World Wide Web Cache Consistency

James Gwertzman and Margo Seltzer
Harvard University

World-Wide Web Cache Consistency

James Gwertzman, *Microsoft Corporation*
Margo Seltzer, *Harvard University*

Abstract

The bandwidth demands of the World Wide Web continue to grow at a hyper-exponential rate. Given this rocketing growth, caching of web objects as a means to reduce network bandwidth consumption is likely to be a necessity in the very near future. Unfortunately, many Web caches do not satisfactorily maintain cache consistency. This paper presents a survey of contemporary cache consistency mechanisms in use on the Internet today and examines recent research in Web cache consistency. Using trace-driven simulation, we show that a weak cache consistency protocol (the one used in the Alex ftp cache) reduces network bandwidth consumption and server load more than either time-to-live fields or an invalidation protocol and can be tuned to return stale data less than 5% of the time.

1.0 Introduction

Network traffic continues to grow at a hyper-exponential rate while network infrastructure does not. This means that existing networks are plagued by ever increasing utilization demands. One approach to coping with the increasing resource utilization is to cache data at non-server sites. As service providers such as America Online introduce millions of subscribers to an already overburdened networking infrastructure, it is nearly assured that systems must cache Web objects to facilitate acceptable service. Caching can be quite effective at reducing network bandwidth consumption as well as server load. Netscape, a vendor of Web servers, claimed in March of 1995 that a single local proxy server can reduce internetwork demands by up to 65% [1].

The value of caching is greatly reduced, however, if cached copies are not updated when the original data change. Cache consistency mechanisms ensure that cached copies of data are eventually updated to reflect changes to the original data. There are several cache consistency mechanisms currently in use on the Internet: time-to-live fields, client polling, and invalidation protocols.

Time-to-live fields are an *a priori* estimate of an object's life time that are used to determine how long cached data remain valid. Each object is assigned a time to live (TTL), such as two days or twelve hours. When the TTL elapses, the data is considered invalid; the next request for the object will cause the object to be requested from its original source. TTLs are very simple to implement in HTTP using the optional "expires" header field specified by the protocol standard [2]. The challenge in supporting TTLs lies in selecting the appropriate time out value. Frequently, the TTL is set to a relatively short interval, so that data may be reloaded unnecessarily, but stale data are rarely returned. TTL fields are most useful for information with a known lifetime, such as online newspapers that change daily.

Client polling is a technique where clients periodically check back with the server to determine if cached objects are still valid. The specific variant of client polling in which we are interested originated with the Alex FTP cache [6] and is based on the assumptions that young files are modified more frequently than old files and that the older a file is the less likely it is to be modified. Adopting these assumptions implies that clients need to poll less frequently for older objects. The particular protocol adopted by the Alex system uses an *update threshold* to determine how frequently to poll the server. The update threshold is expressed as a percentage of the object's age. An object is invalidated when the time since last validation exceeds the update threshold

---

This research was supported by the National Science Foundation on grant CCR-9502156.

times the object's age. For example, consider a cached file whose age is one month (30 days) and whose validity was checked yesterday (one day ago). If the update threshold is set to 10%, then the object should be marked invalid after three days (10% * 30 days). Since the object was checked yesterday, requests that occur during the next two days will be satisfied locally, and there will be no communication with the server. After the two days have elapsed, the file will be marked invalid, and the next request for the file will cause the cache to retrieve a new copy of the file.

There are two important points to note with respect to client polling: it is possible that the cache will return stale data (if the data change during the time when the cached copy is considered valid) and it is possible that the cache will invalidate data that are still valid. The latter is a performance issue, but the former means that, like TTL fields, client polling does not support perfect consistency.

Like TTL, client polling can be implemented easily in HTTP today. The "if-modified-since" request header field indicates that the server should only return the requested document if the document has changed since the specified date. Most web proxies today are already using this field.

Invalidation protocols are required when weak consistency is not sufficient; many distributed file systems rely on invalidation protocols to ensure that cached copies never become stale. Invalidation protocols depend on the server keeping track of cached data; each time an item changes the server notifies caches that their copies are no longer valid. One problem with invalidation protocols is that they are often expensive. Servers must keep track of where their objects are currently cached, introducing scalability problems or necessitating hierarchical caching. Invalidation protocols must also deal with unavailable clients as a special case. If a machine with data cached cannot be notified, the server must continue trying to reach it, since the cache will not know to invalidate the object unless it is notified by the server. Finally, invalidation protocols require modifications to the server while the other protocols can all be implemented at the level of the web-proxy.

In this paper, we examine the different approaches to cache consistency. An ideal cache consistency solution will provide a reduction in network bandwidth and server load at very low cost. In the next section, we discuss cache consistency protocols in general and cache consistency as applied to the Web in particular. Section 3 presents our simulation environment and Section 4 our simulation results. In Section 5, we suggest some areas for future research and conclude in Section 6, with the suggestion that weakly consistent protocols are a good choice for web consistency.

2.0 Related Work

Danzig et al. motivate the need for hierarchical object caches for Web objects on the Internet by examining how strategically located FTP caches affect Internet traffic [9]. They found that FTP traffic across the backbone could be reduced by as much as 42%, simply by caching FTP files at the juncture between the backbone and regional nets. This result inspired the design of the Harvest object cache, which is a hierarchical proxy-cache [7].

Once the need for caching has been established, it is instructive to consider how to maintain consistency among the caches. While there are a number of approaches for maintaining cache consistency in distributed file systems, there has been little work aimed specifically at evaluating cache consistency protocols on the World Wide Web. Blaze explored constructing large-scale hierarchical file systems [5]. While his architecture is similar to the one we posit for the web [10], the systems are sufficiently different that his results cannot be directly applied. In his model clients can also act as servers and can cache files on a long term basis. This is not necessarily true in the web where clients are often personal computers with limited resources.

