xProxy: A Transparent Caching and Delta Transfer System for Web Objects

Matthew Delco and Mihut Ionescu
Computer Science Division
University of California, Berkeley
{delco,mihut}@cs.berkeley.edu

Abstract

The demand for bandwidth in computer networks has been growing at a steady pace for the last several years, and a significant portion of that traffic is for the World Wide Web. At the same time, a large fraction of Internet users continue to rely on slow "last mile" links (such as modems and wireless links). We present a transparent proxy system that uses deltas and compression to increase the performance of web traffic over slow links. By using compression, the average transfer time for HTML objects can be reduced by a factor of 2.3, while using deltas can reduce the transfer time by a factor of 6.0. The techniques used by this system, if utilized by other proxies and/or web servers, could also be used to reduce the amount of web traffic that traverses a network.

1 Introduction

During the last few years the amount of traffic on the Internet has been increasing at a rapid pace. While many people have high-speed access to the Internet through their employer or university, a large fraction of users are still reliant on slow modems for their Internet access. Web caches have been placed in certain networks to help reduce network traffic, but the usefulness of these proxies is hindered by the increasing use of "uncacheable" dynamic content on web servers. To assist end-clients and proxies in determining if a locally cached copy is still up-to-date, the HTTP protocol was extended to allow for "conditional requests" that instruct web servers to transmit an object only if it has changed since a certain date.

HTTP's conditional request mechanism is useful, but this "all-or-nothing" approach ignores the fact that the current version of an object has a significant amount of content in common with prior versions. As a result, a more dynamic approach that uses deltas to transmit only the modifications to an object should have a useful benefit. In this paper we describe the xProxy system that can take advantage of deltas and significantly improve the download performance of web objects over a slow link. In those situations where a delta cannot be used (e.g., the first time that a particular page is visited), using compression is a very viable and worthwhile alternative. The xProxy system is transparent to both web browsers and web servers, and can therefore be used in any network. If the techniques used by xProxy are incorporated into future web servers and clients, then a significant amount of bandwidth savings could be realized in the Internet.

In the next section we describe the architecture of xProxy and provide details on our implementation. Section 3 provides results on the performance improvements we obtained by using the system, and Section 4 describes related work. Section 4 also discusses some of the prior research that pertains to the average characteristics on how people "surf" the web—in this paper we make no claims on how people access the web. Instead, we operate under the assumption (substantiated by these research papers) that a reasonable amount of pages are visited repeatedly. For example, some sites are revisited due to personal preference (e.g., a news site that has relatively few advertisements), while other sites are visited because of necessity (e.g., "home banking" information for a checking account is only available on a particular bank's web site).

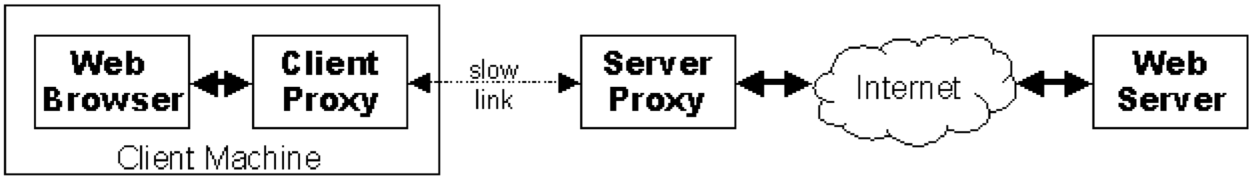

Figure 1: Diagram of the xProxy architecture; this diagram is a simplified representation and does not convey that a Server Proxy can serve multiple Client Proxies or that the Server Proxy need not be located near the Client Proxy.

2 Implementation

2.1 Overview

In order to allow deltas and compression to be used within the existing web infrastructure, we designed the xProxy system to be transparent to both web browsers and web servers. The overall system architecture for this design is depicted above in Figure 1. xProxy consists of two web caching proxies, where each proxy is placed on either side of a slow link. The "Client Proxy" (CP) is essentially a daemon that resides on the end-user's machine, while the "Server Proxy" (SP) can be placed anywhere in the network that has a reasonably high speed connection to the Internet (e.g., on a 100 Mbps subnet that is near the modem bank of the client's ISP). The CP and SP both operate as normal HTTP traffic proxies, except that they have been enhanced to utilize deltas and compression. The CP is intended to serve only the end-client's machine, while the SP could potentially service a large set of users.

2.2 Client Proxy (CP)

To utilize the xProxy system, a web browser must be configured to use the CP as its proxy. Since all common web browsers already support the use of proxies, this configuration change is trivial. In the case of a novice user, a software program (or one of the many automatic proxy detection protocols) could be used to automate this change. By specifying the CP as a proxy, all web requests from the web browser will be routed through the CP. This change allows the CP to determine whether it can satisfy a request without using the slow network connection by utilizing a locally stored copy of the resource as a substitute. If there is not a locally stored copy, or the local copy is not suitable for reuse (as discussed below), then the CP will use the slow network connection to forward the client's request to the SP. Once the CP has received a reply from the SP, it caches a copy of the reply and, if necessary, takes action to produce a conventional response that is suitable for use by the web browser.

2.3 Server Proxy (SP)

The Server Proxy operates in a similar manner as the Client Proxy. In fact, the primary difference between a SP and a CP is that the SP will forward web requests to the originating servers, while the CP will forward all of its requests to a proxy. Like the CP, the SP will first try to satisfy requests using a local copy of the resource. A local copy is only used if the originating server specified, via HTTP caching primitives, that the cached copy is "fresh" (i.e., the object may be used to service other requests for a server-specified length of time). If the proxy has a local copy that can not be considered fresh (i.e., the copy is "stale"), then a unique MD5 identifier of that page is included in the outgoing request in the hope that the recipient (or any transparent proxies residing on the path to the recipient) can use the identifier to create a "delta" response that only contains the differences between the current and locally stored versions of the object. If a delta response is received, then the proxy will use that information to reconstruct the full version of the resource, which is then processed like any conventional response.

When preparing a response to a request, the proxy will create a delta response if it received a MD5 page identifier in the request and a copy of that page is stored locally on the proxy. In cases where a delta response cannot be used for a textual object, the object is compressed before it is transmitted to the requester. Objects that are not text (e.g., images) are not compressed because they are typically represented in a format that is natively compressed. Our system can compute deltas for any file type (e.g., images, other binary files), except that deltas might not be worthwhile for such file types (see below for more details).

2.4 Example of System Execution

We will now consider an example scenario where a web browser makes a request for an HTML page and an image. After the CP receives the two requests, it finds that neither object is stored locally and forwards the requests to the SP. The SP also does not have a local copy of the objects, so it forwards the requests to the originating server. After receiving the replies from the server, the SP stores local copies of the objects, compresses the response for the HTML object, and forwards both replies to the CP over the slow network link. The CP decompresses the HTML object, stores a local copy of both objects, and then forwards a copy of the objects to the web browser.

Supposing that the image was specified by the server as cacheable and the HTML page was specified as not being cacheable, then when the user returns to the page in the near future the request for the image will be immediately serviced by the CP, while the HTML object will make another round-trip to the originating server. However, since the SP and CP now both share a common version of that document, a delta, instead of a compressed file, can be transferred between the two proxies.

2.5 Implementation Features

Our implementation of xProxy contains a large number of features and unique capabilities. Although some of these features are present in other related projects, in most cases those projects only offered a small subset of these features (typically only one).

