Cooperative Web Caching Using Server-Directed Proxy Sharing

Ph.D. Dissertation Proposal

Sandra G. Dykes
April 28, 1998
Technical Report CS-98-01

Division of Computer Science
The University of Texas at San Antonio
San Antonio, TX 78249

Abstract

The World Wide Web suffers from scaling and reliability problems due to overloaded servers and congested routers. Caching at local proxy servers helps, but cannot satisfy more than a third to half of the requests; most requests must still be sent to the original HTTP server or to a higher level cache. This dissertation develops a protocol for cooperative distributed caching between proxy servers, known as *server-directed proxy sharing* (SDP). The goals of SDP are to reduce network contention, end-user response time, and denial of service incidents by distributing requests to caches on routes with more available bandwidth. SDP is designed to match characteristics of the HTTP protocol and web traffic patterns. For example, SDP takes advantage of web page structure by having the server piggyback a list of cache sites onto the page's HTML text. This allows the requestor to retrieve embedded images simultaneously from several cache sites, which decreases response time and better distributes network traffic. Server contact is further reduced by having caches lazily share directory information when they return requested objects.

Dissertation work will consist of three phases. The first task is to characterize HTTP server workloads by collecting local traces, analyzing them and comparing results to published studies. In the second phase, an analytical event-driven simulation will be used to evaluate the cache system. In the final phase a prototype version will be implemented.

The contribution of this dissertation lies both in the protocol design and in the simulation work. Previous simulation studies of Web caching systems modeled single Web servers without including network traffic, and used Web server traces to drive the simulation. Such simulations cannot compute response time or network congestion, and typically rely upon server load metrics such as *bytes/sec* and *requests/sec* to evaluate cache designs. I intend to develop a more predictive network cache simulator by 1) using analytical rather than trace workloads, and 2) adapting a network simulator to model interactions of network traffic generated by multiple Web servers and multiple cache sites. The ultimate goal of this simulation will be to determine whether we can provide realistic predictions of response time and network congestion for various caching protocols and workloads.

SDP is designed to fit into the existing Web infrastructure; it operates at the applications layer and can be built on top of existing HTTP servers and proxy servers. The advantage of this approach is that SDP works with current network layer protocols and Web software, making it easy to distribute and install a prototype across the Internet.

Contents

1 Introduction

1.1 The problem: scaling and reliability of Web servers

Like all large, distributed information systems, the World Wide Web faces issues of scaling and reliability. Unlike most systems, Internet solutions demand cooperation between autonomous, often competing entities. With no controlling authority, independent organizations seldom feel compelled to donate resources for the common good. The Web is also characterized by dynamic, bursty traffic and the phenomenon of popularity spikes, known as "flash crowds". [Sel96], [Sel97], [GwSe96], [Bes97]. Growth in Internet traffic and the dynamic nature of Web request patterns cause two major reliability problems: connection refusals and long, variable response times. A connection refusal or *denial-of-service* occurs when the Web server spawns too few processes or threads to handle the current request load. Large and variable response times occur because congestion at network routers fluctuates rapidly and unpredictably. These problems are exacerbated by the poor scalability of the Web's direct client-server protocol.

Distributed caching and replication are widely viewed as necessary if the Web is to become truly scalable and reliable. However, network caching does not scale if all users access a single cache group. Organizing users into groups and using a different cache for each group solves the scaling problem, but limits hit rates because it restricts the amount of available sharing. This is exactly what happens with Web proxy servers. Proxy servers are typically installed at a local site such as a company or university. The proxy server caches Web objects for all users at the site, and thus shares the cache across the local user community. Unfortunately, there is not enough request duplication within local sites to achieve high hit rates. Even with infinite cache size, the maximum hit rate for proxy caching ranges from approximately 30% to 50%, depending upon the site [AbSt95], [Bes97], [Gl94-1]. The Web needs a method to share cache contents beyond local sites.

1.2 A protocol for cooperative distributed caching

This dissertation develops the *server-directed proxy sharing* (SDP) protocol for cooperative distributed caching which allows Web proxy servers to share their cached objects. SDP's goal is to improve both user and global network performance. To do this, we must understand the charac-

teristics of a remote cache. Unlike memory and disk caches, remote Web caches are not inherently faster than remote Web servers because cache sites do not necessary have more powerful physical hardware or larger network bandwidth. All else being equal, remote network caching actually *increases* average response time if it offers nothing to offset the overhead of discovery and cache miss penalties. To improve response time, a Web caching system must either deliver multiple objects to a user in parallel or reduce delivery time by using caches on connections with more available bandwidth.

Currently most Web browsers simultaneously retrieve multiple embedded images from one Web server. This is a greedy approach which tends to improve user response time at the expense of congestion around the server. SDP simultaneously requests the embedded images, but sends the requests to *different* cache sites. This better distributes network traffic and reduces congestion. In addition, SDP chooses cache sites with the fastest response times, which usually corresponds to less congested routes and produces less interference with other traffic. Consequently, SDP is designed to improve both user response time and overall network performance.

SDP addresses the problem of server availability by reducing the fraction of connections a server handles and the number of bytes returned per connection. As a result, servers can handle more connection requests and are available to more users.

Finally, the SDP protocol takes into account the autonomous administrative nature of the Internet. Server-directed cooperation does not require a layer of dedicated caches maintained by a central authority, nor does it require local caches to donate cache space or change their cache behavior. This makes the protocol administratively, as well as technically, viable. Section 7 describes the SDP protocol in detail.

1.3 Improving simulations of distributed caching protocols

Previous research in Web caching used trace-driven simulation with single server models [BeCu96], [GwSe97]. A trace captures a single workload during one time interval, making trace-driven simulation best suited to systems where workloads do not vary greatly from site to site or over time. As Web workloads are notorious for their changing nature, the flexibility of analytical workload modeling may prove to be a better choice for Web cache simulators. The second limitation of current Web caching simulators is that they model an isolated server and ignore network effects.

Such simulations cannot compute response time or network throughput metrics, and typically rely instead upon server load metrics such as (*bytes/sec* and *requests/sec*) to evaluate cache designs. In this dissertation, I intend to develop a more predictive network cache simulator by 1) using analytical rather than trace workloads, and 2) adapting a network simulator to model interactions of network traffic generated by multiple Web servers and multiple cache sites. The ultimate goal of this simulation will be to determine whether we can provide realistic predictions of response time and network congestion for various caching protocols and workloads.

1.4 Contribution of dissertation

The contributions of this dissertation lie both in the protocol design and in the simulation work. Although caching is widely viewed as being necessary for scalability in large heterogeneous networks, most previous research considers only isolated or hierarchical cache organization. Little research has been done on non-hierarchical cache organizations. In the SDP protocol, we investigate a new cache organization composed of first level caches (proxy servers) connected in a flat mesh design. We develop methods for efficient cooperation between the caches.

The proposed simulation work is unique in that it will be the first Web cache simulation to use analytical workloads, and the first to use network traffic in the simulation model. If our goal of predicting usable response times can be achieved, it will be an important contribution to network cache simulation.

1.5 Phases of dissertation work

The dissertation work consists of three phases. The first phase involves collecting traces on local HTTP servers and using them to characterize HTTP server workloads. Most of the first phase has been completed: traces have been collected on HTTP servers for University of Texas at San Antonio's (UTSA) Computer Science Division and the Computer Visualization and Modeling Laboratory at UTSA. Both UTSA servers exhibit characteristics consistent with those reported in the literature. Consequently, we feel confident these traces can be used to elicit workload characteristics not reported in the literature and to verify, if necessary, the results of our analytical simulation.

The second phase of dissertation work evaluates the SDP protocol using analytical, event driven simulation. The analytical workloads are built from the workload characteristics reported in the

literature and determined from our own traces. In this phase we will investigate how best to incorporate network effects in distributed cache simulators. As network simulation is somewhat complex, the first step uses analytical workloads with an isolated single-server model, similar to simulations reported by other researchers [Bes97], [GwSe97]. This single-server simulation has been completed and is described in Section 8.

The third and final phase of the dissertation implements a prototype version of the protocol and evaluates its performance.

1.6 Organization of this proposal

Section 2 begins by defining Web caching terms used in this proposal. Section 3 explains the inadequacy of proxy server caching and why the Web needs an Internet-wide system of cooperative cache sharing. Section 4 describes important features of Web server workloads, and includes an analysis of UTSA Web server traces. Section 5 discusses design alternatives for cooperative Web caching and propose a taxonomy. Section 6 analyzes related Web caching designs and classifies them according to our taxonomy. The SDP protocol is explained in Section 7. Sections 8 describes the simulation phase, including preliminary simulation work, and Section 9 outlines the prototype implementation. The dissertation proposal is summarized in Section 10.

2 Web caching definitions

Web communication uses the HTTP protocol which is based on a client-server architecture. Caches can be located on the client machine, the server machine, and at intervening sites. A cache site plays the role of server for the original client and of client for the original server. This often leads to confusing and inconsistent terminology in Web caching literature. Therefore we begin by defining the set of Web caching terms in Table 1. The table includes duplicate terms if they frequently appear in the literature or if the more common terms are too confusing.[1]

Proxy server caches provide the building blocks for the current Web cache infrastructure, and for most proposed cooperative cache designs. Typically a proxy server handles requests for a user community located at a single administrative site, such as a company or university, and caches Web

[1] See the footnote on page 19 for an example of really confusing terminology.

Term	*Definition*
User	Browser process sending the original HTTP request.
Proxy server	Internet gateway and HTTP cache for a group of users; a proxy server shares cached HTTP objects across the group. User requests are sent to the proxy server, which attempts to satisfy requests from its cache before forwarding them on. The proxy server is usually on the same local network as the users.
Client	A **user** or its **proxy server**.
Server	HTTP server which owns the requested object.
User cache	HTTP object cache for a single user; located on the same computer as the user.
Client cache	A **user cache** or a **proxy server**
Remote cache	Cache site not on the client's local network.

Table 1: Definitions of Web caching terms.

objects for all its users. Figure 1 illustrates the relationship between Web users, proxy servers and servers.

