3W3 Cache Workshop
University of Manchester
15-17th June 1998

Computer Networks and ISDN Systems 30 (1998) 2245–2252

COMPUTER NETWORKS and ISDN SYSTEMS

LSAM proxy cache: a multicast distributed virtual cache

Joe Touch *,[1], Amy S. Hughes [1]

USC/Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90202-6695, USA

Abstract

The LSAM Proxy Cache (LPC) is a multicast distributed web cache that provides automated multicast push of web pages, based on self-configuring interest groups. The LPC is designed to reduce network and server load, and to provide increased client performance for associated groups of web pages, called 'affinity groups'. These affinity groups track the shifting popularity of web sites, such as for the Superbowl, the Olympics, and the Academy Awards. The LPC's multicast hierarchy is self-configuring, such that these popular affinity groups are automatically cached at natural network aggregation points. This document describes the LPC architecture and the properties of a prototype implementation.

Keywords: Multicast; Web; Proxy; Push; Cache

1. Introduction

The LSAM Proxy Cache (LPC) is Web proxy cache designed to support multicast Web push of groups of related Web pages. LPC reduces the response time for distributed clients that access related Web pages. It uses multicast to distribute these related page sets to a group of caches, reducing both server and network load.

Caching is the most common form of Web performance enhancement; Web caches are typically deployed at the client and at intermediate shared proxies. Client caches often achieve limited benefit, because the users tend to browse a variety of information, much of which is retrieved for the first time by that client. Pages requested for the first time are never in client caches.

Shared proxy caches allow pages to be shared among a group of clients because they aggregate the responses; even first-time clients can hit in the cache, if a sibling client has recently requested a page. Even when only a small set of clients can fill the cache with new information, the rest of the clients benefit. However, shared proxy caches work only where requests can already be aggregated, e.g., near the border router of a domain. These border proxies are not sufficient for some types of traffic, notably which would aggregate across different borders.

There are several examples of groups of related Web pages that become popular over time, such as those of the Superbowl, the Olympic Games, and the Academy Awards. The content of these pages often shifts over time, as new events occur in the Olympics, or during the football playoffs. Even though these page groups become popular, there is no one place a proxy can be placed to avoid a

* Corresponding author.
[1] E-mail: {touch,ahughes}@isi.edu.

0169-7552/98/$ – see front matter
PII: S0169-7552(98)00253-0

hot-spot at the server. We call such groups of popular pages 'affinity groups'.

As a specific example, the winter Olympics Web pages are an affinity group, whose content evolves as the games proceed. Pages for events, such as ice skating, ski jumping, and bobsled, become popular as each event occurs, and as new pages appear with the results of the competition. In current Web cache systems, such pages always generate hot-spots, because they are globally interesting. The pages are also part of popular groups, but the page popularity is not known in advance, so client subscriptions would not exist, defeating page-'cast' systems.

In LSAM, a proxy tunes to the server's Olympic channel if its downstream clients are sufficiently interested in the general topic. When a new page appears, e.g., for downhill skiing, its results are multicast to the entire set of tuned-in proxies the first time it is requested by any client. A client near any of these proxies can retrieve the page from the proxy cache, benefiting as there were one global shared proxy for all pages in the affinity group. Tuning happens automatically, as clients tune to channels that correlate to recent requests, and servers create channels that correlate to groups of popular pages.

LSAM is developing a multicast distributed virtual cache, the LPC, to provide the benefits of a centralized shared proxy cache where no central proxy could suffice, i.e., for these affinity groups. It uses multicast to allow a set of proxy caches to emulate a single, central shared proxy cache.

The LPC pump tracks popular groups of Web pages, and multicasts them to LPC filter caches at natural network aggregation points. Clients can access Web pages locally, even when no nearby client requests a popular page, because the caching system as a whole decides what is popular.

LSAM's technique for addressing these issues is called 'active middleware'. Middleware is a set of services that require distributed, on-going process, and combine OS and networking capabilities, but cannot be effectively implemented in either the OS or the network. LSAM's middleware is active, providing self-organization and programmability at the information access level akin to 'active networking' at the packet level.

