The LSAM Proxy Cache - a Multicast Distributed Virtual Cache

Joe Touch

USC / Information Sciences Institute
touch@isi.edu

Abstract [1]

The LSAM Proxy is a multicast distributed web cache that provides automated multicast push of web pages, based on self-configuring interest groups. The proxy 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 LSAM proxy's multicast hierarchy is self-configuring, such that these popular affinity groups are automatically cached at natural network aggregation points. This document describes the LSAM proxy system architecture and the properties of a prototype implementation.

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 of distributed clients that access related web pages. It uses multicast to distributed 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; these caches are typically deployed at the client and at intermediate shared proxies. Client caches often achieve limited benefit, because the users tend to browse new information. New pages are never in client caches.

Shared proxy caches allow new information to be cached for first-time clients, because they aggregate requests. A small set of clients can fill the cache with new information, and the rest of the clients benefit. However, shared proxy caches work only where requests can be aggregated, e.g., near the border router of a domain.

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 hot-spot at the server. We call these '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. They are globally interesting, so even shared proxy caches do not alleviate the implosion of requests. 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, proxies tune to the server's Olympic channel if their 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 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. Subscription happens automatically, as clients join 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, 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.

LPC tracks popular groups of web pages, and multicasts them to caches at natural network aggregation points.

1. 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 U.S. Government.

The 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 LSAM proxy 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 LSAM Proxy Architecture

The LSAM proxy 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 a filter). The pump, multicasts web pages to the filters, based on affinity groups (Figure 1). The filters together act as a single virtual proxy cache, where the request of any one client benefits the others.

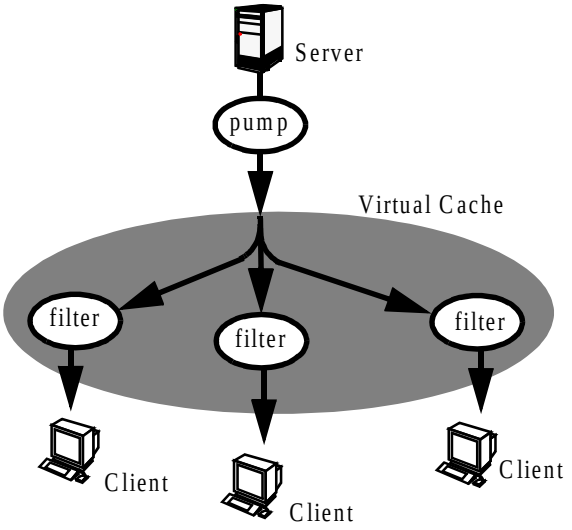

FIGURE 1. LSAM proxy components

The pump monitors web accesses at a server, and creates multicast channels for affinity groups. Requests for web pages in the affinity group are reliably multicast to the associated multicast channel, using AFDP [5]. 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 [12] (Figure 2).

The filter caches web pages for downstream access, 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 join multiple channels and accept multicast downloads without inter-channel cache interference. The cache leaves a channel whose contents are no longer correlated to the cache's ongoing requests.

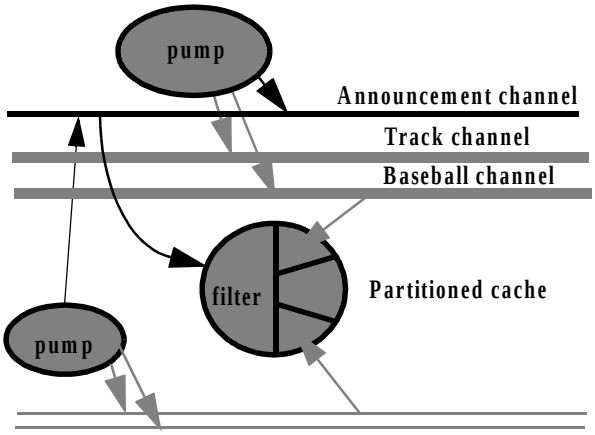

FIGURE 2. Multicast channels in the LSAM system

In the LSAM proxy, 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 (Figure 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 (Figure 3, right).

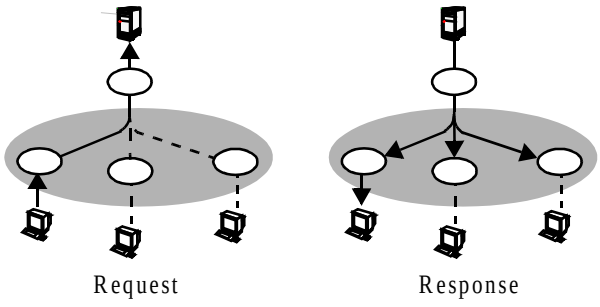

FIGURE 3. First request of affinity group page

Subsequent requests for these popular pages are handled locally, from the filters nearer the clients (Figure 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 generally popular pages.