The Berkeley xFS system [8] suggests a model of cooperative caching that is also similar to the one we propose for the web [10]. However, it relies on clients, not only for long-term caching, but also to retain the master copy of data. Like other distributed file systems (e.g. the Sprite Distributed File System [13], the Andrew File System [11]), it also assumes objects can be changed by any machine while web objects can be modified only on their primary server.

The web is fundamentally different from a distributed file system in its access patterns. The web is currently orders of magnitude larger than any distributed file system. Each item on the web has a single master site from which changes can be made. This suggests that consistency issues may be simpler because conflicting updates should never arise.

The most widely used web cache is the original server distributed by CERN [12]. The CERN server assigns cached objects times to live based on (in order), the "expires" header field, a configurable fraction of the "Last-Modified" header field, and a configurable default expiration time. Cached objects are returned, without further consultation with the server, until they expire, at which point subsequent requests cause an "If-Modified-Since" request to be issued.

One study compares the performance of the CERN proxy cache to a specially designed lightweight caching server [15]. The lightweight cache has an independent process that periodically examines cached objects to determine if they have become stale. Staleness is determined using both TTLs and invalidation callbacks from cooperating primary servers. Proxy caches are registered with the primary server so that they can receive invalidation notices. If one views the CERN proxy cache as implementing an NFS-like consistency protocol [14], the new server implements an AFS-like protocol. The comparison focuses on the performance differences between the two servers and does not examine the relative behavior of the different consistency protocols, which is the focus of this work.

To date, the only other detailed examination of consistency protocols is a study by Worrell that compared TTL fields to invalidation protocols [16]. He showed that the bandwidth savings for invalidation protocols and TTL fields could be comparable if the TTL were set to approximately seven days. Unfortunately, with a TTL of 7 days, 20% of the requests returned stale data. We believed that a simple, but adaptive scheme, such as the Alex protocol, might achieve comparable bandwidth savings with substantially better stale hit rates, so we obtained the same simulator used in Worrell's study and adapted it for a more extensive evaluation. In the process of exploring the Alex protocol, we discovered that the original workload in the Worrell study was inconsistent with the workload we observed in server traces. We hypothesized that, by using a more trace-based workload, the simulation results would change significantly.

The original simulation environment consisted of a cache simulator and a collection of file ages gathered over several months for 4,000 files located around the Web. The simulator modeled a hierarchical caching system and provided both a TTL cache consistency protocol and an invalidation protocol. The invalidation protocol was optimized so that upon receipt of an invalidation message, objects were simply marked invalid, but not immediately retrieved. This increased latency on subsequent accesses, but decreased bandwidth consumption if the object was not accessed again. Finally, the simulator used the average and variance of the file ages to generate a uniform, random stream of file accesses.

3.0 Simulation Environment

We began with Worrell's simulator and modified it in a number of ways. We made two initial modifications to begin the experiments. First, we added the Alex protocol. Then, in order to isolate the effects of cache consistency policy from the effects of hierarchical caching, we flattened the cache hierarchy to model a single cache.

Worrell's simulation analyzed the Harvest cache's hierarchical caching. We wished to separate the issues of hierarchical caching and cache consistency, focusing only on the latter. While eliminating the hierarchy changes the amount of invalidation traffic in the study, in most cases it does not affect the relative traffic of the different invalidation schemes. When it does affect the relative traffic, it does so in a manner that favors invalidation protocols.

Figure 1 shows the cases in which our results may be distorted by collapsing the hierarchy. In all cases where the relative performance of invalidation and time-based protocols is different in the hierarchical and collapsed systems, our simulation favors the invalidation protocols, while our results suggest that time-based protocols are more desirable. Therefore, we expect that time-based protocols in a cache hierarchy will perform even better than our results indicate.

Since we flatten the hierarchy, Worrell's "goodness" metric of *network hops * number of bytes transferred* is no longer a useful measure. We use the number of bytes required to maintain consistency, including invalidation messages, stale data checks, and file data movement. For the remainder of this paper, we will refer to Worrell's modified simulator with Alex and a flattened hierarchy as the *base simulator.*

The base simulator produced results very similar to those reported by Worrell. Our next step was to optimize the Alex and TTL protocols in a manner similar to the invalidation protocol optimization. When a cached datum expired, instead of immediately requesting a new copy, the items were marked invalid. Upon next reference, we issued an "If-Modified-Since" request to the server. The item was only retransmitted if it had, in fact, changed since the last time it was sent. In this manner, we traded the latency of the query request for the bandwidth savings, i.e. not having to retransmit data when a valid copy existed in the cache. By combining the query with the retransmit request to yield a "send this file if it has changed since a specific date" request, we avoided extra overhead and still saved bandwidth where