XDFS

Our implementation utilized an application level file API called XDFS ("xDelta File System") [Mac00] for caching local copies of web objects and computing deltas in the proxies. XDFS is a key component of xProxy that provides the ability to store multiple versions of an object and to store those versions efficiently on disk using deltas. More importantly, XDFS can also compute in linear time a delta between any two arbitrary versions of an object. XDFS uses the xDelta algorithm [Mac98] for computing deltas and stores information on disk using the Berkeley DB [OBS99]. Internally, XDFS stores the various versions of a file by using "forward deltas" from an older base version to the more recent versions of a file. If the size of the most recent delta exceeds a certain threshold, then XDFS will automatically create a new base object against which future deltas will be computed. XDFS also computes an MD5 signature for each version of an object, and as a result we choose to use MD5 as our mechanism for allowing a CP to report its base version to an SP for delta computation.

Since XDFS stores the various instances of a file using deltas, there are certain cases where XDFS does not need to compute a delta "on the fly" and can instead use the delta that is already stored on disk—the most obvious example is where a fresh version of an object is cached locally as a delta against a version which the client happens to possess. The use of a database provided us with some useful features, such as the buffer manager's automatic management of swapping data to and from disk. In order to increase the performance of xProxy, we modified the transactional behavior so that modifications and log records are not forced to disk—before this change we experienced a significant amount of hard disk thrashing (a separate disk was not used for database logs). Although this change will affect the persistency of data, it will not severely affect the system since the cached data is essentially soft state.

The XDFS API currently supports only synchronous calls and is not completely thread-safe (data corruption does not occur, but XDFS can readily cause deadlocks in the database). As a result, it was not feasible for us to implement xProxy by modifying the source code of a pre-existing web caching proxy (e.g., Squid), since those proxies expect an asynchronous API that supports callbacks. For our implementation we resorted to using a global lock for every call to XDFS. Although this lock would be undesirable for an actual high-load deployment of xProxy, we did not observe any significant issues during our performance evaluations because it was not the bottleneck in our evaluations.

Compression

To the best of our knowledge, xProxy is the first actual web proxy implementation to fully utilize compression. Although the HTTP specification has included compression-related mechanisms for a relatively long period of time, it has yet to become widely adopted by web servers. Both Netscape and Internet Explorer (IE) web browsers have built-in support for compression, and the Apache web server must be specially configured to handle compression. We selected *gzip* [D96] as our compression scheme because it is supported by both Netscape and IE, a public implementation of the algorithm is widely available, and the compression performance is superior to other algorithms (e.g., "compress" in UNIX). Our original purpose for adding compression was to lessen the negative performance impact for cases where deltas cannot be used; we were pleasantly surprised by the amount of improvement that was gained (see below for more details).

HTTP 1.1

We invested a significant amount of time in creating a proxy implementation that would conform to HTTP/1.1—our implementation required 5000 lines of C/C++ code (excluding any code for XDFS, compression, and the Berkeley DB). HTTP version 1.1 provides certain features over 1.0 that can increase overall performance and reduce both CPU and network overhead. The current version of xProxy uses persistent connections so that multiple objects can be requested and received over the same network connection. With HTTP/1.0 at most one object can be requested for each connection.

The xDelta proxies also support "pipelining", whereby multiple requests are sent over a single persistent connection without waiting for each response. Pipelining requests serves to reduce latency and also makes it possible for multiple responses to be returned in the same network packet. Pipelining is particularly important for proxies because they handle requests that are destined for many different servers. Without pipelining, a proxy would not be able to service multiple requests from a particular network connection in parallel. For example, by pipelining requests to a proxy for "a.com" and "b.com", the proxy can simultaneously retrieve information from those two sites. This feature is particularly important due to a protocol-defined restriction in HTTP/1.1 that limits each client to at most two connections to the same proxy or web server—this limit does not apply to proxies (which are permitted to use 2*N connections, where 'N' is the number of users) and no such limit exists in HTTP/1.0.

Most web servers currently support persistent connections and pipelining, although we found certain implementations to be better than others. For example, older versions of Apache only allow up to 5 requests to be made over a single persistent connection and many web servers will only return responses in separate network packets even though some responses could have been consolidated into fewer packets. In terms of web browsers, Netscape currently supports only HTTP/1.0, although the HTTP Working Group indicates that the latest beta version of Netscape supports HTTP/1.1.

We did implement a special version of our client proxy that is optimized for HTTP/1.0 clients. The HTTP/1.0 optimized version of our proxy communicates to the client using HTTP/1.0, but all other communication that is performed on behalf of that client (e.g., communication to the SP or servers) is conducted using HTTP/1.1. HTTP/1.0 clients tend to make a large number of simultaneous requests using multiple network connections (Netscape uses up to four), so this particular version of xProxy uses pipelining to multiplex those requests over a single persistent HTTP/1.1 network connection.

Microsoft's IE web browser does support HTTP/1.1 and persistent connections, but it does not pipeline requests. By default, IE uses HTTP/1.0 through a proxy, but we took advantage of a user-specifiable option that causes IE to use HTTP/1.1 with proxies. Since neither Netscape nor IE implement the full HTTP/1.1 functionality, we created a custom test client and used that client to analyze the performance of the system. This test client does not actually render the HTML pages graphically, but we did enhance it with timers that would allow us to accurately measure the duration of each request and the various phases that comprise each request.

Mogul and Douglis have recently published an IETF draft that outlines an approach for using delta encoding in HTTP. The latest version [MKD00] outlines the addition of several new attribute headers and a new status code. Although our implementation does conform to this proposal, it does not use all of the proposed headers. For example, the proposal outlines a means by which web servers can specify a template for a site (against which deltas can be computed) as well as a "clustering" attribute that allows servers to specify how a client can determine if computing a delta against a "neighboring" URL is a worthwhile endeavor for producing an effective (i.e., small) delta. Our current implementation does "cluster" URLs that execute the same CGI program but have a different CGI query string (versions are indexed by the prefix string up to the '?' character).

We also experimented with using deltas on the first-access to a page by using a "neighboring" URL (i.e., a cached object from the same web server) as the baseline version. The resulting deltas not only had a larger variation in size, but they were also an average of 40% the size of the full-version of the object. The compression in our implementation typically results in objects that are smaller than 40%, so we elected not to expand the clustering feature to consider neighboring pages.

Size as Percent of Original Size	Delta			Compression	Compressed Delta		
	Best Case	Worst Case	Average Case	Average	Best Case	Worst Case	Average Case
abcnews.com	0.40%	28.92%	13.91%	39.51%	0.28%	16.66%	9.19%
aol.com	0.39%	5.07%	3.17%	39.03%	0.33%	4.04%	2.57%
cnn.com	0.24%	35.68%	19.50%	43.68%	0.23%	22.53%	13.36%
slashdot.org	0.33%	47.15%	24.91%	44.94%	0.33%	30.33%	17.05%
finance.yahoo.com	0.17%	16.33%	10.11%	43.96%	0.17%	11.68%	7.18%
yahoo.com	0.14%	6.59%	3.05%	43.81%	0.14%	5.66%	2.84%
Average	**0.28%**	**23.29%**	**12.44%**	**42.49%**	**0.25%**	**15.15%**	**8.70%**

Table 1: Average savings from using compression and deltas.