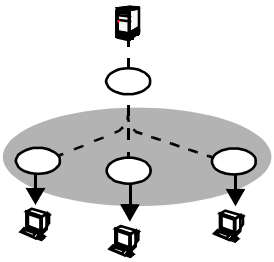

FIGURE 4. Subsequent requests of page

This architecture reduces server load in most cases, and never increases server load. The initial request generates a response in a conventional web cache; LSAM multicasts this response in some cases, for popular pages. Local responses reduce server load; even if no filter hits occur, the initial response would have to occur in either system.

LSAM tries to decrease network load in most cases. The multicast push uses more network bandwidth than the conventional unicast response would; if the page is never hit in local filters, that bandwidth is wasted. The 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. LSAM is designed to be efficient, because it pushes only when there appears to be utility.

3: Self-organization

The LSAM proxy cache system is self-organizing. Filters join 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 joining related multicast channels. At places where the traffic is sufficiently dense (thick lines in Figure 5), filters join the channel. This causes a proxy, and all its upstream parents, to join the channel (filled circles in Figure 5). Note that in this case, the traffic from clients A and B are separately insufficient; only where they join do proxies (and their parents) join.

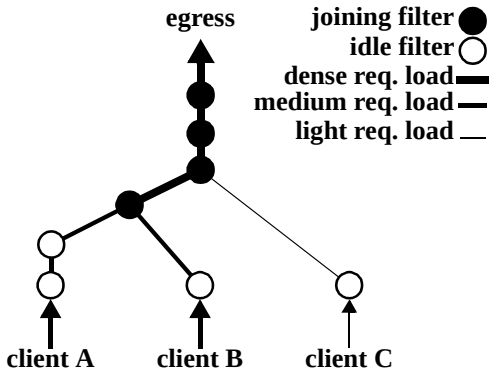

FIGURE 5. Upstream filters join if traffic is dense

All filters that have joined the group receive multicast pushes from the server. Client requests within that web page group will be serviced by the first shared proxy they share, and requests will no longer be forwarded up the tree (dashed upstream path in Figure 6). As a result, filters upstream of the first shared proxy will leave the multicast group (shaded circles in Figure 6). As a result, filters at natural network aggregation points tend to join the multicast tree, at the place where sharing is sufficient to justify shared caching.

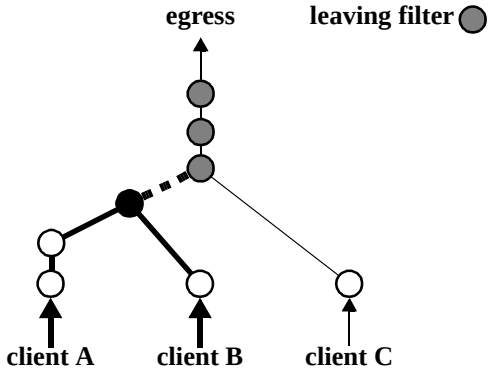

FIGURE 6. Further upstream filters leave

This self-organization limits the number of filters that join a channel, to only the filter closest to the clients that shares traffic. Pushing data closer to the clients improves performance. However, if more filters closer to the client were in the group, the pushed data would be replicated at filters with even light traffic, consuming bandwidth and cache space needlessly. The LSAM proxy can be parameterized to balance these two competing incentives, by adjusting its threshold for 'light' vs. 'heavy' traffic.

4: Implementation Issues

The LSAM proxy cache is implemented as an extension of the Apache proxy cache [1]. 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 a the local cache is checked, and if the URL is not found, the request is forwarded up the proxy hierarchy (steps 1-3 in Figure 7). The pump retrieves the file from the server (step 4), and multicasts it out to the channel (step 5). LSAM uses AFDP to push the file reliably over the multicast channel, directly into the file system of the filter [5]. A redirect response is returned to the filter (step 6), and the file is found in the local file system (steps 7-8).

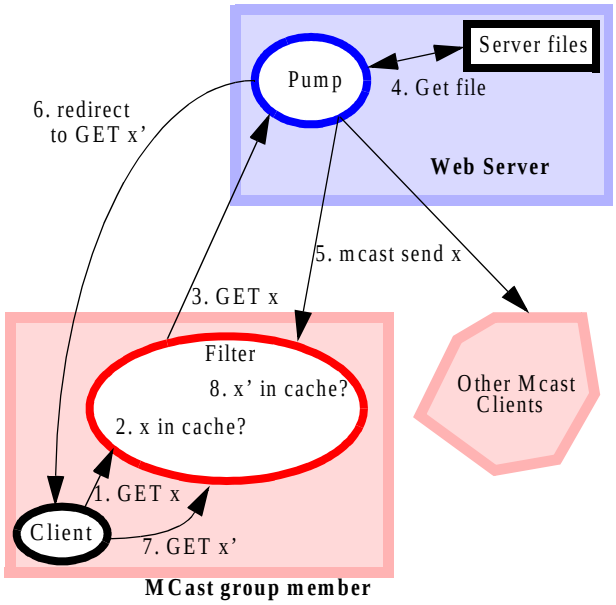

FIGURE 7. Walk-through of a multicast push

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 joined to a page's affinity channel*
- *whether the page has been pushed to that channel*