The remainder of this document describes the LPC architecture and its properties. The feature of self-organization is discussed, as are implementation issues, such as prioritization, channel management, routing, and mobility. Finally, techniques for detecting and reacting to hot-spots are presented, and prior and related work compared.

2. The LPC architecture

The LPC is a Web proxy cache deployed at various places in a network, with two distinct variations — a server proxy (called a pump), and the distributed set of client proxies (called filters). The pump multicasts Web pages to the filters based on affinity groups (Fig. 1). The filters together act as a single virtual proxy cache, where the request of any one client benefits the others.

The filter caches Web pages for downstream access, i.e., nearer the client. It monitors the announcement channel and looks for 'interesting' affinity group channels, i.e., correlated to recent incoming requests. The filter's cache is partitioned, allowing it to tune to multiple channels and accept multicast downloads without inter-channel cache interference. The cache tunes-out (de-tunes) a channel whose contents are no longer correlated to the cache's ongoing requests.

The pump monitors Web accesses at a server and creates multicast channels for affinity groups. Re-

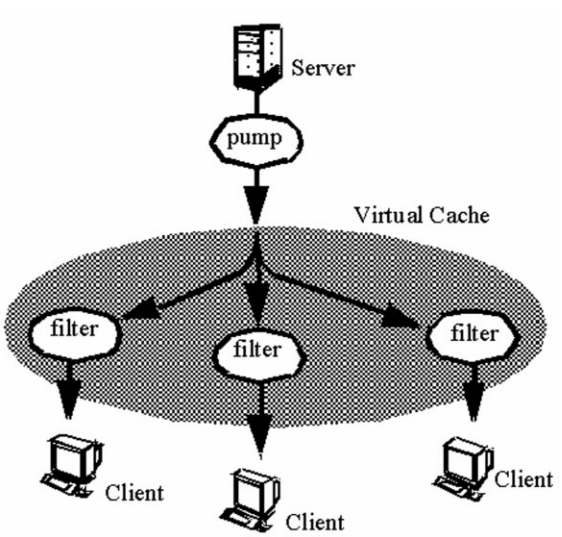


Fig. 1. LSAM components (the pump and filter are LPCs).

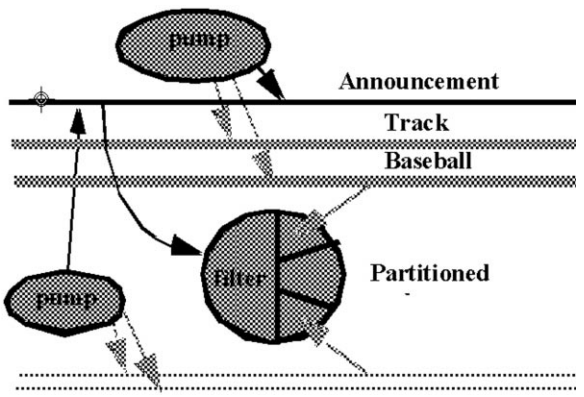


Fig. 2. Multicast channels in the LPC.

quests for Web pages in the affinity group are reliably multicast to the associated multicast channel using AFDP [1]. The pump creates and destroys multicast channels as different affinity groups become more or less popular; these groups are announced on a single, global multicast channel, similar to the teleconference announcement channel in the *sd* teleconference management tool [2] (Fig. 2).

In the LPC, individual requests trigger multicast responses when the pages are members of currently active affinity groups. A request is checked at intermediate proxies, and forwarded through to the server, as in conventional proxy cache hierarchies (Fig. 3, left). The response is multicast to filters in the group by the pump, and unicast from the final proxy back to the originating client (Fig. 3, right). Alternately, the initial response may be unicast back to the originating client, in addition to a multicast response to the group.

Subsequent requests for these popular pages are handled locally, from the filters nearer the clients (Fig. 4). These clients receive pages with in-cache response times, even for pages which they have not requested before. Because of the initial multicast push, network resources and server load are reduced for these popular pages.

This architecture reduces server load in most cases, and never increases server load. Consider what happens in a conventional Web cache — the initial request generates a unicast response. In the LPC, when this request is for a popular page, the response is multicast. This multicast can reduce network load. Even though the initial multicast push uses more network bandwidth than the conventional unicast response, subsequent requests are handled locally.

