The Effectiveness of Cache Coherence Implemented on the Web

Felicia Doswell and Marc Abrams
Department of Computer Science
Virginia Tech
Blacksburg, Virginia 24060
{fdoswell,abrams}@.vt.edu

Abstract—**The popularity of the World Wide Web (Web) has generated so much network traffic that it has increased concerns as to how the Internet will scale to meet future demand. The increased population of users and the large size of files being transmitted have resulted in concerns for different types of Internet users. Server administrators want a manageable load on their servers. Network administrators need to eliminate unnecessary traffic, thereby allowing more bandwidth for useful information. End users desire faster document retrieval. Proxy caches decrease the number of messages that enter the network by satisfying requests before they reach the server. However, the use of proxies introduces a concern with how to maintain consistency among cached document versions.**

Existing consistency protocols used in the Web are proving to be insufficient to meet the growing needs of the World Wide Web population. For example, too many messages are due to caches guessing when their copy is inconsistent. One option is to apply the cache coherence strategies already in use for many other distributed systems, such as parallel computers. However, these methods are not satisfactory for the World Wide Web due to its larger size and range of users. This paper provides insight into the characteristics of document popularity and how often these popular documents change. The frequency of proxy accesses to documents is also studied to test the feasibility of providing coherence at the server. The main goal is to determine whether server invalidation is the most effective protocol to use on the Web today. We make recommendations based on how frequently documents change and are accessed.

Keywords—**Caching, Proxy, Coherence, Consistency, Web.**

I. Introduction

Due to its simple interface to a wide array of media types and its accessibility from multiple platforms, the Web has become a major form of information dissemination. It provides easier access to information via Web browsers, sometimes referred to as "clients". This massive availability of information has lead to a number of concerns. End users experience high latency when attempting to retrieve documents and images. High bandwidth consumption and network congestion are other problems evident. In addition, a large amount of the Web traffic is due to fetching of documents that have recently been retrieved, possibly by users at the same site. Enhancements to control these problems are constantly being proposed by researchers. Compression of documents and sending document differences (DIFF) in response to requests is one option that reduces the number of bytes that are sent over the network as opposed to entire documents. However, it has been shown that only 10 percent of bandwidth is saved by using these methods. Storing HTTP headers (in compressed binary form) separately from the body is another bandwidth improvement. Unfortunately, this would require twice as many files and demand more effort to manage them. Reducing the number of protocol messages that are transmitted over the Internet is another cost saving solution but only applies to reducing network traffic. Document pre-fetching is a technique that provides quicker access to documents for the end user by predetermining and retrieving what the user may access next. However, this technique could possibly introduce unnecessary traffic onto the network by retrieving documents that the user does not want.

Proxy caching is one way to address many of these concerns. They provide benefits to the end user, network administrator, and server administrator by reducing the amount of redundant traffic that circulates through the web. In addition, they act as intermediate agents to allow multiple clients to quickly access a group of popular Web pages. In such an approach, requests for documents are redirected to a cache rather than being serviced directly by a server. When documents are fetched from a server, copies are kept in proxy caches throughout the Web. Although proxy caches provide a significant reduction in network traffic, they also present a new set of concerns that must be addressed. How do we manage the document changes at the server that must be propagated to the proxies? These cached copies should be updated in such a manner that all versions are consistent with the server copy. The technique for accomplishing this is called *cache consistency* or *coherence*.

Several approaches have been proposed to limit the number of stale copies of a document that occur in the Web. They include Time-To-Live (TTL), Client Polling, and Invalidation [7]. In considering cache coherence protocols, it is necessary to determine when changes should be propagated to proxies and whether the server or the proxy should control when modifications are available to clients. The current document consistency mechanisms implemented on the Web is client polling, where periodically the server is contacted when the client request a document. If a document changes frequently, such as online news and sports, then a stale document may be returned if the user requests the document before it is updated. Current Web technology also allows the user to avoid inconsistent documents by specifically requesting the updated version directly from the server. However, this may introduce unnecessary exchange of messages between the client and server, especially when the server documents are changed infrequently.

The purpose of this study is to address the consistency issue. It describes an analytical approach to approximate which cache consistency algorithms work best given certain network parameters. An upper bound study was done to ascertain the rate that documents on the Web change, and to reveal how many messages will be required to communicate these changes to copy holders. This will allow us to enumerate the circumstances in which certain mechanisms are most appropriate. We evaluate whether a server-based invalidation protocol is the most effective choice to solve the problem. Two experiments are performed. First, we measure the fraction of documents that are modified often and try to determine whether the changes occur in a predictable manner. In other words, if a large number of documents change in a short period of time, do they change every [illegible] hours. If [illegible] is large, but the rate of read accesses is small, then HTTP today generates many unnecessary requests to the server. A major part of this study is to determine how many proxies would be notified of changes in a server document if server invalidation were used. We also analyze the data to reveal how many proxies actually access more than one document at a server or access a document multiple times. This will reveal how much overhead is involved in using the invalidation method to disseminate changes at the server. We want to provide consistent documents while minimizing the number of consistency checks necessary to accomplish this task.

The rest of this section is an overview of caching, and the HTTP protocol. The remainder of the paper begins with an overview of related work followed by a presentation of the objectives of this research. Section 4 describes the analytical approach and results to approximate the performance of four proposed algorithms. Section 5 is a discussion of the experimental design and results. The paper concludes with a summary of the results and the conclusions drawn about the merits of server invalidation.

A. Proxy Caching Overview

Caches are designed to keep copies of documents presumably closer to the client (user) in order to handle repeated requests for documents. If located in the network between the client and server, a cache prevents the client from having to always retrieve documents directly from the server. This is called a proxy cache. They allow sharing of documents requested by multiple clients. In a Web configuration, the browser retrieves documents by communicating with a proxy or origin server. First, the client browser forwards a request to the proxy cache it is configured to use. If the cache does not have the requested document, it forwards the request to the server in the same manner that a browser would if a direct connection were made to the server. If the cache contains the requested document, it will check to determine if it has a fresh copy. Fresh documents are served directly from the cache without checking with the server. If the document is stale, the origin server will be asked to verify whether the cached copy is consistent with the server copy. If the server sends an updated copy of the document, the proxy stores a copy of it and forwards the document to the requesting client.

B. HTTP and Consistency

The way that most proxy caches and servers send messages is by using the HyperText Transfer Protocol (HTTP). It is a communication protocol for transferring data throughout the Internet. Although HTTP1.1 is the first version to give explicit rules for consistency on the Web, it is necessary to discuss HTTP1.0 since it paved the way for the next generation communication protocol.

B.1 HTTP1.0 and Consistency

The direct way to retrieve a document is to use the HTTP GET request. To address consistency, HTTP1.0 allows use of a *conditional GET*. With this feature a document is retrieved based on whether it has been modified since the last access. If it has been modified, the new document is retrieved from the server. If it has not been modified, the server will return a *not modified* message that instructs the proxy to return its own cached version. It is apparent that simply using the *conditional GET* every time a request is sent will provide the client with consistent documents. If we assume that most documents do change often, then employing such a technique could give the desired consistency. However, this method could also waste more bandwidth by introducing unnecessary messages onto the Web. This occurs if most document requests result in the return of a *not modified* messages when the cache copy may be sufficient in providing a consistent document.

An alternative is to use the date/timestamps to identify when an entity is inconsistent with the document on the origin server. A client can be notified of a documents potential staleness by using the *Expires* header. Similarly, the age of a document reveals whether a cached document is stale. These directives are used by proxies to determine if a cached document is inconsistent. If so, then the server is polled to determine if the document has changed. One problem with the *last-modified* and the *expires* mechanisms is that document owners often do not use these header fields, which makes it impossible for proxies to control cache consistency in this manner. Finally, the *no cache* option in the HTTP1.0 headers provides a way to tell the proxy that the document should never be cached and therefore should be directly retrieved from the server [10].

B.2 HTTP1.1 and Consistency

In contrast, HTTP/1.1 uses entity tags (ETags) to compare two or more objects from the same requested resource. ETags are unique identifiers that are generated by the server. They change each time the document changes. In addition to these new identifiers, HTTP1.1 still provides the functions specified in the HTTP1.0 version of the protocol. This includes specifying what should be cacheable (public), what may be stored by caches (no-cache), an expire mechanism (max-age), and the reload operation. HTTP1.1 also offers revalidation tags where the server tells the clients exactly how to validate the data. This includes the ability to force each cache along the path to the origin server to revalidate its own entry with the next cache in the path.

B.3 HTTP Browsers and Consistency

In addition to the proxy, the browser cache plays a role in consistency control. Browser caches, also called client caches, are disk storage on the end users computer that only cache documents for its attached browser. The advantage of such caches is to provide immediate access to previously viewed pages, such as when the ”back” button is pressed on the Web browser. The *preferences* dialog of Internet Explorer or Netscape, contain a “cache” setting. There are two mechanisms here that allow the user to specify when a fresh document must be retrieved from the server, or whether it would be sufficient to have a copy from the browser or network cache. Generally, the client user must manually configure which proxy to use thereby giving the permission to retrieve possibly cached documents. In addition, browsers have a cache setting where the user can specify whether it wants the browser to verify a document once per session, every time the document is retrieved, or never. If the document has an expiration time or other age-controlling directive set, the browser can determine if the document is still within the fresh period and will not contact the server. However, requiring the document to be retrieved directly from the server will cause increased delay in getting the document, which may not have been modified since the last retrieval. Also, the user can always use the *reload* button that is available with Web browsers today. This involves use of the *no-cache* header mentioned earlier. If the user feels that a retrieved document is not up-to-date, they can simply press the *reload* button and retrieve the latest server copy.

II. RELATED WORK

A. Consistency in Other Systems

Cache consistency has been an issue long before the World Wide Web made its entrance. Hardware caches, and caches within distributed systems have provided some extensive coverage of cache consistency and have resulted in proven methods to solve the consistency problem. However, the solutions are not directly applicable to the Web. Hardware systems can provide strict cache consistency due to its smaller size [8]. In addition, hardware caches do not require attention to failing network conditions. This is difficult to apply to the Internet due to the need for scalability as more caches are added. Also, hardware systems require handling of multiple reads and writes by users, whereas the Internet is currently a one writer-multiple reader problem [2]. Distributed systems include distributed databases, file systems and main memory. The distributed systems that are most closely related to the Internet, in terms of cache consistency, are distributed file systems where both are implemented in software and involve access to volumes of information. However, the Web is different from distributed file systems in terms of access patterns, its larger size, and its single point of access. The first two issues makes caching more difficult in the Internet than file systems, while the latter one makes it easier. [7].

B. A Survey of Cache Consistency Mechanisms

In general, cached copies should be updated when the originals change. However, users may desire to have an out-of-date copy of a document rather than waiting for the document to arrive from the server. An algorithm that does this is considered a *weak consistency* algorithm. On the other hand, if a stale copy is not tolerable, then a *strong consistency* algorithm is necessary. Such algorithms guarantee that the user will see the same version as the origin server. Most existing Web caches provide weak consistency. Use of such mechanisms requires the user to ensure freshness, when desired, by pressing the reload button on a browser. This causes a burden on the server as well as the user [7]. One advantage of each of the *weak consistency* algorithms is that they are easy to implement in HTTP1.1. Here, we present a few weak and strong consistency protocols that have been proposed in the literature.

