The Age Penalty and its Effect on Cache Performance

Edith Cohen
AT&T Labs–Research
180 Park Avenue, Florham Park
NJ 07932, USA
edith@research.att.com

Haim Kaplan
School of Computer Science
Tel-Aviv University
Tel-Aviv 69978, Israel
haimk@math.tau.ac.il

Abstract

Web content caching is recognized as an effective mechanism to decrease server load, network traffic, and user-perceived latency. An HTTP compliant cache associates with each cached object an expiration time calculated according to directives set by the object's origin server. The cache incurs a *miss* when it has no cached copy of a requested object or when the existing copy had expired (is not fresh). Upon a miss, the cache needs to fetch or validate a copy through exchanges with another cache with a fresh copy or the origin server. Thus, misses generate traffic and prolong service times.

Caches are deployed as proxies, reverse proxies, and hierarchically and as a result, caches often serve other caches. As this happens, content *age* at higher-level caches, in addition to availability and freshness, emerges as a performance factor. The *age* of a cached copy of an object is the elapsed time since fetched from the respective origin. Fresh cached copies of the same object can have different ages and older copies typically expire sooner. Therefore, a proxy cache would suffer a higher miss rate if it receives older objects (e.g., from a reverse-proxy cache). Similarly, reverse-proxy caches that serve proxy-caches receive more requests than an origin server would have received. We refer to the increase in miss rate due to age as the *age penalty.* We use trace-based simulations to measure the extent of the age penalty for content served by content delivery networks and large caches. Even though the age penalty had not been considered previously, we demonstrate that it can be significant, and moreover, can highly vary under different practices.

1 Introduction

Caching and replication of Web contents are widely deployed mechanisms for reducing Web servers load, network load, and user-perceived latency. The cache effectiveness depends on the lifetime durations of cached copies. Web caching is governed by HTTP, which associates with each cached copy of an object a freshness expiration time after which it is considered stale. When a request arrives and there is no cached copy (*content miss*), the cache attempts to obtain a fresh copy through another cache or the origin server. When a request arrives and there is a stale cached copy of the object, the cache must attempt to validate it (ask for validation or a modified copy) before serving it to the user Validations are performed through a conditional (`If-Modified-Since` or `E-tag` based) `GET` request and involve communication with the origin server or another cache. A large fraction of validation requests return "unmodified" (we term such requests *freshness misses*), but very often the latency they incur is comparable to that of a full-fledged content miss. Thus, both content and freshness misses decrease the cache effectiveness and are important to avoid.

Caches determine an expiration time for a cached copy by computing its *freshness lifetime* and its (approximate) *age*. A copy becomes *stale* when its age exceeds its freshness lifetime. The freshness lifetime is essentially determined by the origin server as it is computed using directives and values found in the object's response headers. If explicit directives are not provided, a heuristic is used. The copy's age is the elapsed time since it was sent by its origin server. When a copy is obtained from another cache it has a positive age and thus, typically a shorter time-to-live (TTL) than would have been if fetched directly from the origin server.

Caches are being deployed throughout the network. Proxy caches are placed in ISPs networks close to clients and reverse proxy caches are placed near network exit points and cache objects for hosted Web sites. Content delivery services (CDNs) such as Akamai and Sandpiper place caching servers in multiple locations [1, 10]. Therefore, it is becoming increasingly more common that (explicitly or transparently) content is served to a cache (such as a proxy cache) from another cache (e.g., a reverse proxy). As this happens, another performance factor emerges, namely, the *age* of copies at higher-level caches. If an older copy of an object is received by the low-level cache, the cached copy expires sooner and thus is less likely to remain fresh till a subsequent request arrives for the same object. Therefore, low-level caches that receive older copies incur more misses. We refer to the relative difference in the number of freshness misses incurred by a cache when it forwards requests to a non-authoritative (high-level cache) vs. authoritative (origin server) source as the *age penalty*.

Cache performance issues related to content age had been by and large overlooked. Our main contribution is introducing the age penalty issue and assessing its magnitude. Aging issues take a different spin for content served by CDNs. CDNs such as Akamai act as reverse proxies, but have tighter relation with their clients (Web sites). In particular, they deploy a consistency mechanism other than HTTP/1.1 between them and their clients. Downstream client caches, however, still use the HTTP response headers to determine the object lifetime. The interaction of two coherence mechanism gives rise to different practices that greatly affect object age, and as a result, the effectiveness of client caches, user-perceived latency, and traffic. We discuss observed CDN practices and measure the associated age penalty.

In Section 2 we overview HTTP freshness control and its usage, and introduce aging issues. Section 3 discusses our traces and experimental methodology. Section 4 contains simulation results measuring the age penalty for objects served from a high-level cache. Section 5 is concerned with the age penalty for objects served by content delivery networks. We conclude in Section 6.