3 Why is cooperative Web caching needed?

Before considering our protocol for a cooperative Web caching protocol, we will show that it is necessary. Modern Web browsers contain user caches, and local sites often deploy caching proxy servers. How much can user and proxy caching reduce Web traffic and server load? If localized caching were efficient enough, there would be no need for more complex Internet-wide caching sharing. Unfortunately, this is not the case.

Hit rates at a proxy server cache depend upon the number of users sharing the cache, and the duplication in requests among the users. To estimate the maximum achievable hit rates at a proxy server, Bestavros and coworkers at Boston University recorded client traces over a three month period and simulated a proxy server cache [BeCa95]. Considering only remote requests and assuming infinite cache size, they obtained a 53% byte hit rate.

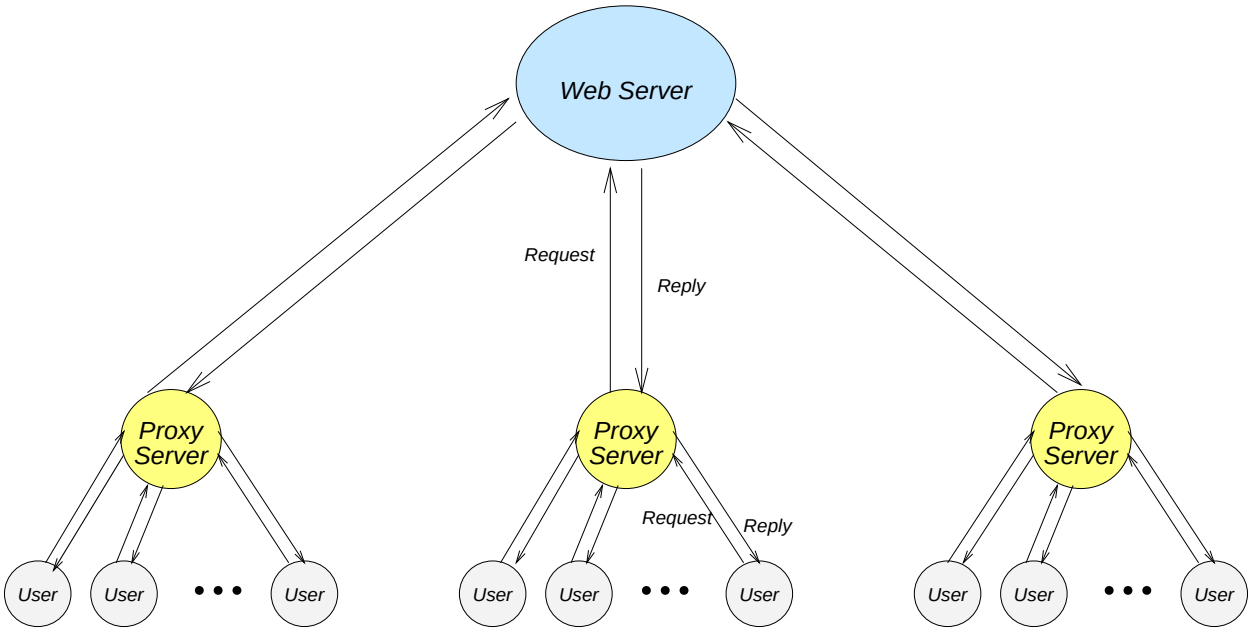


Figure 1: The role of a proxy server.

In a similar study, Abrams and coworkers at Virginia Tech traced requests from three user groups over a semester [AbSt95]. The maximum request hit rates of a proxy server cache for Abram's workloads are 29%, 30% and 46%, where the higher 46% hit rate occurred in a classroom group in which requests are expected to be highly correlated.

Steven Glassman measured hit rates at a combined gopher/HTTP cache on the DEC Systems Research Center gateway [Gl94-1]. The DEC cache was sufficiently large that no object was deleted from the cache over the study period, therefore it acted as an infinite size cache. During the two month study, daily hit rates varied from 30% - 50%.

These three studies indicate that a ceiling exists on the effectiveness of proxy server caching due to limited duplication in requests from the proxy's clients, even over long periods of time. Caching at a proxy server appears to reduce the number of remote requests by at most one-third to one-half. While the reduction is substantial, *most remote requests must still be sent to the server*. The only way to improve cache hit rates is to expand sharing across a wider group of users. It is this need for expanded sharing that motivates designs for cooperating Web caches.

4 Characterization of Web server workload

4.1 Summary of server workload studies

We need to understand Internet traffic patterns and Web server workloads in order to predict which caching approaches best suit the Internet and the HTTP protocol. A cache design performs well only if it matches its workload.

Cache designs are often evaluated using simulation, as we will do. Simulations are driven either by recorded traces or by analytical workload functions. While a trace exactly models one particular situation, it cannot predict performance for other situations. An analytical workload may not model any situation exactly, but by varying function parameters we see how the system performs under many different situations (see Section 8). For this reason we choose to use analytical workloads for our simulation. Consequently, we require analytical functions for interarrival times of requests at the server.

Several researchers have reported statistical summaries of HTTP request traces recorded at client, proxy server and Web server sites (see Table A in Appendix A). We are primarily interested in server traces because our simulation requires the interarrival times of requests at the server. Characterization of Web server traffic appears in [AlBe96], [ArWi96], [Bes97], [GwSe96], [Mo95], [Ta97], [Sel96] and [Sel97]. Of these, Arlitt and Williamson's article [ArWi96] and Seltzer's talk slides [Sel96], [Sel97] are the most comprehensive. Arlitt and Williamson identify characteristics, or *invariants*, common across six server traces. Their study finds object sizes have Pareto distributions and interrequest intervals have exponential distributions. Marge Seltzer examines a wider variety of Web servers, but focuses on fewer features. Her results show 1) object popularity varies by region, and 2) Web invariants apply most strongly to the most popular servers.

Table 2 summarizes characteristics of Web server workloads reported in the above citations that are relevant to cache design. The most important implications for Internet caching are as follows:

- **Web caching is viable**
  - A small percentage of objects satisfy most requests and most bytes transferred.
  - Objects have long lifetimes, reducing cache consistency concerns.
- **Web cache sharing should be widespread**

Web Server Characteristic	**Reference**	**Detail**
Most requests are from remote clients	Arlitt	Remote clients: $\geq$ 70% of requests and $\geq$ 60% of bytes.
Server load is growing exponentially	Seltzer	Number of requests and number of documents stored are growing exponentially.
Arrival time distribution is heavy-tailed	Mogul	Appears log-normal
Object size distribution is Pareto	Arlitt Crovella	$0.40 < \alpha < 0.63$ for transferred objects $\alpha = 1.06$ for transferred objects
Most transfers are small	Arlitt Almeida	Mean size <21 KB Image 13 KB, Text 4 KB, Audio 179 KB, Video 2300 KB, Application 112 KB
Most transfers are images or HTML	Arlitt Gwertzman Almeida	Image 36-78% HTML 20-50% Dynamic 0-7% Image 65% HTML 22% Dynamic 9% Image 75% HTML 19% Dynamic 0% Audio 5% Video 0.4%
Most transfers are duplicates	Arlitt	>97% sent more than once.
Object popularity follows Zipf's law: $freq \sim 1/p,\ p = popularity\ rank$	Arlitt Bestavros	10% of objects sent satisfy 90% of transfers 5% of bytes sent satisfy 85% of byte traffic
Objects have long lifetimes	Bestavros	JPEG 100 days GIF 85 days HTML 50 days
Popularity varies by region	Seltzer	Clients tend to access geographically related servers.
Popularity can change rapidly	Seltzer	Objects and server popularity can change very fast, producing *flash crowds*.

Table 2: HTTP Web servers characteristics summarized from the literature.

- As discussed in Section 3, site caching cannot satisfy most client requests.

- **Web caching systems should contain multiple, regional caches**

  - Object and site popularities vary with geography. Although clients access servers across the Internet, they tend to favor geographically close servers. For example, clients in California access California sites more often than do clients on the East Coast (and during different time periods).

- **Object dissemination should adapt quickly and automatically to request patterns.**

  - Object and server popularity can change very fast and unpredictably; for example, the Web guide Yahoo! publishes a "Cool Site of the Day." A cache system can not keep up with demand unless dissemination occurs automatically and adapts quickly.

4.2 UTSA Web server traces

We analyzed traces from two HTTP servers at the University of Texas at San Antonio: the computer science division Web server (UTSA-CS) and a research-oriented Web server in the Visualization and Modeling Laboratory of Dr. Kay Robbins and Dr. Steve Robbins (UTSA-V). Table 3 summarizes trace statistics and workload characteristics of the servers.

By comparing features of the UTSA traces to published Web server characterizations, we can determine if the UTSA workloads agree with other Web servers. If so, the UTSA traces can be used to elicit other workload features not reported in the literature and, if necessary, can be used in a trace-driven simulation to validate the analytical simulation results.

UTSA server workloads match characterizations reported in the literature with two major exceptions. First, server load for the research site, UTSA-V, is not growing exponentially (see Figure 2). Secondly, the division server, UTSA-CS, received an insignificant number of requests for video objects. Because video objects are large compared to other types, they typically add a bump in the tail of the object size distribution. If video requests are not present, the distribution appears less "heavy-tailed". Figure 2 shows the object size distributions in the UTSA traces. Analytical functions for interarrival times and object size distributions have not yet been determined.