B.1 Time-To-Live

Time-to-live (TTL) [7] is a technique where a priori estimates of a document's lifetime are used to determine how long cached data remains valid. This method is most useful for server administrators who know when a document changes. For instance, if a news page is updated once a day at 7 am, the object can be set to expire at that time, and the cache will know when to get a fresh copy. With the TTL approach, each document is given a fixed "time-to-live" *expire* field associated with it. The time-to-live field is used by the cache to determine if a cached document is fresh. When the TTL elapses, the data is considered invalid. Subsequent requests of invalid data result in the client requesting the newest version from the original server. This weak consistency mechanism is implemented using the HTTP1.0 *expires* field. However, it is not easy to select approximate TTL values. Too short of an interval could cause data to be unnecessarily reloaded. Too long of an interval results in increased staleness. The adaptive TTL approach is a proposed idea where the TTL value is adjusted periodically based on observations of its lifetime. This reduces the possibility of stale documents. Gwertzman and Seltzer [7] have shown that adaptive TTL keeps the probability of stale documents to below 5%.

B.2 Client Polling

In client polling [7], clients periodically check back with the server to determine if the cached documents are still valid. Each client cache sends an *if-modified-since* request to the server. The server then checks to see if the document has changed since the timestamp. If so, a status code of 200, is sent along with the fresh document. Otherwise, the server returns a code of 304 (*document not modified*). ALEX [3], a form of client polling, uses an update threshold to determine how frequently to poll the server. The threshold is a percentage of the document's age. The age is the time since the last access to the document. A document is invalidated when the time since the last validation exceeds ($update\ threshold * document\ age$). This mechanism is implemented using the HTTP1.0 *if-modified-since* request header field. It is also fault resilient for unreachable caches. On the other hand, use of client polling can result in a cache returning stale data or invalidating data that is still valid.

Polling-Every-Read [13], also called Client Invalidation [9] and Polling-Every-Time [2], is a version of client polling where the server is polled every time a request is sent rather than periodically. Netscape Navigator Version 3.0 and above already allows users to select this approach. However, this technique negates the need for caching because it always bypasses the cache even though the document may be cached. In this instance, client polling becomes a strong consistency mechanism at the cost of bombarding the server and network with excessive requests and responses. In addition, the user will experience delay in receiving the document every time a request is made.

B.3 Invalidation

Server invalidation [7], also called callbacks [13], is a mechanism where the server sends out notifications to clients when a document is modified. The advantage of this approach is that it provides stronger consistency while introducing less unnecessary messages than the currently proposed consistency methods. However, this scheme requires the server to keep track of which clients or proxies store copies of a document. This is expensive in terms of storage overhead and processing. In addition, the list of clients maintained by the server may become out of date. Another problem is how to handle recovery when caches or servers become unavailable. If a client becomes unreachable due to failure or lost messages, the servers may have to wait indefinitely to send a new document version to a client. Another issue to consider is that invalidation requires modifications to the server and the HTTP1.1 protocol, whereas TTL and Client Polling are implemented at the proxy level. There are variations of invalidation messages that minimize some of the costs noted above. We list a few below.

B.3.a Update Invalidation. Along with each invalidation, the new document version is returned in the invalidation message. There is no need to contact the server on subsequent requests. The disadvantage is the excessive sending of large documents to caches that may never be contacted again for the said document [5].

B.3.b Delta Invalidation. Along with each invalidation, send the revisions to each proxy rather than the entire update. This would require use of delta encoding (which is not widely used), and although it minimizes the bandwidth usage, there is still a chance that the document may never be accessed again [5].

B.3.c PiggyBack Invalidation. Along with each invalidation message, send additional invalidations for pages that may be accessed in the future. This will decrease the number of subsequent request messages [9]. However, it is possible to send documents that will not be accessed later.

B.3.d Leases. Distributed file systems currently use leases [6] to address the problems with basic invalidation. With leases, every document that is sent to a client contains a lease to specify a length of time that servers will notify clients of modifications to cached data. If a document changes before the lease expires, then the server will use invalidation to notify the client of the modification. After the lease expires, the client will send a renewal request in the form of an *if-modified-since* message. The protocol is a combination of client polling and server invalidation. This decreases the amount of time that a server will have to wait, due to unreachable clients , to complete a write. The server only waits until the lease expires, rather than indefinitely. In addition, leases decrease the possibility of contacting obsolete clients that retrieved the document in the past but no longer access them.

C. Evaluation of Consistency Mechanisms

Several approaches have been applied to maintaining consistency in the Web. Research efforts have included comparisons and proposals of several invariants of the three major protocols: time-to-live, client polling, and invalidation. For example, Worrell's thesis concludes that invalidation is a better approach for cache consistency. The study was restricted to the investigation of consistency in a hierarchical network, which already carries a higher overhead in communication than a simple, flat network [12]. Contradicting the results of Worrell, Gwertzman and Seltzer [7]discuss various cache consistency approaches and conclude that a weak-consistency approach such as adaptive TTL would perform better for Web caching in terms of network bandwidth. Cao and Liu [2], extended this study and studied the implication using other metrics like server load, response time, and message delay. These authors concluded that invalidation performed better than client polling and similar to adaptive TTL in terms of network traffic, average client response time, and server load [2]. Their study however, assumes that all documents are equally important in terms of consistency, and trades this guarantee for strong consistency with the increased bandwidth usage. They also do not address the overhead necessary for servers to invalidate copies. We are concerned with how current consistency mechanisms will affect network performance, particularly bandwidth.

D. Effect of Web Access Patterns on Cache Consistency

Douglis, et al. [5] performed a study to determine the rate at which documents change. Their main goal was to uncover how the rates of modifications affect systems that use delta-encoding. This information is also important to consistency. If we can detect that modifications occur at regular intervals, then we can predict which algorithm will perform better given the rate of modifications. In their study, Douglis, et al. collected traces at the Internet connection of two corporate networks and analyzed this data for a rate of change. This data set restricts the study to measuring requests at a specific point in time (when it was collected) [5]. Similarly, statistics were collected only when a reference to a resource occurred. This does not account for documents that are accessed once and never accessed again. In terms of consistency, it is important to know the access and change pattern of all documents. We measure data periodically using various predetermined intervals. In addition, Douglas et al. [5] discuss how often modifications occur when affected by certain characteristics (e. g. content-type, number of references). We investigate characteristics such as category of URLs, and the time of day that modifications occur. Since people generally determine what they will access based on categories (e. g. news, business) rather than content type (e. g. text, video, GIF), it would be more informative to measure the data from that realistic perspective.

Wills and Mikhailov [11] did a similar study to determine the nature and rate of change of documents using URLs rather than testing log samples. They use MD5 Checksum to obtain the difference in documents [11]. The MD5 checksum algorithm has been found to produce collisions when computing the hash functions [4]. Therefore, our study will make use of the UNIX DIFF and CMP commands to determine the difference among documents. These commands provide less possibility of error than the computation of a checksum. In addition, CMP allows the user to compare binary content of images. Wills and Mikhailov [11] used ".com", ".net", and search (which are special types of ".com") categories. We use more realistic categories of business, family, news, and search. In addition, we have a combined "catch all" category based on a rating by a popular computer magazine. The research data of [11] was collected over a one day period. To have a random sample, we use a longer collection period.

III. Objectives

The primary goal of this study is to determine the effectiveness of server invalidation and its role in providing consistency. We wish to define when invalidation should be used. The objective is to study, and understand the pattern of access to documents in terms of frequency of requests and the number of proxy accesses for a document.

We address the following issues.

- Are the documents modified in a predictable manner (never, sometimes, often, always)?
- How does the access and modification rate affect the number of messages sent?
- How many web sites need to be notified of a change in a document if server invalidation is used?
- How often do proxies contact the server for the same document and for different documents?

IV. Analytical Evaluation of Consistency Mechanisms

In this section, we describe the analytical evaluation of the performance of four basic consistency algorithms. We compare Polling-Every-Read, Time-To-Live and two variations of Invalidation. The metrics used to measure the performance of these algorithms is the number of control messages, *CM*, and the number of file transfers, *FT*. Control messages are the messages that trigger forwarding of requests to the server and responses that do not carry data. These are generally *if-modified-since* requests, invalidations, or *not-modified* (304) response messages. The file transfers are the messages that carry data.

Table I lists the variables that represent the number of requests or modifications that are sent due to consistency. There is one special variable used for TTL, R_t. These requests must be distinguished because a request is only sent to the server if the requested document is considered stale by the cache (document timeout). A regular request, *R*, results when a simple **GET** (M_{get}) or **IMS** (M_{ims}) request is sent without the extra time constraints.

The message variables that begin with a *M* define the message types for each of the requests and file transfers. Our analysis actually started with the form $R(M_{ims}) + R(M_{304}) - W(M_{ft})$. However, since all consistency control messages are the same size, we simplified the equations by using like values for the first two terms. For example, $R(M_{ims}) + R(M_{304}) - W(M_{ft})$ becomes $2 * R(M) - W(M_{ft})$. If the bandwidth is desired, then substitute the size for each of the *M* terms. However, if only the count is needed, then the *M* terms go away. Therefore, $2 * R(M) - W(M_{ft})$ becomes $2 * R - W$. We are only interested in the count for this analysis.

A. Formula Derivations

Next, we describe derivations for formulas used to estimate the number of control and file transfer messages sent for Polling-Every-Read, Time-To-Live, and the two variants of invalidation, Purge Invalidate and Update Invalidate. We use the variables defined in Table I.

TABLE I

Variables representing the number of requests or writes that are sent

Variable	Definition
M_{ims}	If-Modified-Since control message
M_{get}	Regular GET control message
M_{inv}	Invalidation (only) control message
$M_{inv\&ft}$	Invalidation control message with file included
M_{ft}	File transfer response message
M_{304}	304 response message
R	# of request
R_t	# of request after timeout (TTL) (first occurrence only)
R_w	# of request after a write
R_{tw}	# of request after timeout and write (first occurrence only)
R_{inv}	# of request after a write
W_{tw}	# of writes after timeout (first occurrence only)
W	# of writes
T	Length of time of the traffic sampling
t	Timeout value

A.1 Polling-Every-Read

Polling-Every-Read [13] is activated whenever a user requests a document using a *If-Modified-Since* header in the **GET** request. The number of consistency control messages that result from the request is *CM = 2R* if no modifications occurred. However, this number is reduced by the number of file transfer messages resulting after a write has occurred. Therefore, the number of control messages is actually *CM = 2R - FT* where *R* is the number of request. The number of file transfers is *FT = W* if there are more reads than writes generated. However, if there are more writes, then $FT = R_w$ where R_w is the number of reads that occur after a write.

A.2 Time-To-Live

The Time-To-Live [7] consistency mechanism is activated whenever a cache times out its copy of a document. It involves the cache recognizing that a cached copy is stale using the TTL field of the document. Once a copy is determined to be stale it is fetched from the server on a subsequent request for that document. The number of control messages is the number of timeout intervals in a sampling of traffic. That result generally depends on whether there is a read in every interval. Therefore, $CM = T/t$ where T is the length of time of the sampling of traffic and t is the timeout value. If the interval length is less than the rate of reads, then the number of request that generate a message to the server is R_t. This value represents the first requests that occur after a timeout. The number of file transfers is $FT = W_t$ where W_t is the number of writes exactly after a timeout, but before that first read after the timeout. This means that $CM = 2 * R_t - W_t$.