2 Freshness control

Cache freshness control mechanisms are discussed in the specification of HTTP/1.1 and its predecessors. Caches should determine the freshness period conservatively, using parameters in the response headers of the object. If the header does not include specific directives, then it is suggested that caches apply a heuristic, by first matching the URL against some regular expression (e.g., according to suffix which usually indicates the content type), and then determining the freshness lifetime to be some fraction of the elapsed time between the object's date and its last modification time. We provide a quick overview of freshness control at caches. For further details see [5, 2, 11, 8, 7].

The following response headers are used by Squid [11] for freshness control:

- DATE. A time stamp indicating when an object is sent by the **origin server**. All origin servers that have clocks must provide a DATE header. An object received without a DATE header must be assigned one by the recipient if the object will be cached. The DATE header of an object is updated after a 304 (not modified) response to an `If-Modified-Since` GET request. This header is an end-to-end header not supposed to be changed by intermediate caches.
- PRAGMA: NO-CACHE (HTTP/1.0) or CACHE-CONTROL: NO-CACHE (HTTP/1.1).

Explicit directive that the object is not to be cached. Response without this directive is considered cachable.

- Cache-Control: max-age (HTTP/1.1). Explicit freshness lifetime value assignment by the origin server in seconds. This value is typically fixed for copies of the object obtained at different times.
- Expires: (HTTP/1.0). A time stamp beyond which the object stops being fresh. If Max-Age is also present then Max-Age takes priority. In practice, Expires is often either set to be the current time or time in the past (implying freshness lifetime of zero), a far time in the future (years), or is calculated dynamically as Date + T (providing for a freshness lifetime of T) and essentially behaves like a Max-Age directive.
- Last-Modified (HTTP/1.0). The time when the object was last modified by the origin server.
- E-tag (HTTP/1.1). An identifier for the received version of the object. The E-tag is generated at the origin and can be used for validation instead of a Last-Modified time.
- Age (HTTP/1.1). The cumulative time an object spent in caches when received by the current cache. This response header is added and modified by caches and is present if all caches along the response path are HTTP/1.1 compliant.

HTTP/1.1 specification (and Squid) consider every object as cachable unless an explicit no-cache directive is present.[1] The freshness calculation for a cachable object compares the *age* of the object with its *freshness lifetime*. If the age is smaller than the freshness lifetime the object is considered fresh and otherwise it is considered stale. We refer to the difference between the freshness lifetime and the age as the time-to-live (TTL).

Squid calculates the age of an object as the difference between the current time (according to its own clock) and the time specified by the Date header. This method of age calculation is also described in HTTP/1.1 specification. If an Age header is present, the age is taken to be the maximum of the above and what is implied by the Age header.

Squid implements freshness lifetime calculation according to HTTP/1.1 specifications as follows. First, if a Max-Age directive is present, the value is used as the freshness lifetime, and the TTL is the freshness lifetime minus the age (or zero if negative). Otherwise, if Expires header is present, the freshness lifetime is the difference between the times specified by the Expires and Date headers (zero if negative), and thus the TTL is the difference between the time specified by the Expires header and the current time (or zero in case this difference is negative). Otherwise, no explicit freshness lifetime is provided by the origin server and a heuristic is used: The freshness lifetime is assigned to be a fraction (denoted by CONF_PERCENT, HTTP/1.1 mentions 10% as an example) of the time difference (denoted by LM_AGE) between the time specified by the Date header and the time specified by the Last-Modified header, subject to a maximum allowed value (denoted by CONF_MAX and is usually a day, since HTTP/1.1 requires that the cache must attach a warning if heuristic expiration is used and the object's age exceeds a day). Origin servers are supposed to keep their clocks fairly close to real time, and the age calculation assume that this is indeed the case.

2.1 Usage of freshness control

To get an idea for the distribution of freshness control mechanisms used, we took one of the logs we used (6 day NLANR cache trace, see Section 3) and performed separate GET requests to URLs appearing in the log. We then weighted the freshness mechanism by the number of requests to the object recorded in the log. 3.4% of requests were to objects with Max-Age specified. 1.4% were with no Max-Age header and with Expires specified in a relative way (or to a time equal or before the one specified by the Date header), and 0.8% were with Expires specified in an absolute way. The vast majority, 70%, did not have either Max-Age or Expires specified but had a Last-Modified header which

[1] There are few exceptions to this rule such as responses to requests with authorization headers and responses with vary headers that are not yet supported by Squid and are very rarely used.

allowed for a heuristic calculation of freshness lifetime. Other requests either had neither of these 3 header fields, were explicit noncachables (3%), or corresponded to objects with response headers other than 200 (when separately GETted), the most common of which was 302 (HTTP redirect).