Characteristic	UTSA-CS	UTSA-V
Trace statistics		
Dates	Apr 97 - Sept 97	May 96 - Aug 96, Dec 96 - Aug 97
Number of requests	561,292	547,272
Successful requests	63.9%	75.7%
Traffic (bytes transferred)	3.8 GB	2.9 GB
Remote clients	82.4%	77.9%
Server load growth	Possibly exponential (see Fig. 2)	No discernible pattern
Arrival time distributions	not analyzed	not analyzed
Object size distribution	see Figure 2	see Figure 2
Most transfers are small	Mean size <12 KB	Mean size <11 KB
Image	15 KB	6 KB
Text	4 KB	4 KB
Audio	200 KB	81 KB
Video	na	452 KB
Dynamic	1 KB	1 KB
Application	135 KB	386 KB
Most transfers are images or HTML		
Image	transfers 48% bytes 62%	transfers 54% bytes 30%
HTML	transfers 44% bytes 15%	transfers 42% bytes 15%
Audio	transfers 0.4% bytes 8%	transfers 0.1% bytes 1%
Video	transfers 0% bytes 0%	transfers 1% bytes 40%
Dynamic	transfers 4% bytes 0.3%	transfers 0.1% bytes 0%
Embedded images are significant	transfers 37% bytes 43%	transfers 46% bytes 24%
Most transfers are duplicates	>99% sent more than once	>99% sent more than once
Object popularity	10% of objects = 69% of transfers	10% of objects = 79% of transfers

Table 3: UTSA Web server workload analysis.

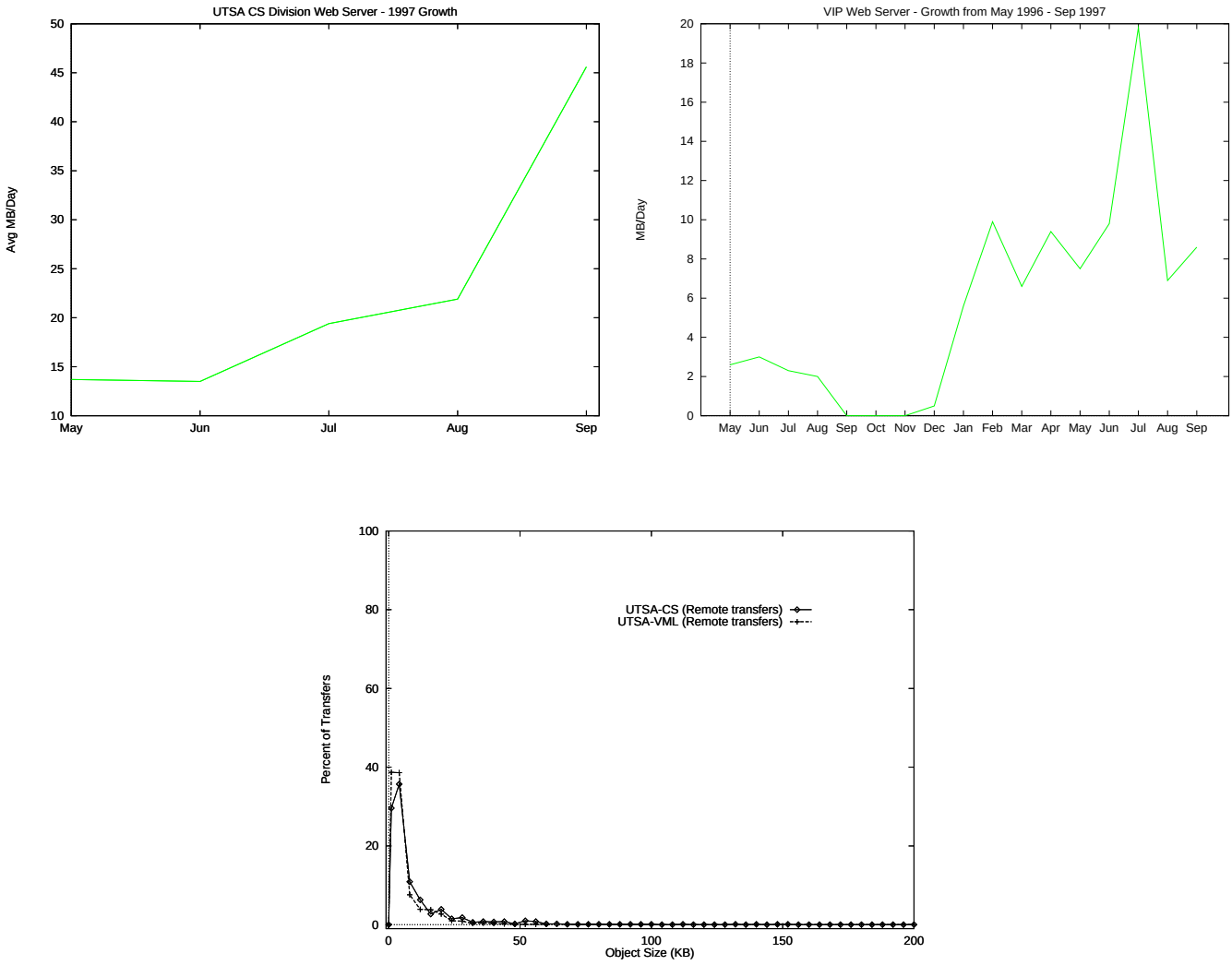


Figure 2: Server load and object size distributions for UTSA Web servers. The top figures plot growth in server load over time (MB/day). The bottom figure shows object size distributions up to 200 KB with the remaining distribution not shown for clarity. Approximately 0.1% of transferred objects exceed this size limit, and 15 are larger than 4 MB.

4.3 Web page statistics

Two important statistics we obtained from UTSA traces are the percentage of Web page requests and the average number of embedded images per page. Although these Web page statistics are not reported in the cited literature studies, they are critical when considering client prefetching or multiple simultaneous retrieval algorithms.

For this analysis, a Web page request is considered to be a request for an HTML text file followed immediately by one or more requests from the same client for GIF or JPEG images. In UTSA-CS and UTSA-V, the percentage of Web page requests is 11.5% and 13.7%, with embedded images accounting for 37% and 46% of the transfers and 43% and 24% of the returned bytes. On the average, 54% of all requests are Web page HTML or embedded images files. The average number of images per Web page is 2.9 and 3.5 for UTSA-CS and UTSA-V, respectively.

5 Taxonomy of Web cache designs

Distributed caches provide three services: discovery, dissemination and delivery of cache objects. *Discovery* refers to how clients locate a cached object. *Dissemination* is the process of selecting and storing objects in the cache; that is, deciding which objects are cached, which cache sites they are stored at, and when the caching occurs. *Delivery* defines how the object makes its way from the server or cache site to the client. Current proposals for Internet-level caches use a surprisingly small number of alternative approaches in each category. These are listed in Table 4 and described in the corresponding sections.

5.1 Why a taxonomy?

Classifying protocols with a taxonomy helps compare designs according to basic design choices rather than implementation details. Just as importantly, using a taxonomy as part of the protocol evaluation helps to focus on aspects that may be overlooked in a simulation or performance study.

Up to now, simulations of Web caching protocols have computed server load instead of response times because it is difficult to model the Internet and because there is no global, time-dependent HTTP data. However, evaluating a protocol *solely* by server load is dangerous because it ignores the protocol's effects on response time and network congestion. For example, response time depends

Service	*Categories*	*Details*
Discovery	Fixed cache	Client always sends request to the same cache.
	Group query	Client locates object by sending queries to a cache group.
	Manual	Group is configured manually.
	Auto	Group is configured automatically.
	Cache Site Directory	Client obtains cache location from directory; many possible directory organizations (centralized, hierarchical, etc.).
Dissemination	Client-initiated	Object cached as a result of client request (*pull-caching*)
	Server-initiated	Server decides what, where and when to cache (*push-caching*)
Delivery	Direct	Object always returned directly to the client.
	Indirect	Object may pass through several sites before reaching client.

Table 4: A taxonomy of Web caching infrastructures.

upon the number of remote transfers in delivery and the current available bandwidth along the delivery route(s). Server load does not reflect delivery times, so is a poor predictor of response time. Because the taxonomy helps to focus on all components of a cache protocol, its helps researchers keep response time and network traffic in mind and mitigates the danger of using incomplete metrics.

5.2 Discovery

5.2.1 Fixed cache discovery

In *fixed cache discovery*, the client always send its requests to the same cache site. Servers can be stateless; it is the client who maintains knowledge of the cache location. Stateless server caching has two advantages: no caching information is lost in the event of server failure and the design requires no changes to existing Web servers. Fixed cache discovery is used in proxy server caching, where browsers are configured to send all requests to their site's proxy server.

5.2.2 Group query discovery

In *group query discovery*, clients locate the object by sending queries to each member of a cache group. This permits servers to be stateless. Clients are responsible for knowing where to send their queries, which requires some method of maintaining group membership information. Two methods have been proposed: *manual configuration* and *automatic configuration*. In manual configuration such as that used in the NLANR national caching system, each client must be informed when the group membership changes [NLANR-98]. Typically this means the systems administrator manually modifies a configuration file. Zhang and coworkers point out manual configuration is error-prone and does not scale. Instead, they propose using IP multicast to automatically configure cache groups [ZhFl97]. Because queries are sent to a multicast address, the clients need not be reconfigured every time a member joins or leaves the group.

The drawback to using group query for discovery is that group size is constrained by competing factors. If the cache group is too large, the client floods the network with query packets. IP multicasting will not relieve this flooding until true multicast routers are in wide-spread use, and even then, replies to the query may cause congestion and overloading. By the same token, Web caching cannot achieve high hit rates *unless* cache groups are large because there is not enough overlap in clients requests within smaller groups. Cache hierarchies attack the size problem by organizing groups into cache levels, where each level exponentially expands cache sharing. Although hierarchies improve overall hit rate, the response time suffers because each cache level miss spawns another remote network transfer of the object. Thus query-based discovery cannot achieve both high hit rates and low overall response time.

5.2.3 Cache site directory discovery

With *cache site directories*, clients locate where object is cached by looking in a directory. The protocol may maintain complete or partial information in cache site directories. If the directory information in not complete, the protocol must specify a backup method for retrieving objects. The directory may be either centralized or distributed, although centralized directories lead to congestion and bottlenecks. One simple distributed cache site directory maintains local, unconnected directories on each proxy server. In this case, the proxy looks for the object only in its local cache site directory. With more complex directory organizations, the proxy has access to location

information in other directories, but must spend time finding and contacting those directories. In other words, retrieving cache location information generates the same discovery, dissemination and delivery issues encountered when retrieving the object. However, directory entries are much smaller than Web objects, which makes prefetching more attractive and information propagation less costly.