A.3 Invalidation

We consider the purge and update invalidation techniques in this analysis. With the invalidation approach, the mechanism is activated when data changes on the server (write). It involves the server notifying the proxies that a cached item has become stale. If we let N denote the number of proxies to contact in the event of an invalidation, and W denote the number of write modifications, then the number of invalidation messages is $N * W$. For the purge invalidation method, the number of file transfers resulting from consistency is $FT = R_w$, where R_w is the number of subsequent requests after an invalidation. Therefore, the total number of control messages due to consistency is $CM = N * W + R_w$. For the update invalidation method, $FT = 0$, which reduces the number of control messages to $CM = N * W$. Although FT is larger for the purge method, there is more bandwidth usage for the update method because the invalidations includes data. To demonstrate that update invalidation has extra overhead of sending the file with the invalidation, we include the $M_{invdeft}$ term. Therefore, for update invalidation $CM = N * W * M_{invdeft}$.

Table II gives the formulas for determining the number of control messages (CM) and file transfer messages (FT) for each pattern fragment of message traffic.

TABLE II
# OF CONTROL MESSAGES AND FILE TRANSFERS FOR FOUR CONSISTENCY MECHANISM PROPOSED IN THE LITERATURE

Method	*Message Count By Read/Write Rate*		
	$R_t \le T/t$, $W_t \le T/t$	$W_t \le R_t$	$W_t > R_t$
TTL	$CM = 2* T/t - FT$ $FT = T/t * (R_{tw})$	$CM = 2* R_t - FT$ $FT = W_t$	$CM = 2* R_t - FT$ $FT = R_{tw}$
Poll Every Read	NA	$CM = 2* R - FT$ $FT = W$	$CM = 2* R - FT$ $FT = R_w$
Purge Invalidate	NA	$CM = 2* W - FT$ $FT = W$	$CM = 2* W - FT$ $FT = R_w$
Update Invalidate	NA	$CM = W * M_{invdeft}$ $FT = 0$	$CM = W * M_{invdeft}$ $FT = 0$

B. *Summary of Analysis*

Using the formula derivations above, we summarize how each of these variables affect consistency. Note that in Table II, the M terms were left in the equations for the update invalidation method. This was done because those message types include control information and file data while the others do not. To simplify comparisons, we let the number of proxies, $N = 1$. Therefore, N is not shown in the formulas for the invalidation methods in Table II.

In Table II, the number of consistency control messages and file transfers depend on whether the writes dominate or the reads dominate. When the reads dominate the number of messages are controlled by both the number of writes and the number of reads for all protocols except update invalidation. When the writes dominate, then the control is based on the number of reads for all protocols except purge and update invalidation. Finally, for the TTL approach, additional control was based on whether the rate of reads and writes were less than the rate of timeouts. Figures 1 and 2 demonstrate how the rate of reads, writes and timeouts control the number of messages, specifically how they affect the CM and FT values.

The most obvious trends in figure 1 is that the Polling-Every-Read starts to increase exponentially as the reads increase, while the other methods start to level off. This is because the control messages for the Polling-Every-Read technique is directly dependant on the rate that reads occur while the other algorithms depend on a timeout value (TTL) or the rate of writes (Invalidation). The Time-To-Live algorithm is dependent upon the threshold and the timeout value. Therefore, it results in a constant value of $T/t = 8$ in figure 1. However, when rate of reads or writes fall outside of the timed interval, t, the counts start to depend on whether there is a $Write..Read$ pattern that occurs immediately after that timeout. Figures 1 and 2 show that Polling-Every-Read is better until the reads become more than the writes or $Reads/Writes \le 1$. At that point, the Update Invalidation becomes better for the control messages. Update Invalidation is shown to perform slightly better than Purge Invalidation because of the count of subsequent requests that result in a miss. The Update Invalidation algorithm returns data when it invalidates, so there are no subsequent misses. However, we must consider the fact that objects that are modified may never be accessed again. For that reason, sending bytes before they are requested become a risk.

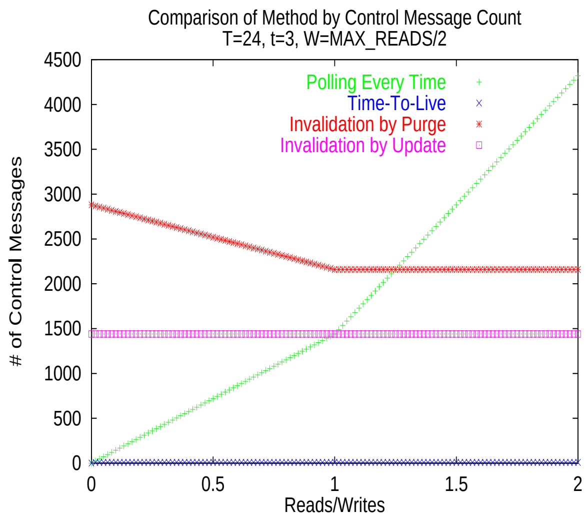

Fig. 1. Number of Control Messages based on ratio of Reads to Writes

We also produced graphs that changed the threshold and time-

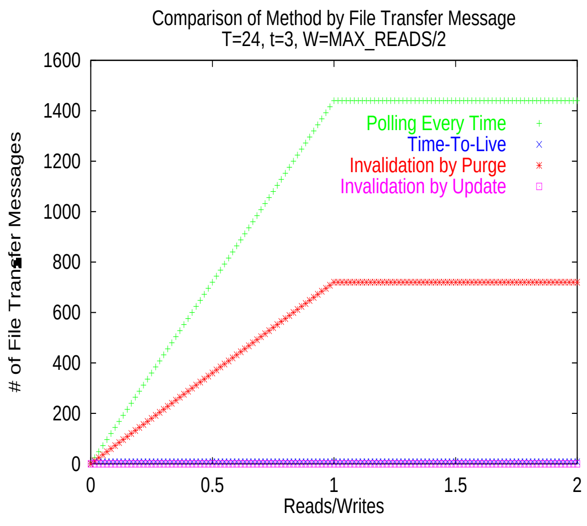

Fig. 2. Number of File Transfers based on ratio of Reads to Writes

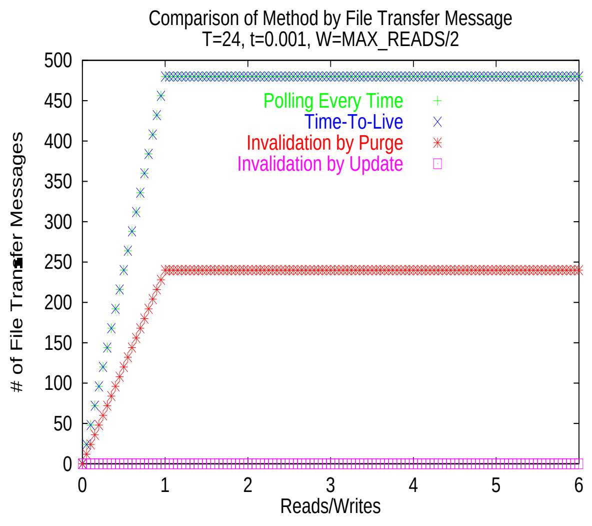

Fig. 4. Number of File Transfers based on ratio of Reads to Writes

out values. They are presented in Figures 3 and 4.

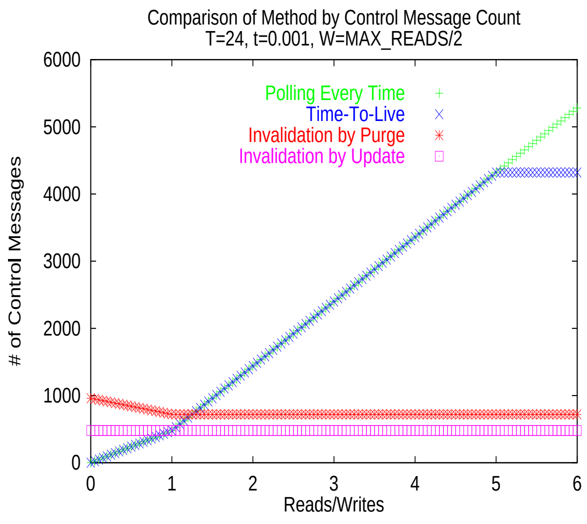

Fig. 3. Number of Control Messages based on ratio of Reads to Writes

V. Experimental Design

The experiments performed to address the objectives are given next. We present the data collected for input, details of the experiment implementation, and the results.

A. Experiment 1: Percentage of URL Changes

A.1 About the Input Data

In this experiment, we analyzed various groups of popular URL listings. Each day for a period of 7 days, we collected popular URLs from two *Hot Spot* sites, *100HOT.COM* and *PC-magazine Top* Listing. We then combined the results of the daily retrievals to produce a comprehensive listing for each site. After getting the original list of URLs (first level), we gather a list of the links at each first level URL and store in separate files (second level URLs). People who browse for documents usually access only the first page of Web sites without exploring links within the page. Therefore, access to the first level pages are expected to be larger than the links within Web sites. Each predetermined interval, the full contents of each retrieved page is saved and compared for differences. For brevity and storage concerns, we decrease the number of URLs of the second level listing, by only traversing the URLs of the first 20 first-level URLs.

We selected five classes of documents to perform this experiment: popular business, family, news, search and a combination of the top 100 sites. The first four categories are extracted from the 100HOT.COM list. The 100hot page ranks the top Web sites based on the analysis of log files from cache servers at strategic points on the Internet backbone [1]. The 100 top sites were retrieved from the final 1998 update of the PCMagazine list. These selections were made by a large group of PC Magazine editors who spend great amounts of time exploring the Web. A summary of the numbers of URLs per category is given in Table III. In addition, the average percentage of URLs that report Last-Modified-Time (LMT) and Content-Length (BYTE) are given in Table III. In the table, Level 1 and Level 2 URLs are represented by ***L1*** and ***L2*** respectively.

TABLE III
Server Log Characteristics

Category	*# URLs*		*% LMT Reported*		*% BYTE Reported*	
	L1	*L2*	*L1*	*L2*	*L1*	*L2*
Business	26	312	45	80	45	84
Family	34	200	58	58	58	62
News	34	305	46	72	37	57
Search	15	428	56	60	42	43
PCMag	100	774	30	54	34	50

A.2 Implementation Description

We measure the percentage of popular documents that change and how popularity affects the number of messages sent by three consistency mechanisms. We also compare the change results calculated from DIFF with the difference using *Last-Modified-Time* and *Content-Length*. With these measures, we report the availability of these directives. This will give us a measure of whether users would specify TTL values and if content length can be an indicator of document change. We measure the first level URLs (initial page) as well as the second level URLs (traversed links within the first page). We acquire statistics that give us the percentage of documents changed over a specified time period. This effort accounts for this by receiving the content of a group of URLs over several intervals and comparing them using the UNIX DIFF command. We determine the percentage that changed over consecutive 3, 6, 12, and 24 hour periods. A Java program was written to obtain these percentages. We save the contents of two consecutive intervals and compare them using DIFF. We also compare the last-modified-time and content-length returned in the header. Using a script, this process is repeated for several intervals over 14 days. Finally, the produced interval files were compared to determine what percent of documents change within each interval.

A.3 Results

We repeat the analysis done by Wills and Mikhailov, but using DIFF as the calculation tool rather than MD5. We collect additional information during data retrieval: Last-modified-time (LMT) and content-length (CL). These values are used to determine any correlation between them and the content differences. Table III reports the percentage of documents that specify LMT and CL. In general, the percent of pages reporting LMT and content-length is 30 to 60 percent for the first level URLs and 50 to 85 percent for the second level URLs. This shows a large gap in how documents report the directives and suggest that there is inconsistency in their usage. Due to the large number of accesses that exclude these parameters and the large gap, we conclude that LMT and CL are insufficient indicators of document change. Our results are significantly different to Wills, et al., who reported that about 85 to 90 percent of LMT and CL values were specified. [11].

