1.1
```

But now the response's max-age has been exceeded, so the proxy revalidates the response with the origin server:

```
GET /bar.html HTTP/1.1
If-None-Match: "abcde"
Host: foo.com
Connection: Meter
Meter: count=1/0
```

thus simultaneously fulfilling its duties to validate the response and to report the one "use" that wasn't forwarded.

The origin server responds:

```
HTTP/1.1 304 Not Modified
Date: Fri, 06 Dec 1996 19:44:29 GMT
Cache-control: max-age=3600
Etag: "abcde"
```

so the proxy can use the original response to reply to the new client; the proxy also zeros the use-count it associates with that response.

Another client soon asks for the resource:

```
GET http://foo.com/bar.html HTTP/1.1
```

and the proxy sends the appropriate response.

After another few hours, the proxy decides to remove the cache entry. When it does so, it sends to the origin server:

```
HEAD /bar.html HTTP/1.1
If-None-Match: "abcde"
Host: foo.com
Connection: Meter
Meter: count=1/0
```

reporting that one more use of the response was satisfied from the cache.

6.2 Protecting against HTTP/1.0 proxies

An origin server that does not want HTTP/1.0 caches to store the response at all, and is willing to have HTTP/1.0 end-system clients generate excess GETs (which will be forwarded by HTTP/1.0 proxies) could send this for its reply:

```
HTTP/1.1 200 OK
Cache-control: max-age=3600
Connection: meter
Etag: "abcde"
Expires: Sun, 06 Nov 1994 08:49:37 GMT
```

HTTP/1.0 caches will see the ancient Expires header, but HTTP/1.1 caches will see the max-age directive and will ignore Expires.

> Note: although most major HTTP/1.0 proxy implementations observe the Expires header, it is possible that some are in use that do not. Use of the Expires header to prevent caching by HTTP/1.0 proxies might not be entirely reliable.

6.3 More elaborate examples

Here is a request from a proxy that is willing to hit-meter but is not willing to usage-limit:

```
GET /bar.html HTTP/1.1
Host: foo.com
Connection: Meter
Meter: wont-limit
```

Here is a response from an origin server that does not want hit counting, but does want "uses" limited to 3, and "reuses" limited to 6:

```
HTTP/1.1 200 OK
Cache-control: max-age=3600
Connection: meter
Etag: "abcde"
Expires: Sun, 06 Nov 1994 08:49:37 GMT
Meter: max-uses=3, max-reuses=6, dont-report
```

Here is the same example with abbreviated Meter directive names:

```
HTTP/1.1 200 OK
Cache-control: max-age=3600
Connection: meter
Etag: "abcde"
Expires: Sun, 06 Nov 1994 08:49:37 GMT
Meter:u=3,r=6,e
```

7 Interactions with content negotiation

This section describes two aspects of the interaction between hit-metering and "content-negotiated" resources:

1. treatment of responses carrying a Vary header (section 7.1).

2. treatment of responses that use the proposed Transparent Content Negotiation mechanism (section 7.2).

7.1 Treatment of responses carrying a Vary header

Separate counts should be kept for each combination of the headers named in the Vary header for the Request-URI (what [4] calls "the selecting request-headers"), even if they map to the same entity-tag. This rule has the effect of counting hits on each variant, if there are multiple variants of a page available.

> Note: This interaction between Vary and the hit-counting directives allows the origin server a lot of flexibility in specifying how hits should be counted. In essence, the origin server uses the Vary mechanism to divide the requests for a resource into arbitrary categories, based on the request- headers. (We will call these categories "request-patterns".) Since a proxy keeps its hit-counts for each request-pattern, rather than for each resource, the origin server can obtain separate statistics for many aspects of an HTTP request.

For example, if a page varied based on the value of the User-Agent header in the requests, then hit counts would be kept for each different flavor of browser. But it is in fact more general than that; because multiple header combinations can map to the same variant, it also enables the origin server to count the number of times (e.g.) the Swahili version of a page was requested, even though it is only available in English.

If a proxy does not support the Vary mechanism, then [4] says that it MUST NOT cache any response that carries a Vary header, and hence need not implement any aspect of this hit-counting or usage-limiting design for varying resources.

> Note: this also implies that if a proxy supports the Vary mechanism but is not willing to maintain independent hit-counts for each variant response in its cache, then it must follow at least one of these rules:
>
> 1. It must not use the Meter header in a request to offer to hit-meter or usage-limit responses.
>
> 2. If it does offer to hit-meter or usage-limit responses, and then receives a response that includes both a Vary header and a Meter header with a directive that it cannot satisfy, then the proxy must not cache the response.
>
> In other words, a proxy is allowed to partially implement the Vary mechanism with respect to hit-metering, as long as this has no externally visible effect on its ability to comply with the Meter specification.

This approach works for counting almost any aspect of the request stream, without embedding any specific list of countable aspects in the specification or proxy implementation.

7.2 Interaction with Transparent Content Negotiation

[A description of the interaction between this design and the proposed Transparent Content Negotiation (TCN) design [6] will be made available in a later document.]

8 A Note on Capturing Referrals

It is alleged that some advertisers want to pay content providers, not by the "hit", but by the "nibble" -- the number of people who actually click on the ad to get more information.

Now, HTTP already has a mechanism for doing this: the "Referer" header. However, perhaps it ought to be disabled for privacy reasons -- according the HTTP/1.1 spec:

> "Because the source of the link may be private information or may reveal an otherwise private information source, it is strongly recommended that the user be able to select whether or not the Referer field is sent."

However, in the case of ads, the source of the link actually wants to let the referred-to page know where the reference came from.

This does not require the addition of any extra mechanism, but rather can use schemes that embed the referrer in the URI in a manner similar to this:

```
http://www.blah.com/ad-reference?from=site1
```

Such a URI should point to a resource (perhaps a CGI script) which returns a 302 redirect to the real page

```
http://www.blah.com/ad-reference.html
```

Proxies which do not cache 302s will cause one hit on the redirection page per use, but the real page will get cached. Proxies which do cache 302s and report hits on the cached 302s will behave optimally.

This approach has the advantage that it works whether or not the end-client has disabled the use of Referer. Combined with the rest of the hit-metering proposal in this design, this approach allows, for example, an advertiser to know how often a reference to an advertisement was made from a particular page.

9 Alternative proposals

There might be a number of other ways of gathering demographic and usage information; other mechanisms might respond to a different set of needs than this proposal does. This proposal certainly does not preclude the proposal or deployment of other such mechanisms, and many of them may be complementary to and compatible with the mechanism proposed here.

There has been some speculation that statistical sampling methods might be used to gather reasonably accurate data. One such proposal is to manipulate cache expiration times so that selected resources are uncachable for carefully chosen periods, allowing servers to accurately count accesses during those periods. The hit-metering mechanism proposed here is entirely complementary to that approach,

since it could be used to reduce the cost of gathering those counts. James Pitkow has written a paper comparing an earlier draft of this hit-metering proposal with sampling approaches [9].

Phillip Hallam-Baker has proposed using a log-exchange protocol [5], by which a server could request a proxy's logs by making an HTTP request to the proxy. This proposal asserts that it is "believed to operate correctly in configurations involving multiple proxies", but it is not clear that this is true if an outer proxy is used as a (one-way) firewall. The proposal also leaves a number of open issues, such as how an origin server can be sure that all of the proxies in the request subtree actually support log-exchange. It is also not clear how this proposal couples a proxy's support of log-exchange to a server's permission to cache a response.

For general background on the topic of Web measurement standards, see the discussion by Thomas P. Novak and Donna L. Hoffman [8]. Also see the "Privacy and Demographics Overview" page maintained by by the World Wide Web Consortium [10], which includes a pointer to some tentative proposals for gathering consumer demographics (not just counting references) [3].

10 Security Considerations

Which outbound clients should a server (proxy or origin) trust to report hit counts? A malicious proxy could easily report a large number of hits on some page, and thus perhaps cause a large payment to a content provider from an advertiser. To help avoid this possibility, a proxy may choose to only relay usage counts received from its outbound proxies to its inbound servers when the proxies have authenticated themselves using Proxy-Authorization and/or they are on a list of approved proxies.

It is not possible to enforce usage limits if a proxy is willing to cheat (i.e., it offers to limit usage but then ignores a server's Meter directive).

Regarding privacy: it appears that the design in this document does not reveal any more information about individual users than would already be revealed by implementation of the existing HTTP/1.1 support for "Cache-control: max-age=0, proxy-revalidate" or "Cache-control: s-maxage=0". It may, in fact, help to conceal certain aspects of the organizational structure on the outbound side of a proxy. In any case, the conflict between user requirements for anonymity and origin server requirements for demographic information cannot be resolved by purely technical means.

11 Acknowledgments

We gratefully acknowledge the constructive comments received from Anselm Baird-Smith, Ted Hardie, Koen Holtman (who suggested the technique described in section 8), Dave Kristol, Ari Luotonen, Patrick R. McManus, Ingrid Melve, and James Pitkow.

12 References

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

2. Anwat Chankhunthod, Peter B. Danzig, Chuck Neerdaels, Michael F. Schwartz, and Kurt J. Worrell. A Hierarchical Internet Object Cache. Proc. 1996 USENIX Technical Conf., San Diego, January, 1996, pp. 153-163.

3. Daniel W. Connolly. Proposals for Gathering Consumer Demographics. http://www.w3.org/pub/WWW/Demographics/Proposals.html.

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

5. Phillip M. Hallam-Baker. Notification for Proxy Caches. W3C Working Draft WD-proxy-960221, World Wide Web Consortium, February, 1996. http://www.w3.org/pub/WWW/TR/WD-proxy.html.

6. Holtman, K., and A. Mutz, "Transparent Content Negotiation in HTTP", Work in Progress.

7. Mogul, J., "Forcing HTTP/1.1 proxies to revalidate responses", Work in Progress.

8. Thomas P. Novak and Donna L. Hoffman. New Metrics for New Media: Toward the Development of Web Measurement Standards. This is a draft paper, currently available at http://www2000.ogsm.vanderbilt.edu/novak/web.standards/webstand.html. Cited by permission of the author; do not quote or cite without permission.

9. James Pitkow. In search of reliable usage data on the WWW. Proc. Sixth International World Wide Web Conference, Santa Clara, CA, April, 1997.

10. Joseph Reagle, Rohit Khare, Dan Connolly, and Tim Berners-Lee. Privacy and Demographics Overview. http://www.w3.org/pub/WWW/Demographics/.

11. Linda Tauscher and Saul Greenberg. Revisitation Patterns in World Wide Web Navigation. Research Report 96/587/07, Department of Computer Science, University of Calgary, March, 1996. http://www.cpsc.ucalgary.ca/projects/grouplab/papers/96WebReuse/TechReport96.html.

12. Wessels, D., and K. Claffy "Internet Cache Protocol (ICP), version 2", RFC 2186, September 1997.

13 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: 1 415 617 3304 (email preferred)

Paul J. Leach
Microsoft
1 Microsoft Way
Redmond, Washington, 98052, U.S.A.

EMail: paulle@microsoft.com

14 Full Copyright Statement