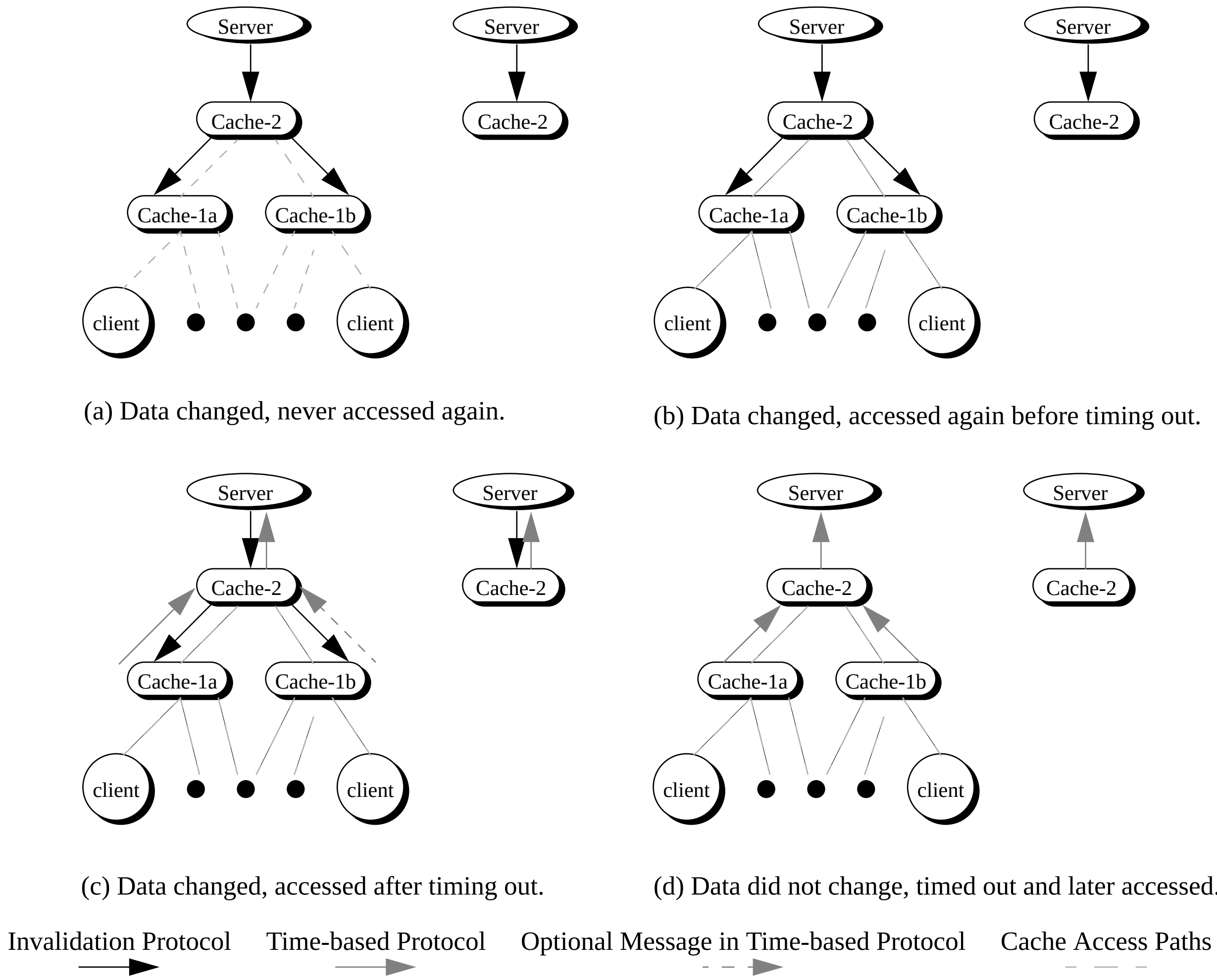


Figure 1. Comparison of Hierarchical Caching and Collapsed Caching. In each diagram, the left picture shows a hierarchical cache and the right picture shows the collapsed hierarchy. The arrows indicate messages sent by the different protocols. In figures a and b, there is no traffic for time-based protocols because the data's time-out has not expired. Therefore, in both simulations, the time-based protocol uses 0% of the bandwidth of the invalidation protocol. In figure c, both time-based and invalidation messages (and files) are sent. If the item is requested from all caches, then the bandwidths of the invalidation protocol and the time-based protocol are equal to each other and the time-based protocol requires 100% of the bandwidth of the invalidation protocol. If some of the caches do not later access the data (e.g. cache-1b), then the time-based protocol will require less bandwidth in the hierarchical model, but still 100% of the bandwidth in the collapsed model. Therefore, when this occurs, we bias the results against the time-based protocols. Figure d shows a similar effect. There is no invalidation traffic, but time-based messages are issued. In the hierarchical case, messages will only be issued from those caches requiring the item, while in the collapsed case, "all" clients request the item. Again, this biases the results *against* the time-based protocols.

possible. We call the simulator with this modification the *optimized simulator*.

Our last change addressed the workload issue mentioned in Section 2. Worrell modeled the file lifetime distribution as a flat distribution between the minimum and maximum observed lifetimes. This means that files were modified with no attention to their type or past modification history. The results of trace analysis from a modified campus Web server show that this is an inappropriate model. Files tend to exhibit bimodal lifetimes. Either a file will remain unmodified for a long period of time or it will be modified frequently within a short time period [10]. (It was this observation that led us to believe that the Alex protocol would be well suited to Web cache consistency.) Additionally, Worrell used a uniform distribution to generate file requests, but Bestavros has shown that the more popular a file is, the less frequently the file changes [4]. We modified the simulator to use a trace-driven workload. This

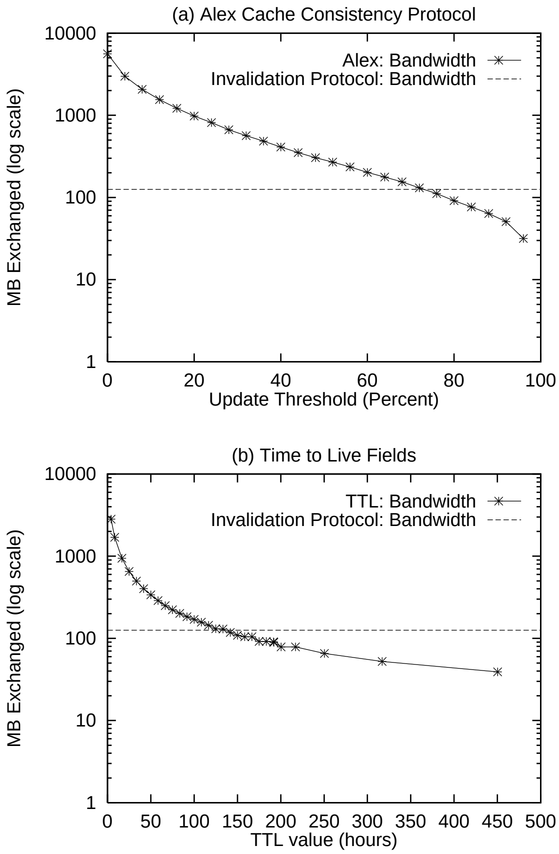


Figure 2. Comparison of bandwidth usage in the base simulator. The cache is pre-loaded with valid copies of all the files held in the primary server. Note the use of a log-scale to display the bandwidth with higher accuracy. The invalidation protocol is superior to both TTL and Alex until the update threshold or TTL is quite large. This result is similar to Worrell's result for TTL protocols and indicates that Alex behaves comparably.