The results of experiment 1 are summarized in figures 5 and 6 [1]. The figures depict trends that occur due to interval length (hours), URL type (business, family, search, news, combined PCMAG), and page level (first level, second level). We found that a large number of popular documents are modified over a very short period of time. For first level pages, PC Magazine's lists of the top 100 sites reported the largest percentage of changing documents (67 to 72 %) with Search (58 % to 60 %) and News (54 % to 60 %) trailing closely in percentages. Business and Family oriented pages have a smaller percentage of documents that change in a small interval block. This means that these URLs, although popular, change less often than the other category of URLs. Special purpose servers could benefit from knowing what consistency method to use based on the type of documents it serves.

[1] The individual graphs of results based on the 3 criteria are given in Appendix A

The percent of URL changes dropped for the second level for Business, News, and PCMag and remains the same for Family and Search sites. The decrease occurs due to the sites that change a small item, such as date or hit counters. Although these first level pages change more often than in second level pages, there is little difference (10%). Overall our results imply that when a front page of a popular site changes, other embedded links within that site also change. According to the browsing nature of users, if a page is accessed, its links will most likely be accessed. Since we know that these links also change as many times as the reference page, we should consider consistency schemes that allow additional invalidations to piggyback onto related invalidations.

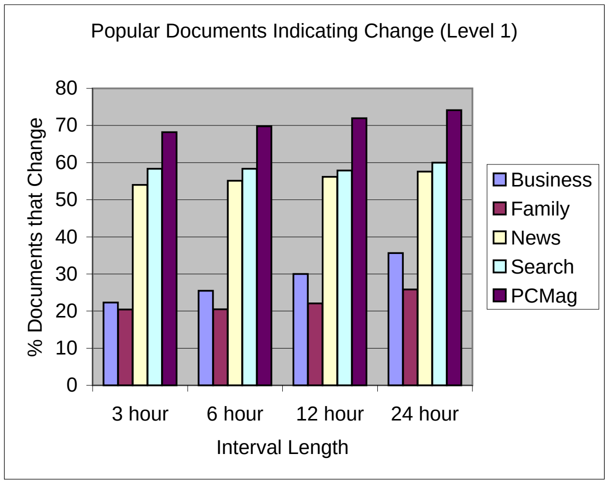

Fig. 5. Document Differences for the First Level

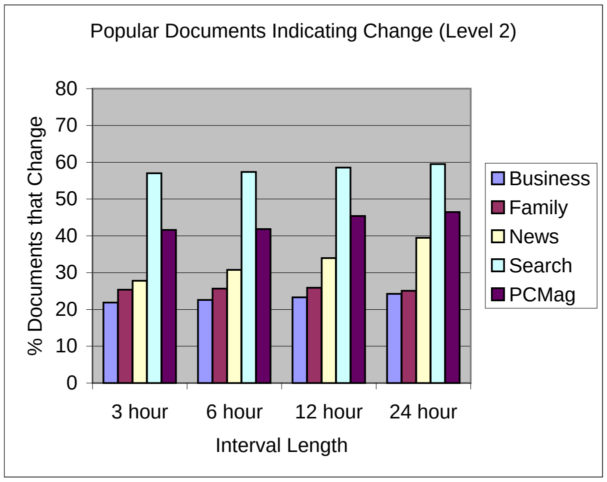

Fig. 6. Document Differences for the Second Level

B. Experiment 2: Proxy Cache Holdings per URL

B.1 About the Input Data

Two alternatives of data collection were considered for this experiment. The first, using data collected by a traffic generator, takes too long and may not properly simulate the real world.

Alternatively, log files give us the necessary access information about each URL in a file such as the number of accesses. We use three days of server log files from Virginia Tech's Computer Science server, *www.cs.vt.edu*. These logs, containing daily collections, include accesses from August 1, 1999 to August 3, 1999. In this paper, we will refer to these logs as VT-CS1, VT-CS2, VT-CS3 to represent the day that the log covers. In addition to the VT-CS logs, we use two weeks of log files from several James Cook University servers in Australia. These files, containing weekly collections, represent servers in the Engineering department (JCU-ENG) and the university library (JCU-LIB). These logs cover the first two weeks in August, and therefore JC-LIB1, and JC-LIB2 represent week 1 and 2 respectively.

B.2 Implementation Description

We study a group of servers to reveal percentage of proxies that access documents at a server. This tells us how many Web sites should be notified of changes in a document maintained by the server (for invalidation). We also measure the number of references made to each server document by each proxy. We calculate the number of proxies that access a certain set of server documents. This is done by attempting to access the IP addresses that occur in a server log entry, and determining if the server at the IP address is a proxy. A program was written to determine how many sites require notification. For instance, we can determine how many URLs were cached by $proxy1, proxy2, ...proxyN$. We examine the server log, and get the IP addresses. Then we attempt to open a socket on port 80, the well-known HTTP port. If we cannot connect, we have not accessed a proxy (a client requested the document directly). Otherwise, it is assumed to be a proxy if it is running a Web server and sending requests.

B.3 Results

Table IV gives the percentage of references to a server. It summarizes the percentage of unique references and accesses to the URLs by three client types. The unique count is the fraction of distinct first time references to the server. The accesses, total duplicate references, give the fraction of accesses to the server including multiple accesses to the same document and multiple accesses by a proxy to several different documents. In addition, the average percentage and standard deviation is given in the table. The data reveals that an average of only 11.20% of the unique accesses and 14.30% of the total accesses come from proxy clients. Therefore, a large percentage of the accesses to servers are done through non-proxy clients.

Table V gives the number of proxy, non-proxy, and unknown clients accessing the server. These numbers do not include the number of references the clients make to the server for unique documents. In other words, the table only represents the number of machines that access the server. This table also gives the number of URLs accessed on the server by proxy, non-proxy, and unknown clients. This data only reflects the actual numbers used to calculate the percentage in table IV.

The experiment also consisted of calculating the number of proxies that access each URL in the server log file. About 92 percent of URLs were accessed by only one proxy. This shows that not many proxies have to be contacted. The actual data is given in Table VI.

TABLE IV
PERCENT UNIQUE AND TOTAL REFERENCES BY PROXY, NON-PROXY (NON-P), AND UNKNOWN CLIENTS (X)

	% Unique 1st References By			% Accesses By		
	Proxy	Non-P	X	Proxy	Non-P	X
VT CS1	8.86	87.61	3.53	8.98	87.51	3.51
VT CS2	13.36	81.07	5.56	12.94	81.56	5.50
VT CS3	12.07	84.01	3.92	12.08	83.87	4.05
JCU Lib1	8.04	91.96	0.00	14.75	85.25	0.00
JCU Lib2	9.33	90.67	0.00	14.03	85.97	0.00
JCU Eng	15.55	84.45	0.00	23.04	76.96	0.00
Avg	11.20	86.63	2.17	14.30	83.52	2.18
Std Dev	0.03	0.04	0.03	0.05	0.04	0.03

TABLE V
# UNIQUE AND TOTAL REFERENCES BY PROXY, NON-PROXY (NON-P), AND UNKNOWN CLIENTS (X)

Trace Trace	# Unique 1st References by			# Total Accesses by		
	Proxy	Non-P	X	Proxy	Non-P	X
VT-CS1	1084	10717	432	1273	12404	498
VT-CS2	3048	18490	1269	3610	22753	1534
VT-CS3	2879	20031	934	3413	23694	1144
JCU-Lib1	2473	27157	0	8534	49329	0
JCU-Lib2	2941	28573	0	8599	52679	0
JCU-Eng	4035	21916	0	11779	39349	0

We also measured the number of proxies that accessed 1 URL, 2 URLs, 3 URLs, and so on. These values give an idea of the number of URLs, owned by this server, that a particular proxy is holding. The results show that between 20 and 30 percent of proxies access a single URL on the server. Therefore, 70 to 80 percent of the proxies that access the Virginia Tech and James Cook servers request multiple URLs from the server. There would clearly be an advantage to using schemes where several invalidations are sent in batches to proxies holding multiple documents owned by servers. Table VII shows the data collected to support these statistics.

Finally, we measured the number of duplicate URLs that proxies accessed. For instance, how many proxies access a document only once, or twice. This gives an upper bound on the number of URLs that were accessed multiple times by the same proxy. We found that a small percentage of proxies accessed a document more than once (between 3 and 18 %). This

TABLE VI

URLs Held by Proxies

# Proxies Holding URLs	# URLs Requested at Server VT-CS1	VT-CS2	VT-CS3	JCU-Lib1	JCU-Lib2	JCU-Eng
1_proxy	317	550	597	218	97	521
2_proxy	19	26	50	16	16	36
3_proxy	2		10	1		5
4_proxy				1		
5_proxy						2
Total	338	576	657	236	113	564

TABLE VII

Count of Proxies Holding Multiple Different URLs

# of URLs Accessed URLs	# of Proxies Requesting From Server VT-CS1	VT-CS2	VT-CS3	JCU-Lib1	JCU-Lib2	JCU-Eng
1_url	67	101	125	53	52	34
2_url	18	31	43	16	24	31
3_url	16	18	31	10	11	127
4_url	28	40	41	11	9	16
5_url	9	22	16	14	15	16
6_url	6	9	8	6	9	3
7_url	8	16	22	17	12	6
8_url	7	5	8	6	3	4
9_url	4	5	6	4	3	4
10_url		4	8	1	2	11
>10_url	31	74	71	57	62	67
Total	194	325	379	195	202	319

is impacted by the existing consistency mechanism used with HTTP1.1.

Table IX provides a summary of the characteristics of proxy accesses to the servers in the workload.

VI. Summary of Results

To summarize, the analytical analysis and experiments revealed the following results.

- Based on the ratio of reads to writes, invalidation performs better than the other strong consistency algorithm, Polling-Every-Read, when there are more reads than writes performed on a document.
- Although first level pages change more often than second level pages, there is little difference. This implies that other pages at the site will most likely be modified. This is an argument for PiggyBack server invalidation where the user sends updates of a group of related documents when invalidating a document.
- Servers administrators can benefit from the knowledge that if given a certain type of document, a given mechanism works best. News and Search Engine documents change more frequently than other categories, but they also require the most consistency. Invalidation would work well for these documents that are known to change often. Business and Family oriented pages would benefit from a weaker consistency mechanism like client polling or adaptive time-to-live.
- The majority of the documents at servers are only accessed by one proxy. In addition, these proxies accessed documents only once. Therefore, invalidation will not consume any more bandwidth than the other mechanisms when used for popular documents, especially since the number of proxies to contact does not impact the network.