Figure 1 plots CDF (Cumulative Distribution Function) of freshness lifetime values weighted by respective number of client requests. The majority of freshness lifetime values are 24 hours (86400 seconds), which is mostly due to the heuristic calculation (using LAST-MODIFIED) with a CONF_MAX setting of 24 hours. About 25% of values are

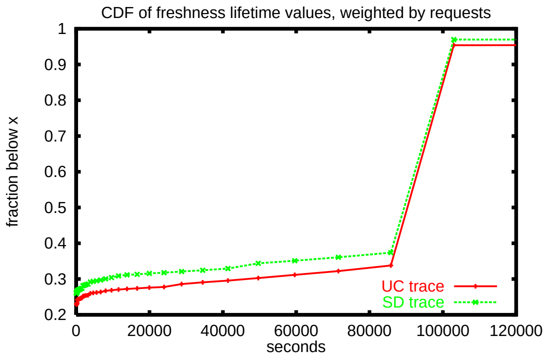


Figure 1: CDF of freshness lifetime values weighted by requests (cachable objects only)

0 (due to MAX-AGE or EXPIRES directive). The linearity of the line between very short freshness lifetime values and a value of 86400 (one day) is due to recently-modified objects and the heuristic expiration calculation (objects modified less than 10 days ago and a CONF_PERCENT value of 10%). These statistics also show that most freshness lifetime values are in effect *fixed*, being the same for copies subsequently obtained from the origin. The freshness lifetime value is fixed when it is computed using MAX-AGE, relative use of EXPIRES, EXPIRES value before the current DATE (freshness lifetime of 0), or with the heuristic when the object is sufficiently old to have freshness lifetime of CONF_MAX (that is, the elapsed time since its creation times CONF_PERCENT is at least CONF_MAX). The two cases without fixed freshness lifetime values are (i) recently-modified objects with no explicit freshness control directive, and (ii) the EXPIRES value is fixed (that is, the server returns the same value for later requests). As we shall see next, these distinctions are relevant for our study.

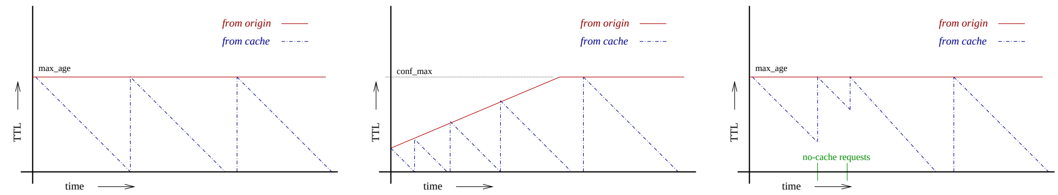
2.2 TTLs via different sources

We consider the TTL of a cached copy of an object at the time it is received by a cache. The age of a received copy, and thus, its TTL, may vary according to the source. A copy fetched through the origin is typically received with zero age whereas a copy obtained from a cache has positive age. [2] The relation between age and TTL depends on the applicable freshness control directive: With a "fixed" freshness lifetime value, (as with MAX-AGE header. dynamically-set EXPIRES header, or heuristic expiration for objects that were not modified very recently), the difference in TTL durations of two fresh copies is equal to the difference in their ages (the older copy becomes stale first). For objects with a "static" EXPIRES header, the TTL is the *same* for copies with different ages. For recently-modified objects with no explicit directives, the difference in TTL values of two copies is in fact *greater* than the difference in their ages.[3]

The following figures illustrate these gaps. Figure 2(A) plots the TTL for an object with MAX-AGE or "relative" EXPIRES freshness control when the object is served from its origin and when it is cached at a top-level cache. The illustrated scenario is such that the cache directly contacts an authoritative source only when its cached copy is stale. Figure 2(B) plots the same for objects with heuristic expiration based on LAST-MODIFIED header. If `no-cache` requests are received by the parent-

[2] Recall that the age of the copy is calculated as the maximum of the age and the difference between the current time and the DATE header value. Hence, if clocks at different hosts are synchronized, copies obtained from a cache should have positive age.

[3] Recall that when EXPIRES or MAX-AGE headers are not present, a heuristic is used to determine the freshness lifetime. The freshness lifetime is determined as a fraction of the difference between DATE and LAST-MODIFIED. In that case the TTL gets penalized twice for the copy's age: first, the freshness lifetime (fraction of the difference between DATE and LAST-MODIFIED subject to a limit) is smaller for older copies, and second, the age of the older copy is larger.



(A) (B) (C)

Figure 2: TTL for an object (i) when fetched from the origin and (ii) when fetched from a cache. (A) the object has Max-Age response header; (B) object has heuristic expiration based on Last-Modified response header, The slope of the line corresponds to CONF_PERCENT value and it levels off at CONF_MAX; (C) object with Max-Age response header when two client requests with a `no-cache` directive where received.

cache, the TTL would look as illustrated in Figure 2(C).

2.3 Age penalty

The age-penalty is measured by a what-if scenario (illustrated in Figure 3). We compare the performance of a (low-level) cache that forwards requests (transparently or explicitly) to high-level caches to the performance if all requests were forwarded to origin servers. We focus here on the freshness hit-rate at the cache. Generally, the age-penalty effect depends on the relation of inter-request times and freshness lifetime duration and kicks in when the low-level cache receives more than one request per freshness lifetime.

