Generation of the Age header field in HTTP/1.1

draft-mogul-http-age-00.txt

STATUS OF THIS MEMO

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

To learn the current status of any Internet-Draft, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

Distribution of this document is unlimited. Please send comments to the HTTP working group at <http-wg@cuckoo.hpl.hp.com>. Discussions of the working group are archived at <URL:http://www.ics.uci.edu/pub/ietf/http/> . General discussions about HTTP and the applications which use HTTP should take place on the <www-talk@w3.org> mailing list.

ABSTRACT

The 'Age' response-header field in HTTP/1.1 [RFC 2068] is intended to provide a lower bound of an estimate of a response message's age (time since generation), by explicitly indicating the amount of time that is known to have passed since the response message was retrieved or revalidated. There has been considerable controversy over when the Age header field should be added to a response. This document explains the issues, rebuts a previous proposal, and provides a set of proposed changes for the revision of RFC 2068.
1 Introduction

RFC2068 [3] defines, in section 14.6, the Age header field for HTTP/1.1:

The Age response-header field conveys the sender’s estimate of the amount of time since the response (or its revalidation) was generated at the origin server. A cached response is "fresh" if its age does not exceed its freshness lifetime. Age values are calculated as specified in section 13.2.3.

\[
\text{Age} = \text{"Age" ":" age-value}
\]

\[
\text{age-value} = \text{delta-seconds}
\]

Age values are non-negative decimal integers, representing time in seconds.

If a cache receives a value larger than the largest positive integer it can represent, or if any of its age calculations overflows, it MUST transmit an Age header with a value of 2147483648 (2^31). HTTP/1.1 caches MUST send an Age header in every response. Caches SHOULD use an arithmetic type of at least 31 bits of range.

Roy Fielding has taken issue with the statement "HTTP/1.1 caches MUST send an Age header in every response." [2]. Fielding correctly points out that the use of the word "caches" in this sentence is incorrect. He lists two possible rewordings for this sentence:
a) An HTTP/1.1 server that includes a cache MUST send an Age header field in every response.

b) An HTTP/1.1 server that includes a cache MUST include an Age header field in every response generated from its own cache.

There is, however, a third possible rewording, if one realizes that the original drafting error was to sloppily substitute the word ‘‘cache’’ where ‘‘proxy’’ was meant (mea culpa):

c) An HTTP/1.1 proxy MUST send an Age header field in every response.

Note that an HTTP proxy does not necessarily include a cache. The other sentences referring to ‘‘caches’’ in that paragraph reflect the same drafting error.

2 Proposed change to RFC2068

For this paragraph in section 14.6 (Age),

If a cache receives a value larger than the largest positive integer it can represent, or if any of its age calculations overflows, it MUST transmit an Age header with a value of 2147483648 (2^31). HTTP/1.1 caches MUST send an Age header in every response. Caches SHOULD use an arithmetic type of at least 31 bits of range.

substitute

If a proxy server receives a value larger than the largest positive integer it can represent, or if any of its age calculations overflows, it MUST transmit an Age header with a value of 2147483648 (2^31). An HTTP/1.1 proxy MUST send an Age header in every response. Proxies SHOULD use an arithmetic type of at least 31 bits of range to represent Age values.

3 Discussion

Because of facts of life such as resolution granularity, clock skews, and the theory of relativity, the Age value can never be exactly accurate; it is always an estimate. We would like to bound the inaccuracy of this estimate. Most important, we would like to ensure that any inaccuracy remaining in the Age estimate not cause avoidable and significant harm.
3.1 Quantifying the error of the Age estimate: Interpretation C

If interpretation (c) is used (i.e., the change proposed above in section 2), then the value of the Age header field might overestimate the actual amount of time since the response was generated at the origin server. One can calculate the size of the estimation error for a path containing N HTTP/1.1 proxies:

\[ \text{Error}_C = \text{Mean}_{\text{RTT}} \times N \times (N + 1)/2 \]

where Mean_{RTT} is the mean round-trip time (RTT) between neighboring pairs of proxies. Note that this measures the error in the Age value at the final recipient client, not the error at the last HTTP/1.1 proxy on the path.

Actually, this analysis assumes that the RTT on each hop is the Mean_{RTT}; in fact, the contribution from the hops closest to the origin server contribute more to the true error value than this simplified formula, and so it is not necessarily an upper bound on the error. If one uses

\[ \text{Error}_C_{\text{bound}} = \text{Max}_{\text{RTT}} \times N \times (N + 1)/2 \]

where Max_{RTT} is the largest of the RTTs over all of the hops, then the Error_C_bound estimate is an upper bound.

In order to quantify this bound, we need parameter values for N and Max_{RTT}. It’s not clear exactly how many proxy caches are used on an arbitrary path through the Internet, especially since many of these paths start within large intranets, but I think it would be realistic to assume that most such paths have somewhere between N=1 and N=6.