In the LPC, multicast occurs only if pages are popular and only to places interested in a particular group. Pumps push only pages in popular affinity groups, as seen from the server's perspective. The multicast tree is limited to filters tuned to popular groups from the client's perspective. The LPC is designed to be efficient, because it pushes only when there appears to be utility.

3. Self-organization

The LPC system is self-organizing. Filters tune-in the multicast channels at natural network aggregation points, reducing unnecessary replication while also reducing client retrieval costs.

The filters, like conventional proxy caches, are configured to forward unserviced requests up a hierarchy, towards each subnets' egress (towards the 'default route'). Traffic, on its way up this hierarchy, causes filters on the way to consider tuning-in related multicast channels. At places where the traffic is sufficiently dense (thick lines in Fig. 5), filters

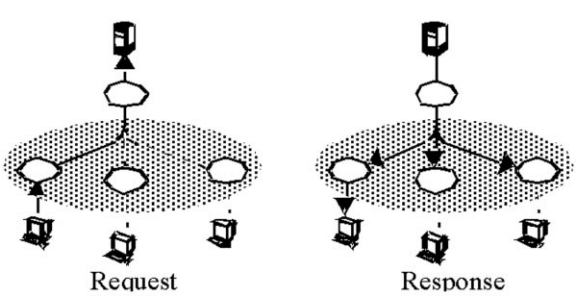


Fig. 3. First request of affinity group page.

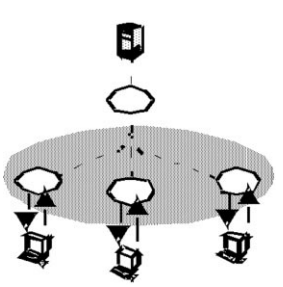


Fig. 4. Subsequent requests of page.

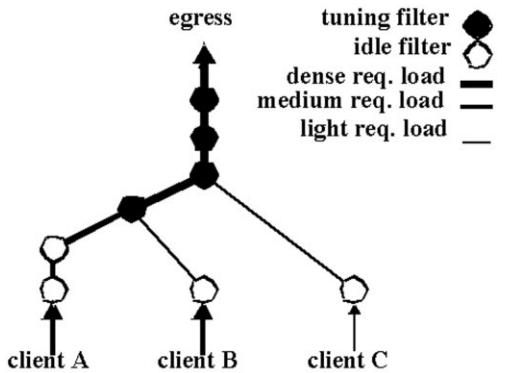


Fig. 5. Upstream filters tune-in if traffic is dense.

tune-in the channel. This causes a proxy, and all its upstream parents, to tune-in the channel (filled circles in Fig. 5). Note that in this case, the traffic from clients A and B are separately insufficient; only where they merge do proxies (and their parents) tune-in.

All filters that have tuned-in the group receive multicast pushes from the server. Client requests within that Web page group will be serviced by the first proxy they share, and requests will no longer be forwarded up the tree (dashed upstream path in Fig. 6). Filters upstream of the first shared proxy will de-tune the multicast group (striped circles in Fig. 6). As a result, filters at natural network aggregation points tend to tune-in the multicast tree at the place where request sharing is sufficient to justify ensemble caching.

This self-organization limits the number of filters that tune-in a channel. The tuning happens only at

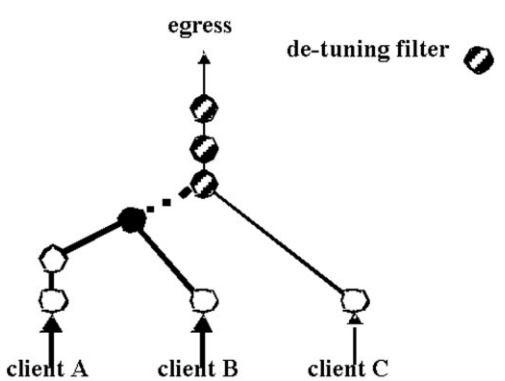


Fig. 6. Further upstream filters de-tune.

filters that are closest to the clients, but far enough to share traffic. In general, pushing data closer to the clients improves response time, at the expense of making potentially needless copies. Pushing data further from the clients increases the effect of sharing and reduces the number of excess copies, but increases the response time. When traffic is light, the former is fine, because bandwidth overuse is not an issue. In heavily-loaded systems, the latter may be preferable. The LPC can be parameterized to balance these two competing incentives, by adjusting its threshold for 'light' vs. 'heavy' traffic.