5.3 Dissemination

5.3.1 Client-initiated dissemination (*Pull-caching*)

Client-initiated dissemination is a client-driven "pull" technology in which clients decide what, when and where to cache. The server may be stateless because it does not need to know client requests patterns; the client-driven nature automatically matches object distribution to request patterns. The problem of cache consistency arises if servers are stateless, but this not associated *per se* with client-initiated dissemination.

The major advantage of client-initiated dissemination is that it automatically adapts to rapidly changing request patterns. This is critical when hot spots or flash crowds occur. Because Internet HTTP traffic exhibits dynamic behavior on a global scale, dissemination *must* be client-initiated if it hopes to match demand. For this reason, our proposed design uses client-initiated dissemination but combines it with stateful servers to improve discovery and reduce cache consistency problems.

5.3.2 Server-initiated dissemination (*Push-caching*)

Server-initiated dissemination is a server-driven, "push" technology in which servers choose what, when and where to cache. However, caches may have the authority to refuse to store objects or remove objects without notifying the server. In server-initiated dissemination a cache acts as a proxy for servers, not for clients.[2]

Replication is a server-initiated dissemination protocol that restricts the authority of the cache sites to refuse or remove objects. On the Internet, replication is commonly associated with "mirror

[2]In the server-initiated dissemination design of Bestavros, et.al. [BeCu96], one set of caches functions as proxies for servers ("service proxies") and a second set of caches functions as proxies for clients ("proxy servers"). The set of service proxies and the set of proxy servers may overlap. That is, a service proxy may or may not be a proxy server.

sites" that duplicate an entire Web site. In general, server-initiated dissemination operates on a finer scale than this by sending individual objects to a cache.

Server-initiated dissemination typically uses stateful servers because they need historical request data in order to decide where to disseminate objects. This server-centric approach provides strong consistency and assures objects with long-term demand are not prematurely replaced by short-term, "hot" items. The major disadvantage of server dissemination is that it does not cope well with rapid or localized changes in request patterns [GwSe95]. A second disadvantage is a resource and security issue: cache sites must agree to store unsolicited objects sent by the server instead of storing only those objects requested by their local users. This raises serious concerns on the Internet, where sharing occurs across different companies and governments. Lastly, a centralized protocol produces a bottleneck and a single point of failure; the clients cannot locate cache sites if the server is unreachable. In summary, server-initiated dissemination works best for objects with long-term or static request distributions, or for objects whose consistency is more important than response time, and is best suited for an autonomous internetwork.

5.4 Delivery

5.4.1 Direct delivery

Direct delivery assures the object is always returned directly from the "hit site" to the client, where the hit site refers to the cache or server where the object is found. This method assures the lowest latency for delivery.

5.4.2 Indirect delivery

Indirect delivery protocols return the object from the hit site to the client through the intervening cache sites. Here delivery usually follows the reverse of the request path. Intervening cache sites may store the object as it passes through; this combines delivery with dissemination. Because indirect delivery often involves more than one remote transfer, it makes for longer and more variable response times and increases the number of packets on the network.

5.4.3 Constraints of HTTP protocol on delivery in multi-level caches

The HTTP protocol has no provision for returning an object to locations other than the one which sent the request. This has repercussions for multi-level cache protocols where the request may be forwarded through several caches before the object is found. Consider the following scenario:

A connects to B and requests an HTTP object

B misses, so B connects to C and requests the object

C hits.

Using HTTP, C cannot return the object directly to A for two reasons: 1) C does not know A requested the object, and 2) C cannot return the object by initiating a connection with A. A expects the object to be returned via its client role in the A-B connection, not via a server role in a new C-A connection. Consequently, multi-level Web caches must either use indirect delivery or must modify HTTP to overcome these problems. To date, no researchers have addressed such modifications.

6 Related research on Web caching infrastructure

Our taxonomy facilitates the comparison of cooperative Web caching projects because it separates basic design from implementation details. If the basic design does not fit Web workloads, no implementation will achieve acceptable performance.

Combining all possible methods in the taxonomy for discovery, dissemination and delivery yields sixteen taxonomy classes. One class is prohibited by the direct connection requirement of HTTP protocol, leaving us with fifteen possible classes (see Section 5). Proxy server caching falls into Class 1, identified by fixed cache discovery, client-initiated dissemination and direct delivery.[3] Table 5 classifies six cooperative Web caching projects and our SDP protocol. Research by other groups falls into three taxonomy classes (Classes 2,3 and 4), while our SDP designs fits into Class 5. Table 5 lists research project names, authors and their methods for discovery, dissemination and delivery.

[3] In this discussion, we assume the use of proxy server caching and analyze protocols from the standpoint of proxy server requests. That is, a protocol is classified as having direct delivery if objects requested by a proxy server are returned directly to the proxy server.

Class	*Discovery*	*Dissemination*	*Delivery*	*Project*	Reference
1	Fixed cache	Client-initiated	Direct	Proxy server caching	[Ap97]
2	Group query, Manual	Client-initiated	Indirect	Harvest/Squid hierarchical caches	[ChDa96]
				NLANR and other national caches	[NLANR-98]
3	Group query, Auto	Client-initiated	Indirect	Adaptive Web Caching	[ZhFl97]
				Cooperating WWW cache servers	[MaLo95]
4	Cache site directory	Server-initiated	Direct	Geographic Push-caching	[GwSe95]
				Demand-based Document Dissem.	[Bes95]
5	Cache site directory	Server-initiated	Direct	Metadata hierarchy	[TeDa97]
	Cache site directory	Client-initiated	Direct	Server-Directed Proxy Sharing	[JeDa97]

Table 5: Classification of some cooperative Web caching projects.

6.1 Hierarchical caches: Harvest, Squid and NLANR

The Harvest project produced an integrated set of tools for gathering information from the Internet, including the Harvest hierarchical proxy server caches [ChDa96]. Although the project ended in 1996, development of its proxy server cache continues as the Squid Internet Object Cache [Sq97]. Harvest and Squid caches can be configured into a static hierarchical structure by assigning *siblings* and *parents* to each cache. When a cache misses, it sends queries to its group of siblings and to its parents, and a "hit" echo to the server. Siblings, parents and the server return either a "hit" or "miss" message. If the cache receives a hit before timing out, it requests the object from the fastest hit site. Otherwise, the cache requests the object from the fastest parent. In the Harvest hierarchies, proxy servers are manually configured to identify siblings and parents. When a cache enters or leaves the system, its siblings and children must be reconfigured. For an Internet-wide system this manual reconfiguration is cumbersome and error-prone.

In the United States, the National Laboratory for Network Research (NLANR) is currently testing a multi-level hierarchical cache system on the Internet built upon Squid proxy caches. NLANR supplies six networked workstations for the top level of the hierarchy. Proxy servers constitute the intermediate levels, and users make up the bottom level. If an NLANR cache

misses, it contacts its sister caches via NSF's high speed vBNS network. If the sister caches miss, the original NLANR cache retrieves the object from the Web server, caches it and returns it to the requesting child cache site. If the request did not originate at an NLANR child, the object is passed back down the request chain until it reaches the original requestor. Cache misses in NLANR are expensive because the request and delivery packets travel through multiple remote HTTP connections: user-proxy, proxy-cache_0, ..., cache_i-NLANR, NLANR-server.

6.2 Automatic query-based discovery using IP multicast

Zhang, Floyd, and Jacobson propose an adaptive web caching protocol which partitions users into multicast groups for discovery and dissemination [ZhFl97]. Multicast allows for automatic group configuration: caches join and leave groups without having to reconfigure other group members. Within a group, the request is multicast to all group members. If no member cache supplies the object, the request is forwarded by a cache which also belongs to the overlapping group closest to the server. Returns from the server follow the reverse route, with the object multicast to every group along the way. Acceptable performance of this design will likely require hardware IP multicast to avoid flooding the network with queries. The other question that arises is how group members decide which overlapping group lies on the shortest route to the server. This seems to require a tight integration between the cache and the router, taking caching out of the applications layer.

Malpani, Lorch and Berger propose a multicast discovery design which uses a single group [MaLo95]. If the group misses, the protocol resorts to contacting the server. Here the problem is scalability: a single group design does not scale to the Internet, but smaller groups will suffer from the same lack of available sharing that limits proxy servers hit rates.

6.3 Push-caching projects

In geographical push-caching, Gwertzman and Seltzer use stateful servers that decide what, when and where to cache [GwSe97]. Clients send requests directly to the Web server, which guides the client to a "nearby" cache. Servers decide which cache is nearest the client by comparing geographic and network topology (hop count) information. The author's simulation estimates adding geographic push-caching to proxy server caching reduces server load (bytes) by 2% as compared to proxy server caching alone. Even if they had produced better results for server load,

the simulation gives no information about response times because on the Internet neither geographic distance nor hop count correlates to latency [CrCa95].

Bestavros and Cunha outline a similar push-caching design [Bes95], termed demand-based dissemination. Using trace-driven simulation, they find their design reduces network traffic (byte-hops) by 40%-50% over no caching. For proxy server caching, they find a 36%-50% traffic reduction. These results suggest that their push-caching design, like that of Gwertzman and Seltzer, offers insignificant improvement over proxy server caching.

7 Server-directed proxy sharing (SDP)

7.1 Design rationale

In previous sections we have shown why the Web needs a cooperative caching infrastructure, we described Web server workloads and we analyzed alternative methods for cache discovery, dissemination and delivery. With this background in place, we describe the server-directed proxy sharing protocol (SDP) and explain the rationale for our design choices. In terms of our Web cache taxonomy, SDP belongs to the class with server-directed discovery, client-initiated dissemination and direct delivery. Each of these choices is explained below.

7.1.1 Why server-directed discovery?

A huge number of servers is attached to the Web. Even though each Web server satisfies most requests from a small fraction of its pages, clients access many servers. Consequently the total number of popular pages is enormous. Web caching can achieve high hit rates and low response times only if

1. objects are distributed across many cache sites,
2. clients can access most of these caches,
3. clients know with high probability where the desired object is cached.