Clearly, the Max_{RTT} value is harder to quantify, since over extremely lossy paths, the RTT can be nearly infinite. As a surrogate for this value, I looked at the total retrieval time for each of 504736 retrievals in a two-day proxy trace made in December, 1996. (See [5] for more discussion of this trace.) Among these retrievals, the mean total retrieval time was 1.25 seconds, with a median of 0.25 seconds and a standard deviation of 13.7 seconds. As far as I can tell, a large part of the variation is due to servers that drop SYN packets due to buggy TCP stacks. Given this, and given that the median is closer to 1/4 second, and given that a full HTTP/1.0 retrieval takes at least two round-trips (and probably more), I think it’s fair to assume a value of 2 seconds as an approximate upper bound for the Max_{RTT} for most paths.

Therefore, doing the math

\[ \text{Error}_C_{\text{bound}}[N=1] = 2 \text{ seconds} \]
\[ \text{Error}_C_{\text{bound}}[N=6] = 42 \text{ seconds} \]

and, just for comparison, if one uses the observed median retrieval
time (0.25 seconds) and assumes the use of no more than 3 HTTP/1.1 proxies, then \( \text{Error}_C\_\text{bound} = 1.5 \) seconds.

### 3.2 Quantifying the error of the Age estimate: Interpretation B

If interpretation (b) is used, then the value of the Age header field might underestimate the actual amount of time since the response was generated at the origin server. The scenario that generates the greatest underestimate is when the path between the origin server and the HTTP/1.1 client’s cache includes an HTTP/1.0 proxy cache, and the client has a skewed clock. In particular, the client’s clock is set into the past by some amount.

In this scenario, since an HTTP/1.0 proxy cache will not add the Age header, there is no indication that the response was delayed by a cache between the origin server and the first HTTP/1.1 cache. The only way for that HTTP/1.1 cache to construct its initial Age estimate is to compare its own clock against the Date value. If the client’s clock is wrong, this error is, in general, impossible to correct. For example, suppose the client’s clock is one hour slow. If the origin server generates the response at 10:00AM, and then the response sits in the HTTP/1.0 cache for 65 minutes, when the response arrives at the HTTP/1.1 client, it will appear to be only 5 minutes old.

The benefit of using interpretation (c) is that if there is a non-caching HTTP/1.1 proxy between the HTTP/1.0 cache and the client, unless its clock is also skewed, it will add a nearly-correct Age header field to the response. I.e., the client will realize that the response is at least 65 minutes old, regardless of its clock skew. So, while the error is impossible to correct "in general", it is possible to correct in certain specific cases, and this is the value of interpretation (c).

### 3.3 Prevalence of clock skew in the Web

Is clock skew a real problem? Unfortunately, I know of no systematic study of HTTP client clock skews. This is difficult, in part, because HTTP requests generally do not include a Date header.

However, since I do have access to a trace of the headers flowing through a proxy whose clock, at the time of the trace, was carefully synchronized using NTP, I was able to look at the clock-skew distribution of a large set of HTTP servers. (The trace covers 22034 distinct server IP addresses.) While this is not the same as a population of HTTP clients, one might actually expect a set of HTTP servers to have better clock synchronization characteristics than a set of HTTP clients. After all, many HTTP clients run on personal computers or workstations, and are managed by non-experts; most Web servers on the Internet have at least some semblance of administration (e.g., someone at least had to obtain a DNS name). In other words, whatever the situation with Web server clocks, one would expect the situation among clients to be worse.
For each response in the trace, I compared the Date header field value (if any) to the proxy's NTP-synchronized timestamps for the start of the connection and the end of the connection. If the server's clock is accurate, the Date value ought to be between those two timestamps. If the server's clock is slow, the Date value would be lower than the start-timestamp; if the server's clock is fast, the Date value would be higher than the end-timestamp.

Because of the 1-second granularity of Date, I treated as "valid" any values less than 1 second in error. I also treated as "obviously bogus" any Date where the server's clock appeared to be more than 1 day wrong, since one could assume that such a badly skewed server clock would be abnormal.

The trace contained 503969 responses with parsable response headers. Of these, only 286779 actually had Date headers (most of the rest appear to be PointCast responses). 1087 of these had Date values that were clearly bogus (by the "1-day-wrong" test). Of the others, 116966 (41%) showed a server with a "slow" clock (by at least one second), and 83782 (29%) showed a "fast" clock. Only 84944 (30%) had apparently-synchronized clocks.

What if we set the threshold for an OK clock at +/- 60 seconds (which, by the earlier analysis, is somewhat larger than the Error_C_bound for N = 6 and Max_RTT = 2)? In this case, we still find 79443 (27%) responses indicating "slow" clocks, and 56429 responses (20%) indicating "fast" clocks. In other words, a lot of the clocks are off by a lot of time.

Using the 1-second threshold, the mean error in the slow clocks is 1287 seconds, with a median error of 113 seconds. For the fast clocks, the mean error is 1383 seconds, with a median of 97 seconds.

Using the 60-second threshold, the mean error in the slow clocks is 1884 seconds, with a median error of 198 seconds. For the fast clocks, the mean error is 2039 seconds, with a median of 152 seconds. (We’re removing the small-error samples from these sets, so we’re left with sets biased towards high-error samples.)