4. Implementation issues

The LPC is implemented as an extension of the Apache proxy cache [3]. There are implementation issues in providing multicast proxy caching in the Apache context. There are also development issues that include multicast channel management, intelligent request routing, object scheduling, dynamic caching algorithms, and support for mobility.

The pump/filter implementation takes advantage of Apache's use of the local file system as a cache. Requests are processed according to conventional proxy rules; first the local cache is checked, and if the URL is not found, the request is forwarded up the proxy hierarchy (steps 1–3 in Fig. 7). The pump retrieves the file from the server (step 4), and multicasts it out to the channel (step 5). The LPC uses AFDP to push the file reliably over the multicast channel, directly into the file system of the filter [1]. A redirect response is returned to the filter (step 6), and the file is found in the local file system (steps 7–8).

As a result of these steps, Web pages that are members of the affinity group are multicast to the filters. There are additional rules governing when responses are unicast or multicast (or both, in some cases), taking into account:

- whether a page is in an active affinity group,
- downstream filters tuned to a page's affinity channel,
- whether the page was recently pushed to that channel.

For example, a request might arrive from a path with no tuned filters, for a page in an affinity group

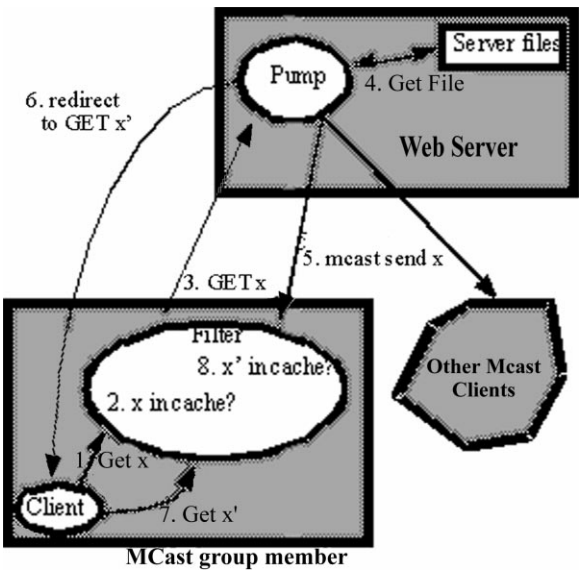


Fig. 7. Walk-through of a multicast push.

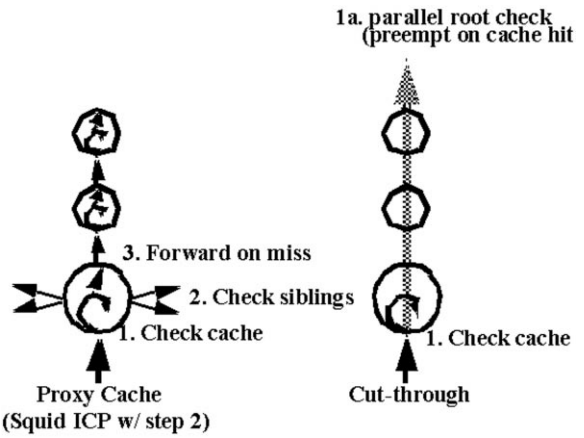


Fig. 8. Check-and-forward vs. cut-through (black is foreground, gray is background).

that has not been recently sent. In that case, the pump would generate a unicast response back to the source of the request, as well as a multicast to the channel.

Multicast channel management is the pump algorithm for creating channels based on affinity groups, and the filter algorithm for tuning channels related to its requests. This channel management determines what channels are active at a pump, and what channels each filter tunes-in or tunes-out (de-tunes). It also includes the management of the announcement channel, where pumps advertise active channels, the parameters of these channels (TTL), and the coordination of address and port space used with other multicast mechanisms.

Intelligent Request Routing (IRR) automatically configures the conventional proxy hierarchy, so that the unicast hierarchy can be used to self-organize multicast tuning. Filters are labelled as being at a client, at an egress, or in the middle; the hierarchy is configured so unicast requests tend to go from clients towards egresses. This auto-configuration system can also be also used for other proxies, such as Squid, generic Apache, or client browsers.