3 Results

The goal of this evaluation was to examine the update patterns of web objects (HTML files, output generated by CGI scripts, images), quantify the amount of change over time (differences between various versions of a file), evaluate the benefits of transferring deltas between file versions as opposed to entire uncompressed or compressed files, and determine the overall performance speedup gained by using xProxy. The following sections detail the experimental setups and the results we obtained.

3.1 Reduction in Bandwidth

To gauge the overall benefit of deltas we initially selected six popular web sites and used a custom program to obtain the homepage from each site on a regular basis for an entire week (April 2nd to April 9th, 2000). The program polled each site on an hourly basis, and at 15 minutes past the hour for even numbered hours. After collecting the pages, we used another program to create a delta between each version of a page and all future versions of that page.

Table 1 presents the overall statistics for these deltas. The table shows that a wide variation exists for the average size of a delta among the web sites. For example, the average size delta for AOL was 3.17%, while the average delta for slashdot.org is nearly 25%. The worst case for slashdot was 47.15%, which is only slightly worse than the average case performance of the UNIX “compress” utility’s 44.94%. Compared to the compress utility, deltas have an obvious benefit; however, the benefits of deltas are less obvious when compared with gzip.

The table also presents results for compressed deltas. The data indicates that compressing deltas does have some benefit, but we feel that the margin of improvement is not large enough to make any definite conclusions on whether all deltas should be compressed. Although xProxy does not currently compressed deltas, it would require only trivial changes in the code to enable the feature.

We also experimented with using deltas against graphical images. In most cases we found that deltas for graphical images are usually 80 to 90% the size of the full version, and is therefore not worthwhile. This result is not entirely dissuasive considering that most images remain unchanged for a large period of time and can be cached using traditional means. In addition, we also found a NASDAQ graph at http://www.golden1.com/ for which the deltas never exceeded 40% the size of the original graph.

Each of the data points for our html delta experiment were plotted so that we could study how the size of deltas increase as more time passes since a page was last visited. Each of these graphs displayed a relatively regular pattern. Figure 2 shows the delta sizes for http://abcnews.go.com/. The graph shows that the size of a delta initially increases at a relatively high rate before leveling off around 15%. Figure 3 shows the average size delta for each two-hour interval. By examining the graph we can see that a delta will first reach the maximum range after two days have passed since a page was last visited.

Figures 4 and 5 show the behavior of delta sizes for aol.com. The graphs show that a significant change occurred to the web page on day 4 of the week that we polled the web site (the average delta size in Figure 5 starts to increase in day 3 because each “day” in that chart is relative to the start of the log, rather the than actual time of day). Although the size increase appears to be significant, a closer inspection of the graph shows that there is only a 2% increase in the size of the deltas (relative to the full-version of the page). In Figure 4 we can also see that the deltas that are computed against a base version occurring after the change have returned to the smaller delta size (e.g., the purple “day 6” deltas which are 24 hours old are smaller than 3%).

Figure 6 is another example of another web site with “well-behaved” deltas. Figure 7 shows the benefits of using deltas for “stock quote” web pages. Rather than compute an average delta size for the same stock over time, we took a more challenging approach and graphed the average size of a delta when a particular stock page is used as the baseline version for other stock quota pages on that same site. With either approach, the benefits of deltas are noticeable.

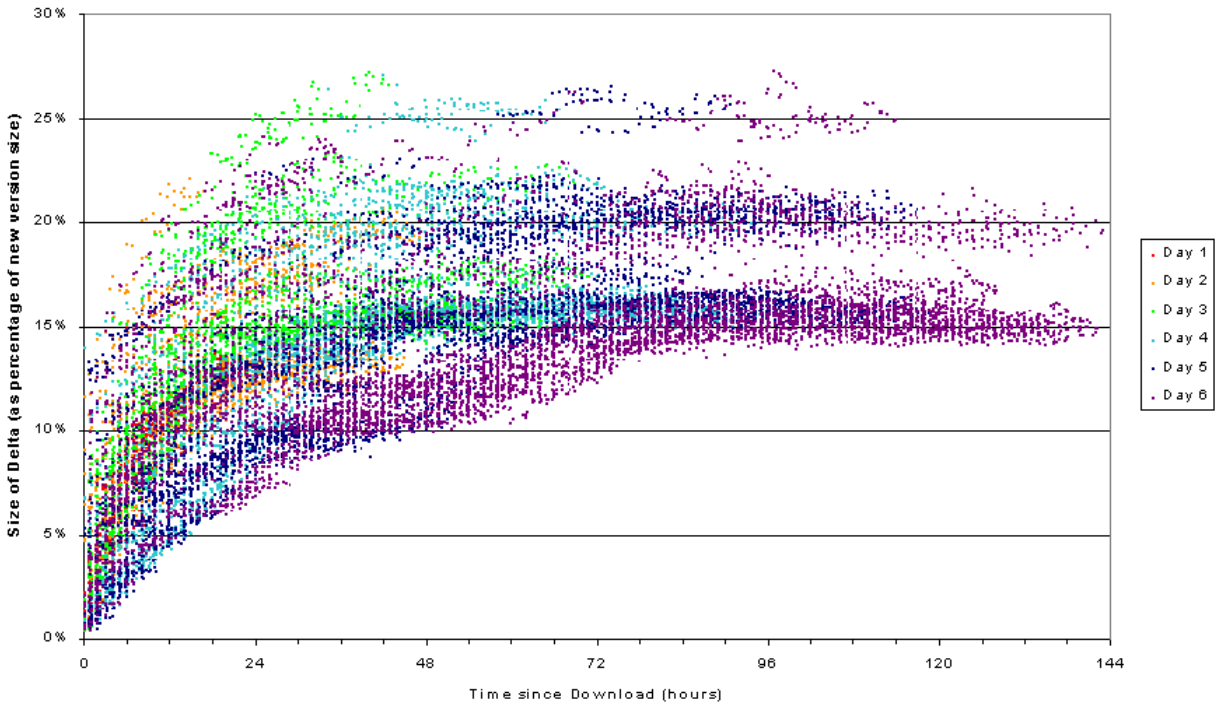

Figure 2: Size of delta (as percentage of new version size) vs. amount of time since the homepage for http://abcnews.go.com/ was last visited.

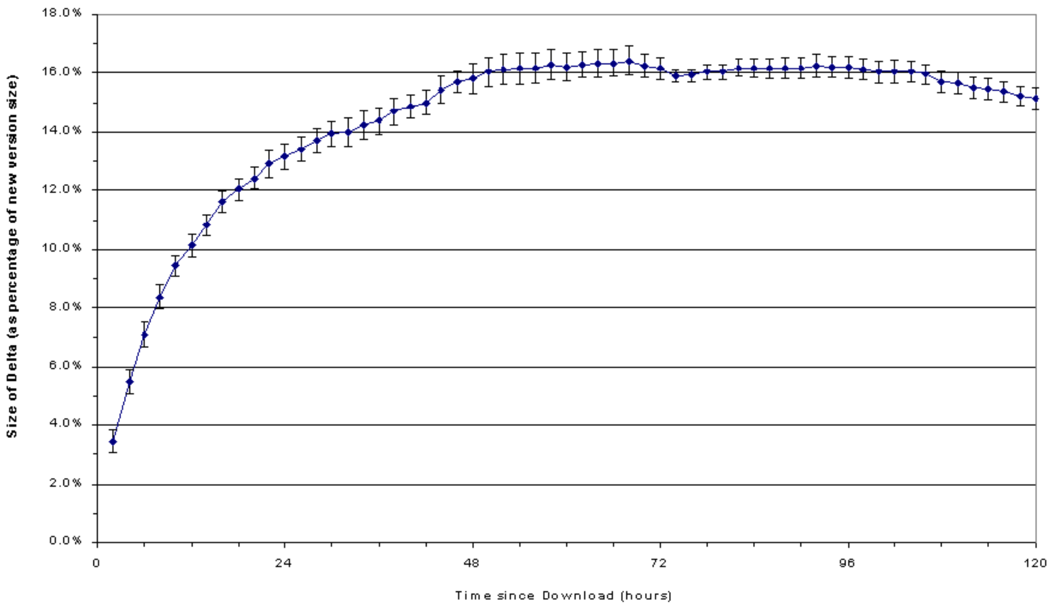

Figure 3: Average Size of delta (as percentage of new version size) vs. amount of time since the homepage for http://abcnews.go.com/ was last visited.