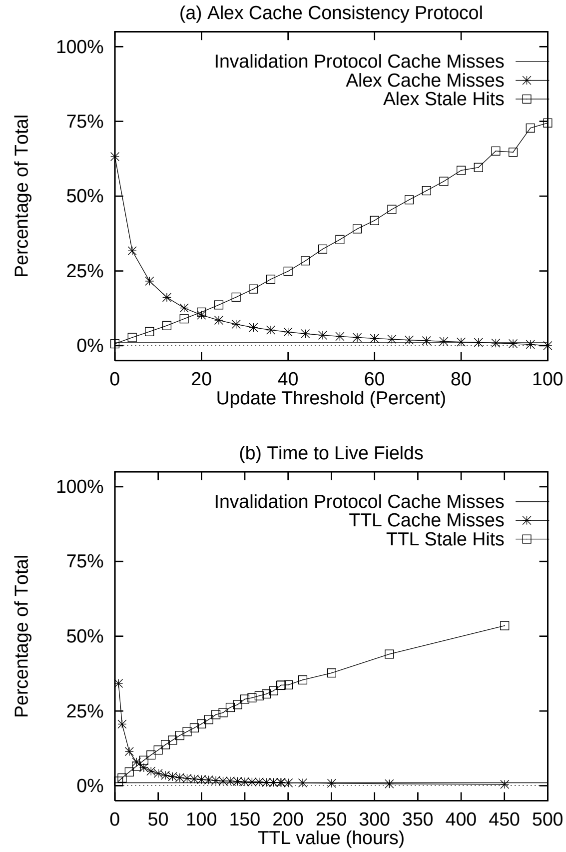


Figure 3. Comparison of cache miss rates in the base simulator. The increases in update threshold and TTL that induced bandwidth savings in Figure 2 also induce an increase in the stale hit rate. The invalidation protocol provides perfect consistency resulting in a 0% stale hit rate (not shown in the figure).

simulator is referred to as the *modified workload simulator.*

4.0 Simulation Results

Figures 2 and 3 show the trade-offs inherent in the parameterization of the Alex and TTL protocols. With Alex, as the update threshold increases, the bandwidth savings also increase (i.e. total bandwidth decreases). However, with this increase in bandwidth savings comes an increase in the number of times stale data is returned to the user (the "stale hits" line in Figure 3). Similarly, with TTL fields, the increase in TTL that induces more bandwidth savings also induces more stale hits. The invalidation protocol is unaffected by parameterization, yielding the constant bandwidth shown in Figure 2, and since valid entries are never evicted from the cache, it also produces the near perfect cache miss rates shown in Figure 3.

Although we expected Alex to outperform TTL, the two figures show that for a specified acceptable stale hit rate, TTL provides greater bandwidth savings. For example, if the acceptable stale hit rate is 25%, then Alex must select an update threshold of approximately 40% (from Figure 3a), inducing a total bandwidth of 400 MB (from Figure 2a). In contrast, to achieve a 25% stale hit rate, the TTL must be set to approximately 125 hours, resulting in a total bandwidth of approximately 130 MB. In both cases, the bandwidth required is greater than that required for the invalidation protocol, and the stale cache rate of 25% is unacceptably high. The difference in bandwidth consumption between Alex and TTL is discussed in more detail in Section 4.2.

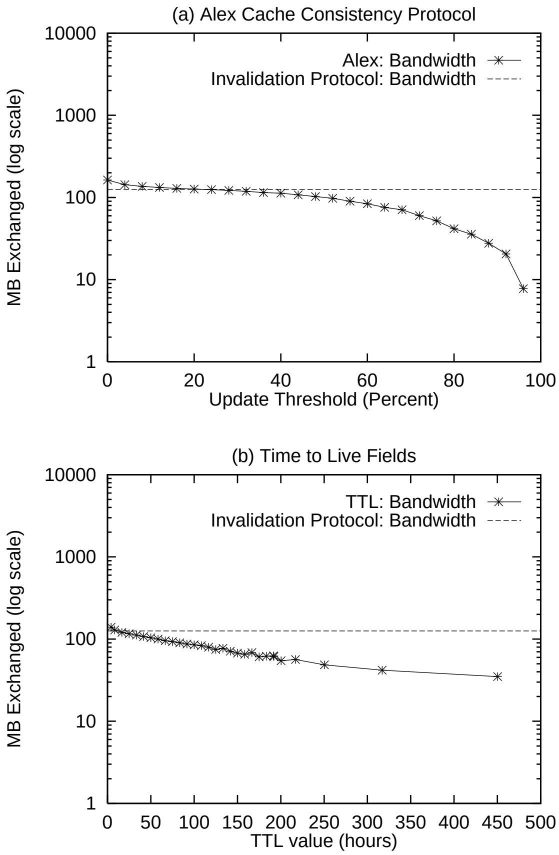


Figure 4. Comparison of bandwidth usage in the optimized simulator. Files are transmitted only when they are truly stale. With this optimization, both TTL and Alex use less bandwidth than the Invalidation Protocol in nearly all cases.