TABLE VIII

Count of Proxies Requesting Duplicates from the Server

# Duplicate URLs Requested	# Proxies Requesting From VT-CS1	VT-CS2	VT-CS3	JCU-Lib1	JCU-Lib2	JCU-Eng
1	991	2739	2581	1879	2178	2661
2	68	219	211	279	394	593
3	11	37	52	115	110	227
4	4	21	10	34	67	147
5	1	6	6	21	29	83
6	1	10	5	17	12	50
7	2	5	6	14	18	32
8	1	5	2	6	17	33
9	2	2		2	5	24
10		1		10	8	26
>10	3	3	6	96	103	159
Total	1084	3048	2879	2473	2941	4035

TABLE IX

Percentage of references to Proxies

Scenario URLs	# of URLs Requested at Server VT-CS1	VT-CS2	VT-CS3	JCU-Lib1	JCU-Lib2	JCU-Eng
% proxies accessing the server	13	13	15	12	13	15
% unique URL accessed by proxies	9	9	10	6	3	18
% total references by proxy (dups)	11	13	12	15	14	23
% proxy references to multiple unique documents	65	68	67	72	74	89
% accesses to previously requested documents	9	10	10	24	26	34

VII. Conclusions

We have investigated some characteristics of Web documents that affect how consistency should be performed in the Web. The results show that a high percentage of popular documents change over a small interval of time. Specifically, 50 to 70 percent of the most popular documents, News and Search sites, change within a three hour period. This rate of modification is less than the rate of access reported in the literature. Using this knowledge, we propose that strong consistency mechanisms can be used for popular documents. Client polling will bombard the server with unnecessary requests if no changes occur between one and three hours. TTL will require that the document creator have a priori knowledge of when a document changes which is not always possible. Invalidation is shown to be most useful of the three general consistency methods proposed in the literature, but how effective is it?

We observed that of the 11 to 20 % proxies that did access the servers, many did not access the servers more than once for the same documents (about 15%). Also, 90 percent of URLs were accessed by only one proxy. In addition, proxies do not generally access documents more than twice, but they do access many different documents at one server. Based on these statistics and the low number of proxies that need to be notified, server-based invalidation will minimally impact the bandwidth used due to consistency. Therefore, server invalidation is recommended for popular documents that change frequently and infrequently. For documents that are accessed infrequently, client polling or TTL would work best. Table X, shown in Appendix B, summarizes the conclusions.

Although we recommend the updating of very popular and frequently changing Web documents using invalidation, we believe our study would have benefited from an analysis of the nature of the document changes. How much of a change constitutes the need for a fresh document? Were there small changes like a date or spelling error? Or were there huge paragraphs being rewritten? These types of decisions can help us to determine when the stale copies are actually acceptable. In addition to the basic algorithms we discuss in this paper, there are many variants of invalidation that can be investigated such as Piggyback Server Validation [9], Invalidation with Delta Encoding [5], Volume lease Invalidation [13]. Our future work involves combining the best elements of existing consistency mechanisms to allow proxies and servers to adapt to conditions of the Web and decide which mechanism to invoke.

References

[1] 100hot.com. <URL: http:www.hot.com>.

[2] P. Cao and Chengjie Liu. Maintaining strong cache consistency in the world wide web. In *Proceedings of ICDCS'97*, pages 12–21, May 1997.

[3] V. Cate. Alex - a global file system. In *Proceedings of the 1992 USENIX File System Workshop*, pages 1–12, May 1992.

[4] B. den Boer and A. Bosselaers. Collisions for the compression function of md5. In *Advances in Cryptology, Proceedings Eurocrypt '93*, pages 293–304, 1994.

[5] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. Mogul. Rate of change and other metrics: A live study of the world wide web. *USENIX Symposium on Internetworking Technologies and Systems*, December 1997.

[6] C. Gray and D. Cheriton. Leases: An efficient fault-tolerant mechanism for distributed file cache consistency. In *In Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pages 202–210, 1989.

[7] J. Gwertzman and M. Seltzer. World-wide web cache consistency. *USENIX Symposium on Internetworking Technologies and Systems*, pages 141–152, January 1996.

[8] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 2nd edition, 1990.

[9] B. Krishnamurthy and C. E. Wills. Piggyback server invalidation for proxy cache coherency. In *In Seventh International World Wide Web Conference*, volume 30, pages 185–193, April 1998.

[10] Ari Luotonen. *Web Proxy Servers*. Prentice-Hall, London, 1998.

[11] C. E. Wills and Mikhail Mikhailov. Toward a better understanding of web resources and server responses for improved caching. *In 8th International World-wide Web Conference*, 1999. <URL:http://www.cs.wpi.edu/~mikhail/papers/www8.ps.gz>.

[12] K. Worrell. Invalidation in large scale network object cache. Master's thesis, University of Colorado, Boulder, 1994.

[13] J. Yin, L. Alvisi, M. Dahlin, and C. Lin. Volume leases for consistency in large-scale systems. *IEEE Transaction on Knowledge and Data Engineering*, 1999.