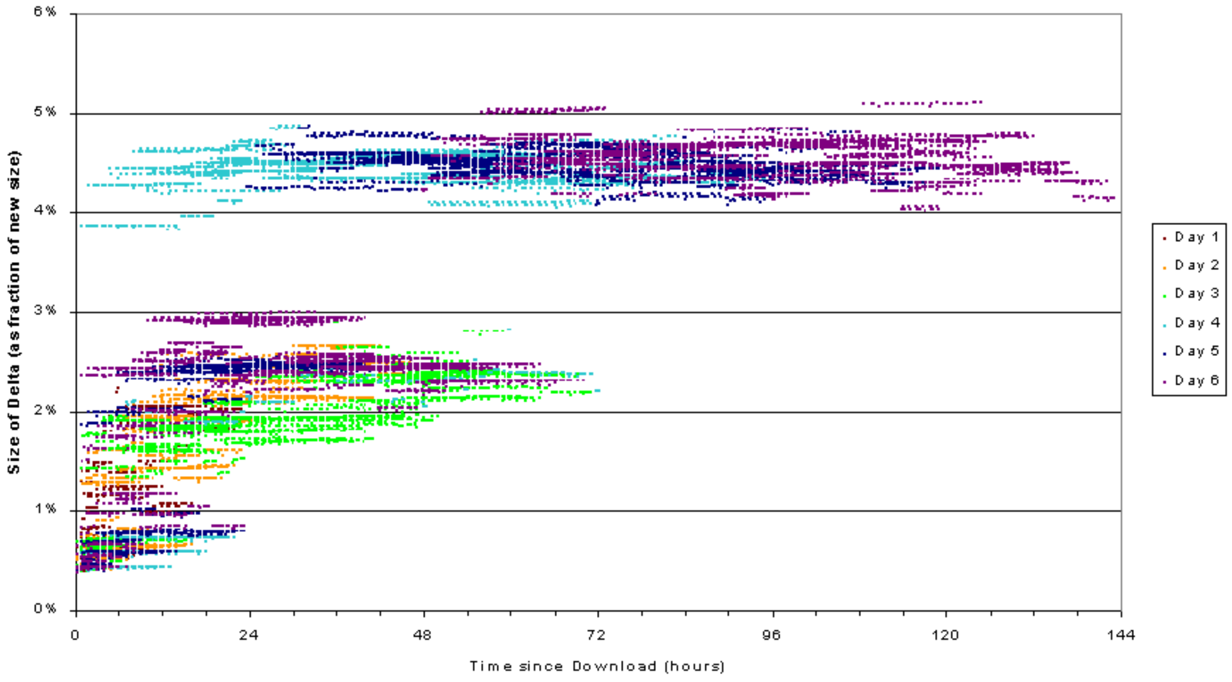

Figure 4: Size of delta (as percentage of new version size) vs. amount of time since the homepage for http://www.aol.com/ was last visited.

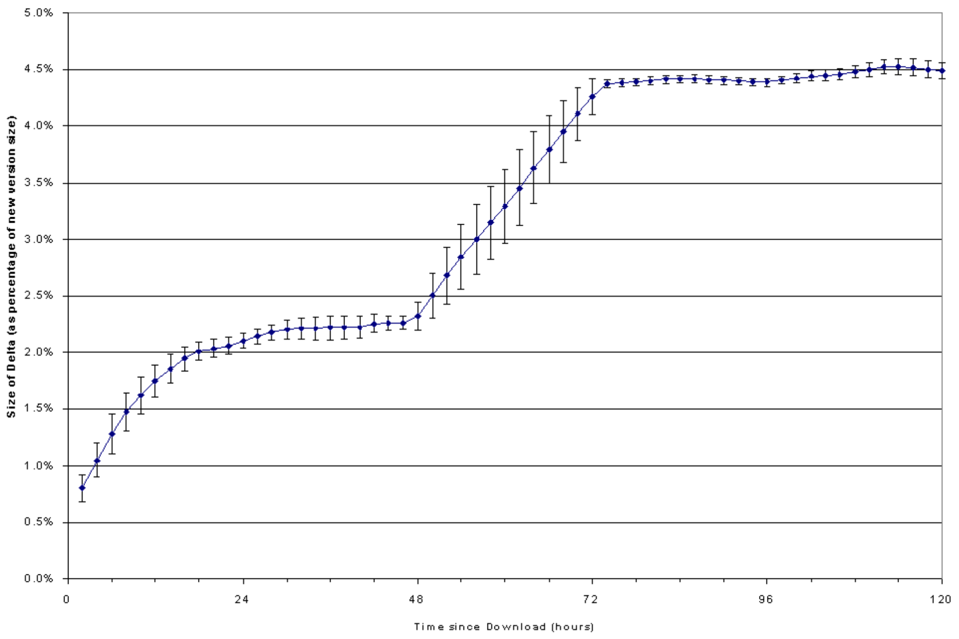

Figure 5: Average size of delta (as percentage of new version size) vs. amount of time since the homepage for http://www.aol.com/ was last visited.

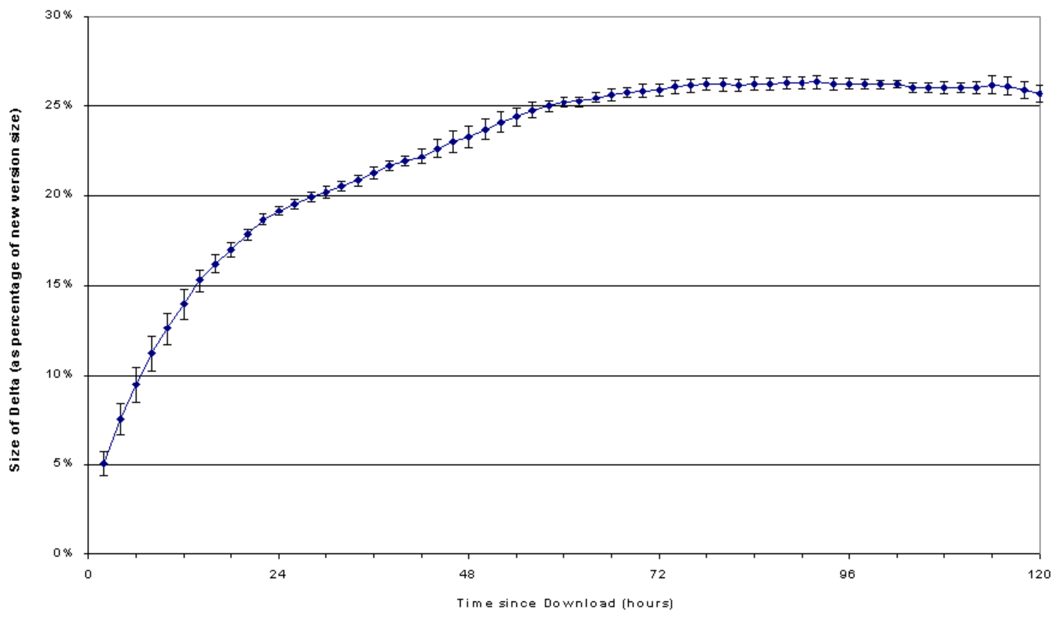

Figure 6: Average size of delta (as percentage of new version size) vs. amount of time since the homepage for http://cnn.com was last visited.