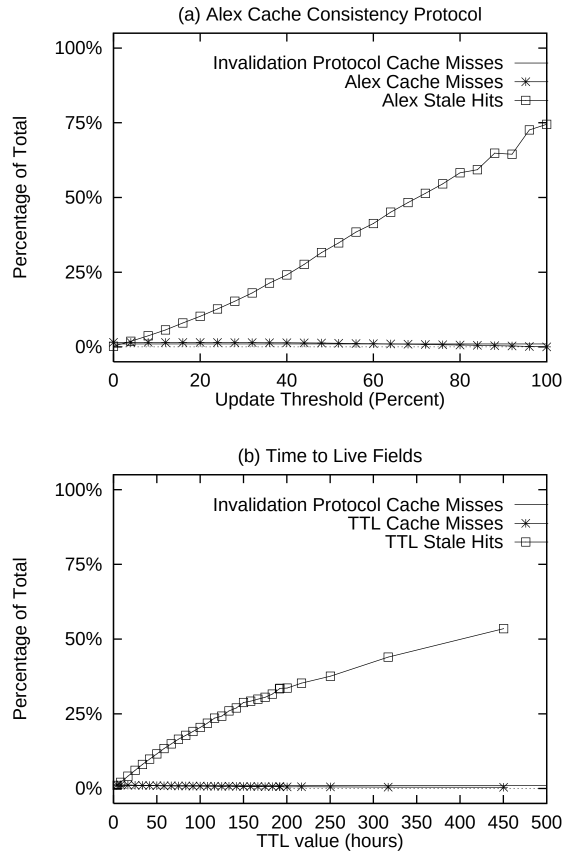


Figure 5. Comparison of cache miss rates in the optimized simulator. The cache miss rates improve dramatically from Figure 3 since invalidated files are left in the cache. All three protocols show miss rates that are indistinguishable from the near perfect miss rate of the invalidation protocol. However, the stale hit rate remains unacceptably high.

4.1 Optimized Retrieval

Our next set of experiments evaluated the conditional retrieval provided by the optimized simulator. The Alex and TTL protocols query the server to determine the validity of their cached, but invalid, data before requesting that a new copy be sent. Figure 4 shows the effect of this change on the total network bandwidth. With this optimization, both protocols outperform the Invalidation Protocol for most parameter settings.

To understand why the protocols save bandwidth, consider the amount of information that must be exchanged in each case. The information can be categorized into messages and file transfers. The invalidation protocol sends an invalidation message every time that a file changes, but sends files only when an invalid file is requested. Both Alex and TTL send messages only after a file has timed out and has been requested again, and send files only when a file that is truly out of date is requested. All three protocols transfer files that are truly invalid. Therefore, neither Alex nor TTL will ever transmit more file information than the invalidation protocol, but could transmit less if stale files are ever returned. On the other hand, the amount of bandwidth consumed by Alex and TTL for messages (queries to the server to check for stale data) could be more or less than that for the invalidation protocol depending on the cache settings. Since each message averages 43 bytes and each file averages several thousand bytes, the effect of saving file transfers is much more pronounced than the effect of sending more server queries. As the number of stale hits increases, the bandwidth consumption decreases.

The more dramatic improvement occurs in the miss rates shown in Figure 5. Both Alex and TTL now achieve near perfect miss rates because the invalidated data are left in the cache, avoiding future

Server	Files	Requests	% Remote Requests	Total Changes	% Mutable Files	% Very Mutable Files
DAS	1403	30,093	84%	321	6.83%	2.61%
FAS	290	56,660	39%	11	2.41%	0.00%
HCS	573	32,546	50%	260	23.3%	5.22%

Table 1: Summary of mutability statistics for various campus servers over a one-month period. Mutable files are defined to be those files that were observed to change more than once over the time period. Very mutable files are those that were observed to change more than 5 times. Any request that was not generated by a client in our campus domain was considered remote, and any files added in the middle of this time period were not included in these statistics. Notice that the most popular server, the FAS server, is also the one with the fewest mutable files.

retrievals. Cache misses are recorded only when a file actually needs to be transferred to the cache. Unfortunately, the stale cache hit rate is unchanged. For example, selecting a TTL of 100 hours saves only 32% of the invalidation protocol's bandwidth but results in a 20% stale cache hit rate. This number of stale hits is probably unacceptable for the moderate bandwidth savings.

4.2 Modified Access Patterns

We expected that an adaptive protocol such as the Alex protocol would do better than the static TTL protocol, so we examined the factors that contributed to Alex's poor performance. The flat lifetime distribution coupled with the fact that all files were assigned equal retrieval probability seemed to be the leading cause. The analysis of traces gathered in our local environment coupled with results by Bestavros [3] convinced us to consider an alternative workload generator.

Bestavros found that on any given server only a few files change rapidly. Furthermore, he observed that globally popular files are the least likely to change. A workload modeled by these characteristics departs significantly from the workload modeled by the base and optimized simulators. If the file request distribution is skewed towards popular files and popular files change less often, then the number of stale hits reported will decrease significantly. An adaptive protocol, such as Alex, will then work well on both rapidly changing files as well as stable ones. While files are changing rapidly, Alex checks frequently; once the files stabilize, Alex checks infrequently.

The modified workload simulator uses Web server logs from our local environment to generate file lifetimes. The server logs were taken from several campus Web servers, modified to store the *last-modified* timestamps with each file request satisfied by the servers. We used the file system's last modification time for the timestamp. The server logs are summarized in Table 1.

The three systems from which we gathered logs are FAS, our university web server, DAS, the web server for the Division of Applies Sciences (think, "College of Engineering"), and HCS, the web server for our local computer society. The statistics from these server logs confirm Bestavros' observation that the most popular files are also the least mutable ones.