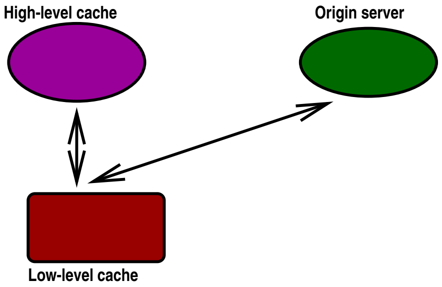


Figure 3: A low-level cache with choice of sources.

3 Experimental methodology

Our data included NLANR cache traces [6]. We used two 6 days traces from NLANR caches collected January 20th till January 25th, 2000 and between July 25 to July 29, 2000 (the second set of logs was used only in measuring CDN performance).

Our experiments required response header values in order to deduce freshness lifetime durations. As these are not typically logged by high-volume caches (including the NLANR traces), we separately performed GET requests shortly after downloading the trace. Timeliness and the distribution of freshness control directives and values (presented in Section 2.1) suggest that the projected values we obtained were fairly close to actual freshness lifetime values that would have been obtained at the time requests were logged.

We run simulation using the original trace and the projected values. The NLANR logs contain various labels that characterize each request by its type and cache performance. (Classification of hits and misses, presence of `no-cache` request header, whether the cached copy was stale or fresh, etc.). In the simulation we accounted for clients requests with `no-cache` header (forcing the cache to forward the request to the server even if the cache has a fresh copy of the object). We used a heuristic to determine on which requests content had changed (treating requests on which the content deemed fresh by the NLANR cache as "no change."). Our heuristic used the labels associated with the request and the logged size of the response to the client. In the simulation we applied the Squid object freshness model (HTTP/1.1 compliant), using a CONF_PERCENT value of 10%, a CONF_MAX value of a day, and a CONF_MIN value of 0 for all URLs. We used the projected freshness lifetime durations. To simulate requests served by origin servers, we set the TTLs to be the respective projected lifetime durations. In various experiments we used TTL values lower than freshness lifetime to simulate access through a cache.

Since we could not GET all URLs in a reasonable time without adversely affecting our environment, we selected a subset. We used all URLs that were requested more than 12 times, and applied some weighted random sampling to others. In total we fetched about 224K distinct URLs (the original logs had millions, most of them requested only once). We factored the sampling back in by scaling the results and grouping URLs by number of requests.

Our simulations assumed infinite cache storage capacity, which is consistent with current trends and with the desired performance on the actual traces we used.

We used the *freshness rate* as our performance metric. We define the freshness rate as the ratio of freshness hits to content hits. We define *content hits* and *freshness hits* as follows. A request for a fresh cached object is considered a content hit and a freshness hit. A request for a stale cached object is considered a content hit only if an authority with a fresh copy (e.g, the server or another cache) certifies that it is not modified. We refer to requests that constitute content hits but not freshness hits as *freshness misses*. Content hit requests exclude requests to explicit non-cachable objects, requests on which the client specified to bypass caches (`no-cache` request header), and requests when the content had actually changed. We also excluded the first request for each object. Content hits include requests for objects with respective freshness lifetime value of 0.

We note that the freshness rate captures only one performance aspect of a cache and we later discuss how to interpret it and combine it with others.

The Squid logs recorded the response code returned to the client and the cache action. We only considered GET requests such that the cache returned a "200" or "304" response code to its client. We further classified these requests as follows, using the listed Squid labels.

- freshness hit (*fhits*): TCP_HIT, TCP_MEM_HIT, TCP_IMS_HIT
- freshness miss (*fmiss*): TCP_REFRESH_HIT
- content miss (*cmiss*): we separately accounted for
  - *cmiss-r* (the cache had a stale cached copy, issued an IMS request, and got a Modified response): TCP_REFRESH_MISS
  - *cmiss-d* (there was no cached copy): TCP_MISS
- `no-cache` request header: TCP_CLIENT_REFRESH_MISS

The following table shows the fraction of requests of each type.

log	fhit	fmiss	cmiss-d	cmiss-r	`no-cache`
UC	23%	10%	56%	1%	10%
SD	19%	15%	56%	3%	7%

Requests classified as fmisses, cmisses, or `no-cache` involve communication with the origin server or another cache. Freshness misses constitute 13% (UC) and 19% (SD) of all requests directed outside. The caches directed most validation requests (fmisses and cmisses-r) outside the NLANR hierarchy (100% in the UC cache and 99.3% in the SD cache). All these requests were directed to an origin server or a transparent reverse proxy. Our measurements suggested that the vast majority of requests directed outside reached the origin server. [4] The great exception was

[4]We distinguish a transparent reverse proxy cache from an origin server by issuing two consecutive GET requests within more that a few seconds apart to the same IP-address of the same server. A cache return a DATE header as obtained from the authoritative server. An authoritative server returns the current time according to its clock. Thus, we examined the DATE header value of the two HTTP responses. The same value indicated a cache. Two different values, where the difference approximates the time between our two HTTP GET request-response pairs, suggest origin server, very short TTL (few seconds or less), or a non-cachable object (that is, no age-penalty effect).

requests directed to CDN servers (this is discussed further in Section 5). Thus, by and large, the original trace was not subjected to age penalty. If more validation requests are directed to a cache we expect to see some of the freshness hits transform to freshness misses. It is also apparent that the vast majority (90% for UC and 95% for SD) of validation requests return Not-Modified.