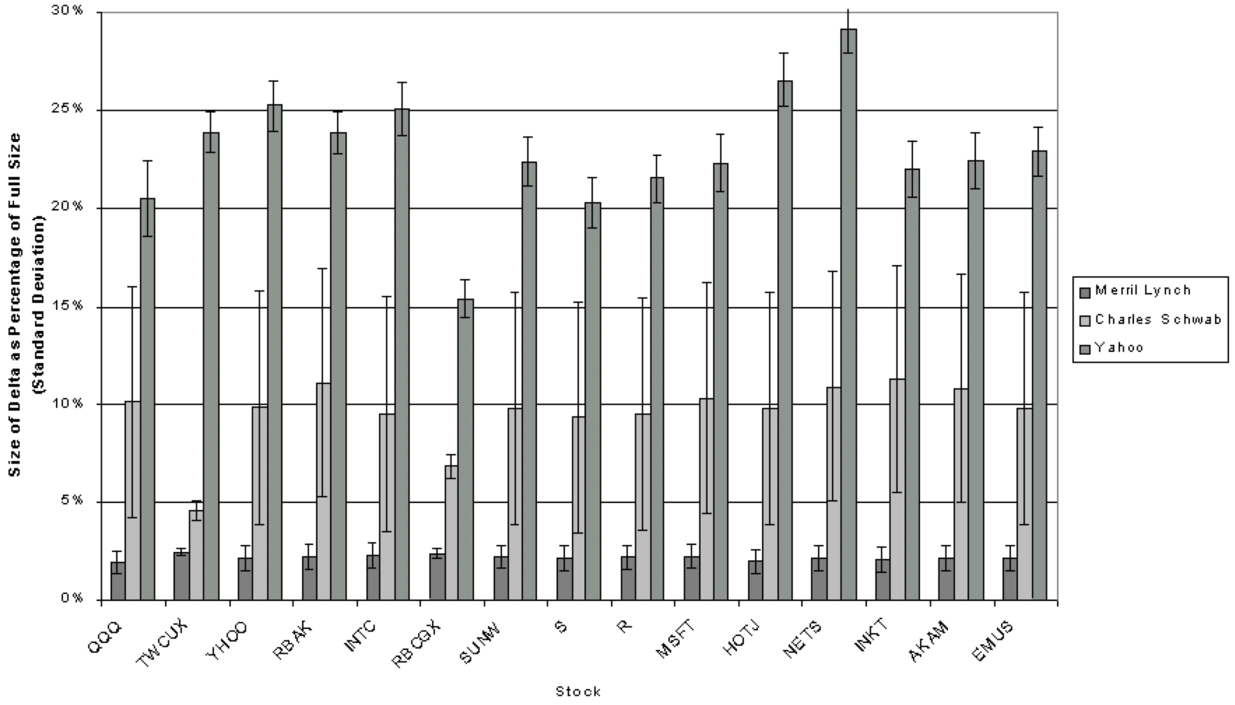

Figure 7: Average size deltas for stock quote pages on three different web sites, where each page is used as a base version for other stock quote pages on the same site

3.2 Reduction in Retrieval Time

To measure the decrease in retrieval time for obtaining web objects using xProxy, we retrieved the main "index.html" pages of 32 dynamic web sites (those pages changed a few times per day, such as news, stock quotes, sports, weather reports, shopping sites, etc.) and measured the retrieval time when using our proxies versus retrieving directly the pages from the source web servers. The client machine was a Pentium II, 333 MHz, 128 MB RAM, running RedHat Linux 2.2 and equipped with an U.S. Robotics 56.6 kbps data/fax modem. The connection to the Internet was established through the U.C. Berkeley general modem pool. The "browser" was a C++ program acting as an HTTP/1.1 client that connected to the locally running client proxy or directly to the source web servers. The server proxy was run on Sun ULTRA 5, 333 MHZ, 128 MB RAM, running SunOS 5.7 and having a 100 Mbps network connection. The server proxy was located five "hops" from the modem pool.

To determine the average retrieval time for downloading the HTML files directly from the source web servers, we retrieved the main HTML page 20 times over the course of one day, with requests being made every 30 minutes from 10am to 7:30pm. A similar experiment was performed for retrieving web pages through xProxy during "cold start" (i.e, when the proxies must use compression because they do not have a cached version of the file). For this experiment the client and server proxies were restarted on every trial to clear their caches. To determine the average retrieval time for transferring deltas between proxies, we retrieved the main HTML pages of the same web sites 5 times during the course of one day (same day as above), with requests being made every 2 hours from 10am to 6pm; the caches of client and server proxies were never flushed during this experiment.

Both proxies and the C++ program acting as the "browser" logged timing information and the amount of data transferred; these logs were later used to determine average retrieval time for downloading the various HTML files, the performance speedup gained when using deltas or compression during cold start, and the amount of time spent in the subcomponents (such as overhead added by the proxies, modem transfer time, and web server response time).

Figure 8 shows the average retrieval time for the 32 web sites. Even during cold start, xProxy is still able to reduce the retrieval time by compressing HTML files. Moreover, the use of deltas further decreases the average retrieval time. Figure 9 illustrates the performance speedup gained using xProxy as opposed to retrieving the files directly from the source web servers: transferring compressed files during cold start reduces the retrieval time by an average factor of 2.3, while transferring deltas improves performance by an average factor of 6.0! Overall, delta transfer reduces the retrieval time by an additional factor (on top of compression) of 2.6. Although these figures are somewhat overstated since they do not consider the transfer of other objects such as images, the level of speedup can still be achieved for pages that contain cacheable images.

Figures 10 shows the breakdown of average retrieval time when transferring whole compressed files through xProxy during cold start. The time components that measured are client overhead, server overhead, modem transfer time, and server response time. The client overhead includes request processing (in this case, the client proxy just pipes the client request to the server proxy since there is no existing version in XDFS) and reply processing (decompression of compressed file and insertion into XDFS). The server overhead includes request processing (same as above) and reply processing time (insertion into XDFS of new version received from the source web server and compression of this version).

The goal of this evaluation was to determine the overhead added by our proxies and, thus, how much improvement can be provided by a more efficient implementation. As can be seen in Figure 10, the client and server proxy overheads are insignificant as compared to the modem transfer and server response times. More specifically, the average total overhead added by both proxies is 7%, with 5.9% of the time spent in the client proxy and 1.1% spent in the server proxy. Thus, the overhead added by xProxy is an insignificant percentage of the total time.