IRR also handles request cut-through. The LPC relies on a deep proxy tree to find appropriate locations to share caches. One result of deep proxy chains is poor response for missed data, because requests do not cut-through the chain; the check-and-forward style cache checking at each level imposes unacceptable delays for more than 2–3 levels [4]. The solution is to allow requests to split at the first proxy, into a foreground request that walks the chain, and a background request that cuts-through to the root server (Fig. 8).

Object scheduling supports the variety of request priorities inherent in the LPC. As in the implementation example above, it is possible that some multicast responses are only anticipatory; the unicast response handles the direct request, and the multicast is in anticipation of future use by filters tuned to the channel. In this case, the multicast response can (and should) be handled with lower priority than the unicast response. In addition, cut-through support requires background processing for the cut-through request; otherwise, the cut-through defeats the traffic reduction gains that caching provides.

There are two other effects of the LPC's pervasive use of caching. First, cache replacement algorithms need to be tuned to match the different characteristics of client, shared proxy, and server caching. The LPC supports dynamic reconfiguration of the replacement algorithm, sensitive to the proxy's configuration and placement. Also, user mobility defeats caching; the LPC includes mechanisms to move cache state to follow the client. This allows the user's home cache (or its upstream filters) to tune to channels based on earlier behavior, so the multicast push follows the user's movement.

5. Finding a affinity group

A key aspect to the LPC is its focus on affinity groups. An affinity group is a set of related Web pages, where retrievals are cross-correlated. These groups determine what channels a pump creates (or deletes), and what channels a filter tunes (or de-tunes). These groups can be determined by a number of factors:

- a-posteri analysis of correlated retrievals,
- syntactic analysis of embedded HREFs,
- analysis of semantic groups.

The LPC uses semantic analysis for its demonstration implementation, specifically a syntactic approximation of semantics, based on URL prefixes. Related Web page groups tend to be clustered in directories. The LPC currently uses that directory structure, as visible in the URL, to determine affinity groups. Other types of groupings are supported via channel management hooks.

The current algorithm uses successive refinement. A server assumes all its pages are members of its '/' (root) affinity group. Subdirectories are valid groups only if they represent significant fractions of their parent group, e.g., 25%. This permits specific groups to be formed, such as '/lsam/proxy/sources/', concurrent with less specific groups, such as '/rfc/'. In the current implementation, we assume affinity groups are subsets of a single server.

The LSAM project is currently developing three algorithms for syntactic approximation of semantic affinity groups: on-line algorithms for the pump and filter, and an off-line algorithm for performance analysis. The latter, used to analyze regional Squid caches [5], will help determine the current potential for performance enhancement by the LPC. It is also possible that the LPC will enable new styles of Web access that favors such groups; such log analysis will help identify such traffic shifts.

6. Prior and related work

There is a wide variety of both research and commercial development of Web cache systems. The LPC's main distinction is its use of multicast push to reduce the first-hit cost of retrieval throughout the system.

The LPC is based on source preloading of a receiver cache, a multicast version of an earlier unicast scheme [6]. It anticipates requests of individual clients by multicasting pages to the channel. Client-side prefetching, using server-provided hints, has also been examined in the unicast domain in the Boston Univerisity Oceans group [7]. Other hints have also been used to direct unicast push, such as geographical hints [8].

Other hierarchical cache systems have been developed, including Harvest [9], and its follow-on Squid [4]. Harvest uses a directory to locate entries in a distributed hierarchy. Squid uses multicast to find cache entries in hierarchical clusters of caches, sending messages via the ICP protocol.

Many other Web cache systems use multicast to distributed pages, supporting explicit subscriptions in NCSA's Webcast [10], and diffusion-based push to move pages closer to their locus of interest in LBNL/UCLA's Adaptive Web Caching [11]. Unicast diffusion push was examined in another tack of the Oceans group [12].

Other large distributed cache systems have been developed; AT&T's Crisp [16] uses a central server to distribute requests to a set of local caches, and both Georgia Tech's CANES [13] and UCL's Cachemesh [14] route requests on their way to the cache.