In query-based discovery, clients locate the object by querying each member of a cache group. If the cache group is too large, the discovery will generate a huge number of query packets that

congests the network and causes long latencies. However, Web caching needs large cache groups in order to achieve high hit rates because clients request so many different documents. Cache hierarchies solve the size problem by moving requests through cache layers, where each layer exponentially expands cache sharing. Although overall hit rate improves, response time suffers because each cache layer traversed by the request adds another remote network transfer of the object. Cache hierarchies have the additional problem that as the number of groups grows, the upper level caches themselves become service bottlenecks. This requires either the higher cache levels be made more powerful or more levels be added to the hierarchy. Adding service power is not a scalable solution, and adding more levels harms response time and network congestion. Thus query-based discovery cannot achieve both high hit rates and low response time, even if caches are organized into hierarchies.

Server-directed discovery fulfills all three criteria listed above: objects are distributed, clients can access most caches, and clients know where to look for objects. The drawback of server-direction is that clients must still initially contact the server. For large objects, server contact overhead is small compared to the savings in server load and network traffic because a small redirection message is returned instead of a large object. For small objects, server-directed discovery may not save much time or traffic. In SDP we avoid contacting the server for many small objects by two methods: 1) when a server returns an HTML text file, it attaches a list of cache sites for the embedded images, and 2) when a cache site returns an object, it piggybacks a list of its most popular Web pages. These methods provide lazy, low overhead distribution of cache site information that substantially reduces the need for server contact.

7.1.2 Why client-initiated dissemination?

Dynamic and unpredictable request patterns are a fundamental characteristic of Web traffic. As discussed in the taxonomy, client-initiated dissemination adapts more quickly to rapidly changing request patterns than does server-initiated dissemination because it is driven by the current request stream rather than by a history of past requests. Better dissemination reduces server load by improving cache hit rates, and reduces network congestion by storing objects at sites nearer the requesting clients.

7.1.3 Why direct delivery?

Indirect delivery increases response time for the current request because it adds extra TCP/IP connections and remote packet transfers. Although direct delivery is obviously faster, HTTP semantics require multi-level Web caches use indirect delivery. These semantics provide an argument for a flat organization of Web caches.

7.1.4 Using Web page organization in the protocol

A Web page consists of HTML text and a set of embedded images (GIF, JPEG, etc.). We treat a Web page as a single entity; that is, we assume a request for an HTML file will often be followed by requests for the embedded images. The SDP protocol anticipates requests for embedded images by piggybacking a list of cache sites for the page when it returns the page's HTML text. Requestors then retrieve the images in parallel from multiple sites on the list. This eliminates server contact for embedded images, reduces response time and distributes network traffic. In the two UTSA traces, 37% and 46% of the transfers are embedded images, so using Web page organization would eliminate from over one-third to nearly one-half of the server contacts.

7.2 SDP components

The SDP design consists of servers, proxy servers and four types of information table: Proxy Tables, Proxy Lists, Cache Site Directories and Popular Lists. Each SDP server maintains a Proxy Table from which it builds the Proxy List for a requested Web object. Each SDP proxy server maintains a Cache Site Directory and a Popular List. We discuss these components before explaining steps in the SDP protocol.

7.2.1 Servers and proxy servers

SDP servers are Web servers which return either the requested Web object, a list of cache sites for the object, or both. SDP proxy servers are proxy servers that share their cached objects with other proxy servers in the SDP mesh.

7.2.2 Proxy Table and Proxy Lists

SDP servers maintain a *Proxy Table* which points to recently requested objects and the SDP proxies that requested them. Specifically, a Proxy Table consists of <*object, proxy, timestamp*> triples, where *object* is an URL or pointer to an URL, *proxy* is an IP address, and *timestamp* records the time of the request.

A *Proxy List* is a list of cache IP addresses for the requested object. When a server receives a request, it compiles a Proxy List for the requested object from entries in the Proxy Table. The server need not include all cache sites in the list; Section 7.3 describes how the server decides which cache sites to include.

7.2.3 Cache Site Directory and Popular List

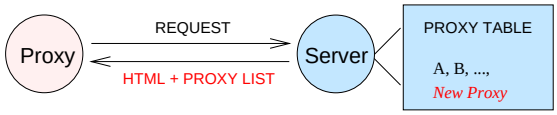
Each SDP proxy server maintains two tables: 1) the *Popular List* containing URLs of the Web pages most frequently requested from the proxy's own cache, and 2) the *Cache Site Directory* which contains Popular Lists from other proxy servers. Entries in the Cache Site Directory have the same <*proxy, object, timestamp*> format as the Proxy Table. The directory is not intended to be complete — it only holds cache site information that was obtained by lazy distribution. Cache information is distributed when a proxy server requests an object from an SDP cache and the cache piggybacks its Popular List onto the returned object. This optimization diffuses cache site information at a low cost because it uses an existing connection and does not send many bytes. Popular Lists will likely contain a small number of object URLs, adding perhaps a few hundred bytes to the transfer. Using this method, clients have a low-cost means of finding cached objects without contacting the server.

7.3 SDP protocol

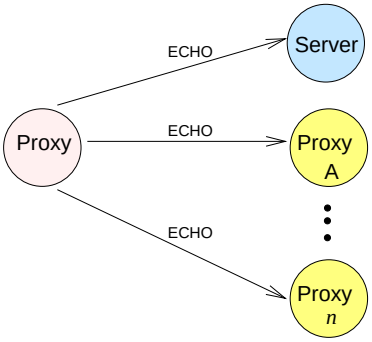
In the discussion below, the "client proxy" is the SDP proxy server requesting the object on behalf of its local users. Cache sites are SDP proxy servers that have cached the desired object. To follow the discussion, refer to the SDP scenario illustrated in Figure 3.

1. **User sends request to proxy server**

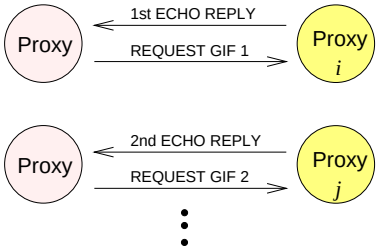
1. Client proxy requests Web page from Server.
   Server returns HTML file and a list of proxies for embedded images.
   Server enters <proxy, URL, time> in Proxy Table.



2. Client proxy selects candidates from list by timing ICMP Echoes.



3. Client proxy requests different images concurrently from the fastest candidates.



4. Each proxy returns an image and a list of popular Web pages in its cache.
   Client adds popular lists to its Cache Site Directory.

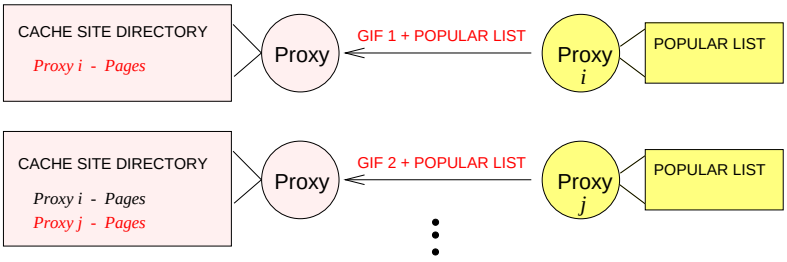


Figure 3: Server-directed proxy sharing after a Cache Site Directory miss.

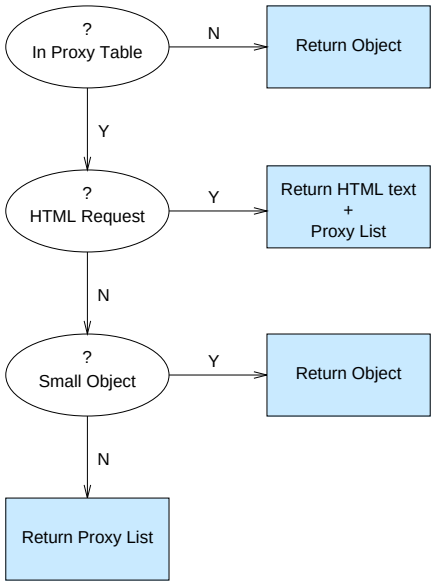


Figure 4: How an SDP server decides whether to return object, Proxy List or both.

When a browser cannot satisfy a request from its own cache, it contacts its proxy server. The proxy server checks its cache and returns the object, if it is there.

2. **Client proxy searches its Cache Site Directory**

   If the proxy server does not have the object, it looks in its Cache Site Directory for other SDP proxies that have cached it. If the directory lookup is successful, the proxy does not need to get cache sites from the server and skips to latency measurements in Step 6.

3. **Client proxy sends request to SDP server**

   When the proxy's request arrives, the SDP server decides whether to return the object or the Proxy List or both. The server bases its decision on three factors (see Figure 4):

   Is the object listed in the Proxy Table and are the cache copies current?

   Is the object an HTML file?

   Is the object below a size threshold?

   If no proxies have a current copy, the server returns the object. The server determines if a cache copy is current by comparing the object's age to the cache timestamp recorded in the Proxy Table. To avoid accessing the file system for each request, the server may assume all

entries are current when it processes the requests and periodically removes out-of-date entries from the table during slack usage periods. Although checking dates off-line weakens cache consistency by sometimes returning out-of-date files, it is more efficient and speeds up server response. In addition, the probability of returning an out-of-date file is small because Web documents usually have long lifetimes. Gwertzman and Seltzer report the average lifetime to be 50 days for HTML files, 85 days for GIF files and 100 days for JPEG files [GwSe96]. Bestavros indicates that more popular Web documents are updated less frequently; he reports the update probability for a popular Web document is less than 0.5% per day [Bes97]. Another problem arises because servers do not know when a cache deletes an object. We use a "cache life" parameter to estimate how long an object will remain cached before being replaced. Proxy Table entries older than the cache life are considered no longer valid and are removed.

If the requested object is an HTML file, the server returns both the HTML text and a Proxy List because it assumes the client usually wants the whole Web page. The client proxy can then use the Proxy List to retrieve embedded images from cache sites.

If the object is below a size threshold, the server returns it directly; otherwise the server returns a Proxy List for the objects. An SDP server establishes its own threshold size based upon current server utilization, network conditions and local policy. By taking into account current conditions, the SDP protocol adapts to variations in server load and network traffic. For example, if the server is busy or its backbone connection appears congested, it reduces the threshold size and thereby puts fewer bytes onto the network.