APPENDIX

I. ADDITIONAL FIGURES

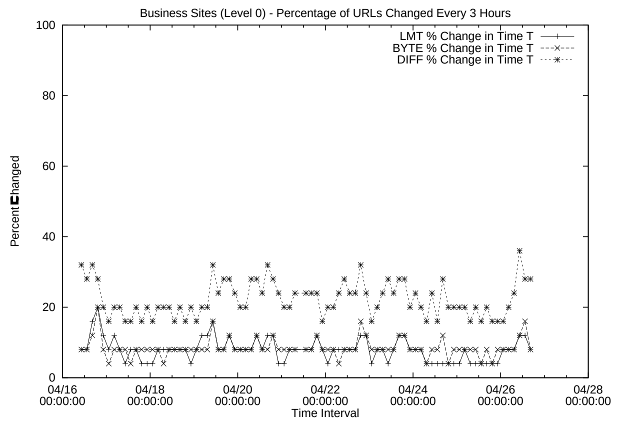

Fig. 7. 1st Level Business URLs Compared Every 3 Hours for Differences

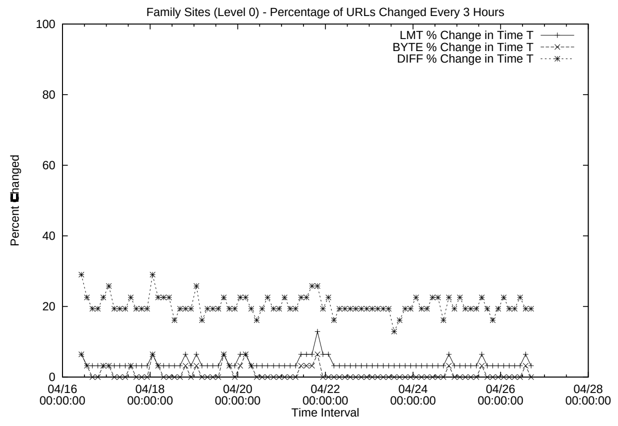

Fig. 8. 1st Level Family URLs Compared Every 3 Hours for Differences

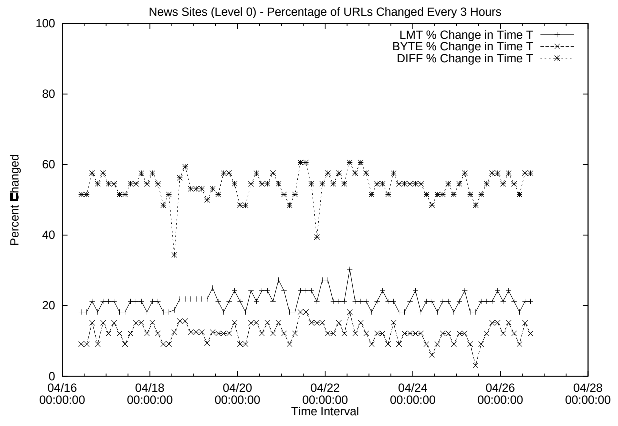

Fig. 9. 1st Level News URLs Compared Every 3 Hours for Differences

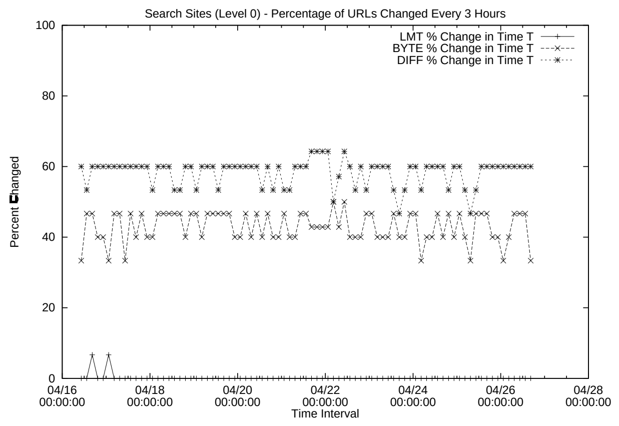

Fig. 10. 1st Level Search URLs Compared Every 3 Hours for Differences

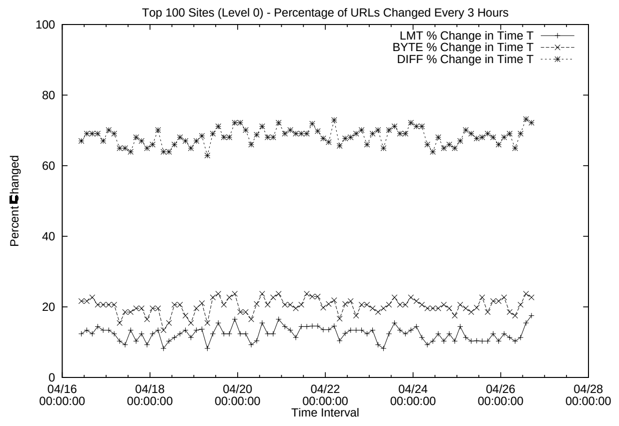

Fig. 11. 1st Level PCMAG URLs Compared Every 3 Hours for Differences

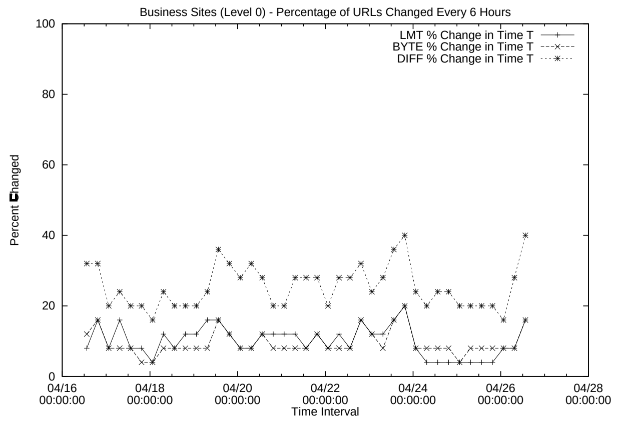

Fig. 12. 1st Level Business URLs Compared Every 6 Hours for Differences

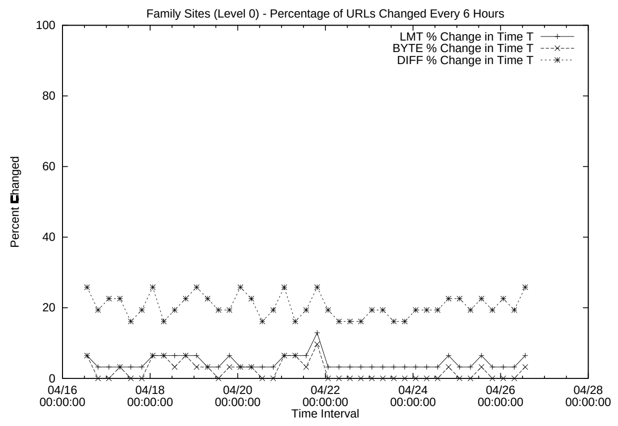

Fig. 13. 1st Level Family URLs Compared Every 6 Hours for Differences

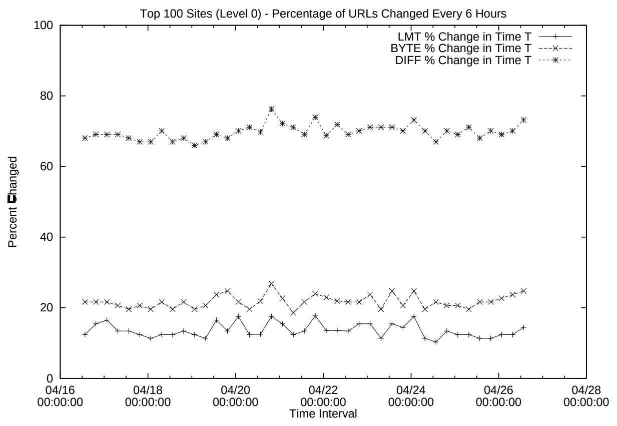

Fig. 16. 1st Level PCMAG URLs Compared Every 6 Hours for Differences

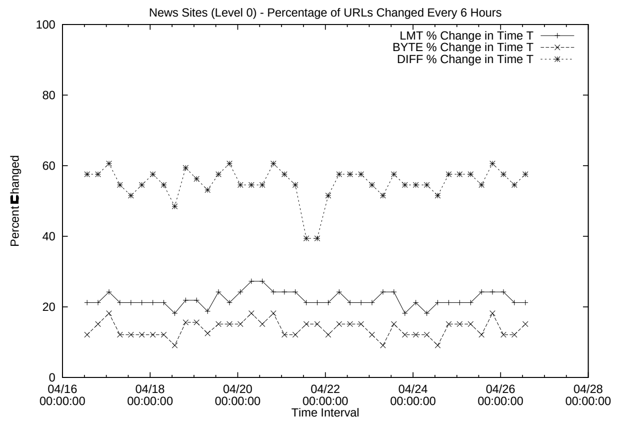

Fig. 14. 1st Level News URLs Compared Every 6 Hours for Differences

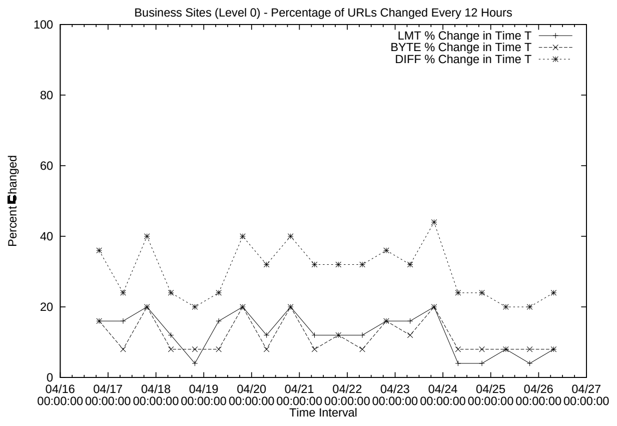

Fig. 17. 1st Level Business URLs Compared Every 12 Hours for Differences

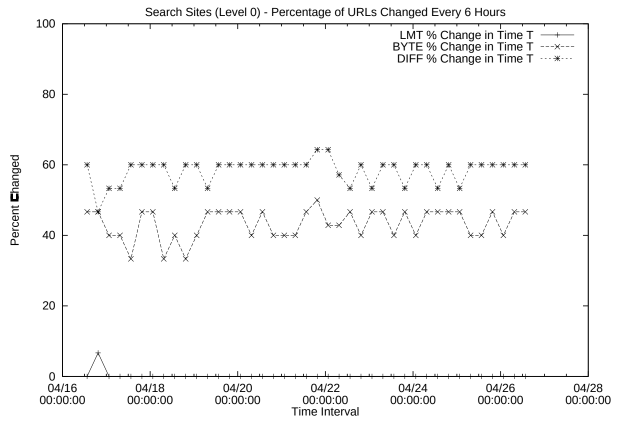

Fig. 15. 1st Level Search URLs Compared Every 6 Hours for Differences

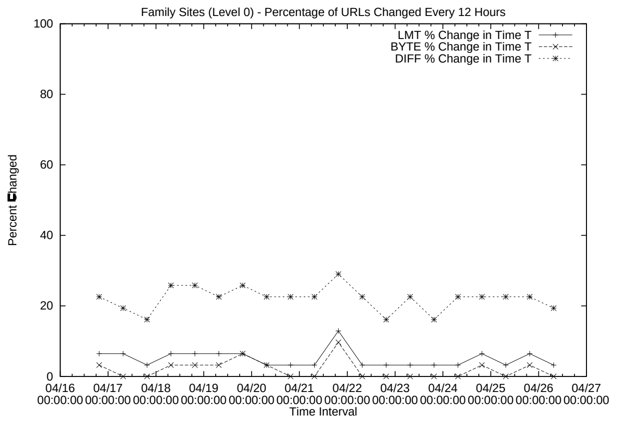

Fig. 18. 1st Level Family URLs Compared Every 12 Hours for Differences

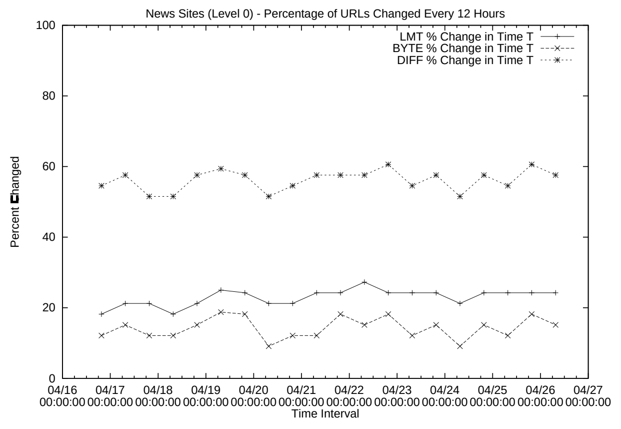

Fig. 19. 1st Level News URLs Compared Every 12 Hours for Differences

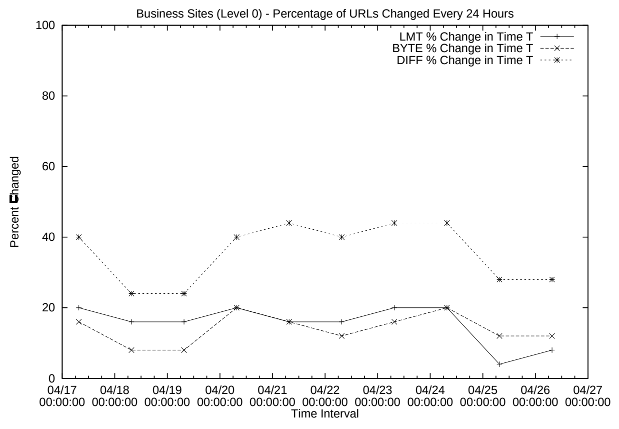

Fig. 22. 1st Level Business URLs Compared Every 12 Hours for Differences

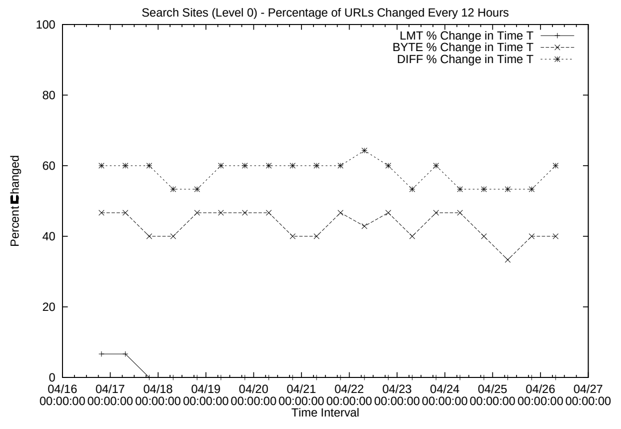

Fig. 20. 1st Level Search URLs Compared Every 12 Hours for Differences

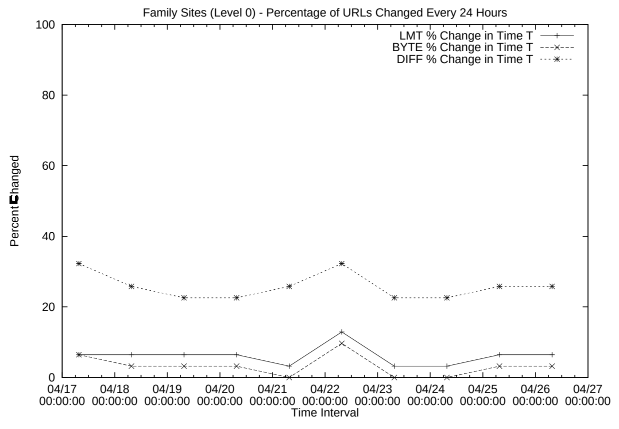

Fig. 23. 1st Level Family URLs Compared Every 24 Hours for Differences

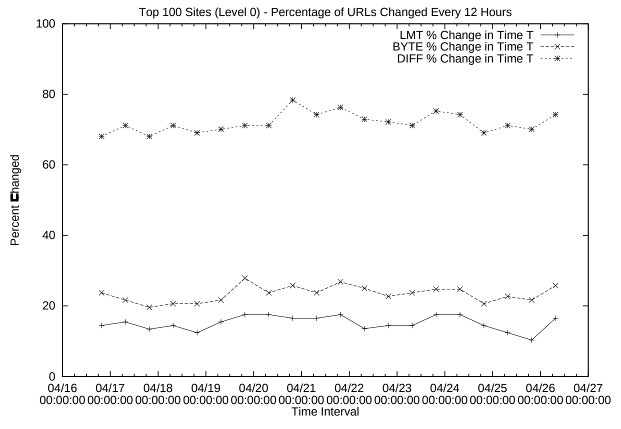

Fig. 21. 1st Level PCMag URLs Compared Every 12 Hours for Differences

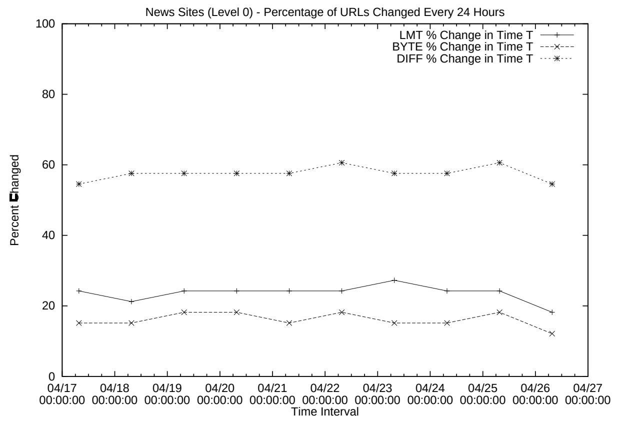

Fig. 24. 1st Level News URLs Compared Every 24 Hours for Differences

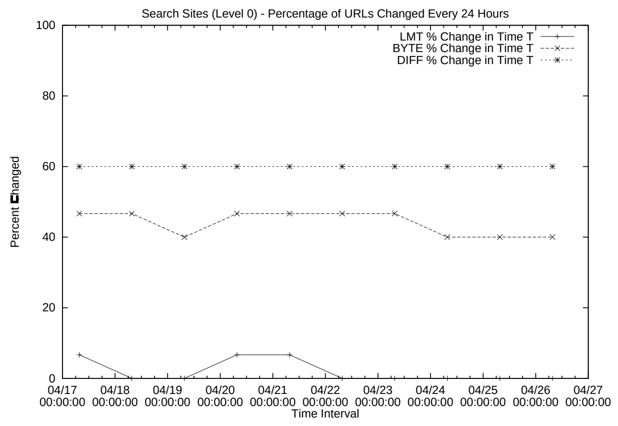

Fig. 25. 1st Level Search URLs Compared Every 24 Hours for Differences

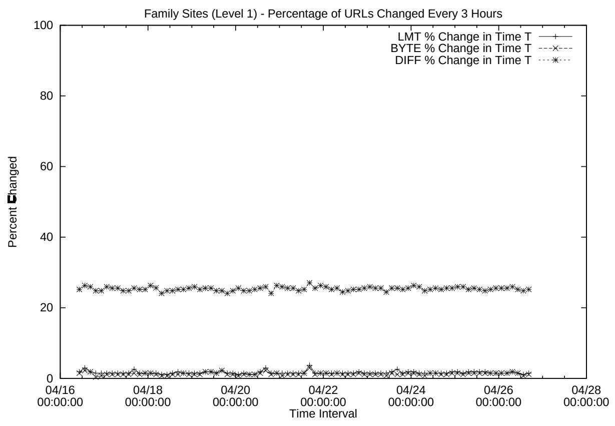

Fig. 28. 2nd Level Family URLs Compared Every 3 Hours for Differences

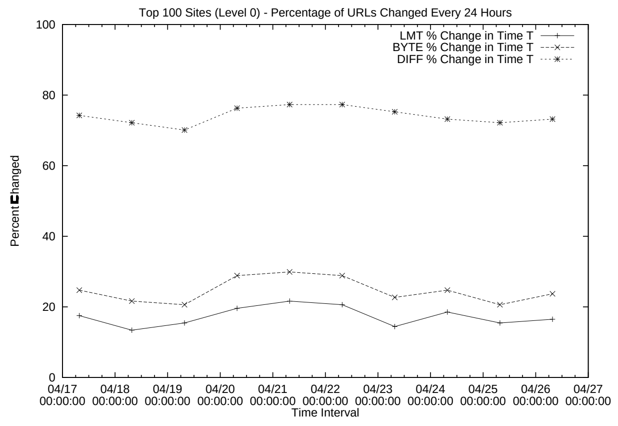