In summary, clock skew seems to be prevalent among HTTP servers, and the skews seem to be fairly large. One might be justified in guessing that the situation is worse among HTTP clients.

NOTE: I should reanalyze this data, breaking it down by server address, rather than by response, but that will have to wait for another draft of this document.

3.4 Implications of overestimating the Age value: Interpretation C
What happens if the Age value is overestimated? If this happens, some "fresh" responses appear to be "stale", and so unnecessary cache misses may be generated. Except in the case where the network is
partitioned, this is a performance problem, but does not lead to the
delivery of responses with the wrong body or headers. (When the
network is partitioned, caches are allowed to return stale values
with an appropriate Warning, so this is irrelevant to the current
discussion).

Further, only a very small subset of the fresh, cachable responses
will be affected. In particular, this can only affect a response
whose actual remaining freshness lifetime is smaller than the error
in the Age estimate. Presumably, this is only a significant issue
for responses whose initial max-age value is small (since otherwise
it would be a low-probability coincidence that a request arrives at
the cache just before the freshness lifetime expires). (Remember
that the Error_C_bound = 42 for N = 6 and Max_RTT = 2, and is a lot
lower for shorter proxy paths.)

We have no good information on what kinds of lifetimes HTTP/1.1
origin servers will be assigning to their responses (very few current
servers send Expires), but we can make a guess that of the responses
that are likely to be cachable, most will be assigned fairly long
lifetimes. This is because existing caches already assign, by
heuristic, fairly long lifetimes to responses without Expires or
max-age values, and (for the most part) this kind of caching seems to
be tolerated.

Certainly, one would hope that forthcoming HTTP/1.1 server
implementations will make it easier for service authors to specify
small freshness lifetimes. However, it is unlikely that this could
lead to a great improvement in the number of cache hits. Several
trace-based studies [1, 4] have shown that there is an upper limit to
the performance of straightforward HTTP caches, because for many
resources, no two requests result in the same response body.
Therefore, one should not expect the currently-observed cache hit
rates to get a lot better.

3.5 Implications of underestimating the Age value: Interpretation B

What happens if the Age value is underestimated? If this happens,
some "stale" responses appear to be "fresh", and are returned to the
client without any Warning. The client, in this case, naively
obtains the wrong response value.

This is a far more serious error than causing an extra cache miss.
Delivering the wrong response is not always a significant error, but
in some cases it can lead to serious external consequences.

Moreover, the set of cachable responses vulnerable to this error is
much larger than for an overestimation error. The reason is, as we
saw above, clock-skew errors appear to be quite common, especially
slow-clock errors, and the mean error is measured in tens of minutes.
This is likely to be much closer to the magnitude of freshness
lifetimes than the approximately 1-minute overestimation error.
The adoption of interpretation (b), in an attempt to avoid unnecessary cache misses, could have a perverse effect: by significantly increasing the likelihood of undetected delivery of stale responses when the freshness lifetime is short, this might discourage service authors from allowing such responses to be cached at all. In other words, service authors generating responses with short cachable lifetimes might be driven to declare these responses as totally uncachable, to avoid the possibility of unwarranted caching.

One should also note that a small (but certainly non-empty) fraction of the bogus Date values in our trace are wrong by days, not hours. If such a problem afflicted many clients, this could affect the willingness of service authors to allow caching even of responses with relatively long lifetimes.

3.6 So what’s all the fuss about, anyway?

It might seem that arguing over the issue of caching responses with relatively short freshness lifetimes is a tempest in a teapot. At some level, it is: straightforward HTTP caching will probably never yield the kind of hit rates seen for CPU caches, and this is especially true for short-lifetime values.

However, there are a number of ways in which the effectiveness of an HTTP cache could be improved; for example, prefetching seems to be useful for decreasing latency (although it doesn’t help if the path is already bandwidth-limited). Prefetching might take advantage of the cachability of responses with relatively short lifetimes, but it will not be feasible if service authors cannot trust the prefetching caches to be punctilious about freshness lifetimes.

In other fields of computer science (for example, compilers or multiprocessor caches), aggressive optimization always depends on a reliable understanding of the situation at hand. That is, if one cannot be sure that the transformation done by the optimization preserves the semantics of the system, one cannot safely do that optimization (and, consequently, one may be prevented from doing a lot of other optimizations that are enabled by the first one). Even when optimizations are done "speculatively", this always involves being able to check the results for semantic correctness before committing them.

Attempts to make things faster at the cost of semantic transparency might have some short-term attractions, especially in a setting where we have historically been sloppy about semantic correctness. But this is ultimately a dead-end path; after once giving up on semantic correctness, it’s almost impossible to go back.
4 Acknowledgements

I would like to thank Jim Gettys for comments on this draft.

5 References


6 Author’s address

Jeffrey C. Mogul
Western Research Laboratory
Digital Equipment Corporation
250 University Avenue
Palo Alto, California, 94305, USA
Email: mogul@wrl.dec.com