4. **Server culls the Proxy List**

Suppose the server decides to return a Proxy List and finds a large number of proxies have cached the object. How many proxies should the server return and which ones should it return? Proxies are identified by their 4 byte IP address so each proxy adds only 4 bytes to the return message. The size of the Proxy List is limited more by the method clients use to select proxies from the list (see below) than by the extra message bytes. For example, a maximum size of 50 proxies adds fewer than 200 bytes to the return message.

How a server culls the Proxy List is not too critical because this is just a first pass. The client proxy makes the final selections by measuring current latencies from itself to the candidate

proxies. The server need only roughly estimate which proxies might be faster. Several static criteria could be used, including the following:

- *Bottleneck link capacity between the candidate proxy and an Internet backbone*

  (Does the candidate have a 56 Kbps connection, a T1 line, or a T3 line?)

- *Network proximity*

  (Are the client proxy and candidate proxy on the same regional network or backbone?)

5. **Server updates its Proxy Table**

   The server assumes the client proxy will cache the object, and adds the <object, proxy, timestamp> entry to its Proxy Table, regardless of whether the server returned the object or a Proxy List.

6. **Client proxy measures cache latencies**

   At this point the client proxy has a list of candidate SDP proxies, obtained from either its own Cache Site Directory or from the server's Proxy Table. Now the client proxy must select sites from the list. Its main concern is speed — the client wants the site with the fastest response time. Because response times fluctuate rapidly and unpredictably [CrCa95], the client proxy selects a cache site from the list by measuring the current round trip latencies for ICMP echo packets.

7. **Client proxy sends concurrent requests for embedded images**

   If the client proxy wants just one object, it sends the request to the first candidate proxy that responds to the echo. The request is sent as soon as the echo response arrives, without waiting for responses from other candidates.

   Typically, however, users want a complete Web page consisting of an HTML file and a set of embedded image files. The client proxy retrieves page components concurrently by requesting different objects from different SDP proxies. As soon as response arrives from a candidate proxy, the client proxy sends it a request for the next page component without waiting for other responses to arrive. For example, the fastest candidate proxy gets the request for image 1, the second fastest candidate proxy gets the request for image 2, and so on. A proxy can

receive more that one request if it finishes while more requests remain and all other proxies are still working on their requests (or have not responded to the echo packet). This selection process essentially functions like a single queue at a multi-server facility.

Two features safeguard against excess delay. First, the server is included in the candidate list and, if fast enough, may be sent a request.[4] This makes sense because by being faster than some proxies, the server demonstrates it has more load capacity or a less congested route and *should* be used. Second, the client proxy sets reply timers. If a timer expires, the client sends a duplicate request to another proxy (or to the server). The SDP protocol includes a provision to cancel the slower request after its duplicate response arrives.

8. **Cache returns object or refuses request**

   Caching proxies have the option of refusing an SDP request. Requests may be refused, for example, because the cache removed the object, the proxy is busy or, as a matter of policy, the proxy does not service any requests from this client. If the request is refused, the client proxy sends the request to the next fastest cache site.

9. **Cache piggybacks its Popular List to object**

   When an SDP proxy returns an object, it attaches a list of the p most popular items in its cache, where the client proxy specifies p in its request. The list contains a small number of URLs, requiring perhaps a few hundred bytes. The client proxy stores this information in its Cache Site Directory for possible future use.

8 Analytical simulation

The second phase of dissertation work models the SDP protocol with an analytical event-driven simulator. A simulator allows us to study how the protocol performs under various workloads, to fine-tune details of the protocol and compare the protocol to other cooperative Web caching designs.

[4] At this stage in the protocol the server is sent an HTTP request rather than an SDP request to guarantee it returns the object instead of a Proxy List.

8.1 Including network traffic in the simulation model

Because the behavior of a web caching system depends heavily on the network topology and traffic, an ideal simulator would include the network in its model and would compute end-user latency. To date, web caching simulations reported in the literature have used much simpler single server models with trace-driven workloads, and have computed server load in *bytes/sec* or *requests/sec* rather that response times and denial-of-service probabilities [Bes95], [GwSe95], [MaLo95], [MaDu97], [ZhFl97]. One aspect of our work is to investigate the modeling of a web-caching system using a network simulator. From this simulation, we hope to predict relative response times. As this involves a large and rather complex simulation, we first built the isolated single server simulator described below. Our next step will be to study a detailed network simulation package and investigate how to adapt it to model web-caching systems, and how to build a global, multi-server request stream.

8.2 Analytical vs. trace-driven workloads

The advantage of analytical simulation over empirical trace-driven simulation is that analytical workloads are easily varied, allowing systematic investigation of the protocol and predictions for different workloads. In addition, analytical studies help us better understand the underlying system by separating the effect of each workload variable. The ability to predict system behavior is especially important in systems where the traffic patterns vary between sites or over time. This is certainly the case with the Internet and its HTTP traffic. The disadvantage of analytical modeling is that it is less accurate and less detailed than empirical modeling, which captures a precise movie of a single workload over a limited period of time. Previous simulation studies in web caching have relied on traces [BeCu96], [GwSe97]. We deemed the flexibility of analytical simulation more important than the precision of trace-driven simulation because we wish to evaluate protocol performance as a function of offered load, and because we expect the volatility of web traffic to continue over the foreseeable future.

8.3 Isolated single server model

Our single server simulation models the SDP protocol and proxy server caching from the server's point of view, using distributions and probabilities compiled from server traces in the literature [ArWi96], [GwSe97], [Bes97], [Mo95] and from our own UTSA server traces. Table 6 lists

Workload Variable	*Model Distribution*	*Parameter Values*	Request Probability
Interarrival times (sec)			
Session	Exponential(a)	*varied*	
Embedded images	Log_{10}-normal(a, b)	$a = -0.7$, $b = 0.3$ ($mean = 221ms$)	
Connection duration (sec)	Log_{10}-normal(a, b)	$a = -0.75$, $b = 0.65$ ($mean = 289ms$)	
File size and probability			
HTML	Pareto(a)	$a = 4$ KB	$P = 0.430$
Image	Pareto(a)	$a = 11$ KB	$P = 0.506$
Audio	Pareto(a)	$a = 140$ KB	$P = 0.003$
Video	Pareto(a)	$a = 452$ KB	$P = 0.004$
Application	Pareto(a)	$a = 260$ KB	$P = 0.007$
Dynamic	Pareto(a)	$a = 1$ KB	$P = 0.019$
Other	Pareto(a)	$a = 11$ KB	$P = 0.031$
Page Request	Fixed		$P_{page} = 0.13$
Embedded images per page	Normal (a, b)	$a = 3.2$, $b = 1$.	

Table 6: Simulation workload parameters.

the workload distributions, parameters and probabilities used in our simulation. Because SDP treats web pages differently from other requests, we used the UTSA traces to determine the probability of web pages requests (0.13) and the average number of embedded images per page (3.2). Web pages consist of HTML text plus one or more embedded images. If a request is not for an embedded image, it is considered a *session* request. Session requests are web page HTML files or non-page objects such as cgi, video, audio or non-embedded image files. Table 7 shows the percentages of session, embedded, and page requests in our UTSA traces, as well as the average number of embedded images per page.

In our simulations, we assume all clients use proxy server caches and all proxy servers belong to the SDP mesh. Although both assumptions are unlikely, we wish to show the performance that would result from using proxy server caching and SDP. If fewer clients use proxy server caches or fewer proxy servers belong to the SDP mesh, the performance would be lower than predicted.

Request category	*Trace UTSA-1*	*Trace UTSA-2*
Session	59%	54%
Embedded image	37%	46%
HTML total	43%	42%
HTML Web page	12%	13%
Embedded images per page	2.9	3.5

Table 7: Percentages of request types measured in UTSA traces.

8.3.1 Interarrival times (T_a) and service times (T_s)

We use two distributions for interarrival times (T_a): Poisson for session requests and log-normal for embedded images within a session. Previous work shows overall HTTP interarrival times may not be Poisson, but these analyses did not distinguish between session requests and embedded image requests. Paxson and Floyd did this study for FTP traffic and showed FTP session arrivals appear Poisson, while FTPDATA connections spawned by the session are better approximated by a heavy tailed distribution such as log-normal or log-logistic [PaFl94]. For proxy server caching, we assumed the client uses parallel HTTP; that is, all embedded image requests are sent concurrently rather that sequentially. We model the average interarrival time of a group of parallel embedded requests as a heavy-tailed log-normal distribution with respect to the end of the corresponding HTML transfer, and randomly distribute arrivals of individual image requests around this mean. As our embedded image arrival times model the time it takes for the HTML file to reach the client plus the time for the corresponding embedded image request to reach the server, we used arrival time parameters that approximate the average time (207 ms) for 536 byte `pings` reported by Mogul [Mo95].

The distribution of service time (T_s) is based on our interpretation of Mogul's graph of connection durations in a study of DEC server traces [Mo95]. For well-connected servers, the connection duration will likely depend upon the proxy's network bandwidth, and as such the duration distribution should be similar for most servers.

8.3.2 Simulation metrics

Our parameter for offered request load is the *request intensity* (T_s/T_a), where T_a reflects the request rate generated by client proxies. In proxy server caching, the server receives all these requests. In

SDP many of the embedded image requests are never seen at the server because they are sent to cache sites. Hence the request rate at the SDP server is lower than the request rate generated by the client proxies. The larger the request intensity, the busier the server. For proxy server caching, the server becomes overloaded and starts refusing connections at $T_s/T_a = 1$.

We compared SDP to proxy server caching by computing server load metrics (*arrivals/sec* and *MB returned/sec*) and the percent of refused connections as a function of request intensity. Previously reported web caching simulations concentrate on server load and do not compute a denial-of-service metric. We feel that the percent of refused connections is important to users and should be estimated when evaluating Web caching protocols.