Georgia Tech's CANES also targets one of the LPC's goals, to provide caching at optimal network aggregation points. CANES deploys caches at routers using Active Networks technology. The LPC relies on multicast to achieve similar benefits using only application-layer code. CANES also uses *modulo caching* to cache pages at every Nth proxy on the unicast return path; the LPC uses announcements and affinity group channels to similarly limit caching to a subset of proxies.

7. Current status

The LPC is implemented as a modified version of the Apache proxy cache [3], with additional control scripts and daemons written in Perl. It can be instantiated as a pump or as a filter via command-line arguments, and currently runs on FreeBSD 2.2.x.

The current implementation of the LPC, v0.8,

supports multicast push for a number of channels, based on manual configuration or scripted program control. In this release, the automated control is limited to suggest manual actions (via highlighting on the control page). The final release, available at the end of August 1998, contains automated channel control for both pumps and filters. It supports dynamic auto-configuration of the unicast proxy hierarchy, which can be exported to other proxy caches and clients. Six different cache replacement algorithms have been implemented, selected in the configuration file at proxy boot time. Several different object scheduling mechanisms have also been implemented, and compared in network-limited and processor-limited environments. This release, v0.8, is currently available on the LSAM Web pages [15].

The current system has been demonstrated in a lab, using artificial bandwidth limiters and delay inducers. A demonstration is also available, implemented in the *ns* network simulation tool. In both cases, client access is equivalent to a local cache hit, even for pages not yet accessed.

Future work on the LPC focuses on standardizing and generalizing the API and channel announcement protocols, unifying them with other efforts in the IETF and the multicast Web cache community. Other work examines the implications of partitioned caching, notably on the use of different cache replacement algorithms in each partition and the interactions among the partitions. Finally, many of the components required to support the LPC, such as intelligent request routing and object scheduling, have more general implications on Web caching hierarchies and differentiated application services, which are also being examined.

8. Summary

The LSAM proxy cache (LPC) provides a distributed cache system that uses multicast for automated push of popular Web pages. LPC uses proxy caches that are deployed as a server pump and a distributed filter hierarchy. These components automatically track the popularity of Web page groups, and also automatically manage server push and client tuning. The system caches pages at natural network aggregation points, providing the benefits of a single, central shared proxy cache, even where no such central proxy could exist. The system has an initial prototype implementation, and it is evolving to mitigate the effects of a deep proxy hierarchy, needed for the multicast self-organization. LPC reduces overall server and network load, and increases access performance, even for the first access of a page by a client.

Acknowledgements

The LSAM project is a collaborative effort, including:

- the authors (overall design and partitioned caching),
- Theodore V. Faber and Wei Yue (mobility support),
- Gregory G. Finn (log analysis),
- John Heidemann and Lars Eggert (object scheduling),
- Steve Hotz and Stephen Suryaputra (intelligent request routing),
- Anne Hutton (channel management).

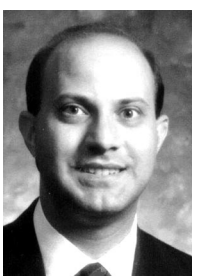
We would like to thank former project co-architect Paul V. Mockapetris, and past project staff including Katia Obraczcka, William L. Scheding, Brian Tung, Vikram Visweswaraiah, Kedar Jog, and Anand Oswal. We also thank Jon Crowcroft of UCL, Ellen Zegura of Georgia Tech., Fred Bauer of SRI, and Ron Larsen of DARPA for their constructive comments on this work.

This work is supported by the Defense Advanced Research Projects Agency through FBI contract #J-FBI-95-185 entitled ‘Large-Scale Active Middleware’. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Department of the Army, the Defense Advanced Research Projects Agency, or the US Government.