Figure 11 shows the breakdown of where the time is spent when transferring deltas through xProxy. The time components that we measured are the same as above. On the client proxy, the request processing includes the extraction of a version MD5 from XDFS, and the reply processing includes the insertion into XDFS of the received delta, which automatically reconstructs the most-up-to-date version, and extraction of this version from XDFS. On the server proxy, the request processing remains the same, but the reply processing includes the extraction of a delta based on the version MD5 sent by the client. The average overhead added by both proxies is 8% of the total time, with 6.8% of the time spent in the client proxy and 1.2% spent in the server proxy. Again, the overhead added by xProxy is an insignificant percentage of the total retrieval time.

From Figures 10 and 11, we can observe that the client proxy adds more overhead than the server proxy, even though they use the same source code. One explanation is that Solaris provides a more efficient threads implementation than Linux. However, further investigation is needed to determine the exact cause of this slight discrepancy.

Overall, xProxy provides great reduction in time for retrieving web objects. We used it regularly on Linux with Netscape Communicator 4.6 when connecting to the Internet via modem, and our user experience has been very satisfying until now. We also tested it (successfully) for correct functionality with Internet Explorer 5.0 on Windows NT, while running both proxies on a Solaris machine.

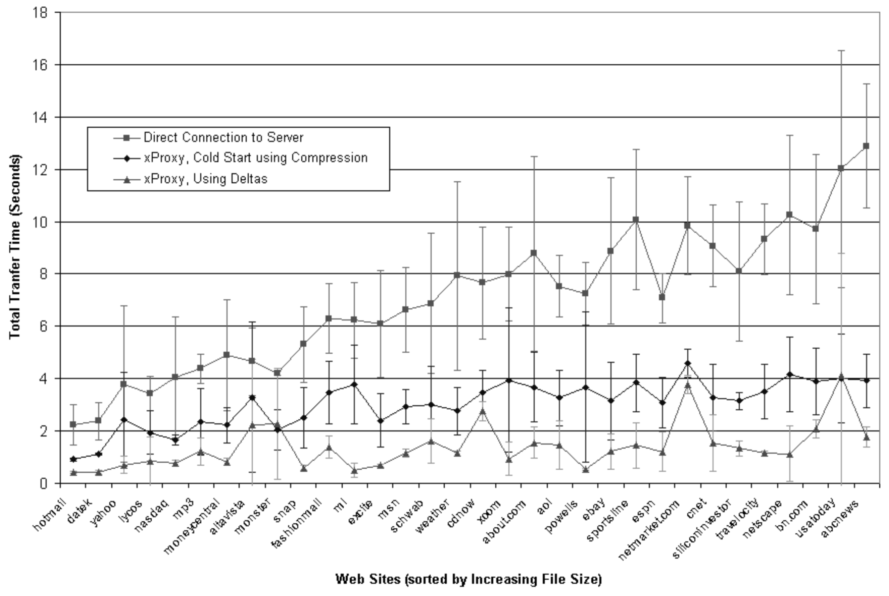

Figure 8: Average retrieval for each HTML file retrieved 1) using a direct connection to the source web servers, 2) using xProxy to transfer whole compressed files during cold start, and 3) using xProxy to transfer deltas.

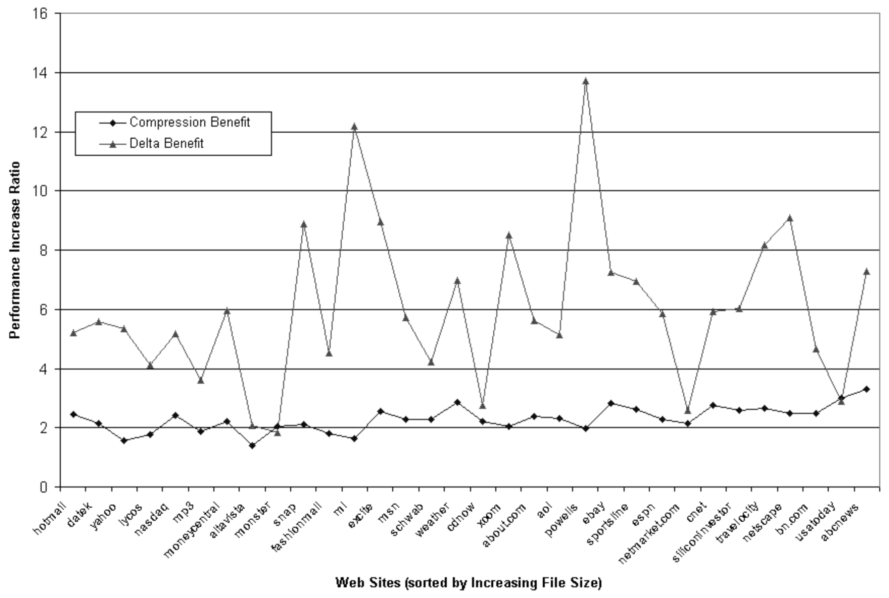

Figure 9: Performance speedup of xProxy (transferring deltas or whole compressed files during cold start) over direct retrieval from the source web servers.

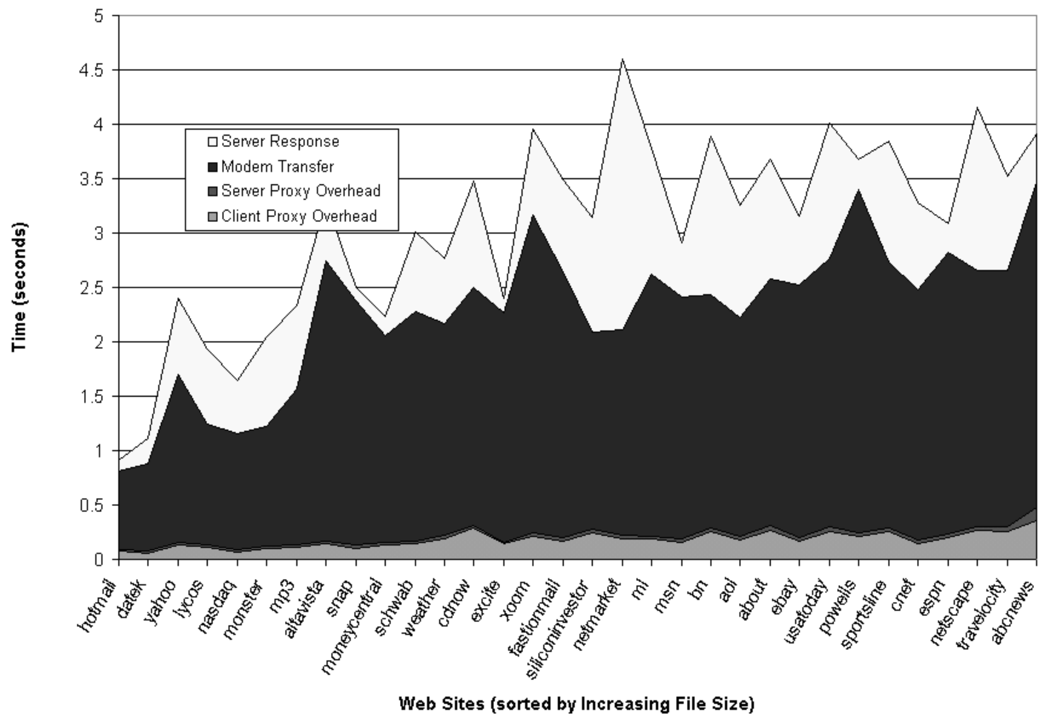

Figure 10: Breakdown of average retrieval time when transferring whole compressed files through xProxy during cold start; the measured components of the time are client proxy overhead, server proxy overhead, modem transfer time and server response time.