8.3.3 Simulation results

Figures 5 and 6 show results for server load (*MB returned/sec* and *arrivals/sec*), and Figure 7 shows results for denial-of-service probability as a function of the request intensity T_s/T_a. We believe the region of interest lies in the range $0.5 \leq T_s/T_a \leq 1.5$, where servers become busy and begin to refuse connections. Our results indicate that SDP can help substantially, reducing both server load and the fraction of denied requests. In the critical range 0.5 to 1.5, SDP improves upon proxy server caching by 41% to 63% for traffic load (MB/sec) and by 26% to 38% for request load (requests/sec). Past the critical point of $T_s/T_a = 1$, the advantage of SDP levels off to approximately a 35%-40% reduction in traffic load and a 10%-15% reduction in request load. More importantly, SDP starts refusing connection requests well after proxy server caching begins refusals, and continues to refuse a smaller number of requests as intensity increases.

9 Prototype implementation

The third phase of the dissertation work implements and tests prototype SDP server and proxy server software. To implement the prototype, we will modify the publically available Apache Web server and proxy server. [Ap97], [La97].

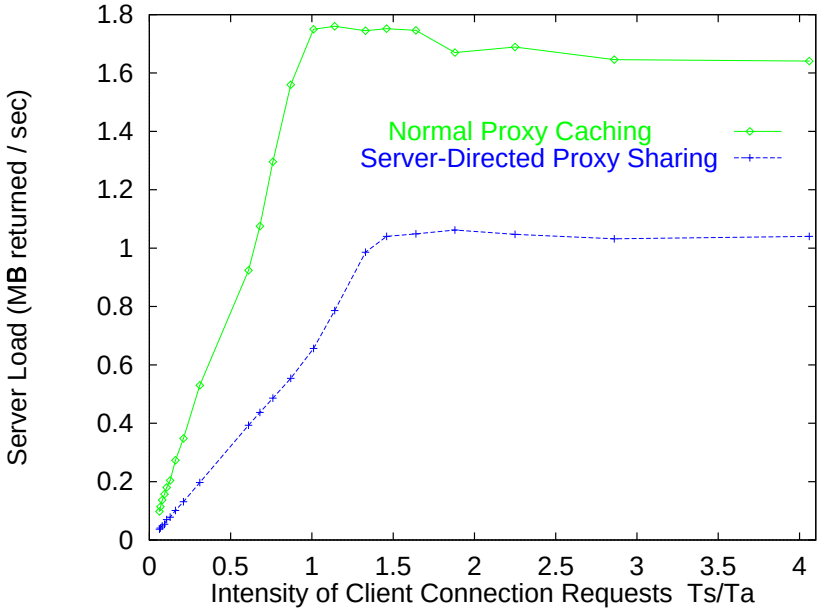


Figure 5: Simulation results: server load (MB transferred /sec) as a function of request intensity.

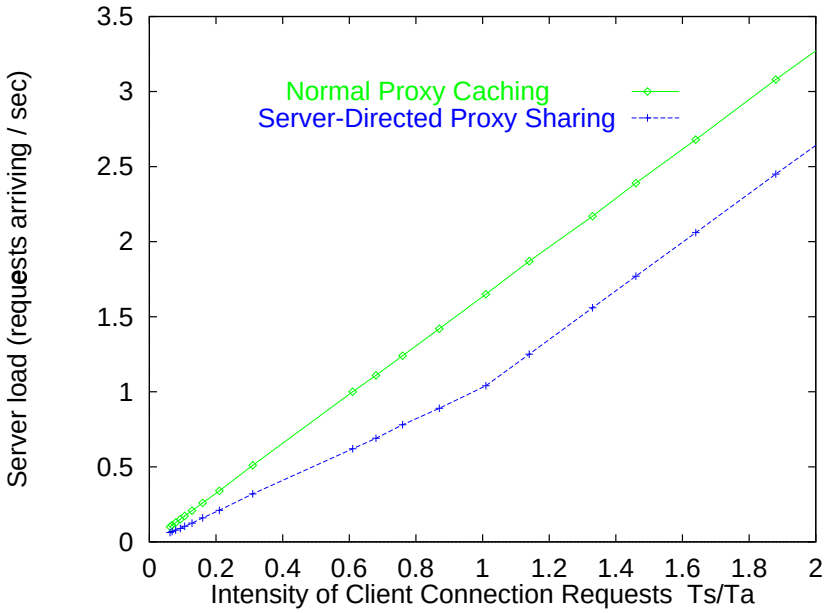


Figure 6: Simulation results: server load (requests arriving /sec) as a function of request intensity.

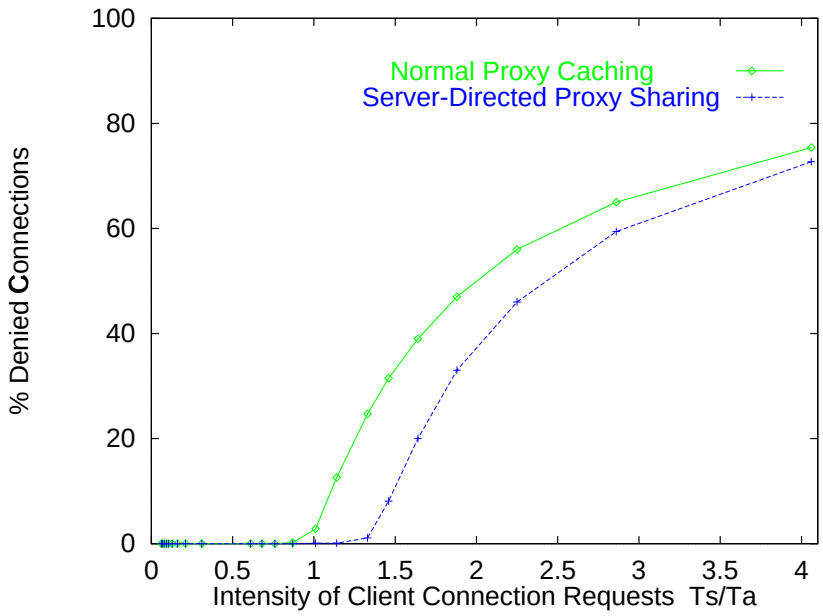


Figure 7: Simulation results: denial-of-service probability as a function of request intensity.

9.1 Prototype SDP server

In the envisioned prototype implementation, the SDP server sits between arriving SDP requests and an unmodified HTTP server. The SDP and standard HTTP servers coexist on the same machine, each listening at its own designated port. With this design we need only to implement SDP functionality and can rely on the HTTP server for normal Web services. When an SDP request arrives, the SDP server spawns a thread and goes back to listening at the port. If the SDP thread wishes to return a proxy list, it looks up the object in the SDP proxy table and checks its timestamp. If the SDP server wishes to return an object, it first retrieves the object from the HTTP server then returns it to the SDP client. SDP uses HTTP error codes, allowing the SDP server to return the same error code it received from the HTTP server. The HTTP server is not modified as it need not distinguish between requests from an SDP server and requests from other clients.

9.2 Prototype SDP proxy server

The second component of the prototype is the SDP proxy server. The SDP proxy server sits between a standard proxy server and the Internet. It has several jobs:

1. receive requests from the HTTP proxy server, convert them into SDP requests and send them to an SDP server,

2. implement the SDP client-side protocol, including run-time latency measurements of candidate SDP cache sites,

3. return retrieved objects to the HTTP proxy server,

4. listen for incoming requests from other SDP proxy servers and supply them with objects from the cache of the HTTP proxy server.

Conceptually, the SDP proxy server builds a shell between the Internet and the HTTP proxy server. Similar to the HTTP server in the prototype, the HTTP proxy server does not need to be modified. However, the HTTP proxy server must be configured to direct all outgoing requests to the SDP shell rather than to remote Web sites. This configuration option is available in most popular proxy servers, including Apache and SQUID.

Separation of SDP and HTTP components should simplify the prototype implementation and speed up development time. In addition, separating SDP and HTTP servers makes the SDP prototype compatible with a variety of HTTP servers, which increases the chances of other sites installing an SDP server to try it out. Performance may be better in a more complex implementation where SDP and HTTP are combined in one server, but this is not planned as part of the dissertation work.

9.3 Evaluation of prototype

We can test the prototype for correctness and evaluate some aspects of its performance on the UTSA campus network. A more in-depth evaluation of the prototype would require SDP be installed at several remote sites, for which we would need to enlist the support of other researchers. While we hope other sites will install SDP, we cannot count on widespread distribution for the purpose of evaluating our prototype. In the worst case, our evaluation of SDP will rely on simulation work and prototype testing on the local campus network.

10 Summary

The SDP protocol is designed to match known characteristics of the Web. Among the most important of these are the limited overlap of requests within a user community, the unpredictability of object popularity, the rapid fluctuations in Internet traffic congestion, and the grouping of objects into Web pages. By organizing cache sites into a flat mesh and using server-direction, SDP can share cached objects across the entire Internet with a very high cache hit rate. A cache request only fails if the cache has deleted the object before the server anticipates it would be deleted. The main disadvantage of server-direction is that clients still contact the server for each request. SDP reduces this problem by piggybacking cache site information onto object transfers, by having the server return small Proxy Lists instead of larger objects and by treating Web pages as single entities.

SDP adapts to changing request patterns and network congestion by choosing a cache site based upon current round trip packet times. This overlaps user self-interest (low response time) with global network interest (low contention). Our preliminary simulation suggests SDP substantially reduces both server load and the number of connection refusals when compared to proxy server caching, and is especially effective in the range where servers begin to experience overloading. Although our preliminary simulation did not compute response times, they are a primary consideration in the SDP design and will be the subject of more detailed simulation study.

Finally, SDP is an administratively and economically viable approach to Web caching because it does not require autonomous local sites to change the contents of their caches, nor does it require monies for extra resources. In push-caching proposals, the local cache sites store unsolicited objects that their local user community may not need, raising security and resource-sharing questions. Hierarchical cache designs require a layer of dedicated cache sites which are supported by government funding, or potentially by a "pay-for-services" plan; neither is a particularly attractive option. In contrast, SDP requires only that local proxy servers share information they previously cached for their local users. Thus SDP matches both technical *and* administrative characteristics of the Web.