It is instructive to compare our trace characteristics with those of the workload simulated by the base simulator. The traced files change far less often than the files with randomly generated lifetimes. For example, one run of the base simulator included accesses to 2085 files over a 56 day simulated run. Those 2085 files changed 19,898 times yielding a 17% average probability that on any given day a particular file changed. Our HCS trace, which changed the most frequently, involved 573 files changing 260 times over 25 days. This yields a 1.8% average change probability, which is consistent with Bestavros' per-day file-change probability of 0.5% – 2.0%, with more popular files changing less often than other files.

While the simulation of our trace data modeled the exact modification behavior on our servers, the change probability computed above is based on a small sample size. Bestavros offers another data point, but it is only accurate between one-day intervals. It is possible that the one day granularity masked a number of changes equivalent to those used by Worrell, but it is unlikely, since Bestavros' data reflected an order of magnitude less change than the simulated workload. Each file that was recorded as changed would have had to have changed not once, but 10 times between samples to produce an equivalent rate of change. Given the significant difference in the rapidity of change between the trace data and the simulated workload, we expected to observe far fewer stale cache hits with the Alex and TTL protocols using the trace data than we did with the random lifetime generation.

In order to verify that the data from our traces is representative of "typical" web usage, we gathered both information on the distribution of accesses to different types of files as well as the average life-spans of these file types. We gathered this data from two different sources. We obtained information about the distribution of accesses to different types of web objects from a proxy cache at Microsoft. We obtained information about the life-span of different file types from modification logs of the Boston University web server.

The Microsoft proxy cache sits between all Microsoft employees and anything outside of Microsoft. The access logs for the server contain the types and sizes of files accessed, but not the last-modified date for files retrieved, so we could not simulate this log. Instead, we used the data to characterize access patterns by file type. On an average week day, the Microsoft proxy cache server receives approximately 150,000 requests for web objects. Of these, 65% are for image files (gif and jpg). The file type breakdown is shown in the second and third columns of Table 2.

To understand what files are the most likely to change, we analyzed the data gathered from the Boston University web server. Each day between March 28 and October 7, Bestavros sampled the server and recorded all the files that were modified since the previous day [4]. The logs contain approximately 2,500 file references and 14,000 changes during that 186 day time period. Categorizing this data by file type, we can determine the average life span per file type. This data is shown in the last two columns of Table 2.

In computing these life-spans, we err on the side of conservatism, overestimating the rate of change by assuming that all data changed at least once during the measurement interval. This biases the results because the longest life-span we consider is 186 days and there almost certainly exist files with longer life-spans. However, ignoring files that did not change and considering only those files that did change would have skewed the results far more.

Images, which represent 65% of the accesses in the Microsoft data, have the longest lifetimes, living 85–100 days. Surprisingly, image files are also relatively small, so caching them is feasible. This supports our hypothesis that weak consistency caching will be effective, since the most popular web objects also have the longest life-span.

	Microsoft		**Boston University**	
File type	**%-age of total accesses**	**Average file size**	**Average life-span (days)**	**Median Age (days)**
gif	55%	7791	85	146
html	22%	4786	50	146
jpg	10%	21608	100	72
cgi	9%	5980	NA	NA
other	4%	NA	NA	NA

Table 2: Tabulation of Microsoft and Boston University server log summaries. The Microsoft data provides information on file access patterns while the Boston University data provides information on file type lifetimes.

While we still need to collect better data from a single server, the behavior observed at Microsoft and Boston University convinced us that our own local traces were representative of the rate of change observed on the web. We then simulated the three different consistency algorithms using a workload based on the trace data summarized in Table 1.

Figures 6 and 7 show dramatically different results from those in Figures 2 through 5. Both Alex and TTL produce less bandwidth usage than the invalidation protocol with few stale cache hits, reflecting the fact that few files change frequently on the server. Since files do not change often, they do not cause stale data to be returned. In contrast to the earlier calculations, we find that with an acceptable stale hit rate of less than 5%, both Alex and TTL demand less bandwidth than the invalidation protocol for nearly all parameter settings and that Alex and TTL offer similar savings in bandwidth.

Having established that the weakly consistent schemes are competitive with invalidation protocols in terms of bandwidth and stale cache hits, it is useful to examine the server load created by the various protocols. Figure 8 shows the number of server operations (i.e. requests for documents, queries to determine whether documents are stale, and invalidation messages) for each of the protocols. While TTL offers bandwidth savings and acceptably low stale cache hit rates, it induces a higher server load than either Alex (with properly tuned parameters) or the invalidation protocol. The number of server queries generated by Alex with an update threshold of 0 is particularly noteworthy. This configuration represents the case where the cache checks with the server on every client request as some poorly designed servers currently do. Not only is this unnecessary, since an update threshold as low as 5%