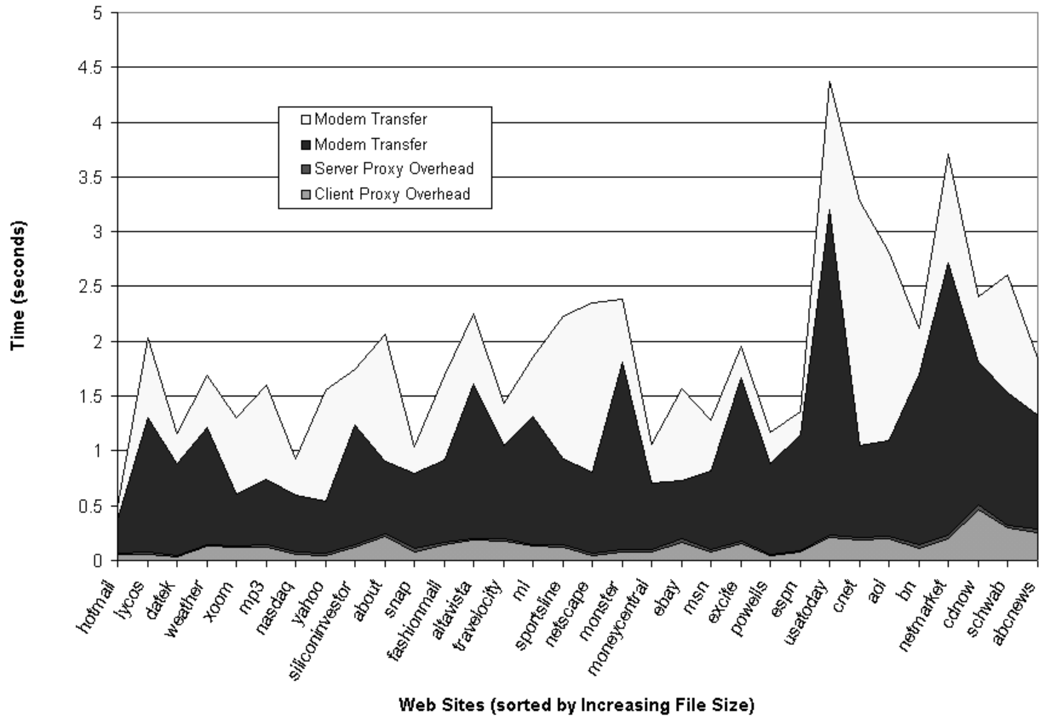

Figure 11: Breakdown of retrieval time when transferring deltas through xProxy; the measured components of the time are client proxy overhead, server proxy overhead, modem transfer time and server response time.

4 Related Work

For this paper we opted not to make any claims regarding the characteristics of how the "average" person surfs on the World Wide Web. A number of papers have already attempted to address the issue ([FCD99], [WM98]), and to a large extent these papers draw differing (and sometimes opposing) conclusions because their underlying analysis is based on traces obtained from their employer or university. Many of these papers are also misleading because they implicitly or explicitly dismiss accesses to dynamic content (or content containing cookies) because they assume these objects can have no useful or tangible utility for future accesses.

Jeff Mogul and Fred Douglis have authored a number of papers on the user access behavior and content update behavior of the web. In [DFKM97] they report that 1) nearly 70% of accesses are for images, and 20% are for HTML, 2) images almost never change, 3) the most frequently accessed resources have a shorter interval between modifications, 4) 60% of the items that are referenced multiple times have changed since the last access (16.5% are modified on every access), and 5) the mean inter-arrival time between accesses is 25.4 hours (although spikes in access probability are present at certain intervals such as 1 minute and 1 hour).

Mogul and Douglis also published a paper on the potential benefits of deltas between multi-user web proxies [JMFK97]. Their experiments involved an analysis based solely on web traces and did not include an actual implementation nor considered the tradeoffs of computation time. Although the technical report is nearly sixty pages in length, its usefulness is limited because their use of pronouns renders many of their statements ambiguous. What is clear from the paper is that one-third of HTML references are "delta-eligible" (meaning that deltas could be used if both proxies still have cached copies of a previous version), and a significant number of references are to CGI's that produce dynamic content. In addition, they conclude that 90% of repeat accesses occur within 14 hours of a previous access and that compression can be beneficial in cases where deltas are not feasible. [WM98] points out that pages with a ".com" domain name are more likely to change than those with ".net", ".org" or ".edu".

The WebExpress project [HL96] predates the work of Mogul and Douglis and is commonly considered the first published description of delta encoding for accessing web servers. WebExpress is targeted for salespersons who must use wireless devices to submit web-based transactions over a wireless interface, and it uses an optimized, non-HTTP, message stream for the wireless connection. Since WebExpress only caches a single version of each page, a client that is not in possession of that particular version must be sent a copy of that version, uncompressed, before deltas can be used. The project only considered two different benchmark applications and all items are cached by WebExpress for a fixed period of time. Since WebExpress uses a non-HTTP protocol, it would not be appropriate for more general use in a network.

A later project [BDR96] advocated the use of "optimistic deltas". Their main premise that, in the case of servers with high latency, a person accessing the web via a modem would see better performance if an old version of the page was transmitted to the user while he/she is waiting for the server to respond. Once the server does respond to the request, a proxy could transmit to the user a delta between the old version and the new version. The authors greatly underestimated the performance of the Internet and concluded that this approach would only improve performance in situations with extreme latency (such as five seconds). In all other cases, the "optimistic" approach performed worse. This project, like many others that have attempted to trade reduced latency for more network traffic ([PM96], [RBR], [FCL99]), essentially resulting in failure because many pre-fetched objects are not actually used and, in certain cases, the pre-fetching is useless because the object has been modified since the last pre-fetch occurred.

XDFS does not currently allow deletions of objects, so we were not able to consider the issue of cache replacement policies in a proxy. However, there are a large number of papers that already consider these issues. [CI97] provides an overview of nine different algorithms that are commonly used or discussed in literature, and the authors also present a new algorithm of their own creation. We expect that the conventional replacement policies for a web cache should also be appropriate for deltas, although a reference count for each delta may have a useful benefit.

5 Future Work

The current implementation of xProxy is fairly complete but can still be improved upon. Based on the experience we have gained, the following is a list of future work that we propose to do in this area:

- The client proxy currently keeps all versions of a file since we choose to reuse the code for the server proxy. The client proxy only needs to keep the most recent version of a file and, thus, a storage that keeps only the most recent version should be enough for the client proxy. XDFS can be instrumented to keep only one version or a simpler (and possibly less space consuming) storage, such a hash table, can be used for this purpose.
- Implement logic to decide whether it is better to send a delta or the entire file (compressed or uncompressed).
- Eliminate the "store and forward" technique that xProxy currently implements. Inserting a version/delta into XDFS as the data is being read from the network or sending data onto the network as the version/delta is being extracted off XDFS would cut down the overhead added by the two proxies.
- Implement a delta management policy to reduce storage requirements on the server proxy.
- Deploy a server proxy at the departmental or university level. This will allow extensive tracing and, thus, better understanding of the benefits provided by transferring deltas.
- Implement the server proxy as a cluster-based service to provide good scalability, fault-tolerance and availability. Such an approach would allow for incremental cache increase (and software upgrades) without discarding the previous cache or making the service unavailable.
- Since browsers already keep caches and can handle compression, it would be natural to include a mechanism for reconstructing a page using a delta into the browser, thus, eliminating the client proxy. On the other hand, web servers could send deltas if clients provide MD5's identifying the version they have; thus, bandwidth requirements for web servers would decrease at a slight increase in CPU load required to extract the necessary deltas.