Fig. 26. 1st Level PCMag URLs Compared Every 24 Hours for Differences

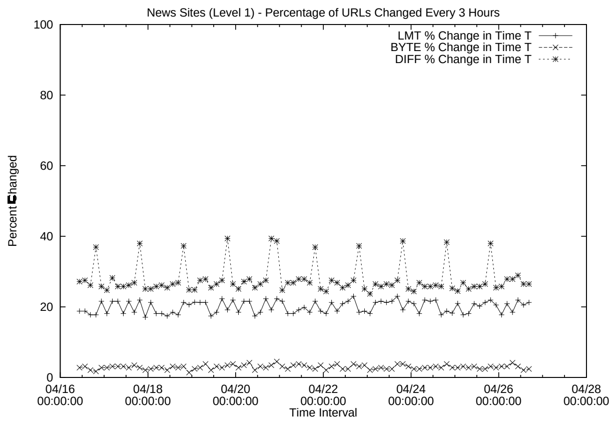

Fig. 29. 2nd Level News URLs Compared Every 3 Hours for Differences

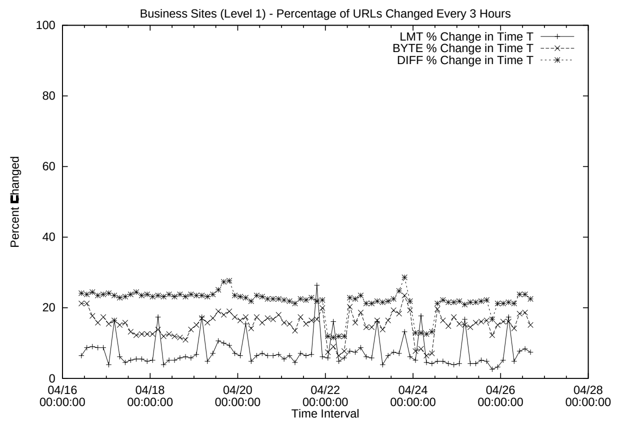

Fig. 27. 2nd Level Business URLs Compared Every 3 Hours for Differences

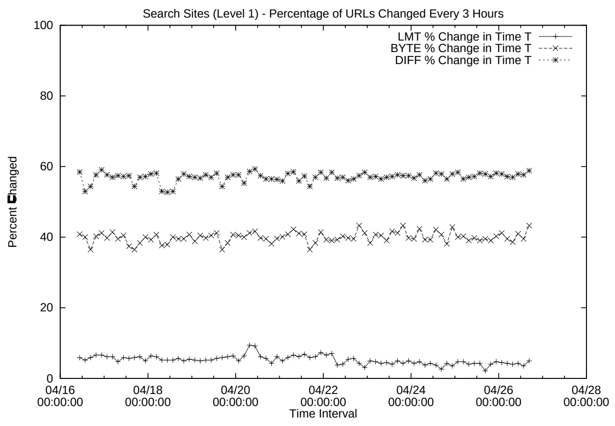

Fig. 30. 2nd Level Search URLs Compared Every 3 Hours for Differences

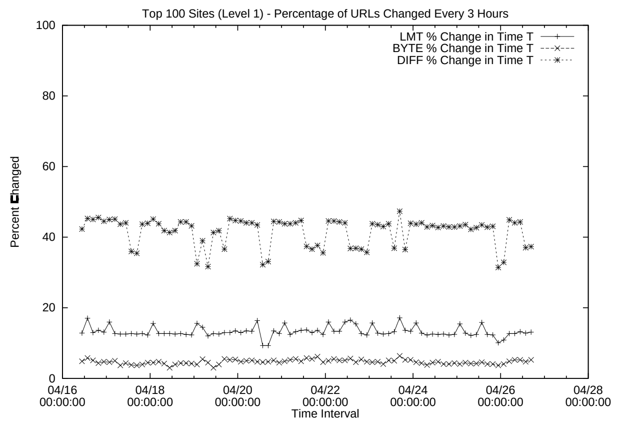

Fig. 31. 2nd Level PCMag URLs Compared Every 3 Hours for Differences

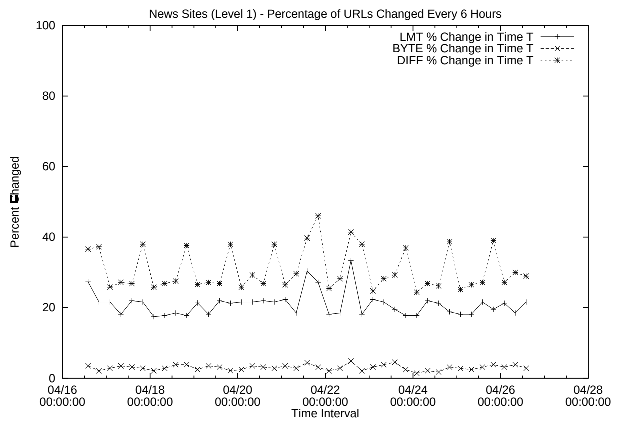

Fig. 34. 2nd Level News URLs Compared Every 6 Hours for Differences

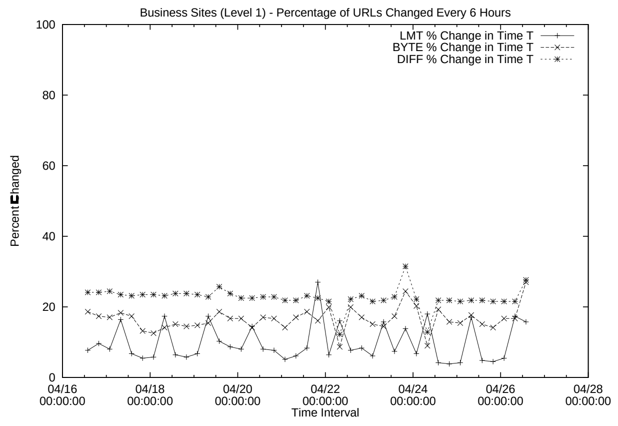

Fig. 32. 2nd Level Business URLs Compared Every 6 Hours for Differences

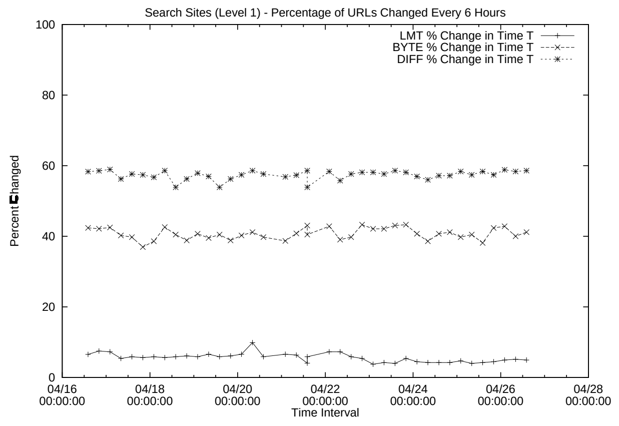

Fig. 35. 2nd Level Search URLs Compared Every 6 Hours for Differences

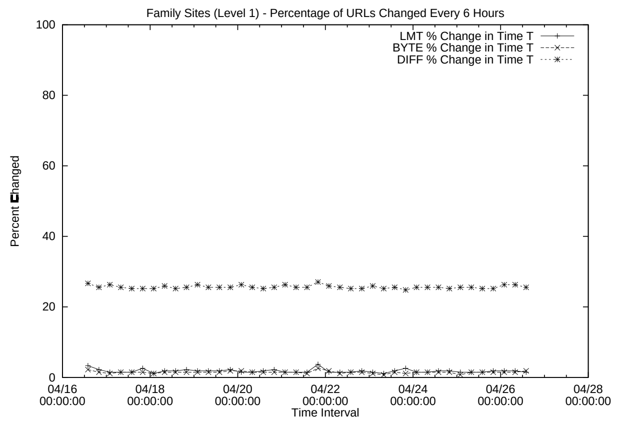

Fig. 33. 2nd Level Family URLs Compared Every 6 Hours for Differences

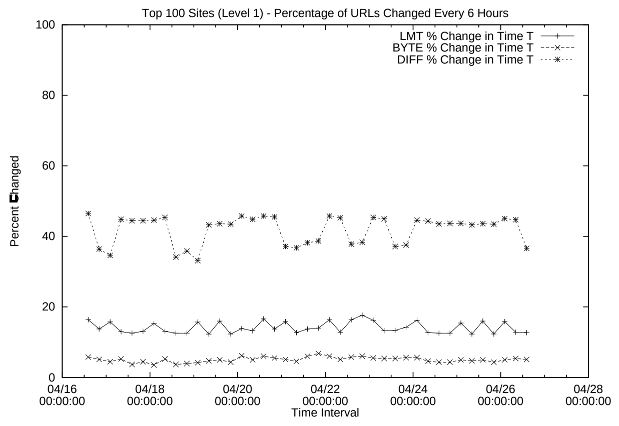

Fig. 36. 2nd Level PCMag URLs Compared Every 6 Hours for Differences

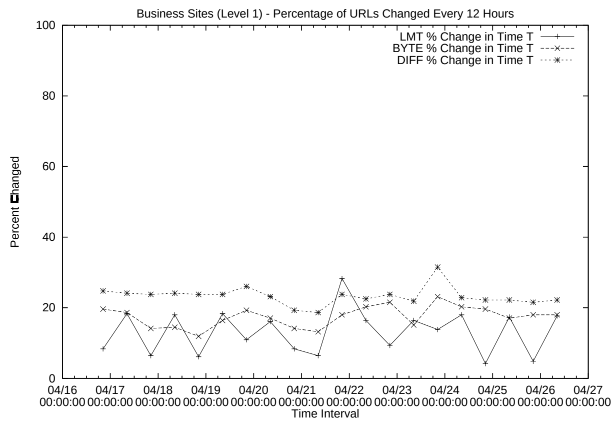

Fig. 37. 2nd Level Business URLs Compared Every 12 Hours for Differences

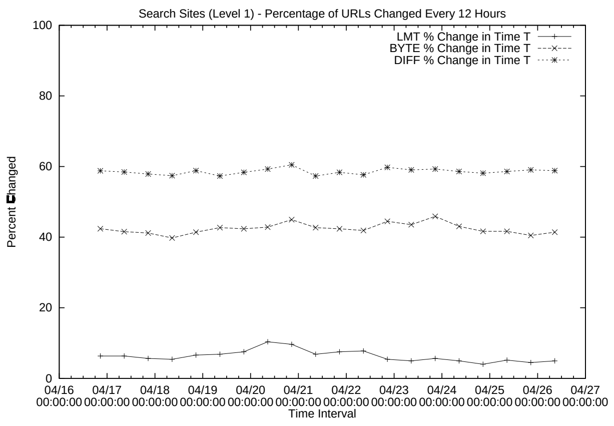

Fig. 40. 2nd Level Search URLs Compared Every 12 Hours for Differences

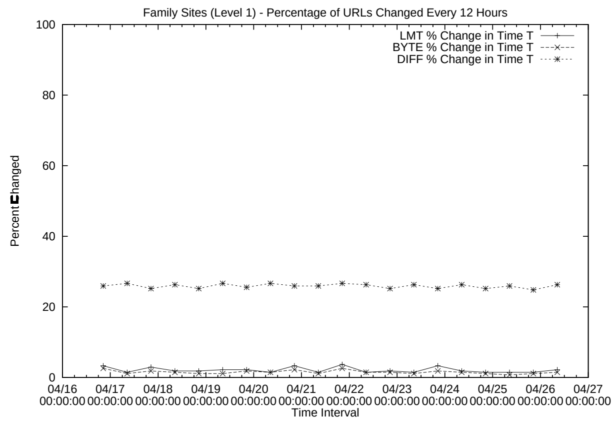

Fig. 38. 2nd Level Family URLs Compared Every 12 Hours for Differences

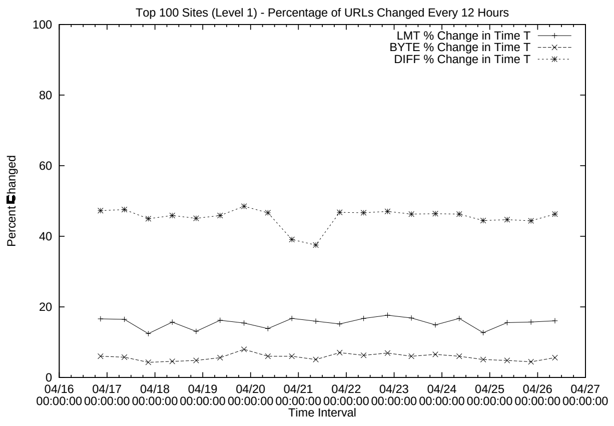

Fig. 41. 2nd Level PCMag URLs Compared Every 12 Hours for Differences

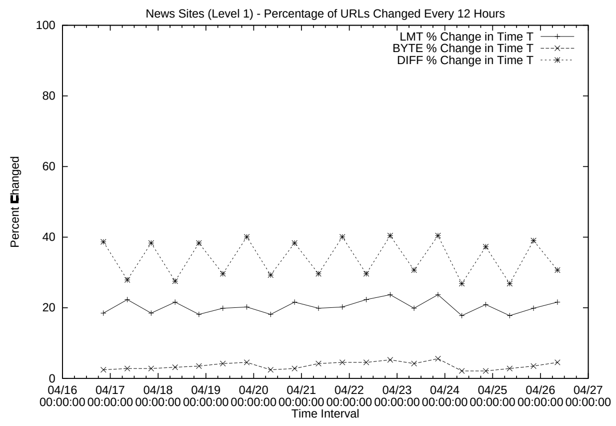

Fig. 39. 2nd Level News URLs Compared Every 12 Hours for Differences

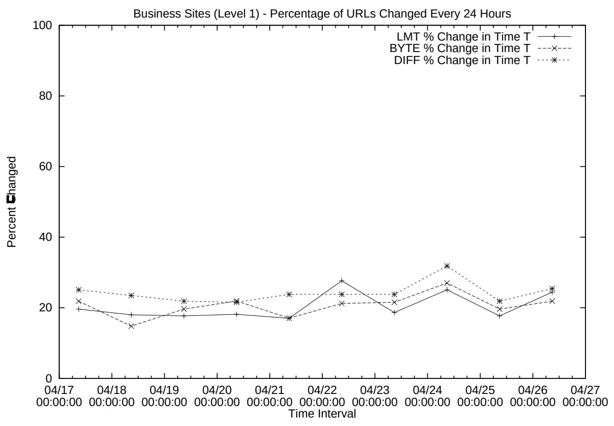

Fig. 42. 2nd Level Business URLs Compared Every 24 Hours for Differences

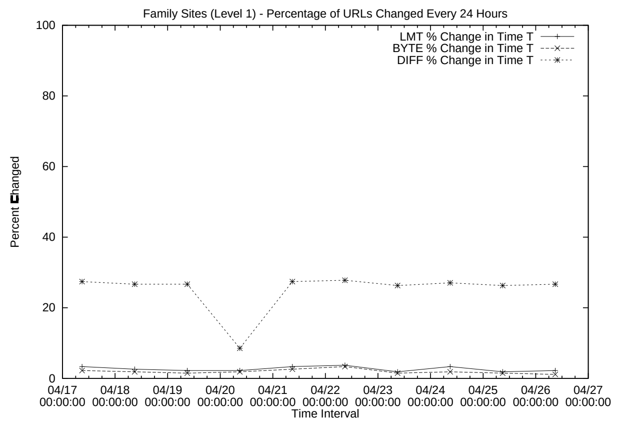

Fig. 43. 2nd Level Family URLs Compared Every 24 Hours for Differences

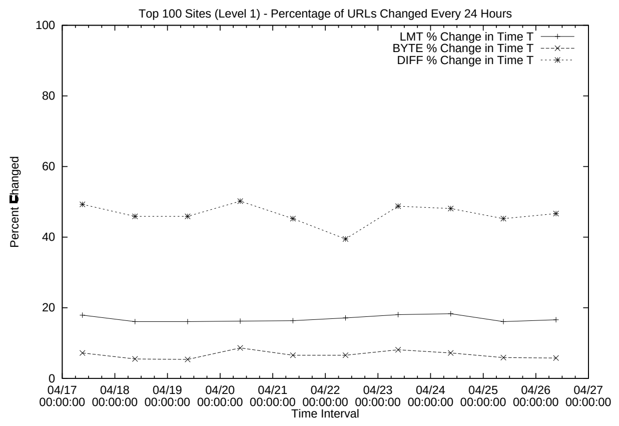

Fig. 46. 2nd Level PCMag URLs Compared Every 24 Hours for Differences

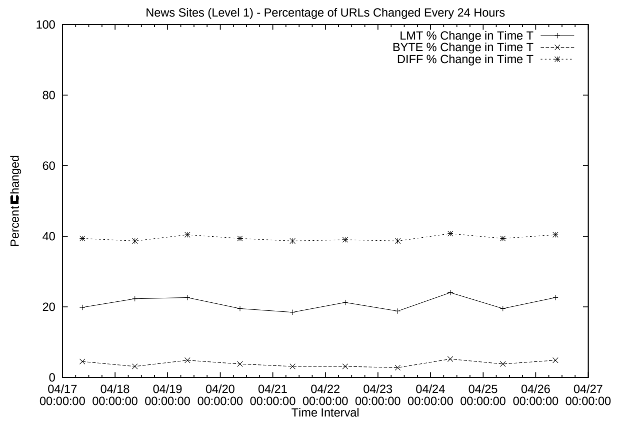

Fig. 44. 2nd Level News URLs Compared Every 24 Hours for Differences

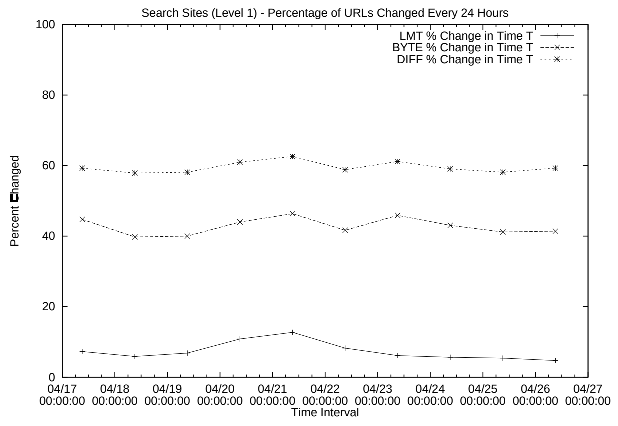

Fig. 45. 2nd Level Search URLs Compared Every 24 Hours for Differences

II. Summary of Conclusions

TABLE X
Selection of a Consistency Approach