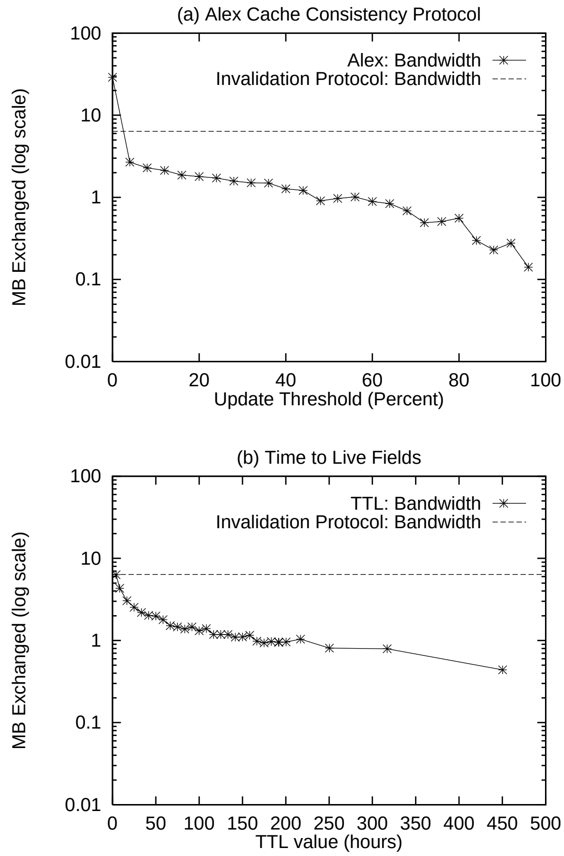


Figure 6. Comparison of bandwidth using the workload modified simulator. These results depict the averages of the FAS, HCS, and DAS traces. Files that were not in the primary host at the beginning of the month were not simulated. Both Alex and TTL use less bandwidth than the Invalidation Protocol for nearly all parameter settings.

returns stale data less than 1% of the time, but it is excessively wasteful of server resources since it creates nearly two orders of magnitude more server queries.

5.0 Future Work

Our simulations indicate that maintaining cache consistency in the World Wide Web need not be expensive. However, there are still important issues to be examined. The time-based protocols (Alex and TTL) both rely on careful tuning of parameters. Leaving this tuning to manual intervention is guaranteed to result in suboptimal performance. Furthermore, as the Boston University and Microsoft trace data indicate, different types of files exhibit different update behavior. One important area of further research is to investigate tuning cache consistency protocol parameters.

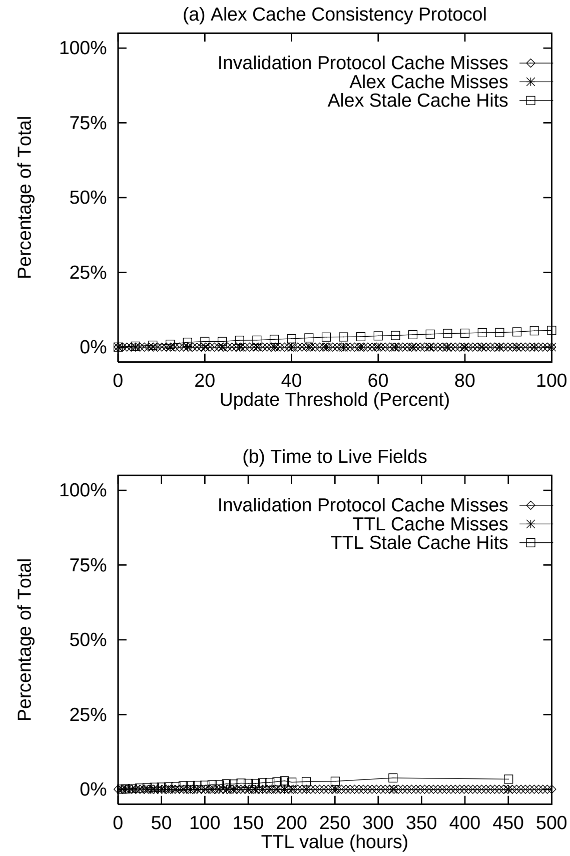
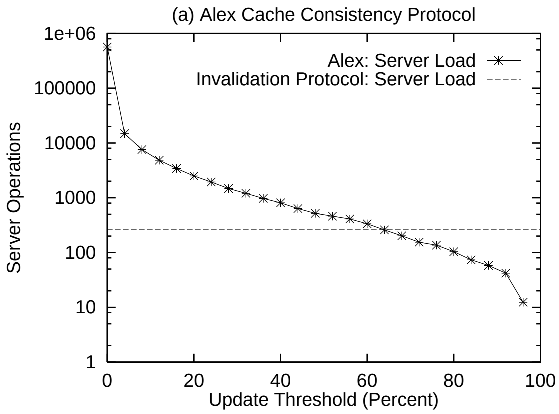


Figure 7. Comparison of cache miss rates using the modified workload simulator. Both protocols provide extremely low stale data rates using trace-driven simulation. The cache miss rates for the invalidation protocol, Alex, and TTL are all less than 0.04%, producing the overlapping lines near 0%.

We are investigating algorithms by which caches can be self-tuning, by adjusting parameters based on the data type and the history of accesses to items of that type.

Another trend in web usage that has an affect on proxy caching is the increasing number of web objects that are dynamically generated. The Microsoft trace logs revealed that 10% of the requests were for dynamically generated pages. This represents a tenfold increase from only six months ago. As the number of dynamic objects increases it will become critical to devise ways to cache the actual scripts that generate dynamic pages. Web scripting languages such as Java and Tcl offer one possible approach, but autonomously replicating the databases that underlie most dynamic content is non-trivial.



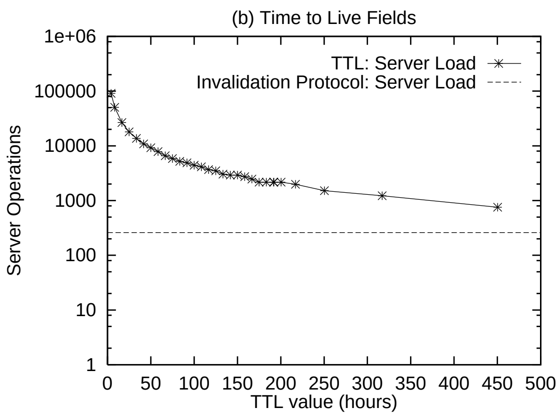


Figure 8. Comparison of server loads on the three consistency protocols. Notice that parameterization is critical for efficient operation of either Alex or TTL and that Alex imposes less load on the server than TTL. TTL always imposes more load than the invalidation protocol while Alex requires an update threshold of at least 64% in order to achieve the same server load as the invalidation protocol. At this 64% threshold, the stale cache miss rate is 4%.