4 High-level cache simulation

The following experiment attempts to measure the age penalty when content is fetched from nonauthoritative servers such as reverse proxy caches. As discussed above, a given request can constitute a hit or a miss at the higher-level cache. For request constituting cache misses, performance would have been better had the higher-level cache not been there, as longer latency and more traffic is incurred than through direct communication between our cache and the origin server. The higher-level cache is also not as effective on request constituting content misses. Thus, in order to distill the performance effects of age, we simulate an optimistic scenario where all requests constitute cache hits (the higher-level cache always keeps a fresh copy). Note that overall performance would be worse when this is not the case.

Our simulation corresponds to a situation where all requests to a given URL are forwarded from our cache consistently through the same top-level cache (e.g., a reverse proxy) [5]. These top-level caches maintain continuous freshness by refreshing copies through the respective origin servers as soon as it becomes stale. Under these assumptions, for objects with fixed freshness lifetime durations, the TTL of the copy at the top-level cache cycles from the lifetime duration to 0. We denote the freshness lifetime value by T and compute the TTL obtained after each (content or freshness) miss as

$$\mathrm{TTL} = T - (\mathit{time} - \mathit{log\_start\_time}) \bmod T \ .$$

For requests labeled as CLIENT_REFRESH in the trace (arriving with `no-cache` request header), we set the TTL to T in order to simulate a situation where misses are forwarded to the origin server.

[5]Interestingly, analysis shows that otherwise the age penalty is higher [3]

NLANR log & requests fraction	object source	freshness rate $fhits/chits$
UC 1	origin	52%
UC 1	cache	43%
UC 0.1	origin	35%
UC 0.1	cache	28%
SD 1	origin	47%
SD 1	cache	38%
SD 0.1	origin	30%
SD 0.1	cache	24%

Table 1: Freshness rates when requests are directed to (i) a cache or (ii) origin servers.

The respective freshness rates in the two scenarios are listed in Table 1, and show that the overall age penalty of going through higher-level caches amounts to 20%-25% decrease in freshness hits. The logs UC0.1 and SD0.1 are reduced traces that included a random sample of 10% of the requests.

5 Content Delivery Networks

Many sites now use content delivery networks (CDNs) to distribute some of their content [1, 10]. CDNs place multiple servers distributed inside different ISP's networks. Each of these servers is essentially a cache. Like reverse proxy caches, these servers only process URLs within the CDN domain, but like proxy caches, they are located close to clients. Since clients can reach a CDN server through fewer router hops and peering points, the response time, packet loss, and the number of data retransmissions typically improves. CDNs also off-load origin sites.

The current architecture used by popular CDNs (such as Akamai [1]) involves URL substitutions. The origin sites substitute the original object URL to one within the CDN domain. For example, the origin site substitutes the URL of the embedded image `http://cnn.com/images/icons/video.gif` with the URL `http://a388.g.akamaitech.net/7/388/21/e0a3b4215f9e4b/cnn.com/images/icons/video.gif`. The substituted URL is termed by Akamai the *Akamaized URL* or *ARL*.

Clients are then directed to a "good" CDN server (one that is likely to be less loaded, have a cached copy of the object, and be close to the client). When the client local DNS server resolves the hostname of the object, the Akamai DNS server returns an IP-address of a "good" CDN server. CDNs mainly serve relatively static content such as images and java applets but are striving to serve additional content types including streaming media, authenticated content and dynamic content.

The URL substitution architecture forces the deployment of some coherence protocol between the CDN and the origin sites. One natural possibility is to adhere to HTTP/1.1 freshness control - where copies on the CDN servers must be refreshed when they expire. This approach would make the aging issues similar to plain reverse proxy caches. We shall see, however, strong indication that a different mechanism is used.