6 Conclusion

The "all-or-nothing" approach of today's web ignores the fact that the current version of an object has a significant amount of content in common with prior versions. xProxy is a transparent proxy system that takes advantage of the similarity between versions, and uses deltas and compression to decrease the retrieval time for web objects. It is mainly targeted for users behind slow links, such as modems and wireless links, but it can also be used as a general mechanism to reduce the amount of web traffic across any network. Using compression, xProxy reduced the retrieval time for HTML objects by an average factor of 2.3; using deltas, xProxy reduced the retrieval time by a factor of 6.0.

Acknowledgments

We would like to thank Joshua MacDonald for providing us with XDFS.

References

[BDR96] Gaurav Banga, Fred Douglis, and Michael Rabinovich. "Optimistic deltas for WWW latency reduction." In Proceedings of 1997 USENIX Technical Conference, pages 289-303. Anaheim, CA, January 1997.

[BEOW] Richard Bunt, Derek Eager , Gregory Oster, Carey Williamson, "Achieving Load Balance and Effective Caching in Clustered Web Servers". Department of Computer Science, University of Saskatchewan.

[CDF98] R. Caceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich, "Web Proxy Caching: The Devil is in the Details", Performance Evaluation Review, Vol. 26, No. 3, pp. 11-15, December 1998.

[CI97] Pei Cao, and Sandy Irani, "Cost-Aware WWW Proxy Caching Algorithms". In Symposium on Internet Technology and Systems. USENIX Association, December 1997.

[D96] P. Deutsch. GZIP file format specification version 4.3. RFC1952, Network Working Group, May, 1996.

[DAP] John Dilley, Martin Arlitt, Stephane Perret, "Enhancment and Validation of Squid's Cache Replacement Policy", HP Labratories, Technical Report.

[DFKM97] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffery Mogul. "Rate of change and other metrics: a live study of the World Wide Web." Technical Report #97.24.2, AT&T Labs-Research, Florham Park, NJ, December 1997.

[FCD99] Anja Feldmann, Ramon Caceres, Fred Douglis, et al. "Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments". In

Proceedings of the IEEE Infocom '99 Conference, New York, NY, March 1999. IEEE.

[FCL99] Li Fan, Pei Cao, Wei Lin, et at. "Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance." Proceedings of the 1999 ACM SIGMETRICS Conference, Atlanta, GA, May 1999.

[FGC97] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, Paul Gauthier. "Cluster based scalable network services", Proceedings of the 16th ACM Symposium on Operating System Principles, San Malo, France, October 1997.

[GB97] Steven D. Gribble and Eric A. Brewer. System design issues for internet middleware services: Deductions from a large client trace. In Proceedings off the Symposium on Internetworking Systems and Technologies. USENIX, December 1997.

[HL96] Barron C. Housel and David B. Lindquist. "WebExpress: A System for Optimizing Web Browsing in a Wireless Environment." Proc. 2nd Annual Intl. Conf. on Mobile Computing and Networking, ACM, Rye, Net York, November 1996, pp. 108-116. Document was later published in 1998 without any significant advances.

[HTTP1.1] Roy Fielding, Jim Gettys, Jeff Mogul, et al. Hypertext Transfer Protocol -- HTTP/1.1, RFC 2616 (Proposed Standard), June 1999.

[JMFK97] Jeffrey Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. "Potential benefits of delta encoding and data compression for HTTP", Proceedings of ACM SIGCOMM '97, pp. 181-194, Cannes France, September 1997. A 60-page extended (and corrected) version appears as Digital Equipment Corporation Western Research Lab TR 97.4, July, 1997.

[Mac98a] Josh MacDonald. Program Source for Xdelta. ftp://ftp.xcf.berkeley.edu/pub/xdelta, 1998.

[Mac98b] Josh MacDonald. "Versioned File Archiving, Compression, and Distribution". UC Berkeley. Available via http://www.cs.berkeley.edu/~jmacd/.

[Mac00] Josh MacDonald. "Delta Compression for Storage and Transport". Master's Thesis (draft). UC Berkeley. Available via http://www.cs.berkeley.edu/~jmacd/.

[MKD00] Jeffrey Mogul, Balachander Krishnamurthy, Fred Douglis, et al. "Delta encoding in HTTP", IETF Internet Draft, Network Working Group. Available as "draft-mogul-http-delta-04.txt" from ftp.isi.edu.

[ML97] J. Mogul, P. Leach, "Simple Hit-Metering and Usage-Limiting for HTTP", Network Working Group, RFC2227. October 1997

[Mog94] Jeffrey Mogul. "Improving HTTP latency". Available from http://www.ncsa.edu/SDG/IT94/Proceedings/DDay/mogul/HTTPLatency.html"

[Mog95] Mogul, J. "The Case for Persistent-Connection HTTP", Western Research Laboratory Research Report 95/4, Digital Equipment Corporation, May 1995.

[NGB97] H. Nielsen, Jim Gettys, Anselm Baird-Smith, et al. "Network Performance Effects of HTTP/1.1, CSS1, and PNG". June 42, 1997. Available from http://www.w3.org/TR/NOTE-pipelining.

[OBS99] M. Olson, K. Bostic, and M. Seltzer. Berkeley DB. In Proceedings of the 1999 USENIX Annual Technical Conference, FREENIX Track (June 1999), pp. 183-192.

[PM96] V. Padmanabhan, and Jeffrey C. Mogul. "Using Predictive Prefetching to Improve WWW Latency", ACM SIGCOMM, Computer Communication Review, Vol. 26, No. 3, pp. 2-36, July 1996.

[RBR] Pablo Rodriguez, Ernst W. Biersack, Keith W. Ross. "Improving the WWW: Caching or Multicast?"

[RRB99] Pablo Rodriguez, Keith Ross, Ernst Biersack, "Distributing Frequently-Changing Documents in the Web: Multicasting or Hierarchical Caching?". Submitted to Infocom '99. August 28, 1999.

[SS94] Spero, S., "Analysis of HTTP Performance Problems," July 1994. Available at http://metalab.unc.edu/mdma-release/http-prob.html.

[THO96] Touch, J., J. Heidemann, K. Obraczka, "Analysis of HTTP Performance," USC/Information Sciences Institute, June, 1996. Available at http://www.isi.edu/lsam/publications/http-perf/index.html.

[WAS96] S. Williams, M. Abrams, C. Standridge, G. Abdulla and E. Fox, “Removal Policies in Network Caches for World-Wide Web Documents”, Proceedings of the 1996 ACM SIGCOMM Conference, Stanford, CA, pp. 293-305, August 1996.

[WM98] Craig E. Wills and Mikhail Mikhailov. Towards a Better Understanding of Web Resources and Server Responses for Improved Caching. Technical Report WPI-CS-TR-98-27, Computer Science Department, Worcester Polytechnic Institute, December 1998.