References

[1] J. Cooperstock, S. Kotsopoulos, Why use a fishing line when you have a net? an adaptive multicast data distribution protocol, in: Proc. USENIX 1996, http://www.ecf.utoronto.ca/afdp/

[2] *sd* tool Web pages, March 1998, ftp://ftp.ee.lbl.gov/conferencing/sd/
[3] Apache HTTP Server Project Web page, March 1998, http://www.apache.org
[4] D. Wessels, K. Claffy, ICP and the Squid Web cache, NLANR, August 1997, http://www.nlanr.net/%7ewessels/Papers/icp-squid.ps
[5] NLANR global Internet cache Web pages, March 1998, http://ircache.nlanr.net/
[6] J. Touch, Defining 'high speed' protocols: five challenges and an example that survives the challenges, IEEE J. Selected Areas Comm. 13 (1995) 828–835, http://www.isi.edu/touch/pubs/jsac95.html.
[7] A. Bestavros, Speculative data dissemination and service to reduce server load, network traffic, and service time, in: Proc. Int. Conf. on Data Engineering, New Orleans, LA, March 1996, http://cs-www.bu.edu/faculty/best/res/papers/icde96.ps
[8] J. Gwertzman, M. Seltzer, The case for geographical push-caching, in: Proc. 5th Annual Workshop on Hot Operating Systems, Orcas Island, WA, May 1995, http://www.eecs.harvard.edu/~vino/web/hotos.ps
[9] A. Chankunthod, P. Danzig, C. Neerdaels, M. Schwartz, K. Worrell, A hierarchical Internet object cache, in: Proc. USENIX 1996, San Diego, CA, January 1996, http://catarina.usc.edu/danzig/cache.ps
[10] NCSA, Overview of Webcast, Web pages, March 1998, http://wwwdoctest.ncsa.uiuc.edu/SDG/Software/XMosaic/CCI/webcast-doc.html
[11] L. Zhang, S. Floyd, V. Jacobson, Adaptive Web caching, in: Proc. 2nd NLANR Web Cache Workshop, Boulder, CO, June 1997, http://ircache.nlanr.net/Cache/Workshop97/Papers/Floyd/floyd.ps
[12] A. Heddaya, S. Mirdad, D. Yates, Diffusion based caching along routing paths, in: Proc. 2nd NLANR Web Cache Workshop, Boulder, CO, June 1997, http://www.cs.bu.edu/faculty/heddaya/Papers-NonTR/webcache-wkp.ps.Z
[13] S. Bhattacharjee, K. Calvert, E. Zegura, Self-organizing wide-area network caches, in: Proc. Infocom 1998, IEEE, San Francisco, CA, April 1998, http://www.cc.gatech.edu/fac/Ellen.Zegura/papers/git-cc-97-31.ps.gz
[14] Z. Wang, Cachemesh: A distributed cache system for the World Wide Web, in: Proc. 2nd NLANR Web Caching Workshop, Boulder, CO, June 1997, http://www.bell-labs.com/user/zhwang/papers/cache.html
[15] LSAM Web pages, March 1998, http://www.isi.edu/lsam/
[16] S. Gadde, M. Rabinovich, J. Chase, Reduce, reuse, recycle: an approach to building large Internet caches, in: Proc. Workshop on Hot Topics in Operating Systems (HotOS), May 1997, http://www.research.att.com/~misha/crisp/distrProxy/hotos.ps



Joe Touch received his from the University of Pennsylvania in 1992, when he joined USC/ISI where he is a Project Leader in the Computer Networks Division and a Research Assistant Professor at USC in the Department of Computer Science. He currently leads the LSAM and X-Bone projects; LSAM is a multicast Web cache system, and the X-Bone is a system for the automated deployment and management of overlay networks. His primary research interests include multicast variants of network management, high-performance protocols, and protocols for latency reduction. Joe is a member of Sigma Xi (since 1984), IEEE, and ACM, and is the Technical Activities and Vice Chair of the IEEE TCGN (gigabit networking). He also serves on the editorial board of IEEE Network, and is a member of several program committees, including IEEE Infocom (since 1994), and is co-chair of the IFIP/IEEE Protocols for High Speed Networks Workshop 1999, and chair of the 1998 IEEE Gigabit Network Workshop.

Amy S. Hughes received her A.B. (Hons.) degree in Computer Science from Bryn Mawr College in 1995, and an M.S. degree in Computer Science from the University of Southern California in 1996. She is currently working on her Ph.D. at USC. She is a member of the LSAM project, developing a multicast Web cache system. She is also researching cross-domain caching. Her research interests include protocols for caching, protocols for latency reduction, and human–computer interaction.