Experimentation shows that the content of the response is not sensitive to the particular Akamai hostname used (e.g., when `a388.g.akamaitech.net` is substituted with `a534.g.akamaitech.net`). Furthermore, the response content is not sensitive to changes in the 3 fields of the ARL preceding the part that contains the origin URL (e.g., `/388/21/e0a3b4215f9e4b/` in the example above). Values in the last of these field suggest that it encodes a time duration, counter, or an object identifier. It seems that the redundancy in URL to ARL translation is used to encode freshness lifetime and that the Akamai servers use the information embedded in the requested ARL to determine when to refresh (through the original URL). In some cases (as in the example above) the ARL seems to be unique per version of the URL, as it encodes the version identifier (`e0a3b4215f9e4b`). Measuring latency when requesting different ARLs [6] strongly suggest that CDN servers refresh a copy when a request is received with a new "Akamization" of the same URL.[7] It is also evident that Akamai servers do not conduct any checks to determine if a particular "Akamization" was actually used by the origin site, since they responded to requests with arbitrary values (even in the URL portion of the ARL, e.g., in the ARL above, the URL portion `cnn.com/images/icons/video.gif` can be substituted with an arbitrary URL). With CDN servers not performing checks, freshness control can be performed either by (i) CDN servers considering a copy as expired after a pre-agreed duration had elapsed, or as the content of a particular embedded URL is modified at its origin site, the origin site has to (ii) re-substitute a new ARL for it or (iii) to trigger removal of all cached copies at all CDN servers. The most plausible possibility seems to be the first, that is, the use of pre-agreed durations, most likely encoded within the ARL itself.

We discuss how age affects performance under past and present CDN practices.

5.1 Leave response headers intact

The first practice we discuss was implemented by the two studied CDN sites until about March of 2000. The CDN servers, like plain HTTP caches should, populated the end-to-end header values of their responses with the original settings provided for the object when it was fetched from the origin site. As a result, when an object is fetched to a cache from a CDN server, its age includes the

[6] We conducted the following measurements with several URLs: We compared the latency of 500 GET requests of the URL from the same CDN server when (i) the same ARL was used each time (ii) a different ARL was used each time (e.g., by replacing the string `e0a3b4215f9e4b`). The cumulative latency in the latter measurement was 25%-40% slower than in the first. This suggests that when seeing a new Akamization, the CDN server obtains a new copy of the URL from the origin server prior to sending the response.

[7] As of 11/2000, Akamai seem to use HTTP redirect (302 response code) to the original URL in some cases when a new Akamization is seen.

duration it resided on the CDN server. More specifically, when a CDN server responded to an HTTP request it returned the DATE, EXPIRES, MAX-AGE, and LAST-MODIFIED response headers as they were when the object was fetched to the CDN server from the true origin site. The CDN servers, however, differ from HTTP caches since they are considered by caches to be the authoritative sources for the substituted URLs (ARLs). Hence, their responses are always considered valid for the present request (even if already expired) and they constitute a final destination of requests with `no-cache` request headers.

To experimentally measure the age penalty effect for CDN-delivered objects we extracted from the (January, 2000) NLANR logs requests that used two CDNs Akamai [1] and Digital Island (Sandpiper) [10]. To that end we extracted URLs whose hostname included the strings "akamai," "sandpiper", or had the prefix "fp.cache." We note that this is only a subset of requests served by these providers because sometimes the domain name does not include these strings. We used the same simulation methodology as outlined in Section 3. For these URLs we calculated TTLs in two ways: 1) According to the headers provided by the content provider, 2) As they would have been calculated if a current copy was obtained from the origin (with the DATE set to current time and adjusting for dynamically-generated EXPIRES headers.) Figure 4 shows the fraction of URLs with TTL value below x for varying x. When fetched directly from the origin, most objects have TTL value of a day (this is consistent with what we get when considering all objects in the log). When fetched from the CDN server, TTL values drop drastically, and most of them become zero.

The table below includes for each provider and cache, the respective fraction of requests in the log, the freshness rate as is (fetched from the CDN behaving as an HTTP cache with respect to the headers), and the freshness rate as would-have-been if the origin server was serving the object.

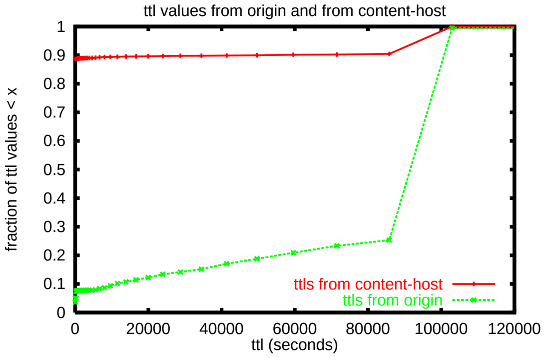


Figure 4: CDF of TTL values when content is fetched from origin vs. content-host

CDN	*cache*	*% of log requests*	*fhits/chits* thru: *CDN*	*origin*
Sandpiper	UC	0.4%	5%	76%
Sandpiper	SD	0.5%	6%	67%
Akamai	UC	1.7%	5%	61%
Akamai	SD	1.1%	6%	63%

These results show that requests to objects served through CDNs incur a ten to fifteen fold decrease in freshness hits and 2-3 fold increase in freshness misses relative to the same requests directed to respective origins.

Freshness misses constitute 26%-38% (UC and SD Sandpiper) and 26%-40% (SD and UC Akamai) of all requests forwarded by the UC cache to the CDN, which is considerable more than general statistics across all logged requests. Our simulation suggests that most of these requests are due to the age penalty effect.