Criterion	*Suggested Consistency Method*		*Explanation*
	Strong	*Weak*	
A priori knowledge of lifetime	Invalidation	TTL	This assumes that the proxy has prior knowledge of exactly when a document changes. This is usually impossible even with documents that have a set time to change. However, it would be best to use TTL unless strong consistency is required
Popular documents that are modified frequently	Invalidation	Client Polling	The number of requests (R) is at least equal to the number of modifications (M). According to Douglis, et al. and our results, the rate of requests are higher than the rate of modification. This implies that many more reads are performed in comparison to writes. In addition, an average of one proxy needs to be contacted for invalidation. Using our results and those in the literature, we conclude that invalidations produce less messages than the strong consistency mechanism, Polling Every Time. Client Polling decreases the need to poll the server at the expense of acquiring possibly stale documents.
Popular documents that are modified infrequently	Invalidation	Client Polling	The number of requests (R) and the number of modifications (M) are greater than 1, however $R > M$. According to our analysis, Business and Family sites fall in this category. Invalidation would be the best algorithm in general.
Unpopular documents that are modified frequently	Polling Every Time	Client Polling	The number of requests (R) is less than the number of modifications (M). This scenario implies that there are many more writes performed than reads. If documents are not frequently requested, then there are less IMS and 304 messages generated. Polling the server gives fewer messages while invalidations would produce unnecessary messages. Since it is highly probable that most of the writes will occur in succession with no intercepting reads, w may not have to poll the server as often as required by Polling-Every-Read
Unpopular documents that are modified infrequently	Polling Every Time	Client Polling	The number of requests (R) and number of modifications (M) are small and equally likely. This scenario implies that the documents are not requested very often and hardly ever change. These are usually old documents hanging around on the Web. The current Web mechanism, Client Polling would work best.