A Table of reported Web client, proxy server and server traces

Type	Trace	Referenced in
FTP	NSFNET	[Da93]
USER	Boston University	[CrBe96] [CuBe95]
	Virginia Tech	[AbSt95] [AbFo97]
PROXY SERVER	AOL	[AbSt95] [AbFo97]
	Auburn H.S.	[AbSt95] [AbFo97]
	Boston Univ.	[AbSt95] [Bes97] [CrBe96]
	Community college	[AbSt95] [AbFo97] [MaDu97]
	DEC	[AbSt95] [AbFo97] [Gl94-1] [MaDu97]
	Korea	[AbSt95] [AbFo97]
	Virginia Tech	[AbSt95] [AbFo97]
HIGHER LEVEL CACHE	NLANR	[NLANR-98][MaDu97]
SERVER	Boston Univ.	[AlBe96] [Bes95] [BeCu96] [Bes97] [GwSe96] [Ta97]
	ClarkNet (ISP)	[ArWi96]
	DEC	[Mo95]
	EPA	[AlBe96]
	Harvard	[GwSe96] [Sel96] [Sel97]
	Microsoft	[GwSe96] [Sel96] [Sel97]
	NASA Kennedy Space Center	[ArWi96]
	NCSA	[AlBe96] [ArWi96] [Sel96] [GwSe96] [Ta97]
	Rolling Stones	[Bes97] [BeCu96] [Sel97]
	SD Supercomputer Center	[AlBe96]
	Univ. of Calgary	[ArWi96]
	Univ. of Saskatchewan	[ArWi96]
	Univ. of Waterloo	[ArWi96]

Table 8: HTTP traffic traces.

References

[AbFo97] G. Abdulla, E. A. Fox, M. Abrams, and S. Williams, "Shared User Behavior on the World Wide Web", *Proc. of WebNet97*, Toronto, October, 1997.
http://www.cs.vt.edu/ chitra/docs/97webnet/

[AbSt95] Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, Stephen Williams, Edward A. Fox, "Caching Proxies: Limitations and Potentials", *Proc. of the 4th Inter. World-Wide Web Conference*, Boston, MA, Dec. 1995, pp 119-133.

[AlBe96] V. Almeida, A. Bestavros, M. Crovella, A. de Oliveira, " Characterizing Reference Locality in the WWW", *Proc. IEEE Conference on Parallel and Distributed Systems (PDIS'96)*, Miami Beach, FL, December 1996.

[Ap97] Apache Web Server,
http://www.apache.org/

[La97] B. Laurie and P. Laurie, *Apache, The Definitive Guide*, O'Reilly & Associates, Inc., Sebastopol, CA, 1997.

[ArWi96] M. F. Arlitt and C. L. Williamson, "Web Server Workload Characterization: The Search for Invariants", *Proc. ACM SIGMETRICS*, Philadelphia, PA, April 1996.

[Bes95] A. Bestavros, "Demand-based Document Dissemination to Reduce Traffic and Balance Load in Distributed Information Systems", *Proc. Seventh IEEE Symposium on Parallel and Distributed Processing (SPDP'95)*, San Antonio, TX, October, 1995.

[Bes97] A. Bestavros, "WWW Traffic Reduction and Load balancing through Server-Based Caching", *IEEE Concurrency*, Jan-Mar 1997, pp.56-67.

[BeCu96] A. Bestavros and C. Cunha, "Server-initiated Document Dissemination for the WWW", *IEEE Data Engineering Bulletin*, 19(3):3-11, September 1996.

[BeCa95] A. Bestavros, R. L. Carter, M. E. Crovella, C. R. Cunha, A. Heddaya, and S. A. Mirdad, "Application-Level Document Caching in the Internet", *Proc. of the Second Intl. Workshop on Services in Distributed and Networked Environments (SDNE'95)*, 1995.

[Bla95] M. A. Blaze, "Caching in Large-Scale Distributed File Systems", Technical Report TR-397-92, Princeton University, January 1993.
http://ncstrl.cs.princeton.edu/techreports/

[ChDa96] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, K. Worrell, "A Hierarchical Internet Object Cache", *Proc. of the 1996 USENIX Technical Conference*, San Diego, CA, January 1996.

[Co96] A. Coromack, "Web Caching", *Report to the United Kingdom Advisory Committee on Networking*, Sept. 1996

[CrBe96] M. E. Crovella and A. Bestavros, "Self-Similarity in World Wide Web Traffic Evidence and Possible Causes", *Proc. of the 1996 ACM SIGMETRICS Intl. Conf. on Measurement and Modeling of Computer Systems*, Philadelphia, PA, May 1996.

[CrCa95] M. E. Crovella and R. L. Carter, "Dynamic Server Selection in the Internet", *Proc. of the Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS'95)*, IEEE Communication Soc., New York, August 1995, pp.158-162. Also available as TR-95-014, Boston University Computer Science Department.

[CaCr96] R. L. Carter and M. E. Crovella, "Measuring Bottleneck Link Speed in Packet-Switched Networks", Technical Report TR-96-006, Boston University Computer Science Department, March 15, 1996.

[CuFi95] R. M. Cubert and P. Fishwick, "Sim++, Version 1.0", Technical Report, Dept. of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, 28 July, 1995.

[CuBe95] C. Cunha, A. Bestavros and M. E. Crovella, "Characteristics of WWW Client-based Traces", Technical Report TR-95-010, Computer Science Dept., Boston Univ., Boston, MA, 1995.

[CuJa97] C. R. Cunha and C. F. B. Jaccoud, "Determining WWW User's Next Access and Its Application to Pre-fetching", *Proc. of the Intl. Symposium on Computers and Communication '97*, Alexandria, Egypt, July 1-3, 1997.

[Da93] P. Danzig, R. Hall and M. Schwartz, "A Case for Caching File Objects Inside Internetworks", Technical Report CU-CS-642-93, University of Colorado, Boulder, 1993.

[DEC] Digital Equipment Corporation, "Digital's Web Proxy Traces",
ftp:
ftp.digital.com/pub/DEC/traces/proxy/webtraces.html/

[Gl94-1] S. Glassman, "A Caching Relay for the World Wide Web", *Proc. of the First Intl. World Wide Web Conference*, 1994, pp. 69-76.
http://www1.cern.ch/PapersWWW94/steveg.ps/

[Gl94-2] S. Glassman, "A Caching Relay for the World Wide Web", *Computer Networks and ISDN Systems* 27, No. 2, 1994

[GwSe97] J. Gwertzman and M. Seltzer, "An Analysis of Geographical Push-Caching",
http://www.eecs.harvard.edu/~vino/web/

[GwSe95] J. Gwertzman and M. Seltzer, "The Case for Geographical Push-caching", *Proc. of the 1995 Workshop on Hot Operating Systems (HotOS-V)*, 1995, pp. 51-55.

[Gwe95] J. Gwertzman, "Autonomous Replication in the Wide-Area Distributed Information Systems", *Technical Report TR-95-17*, Harvard University Division of Applied Sciences, Center for Research in Computing Technology, 1995.

[GwSe96] J. Gwertzman and M. Seltzer, "World-Wide Web Cache Consistency", *Proc. of the 1996 Usenix Technical Conference*, San Diego, CA January 1996.

[HeMi97] A. Heddaya, S. Mirdad and D. Yates, "Diffusion-based Caching Along Routing Paths", *Proc. 2nd Web Cache Workshop*, Boulder, Colorado, June 9-10, 1997.

[JeDa97] C. Jeffery, S. Das and G. Bernal, "Proxy Sharing Proxy Servers", *Proc. of the IEEE etaCOM Conference*, Portland, Oregon, May 1996.

[MaLo95] R. Malpani, J. Lorch and D. Berger, "Making World Wide Web Caching Servers Cooperate", Proc. of the Fourth International World-Wide-Web Conference, Boston, Massachusetts, December 1995.

[MaDu97] D. Marwood and B. Duska, "Web Proxy Traces and Simulator (SPA)", *Proc. of the 2nd Web Cache Workshop*, Boulder, Colorado, June 9-10, 1997.

[Mo95] J. Mogul, "Network Behavior of a Busy Web Server and its Clients", *Research Report 95/5*, Digital Equipment Corporation, October 1995.

[NLANR-98] "A Distributed Testbed for National Information Provisioning", http://www.ircache.nlanr.net/

[NLANR-97] "Tutorial: Insight Into Current Internet Traffic Workloads", http://www.nlanr.net/NA/tutorial.html/

[PaFl94] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling", *Proc. of the 2nd Web Cache Workshop*, SIGCOMM '94, September 1994.

[Ta97] I. Tatarinov, A. Rousskov and V. Soloviev, "Static Caching in Web Servers", *Proc. of the 6th IEEE Intl. Conf. on Computer Networks (IC3N'97)*, Las Vegas, Nevada, September 1997, pp. 410-417.
http://www.cs.ndsu.NoDak.edu/šoloviev/paperStatic.ps/

[Sel97] M. Seltzer, "Issues and Challenges Facing the World Wide Web", Presented at Lotus, March 1997,
http://www.eecs.harvard.edu/m̃argo/slides/lotus.html/

[Sel96] M. Seltzer, "The World Wide Web: Issues and Challenges", Presented at IBM Almaden, July 1996,
http://www.eecs.harvard.edu/m̃argo/slides/www.html/

[Sq97] "Squid Internet Object Cache",
http://www.squid.org/

[TeDa97] R. Tewair, M. Dahlin, H. M. Vin and J. S. Kay, "Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet",
Technical Report TR98-04, University of Texas at Austin, Austin, TX,

[WWW-1] "Replication and Caching Position Statement",
http://www.w3.org/pub/WWW/Propagation/Activity.html/

[WWW-2] "Propagation, Caching and Replication on the Web", http://www.w3.org/pub/WWW/Propagation/

[ZhFl97] L. Zhang, S. Floyd and V. Jacobson, "Adaptive Web Caching", *Proc. of the 2nd Web Cache Workshop*, Boulder, Colorado, June 9-10, 1997.