What caused the age-penalty through CDN servers to be so much worse than through HTTP-compliant caches (see Section 4) was deployment of a freshness control mechanism other than HTTP/1.1 to decide when to refresh their copies of hosted objects. The freshness lifetime value used by the CDN server was typically considerably longer than the HTTP-implied value, and thus, the typical hosted object HTTP-expired well before it was refreshed by the CDN server. For example, often the content-host-served objects have an age exceeding MAX-AGE value, resulting in a TTL of 0 and them becoming stale almost immediately at an HTTP/1.1 compliant cache.

Around March, 2000, one of the CDNs, Akamai, addressed the issue by rewriting some of the end-to-end response headers of hosted objects. In particular, the Date header value was set to the current time at the Akamai server, and the Expires is made consistent with the Max-Age directive. The returned headers are then such that an HTTP/1.1 compliant cache assigns the same TTL as it would have if it had contacted the origin server[8]. This rewriting of response headers eliminated much of the unnecessary freshness misses. However, the interaction of the two freshness control protocols gives rise to further questions.

5.2 ARL freshness control

It is evident from the previous measurements that the freshness control mechanism between the CDN servers and the sites allows considerably longer freshness lifetime durations than HTTP freshness control. Thus, the freshness rate of client caches can be significantly improved if the freshness lifetime known to the CDN servers is translated into the HTTP response headers. In particular, when the "Akamization" is unique per version, the Expires header can be set accordingly to a far time in the future, and as a result, greatly reduce the number of freshness misses at downstream caches and network load due to validations.

To asses the potential gain we used a 5 day long trace from the UC NLANR cache taken between 25 and 29 of July, 2000. Note that this trace was downloaded after Akamai implemented the change of rewriting response header values. The trace included for each request the response code returned from the UC cache to the client, the UC cache action taken to process it, and service times (the total processing time from the UC cache perspective).

In total there were 19K requests to 6K different ARLs in the akamaitech.net domain for which the UC cache returned a 200 (OK) or 304 (Not-Modified) response code to the client. 304 responses are freshness misses at the client cache and 200 responses are content misses at the client cache. The breakdown of requests is given in Table 2. In each category, we further breakdown requests according to actions taken at the UC cache. In particular, we list the percentage of freshness hits at the UC cache (LOCAL), misses forwarded to a sibling cache (SIBLING), and misses forwarded to the Akamai server (DIRECT)[9]. All request directed to a SIBLING were cache content misses (TCP_MISS in Squid terminology). For DIRECT requests we accounted separately for those that arrived from the client with a no-cache request header (TCP_CLIENT_REFRESH_MISS), those that were freshness misses at the UC cache (TCP_REFRESH_HIT), and other cache misses. We calculated average service time for requests in each category excluding values exceeding 10 seconds, since otherwise the average is distorted by few outliers. Service times of less than 10 seconds included all local responses and about 99.5% of all responses. It is evident that average service times highly varies by request category. LOCAL requests are handled significantly (70%) faster than requests on which the cache contacted another server. The response time on 304 responses is about 40% shorter than on 200 responses.

We are now able to asses the potential reduction in freshness misses, both at the UC cache and at caches of its clients. 10.4K out of the 19.4K (over 50%) of the requests directed to the UC cache from client caches were validation requests with 304 response. All of these requests except for those with a no-cache request header could have been eliminated. Thus, 45% of total requests between client caches and the UC caches would have been eliminated. Considering average service times shows that the average latency gain would have amounted to about 150ms (plus RTT) per request. Requests from the UC cache to Akamai servers that potentially could have been eliminated are all validation requests without a no-cache request header. This included about 19% (11% for 200 client

[8]This is true in all cases except when the Expires header is used in a relative way and there is no Max-Age directive (see Section 2), and when clocks on the origin site and the Akamai server are not synchronized

[9]no requests were forwarded to a parent cache

at client cache	*at UC cache*				
content miss (200)	*LOCAL*	*DIRECT*			*SIBLING*
9K 208ms	35% 80ms	59% 269ms			6% 360ms
		no-cache	304	*miss*	
		20% 1.1K	12% 0.62K	68% 3.6K	
freshness miss (304)	*LOCAL*	*DIRECT*			*SIBLING*
10.4K 157ms	28% 51ms	67% 196ms			5% 201ms
		no-cache	304	*miss*	
		18% 1.3K	29% 2K	53% 3.5K	

Table 2: Breakdown of requests according to cache action.

responses and 29% for 304 responses) of total requests issued to an outside server (sibling cache or origin).

On a side note, it is evident that the UC cache evicts items more aggressively than its clients caches: the majority of requests where 304 response was sent to the client were content misses at the UC cache and on the other hand 60% of the freshness misses at the UC cache were freshness misses (and content hits) at the client. This suggests that UC cache performance on these items (and the possible gain from ARL uniqueness) would benefit from increased storage or better replacement policy.