6.0 Conclusions

While Worrell's results presented a strong preference for invalidation protocols relative to TTL, our results differ significantly. If network bandwidth is the driving force, then TTL is an attractive alternative, offering reduced network bandwidth and a low stale hit rate. It does present a significantly higher load to the server, which makes it unattractive. However, in general, the Alex protocol provides the best of all worlds in that it can be tuned to:

- reduce network bandwidth consumption by an order of magnitude over an invalidation protocol,
- produce a stale rate of less than 5%, and
- produce server load comparable to, or less than, that of an invalidation protocol with much less bookkeeping.

Although Alex is preferable to TTL, there are cases where TTL might still be suitable. For example, when object lifetimes are known *a priori,* as is the case with daily news articles or weekly schedules, TTL is the right choice.

Although invalidation protocols are still required when perfect cache consistency is a necessity, the weakly consistent protocols are particularly attractive for a number of reasons. They are both much simpler to implement. They are both more fault resilient when machines become unreachable; the right thing automatically happens. Documents eventually become invalidated and the server is contacted upon subsequent requests. With an invalidation protocol, recovery is much more complicated. The changes required to implement an invalidation protocol in existing web servers and clients is more significant than the effort to implement either TTL or Alex.

7.0 Acknowledgments

We would like to thank Kurt Worrell for inspiring this work and for giving us his simulator, the Harvard Arts and Sciences Computer Services Organization, the Division of Applied Sciences computing staff, and the Harvard Computer Society for running our modified web server, the Microsoft Gibralter team for providing us with trace logs, Keith Smith and Anna Watson for their careful reviews, and Azer Bestavros for his data and continued excellent advice.

8.0 Bibliography

[1] Andreessen, M., private email correspondence.

[2] Berners-Lee, T., “Hypertext Transfer Protocol HTTP/1.0,” HTTP Working Group Internet Draft, October 14, 1995.

[3] Bestavros, A., “Demand-based Resource Allocation to Reduce Traffic and Balance Load in Distributed Information Systems,” to appear in *Proceedings of the SPDP’95: The 7th IEEE Symposium on Parallel and Distributed Processing*, San Antonio, TX, October 1995.

[4] Bestavros, A., “Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic and Service Time for Distributed Information Systems,” *Proceedings of 1996 International Conference on Data Engineering*, New Orleans, Louisiana, March 1996.

[5] Blaze, M., "Caching in Large-Scale Distributed File Systems," Princeton University Technical Report, TR-397-92, January 1993.

[6] Cate, V., "Alex— A Global Filesystem," *Proceedings of the 1992 USENIX File System Workshop*, Ann Arbor, MI, May 1992, 1–12.

[7] Chankhunthod, A., Danzig, P., Neerdaels, C., Schwartz, M., Worrell, K., "A Hierarchical Internet Object Cache," *Proceedings of the 1996 USENIX Technical Conference*, San Diego, CA, January 1996.

[8] Dahlin, M., Mather, C., Wang, R., Anderson, T, Patterson, D., "A Quantitative Analysis of Cache Policies for Scalable File Systems," *Proceedings of the 1994 Sigmetrics Conference*, May 1994, 150–160.

[9] Danzig, P., Hall, R., Schwartz, M., "A Case for Caching File Objects Inside Internetworks," Technical Report, University of Colorado, Boulder, CU-CS-642-93, 1993.

[10] Gwertzman, J., "Autonomous Replication in Wide-Area Distributed Information Systems," Technical Report TR-95-17, Harvard University Division of Applied Sciences, Center for Research in Computing Technology, 1995.

[11] Howard, J., Kazar, M., Menees, S., Nichols, D., Satyanarayanan, M., Sidebotham, R., West, M., "Scale and Performance in a Distributed File System," *ACM Transactions on Computer Systems, 6*, 1, February 1988, 51–81.

[12] Luotonen, A., Frystyk, H, Berners-Lee, T., "W3C httpd," http://www.w3.org/hypertext/WWW/Daemon/Status.html.

[13] Nelson, M., Welch, B., Ousterhout, J., "Caching in the Sprite Network File System," *ACM Transactions on Computer Systems, 6*, 1, February 1988, 134–154.

[14] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., and Lyon, B., "Design and Implementation of the Sun Network Filesystem," *Proceedings of the Summer 1985 USENIX Conference*, Portland OR, June 1985, 119–130.

[15] Wessels, D., "Intelligent Caching for World-Wide Web Objects," *Proceedings of INET-95, 1995*.

[16] Worrell, K., "Invalidation in Large Scale Network Object Caches," Master's Thesis, University of Colorado, Boulder, 1994.

James Gwertzman is a program manager at Microsoft Corporation where he works on the Microsoft Network. His research interests include distributed systems, online communities, and data replication. He received an A.B. degree from Harvard College in 1995, and was the recipient of a Hoopes prize for his senior thesis. He promises to attend graduate school in the near future.

Margo I. Seltzer is an Assistant Professor of Computer Science in the Division of Applied Sciences at Harvard University. Her research interests include file systems, databases, and transaction processing systems. She is the author of several widely-used software packages including database and transaction libraries and the 4.4BSD log-structured file system. Dr. Seltzer spent several years working at start-up companies designing and implementing file systems and transaction processing software and designing microprocessors. She is a Sloan Foundation Fellow in Computer Science and was the recipient of the University of California Microelectronics Scholarship, The Radcliffe College Elizabeth Cary Agassiz Scholarship, and the John Harvard Scholarship. Dr. Seltzer received an A.B. degree in Applied Mathematics from Harvard/Radcliffe College in 1983 and a Ph. D. in Computer Science from the University of California, Berkeley, in 1992.