In an attempt to try to consider only ARLs that are used in a unique fashion per URL version we considered only requests to ARLs where the value in the last field preceding the URL seemed to contain a time-stamp, counter, or version identifier. We identified 5.8K such requests which included 2.9K with 304 response and 2.9K with 200 response. Out of the 304 responses there were 0.8K with no-cache headers. Thus, about 35% of total requests from clients to the UC cache would have been eliminated. Out of the 5.8K requests, about 60% were DIRECT and about 13% were DIRECT freshness hits. Thus, about 20% of requests to the Akamai server could have been eliminated.

The above discussion asses the potential gain from translating the proprietary CDN freshness control into the HTTP headers, but leaves aside possible reasons for keeping two sets of directives. For example, hit-count and collecting statistics, but these can usually be performed through the referring HTML page or using a single embedded object on each page.

6 Conclusion

We discussed the effects of object-age on the performance of cascaded caches, and showed, through trace-based simulations, that the age penalty can be significant. We remark that the age penalty can be eliminated altogether if strong consistency is maintained between high-level caches (e.g., reverse-proxies or CDN servers) and origin servers. Strong consistency, however, is expensive and not facilitated through HTTP or otherwise widely supported.

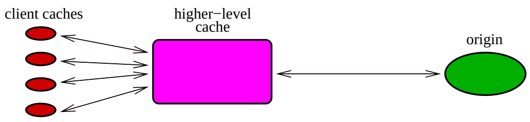


Figure 5: Client caches, high-level cache, and the origin

For future work, we propose two approaches to alleviate the age penalty while working within the current consistency infrastructure. The first, *source selection*, is deployed by "low-level" caches. In some architectures a cache may have a choice of where to forward requests on which a miss occurred. If so, it is preferable to forward requests to a server that is more likely to have the "youngest" fresh copy of the object. More generally, source selection should balance distance, likelihood of fresh cached copy, and age. The second approach, *rejuvenation*, is deployed by "high-level" caches. We use the term *rejuvenation* for pre-term validation of selected copies (well before they expire), as a mean to decrease age. At the limit, frequent rejuvenation amount to strong consistency between the cache and the origin and thus, no age penalty. Rejuvenations reduce traffic between the cache and its clients (and user-perceived latency), but increase traffic between the cache and the origin servers (see Figure 5). When the cache serves many clients which request an object frequently, the benefit of decreased age penalty could significantly outweigh the cost. We analyse and experimentally-evaluate rejuvenation in subsequent work [4, 3].

The age penalty can widely vary for content served by CDNs. The practice of intact end-to-end HTTP response headers resulted in an order of magnitude decrease in freshness hits and in a 2-3 fold increase in validation traffic. With edited response headers, it seems that performance could greatly improve if ARL freshness control is translated to HTTP freshness control in the rewritten HTTP response headers. Generally, it is desirable that CDN clients would need to use only one set of freshness control specifications, either through HTTP headers, or through the CDN proprietary mechanism. CDNs can then either adhere to HTTP directives or re-write them to be consistent with the proprietary protocol. A related discussion on defining the role of CDNs, possibly by incorporating specific CDN directives into HTTP headers, is given in [9].

We conclude with the hope that our work would contribute to better understanding, by researchers and practitioners, of the age-related facet of cache performance, and ultimately, spur further work aimed at improving performance of cascaded caches.

Acknowledgment

We thank Duane Wessels for answering questions on a (since corrected) Squid logging bug, and Bruce Maggs and Mark Nottingham for answering questions on observed Akamai practices. We also thank Jeff Mogul for comments on an earlier version of this paper.

References

[1] Akamai. `http://www.akamai.com`.

[2] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol — HTTP/1.0. RFC 1945, MIT/LCS, May 1996.

[3] E. Cohen, E. Halperin, and H. Kaplan. Performance aspects of distributed caches using TTL-based consistency. Manuscript, 2000.

[4] E. Cohen and H. Kaplan. Aging through cascaded caches: performance issues in the distribution of web content. Manuscript, 2000.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, and T. Leach, P. Berners-Lee. Hypertext Transfer Protocol — HTTP/1.1. RFC 2616, ISI, June 1999.

[6] A Distributed Testbed for National Information Provisioning. `http://www.ircache.net`.

[7] J. C. Mogul. Errors in timestamp-based HTTP header values. Technical Report 99/3, Compaq Western Research Lab, December 1999.

[8] M. Nottingham. Optimizing object freshness controls in Web caches. In *The 4th International Web Caching Workshop*, 1999.

[9] M. Nottingham. On defining a role for demand-driven surrogate origin servers. In *The 5th International Web Caching and Content Delivery Workshop*, 2000.

[10] Digital Island (Sandpiper). `http://www.sandpiper.com`.

[11] Squid internet object cache. `http://squid.nlanr.net/Squid`.