For example, a request might arrive from a path with no joined filters, for a page in an affinity group 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 joining channels related to its requests. This channel management determines what channels are active at a pump, and what channels each filter joins or leaves. 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 use with other multicast mechanisms.

Intelligent request routing automatically configures the conventional proxy hierarchy, so that the unicast hierarchy can be used to self-organize multicast joins. 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. LSAM 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 [15]. 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 (Figure 8).

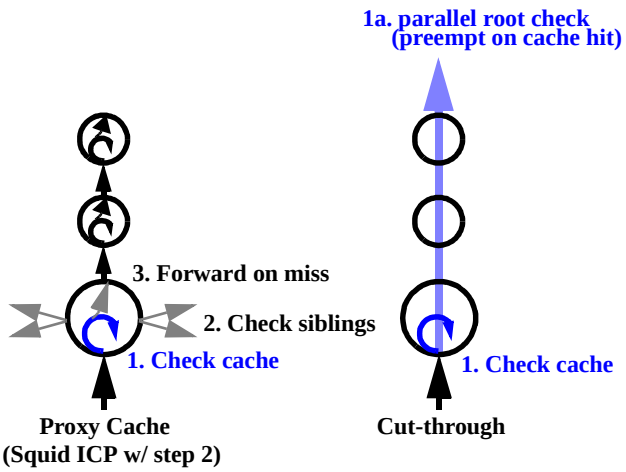

FIGURE 8. Check-and-forward vs. cut-through

Object scheduling supports the variety of request priorities inherent in the LSAM system. 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 joined 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 LSAM'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 LSAM proxy supports dynamic reconfiguration of the replacement algorithm, sensitive to the proxy's configuration and placement. Also, user mobility defeats caching; the LSAM proxy includes mechanisms to move cache state to follow the client. This allows the user's home cache (or its upstream filters) to join channels based on earlier behavior, so the multicast push follows the user's movement.

5: Finding a affinity group

A key aspect to the LSAM proxy cache 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 joins (or leaves). 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*

LSAM chose 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; LSAM 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 [11], will help determine the current potential for performance enhancement by LSAM. It is also possible that LSAM 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 LSAM proxy's main distinction is its use of multicast push to reduce the first-hit cost of retrieval throughout the system.

LSAM is based on source preloading of a receiver cache, a multicast version of an earlier unicast scheme [13]. 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 Univ. Oceans group [2]. Other hints have also been used to direct unicast push, such as geographical hints [7].

Other hierarchical cache systems have been developed, including Harvest [4], and its follow-on Squid [15]. 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 [16]. Unicast diffusion push was examined in another tack of the Oceans group [8].

Other large distributed cache systems have been developed; AT&T's Crisp uses a central server to distribute requests to a set of local caches, and both Georgia Tech's CANES [3] and UCL's Cachemesh [14] route requests on their way to the cache.

Georgia Tech's CANES also targets one of LSAM's goals, to provide caching at optimal network aggregation points. CANES deploys caches at routers using Active Networks technology; LSAM 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; LSAM uses announcements and affinity group channels to similarly limit caching to a subset of proxies.

7: Current status

The LSAM proxy cache is implemented as a modified version of the Apache proxy cache [1], 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 2.2.5.

The current implementation supports multicast push for a single, static (preconfigured) channel. It also 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, v0.7, is currently available on the LSAM web pages [9]; additional releases are planned on 3-month intervals.

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.

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 subscription. 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.

9: References

[1] Apache HTTP Server Project,
http://www.apache.or

[2] A. Bestavros, “Speculative Data Dissemination and Service to Reduce Server Load, Network Traffic, and Service Time,” Proc. International Conference on Data Engineering, New Orleans, LA (March 1996),
http://cs-www.bu.edu/faculty/best/res/papers/icde96.ps

[3] S. Bhattacharjee, K. Calvert, E. Zegura, “Self-Organizing Wide-Area Network Caches,” Proc. Infocom 1998, IEEE, San Francisco, CA (March 28 - April 2, 1998),
http://www.cc.gatech.edu/fac/Ellen.Zegura/papers/git-cc-97-31.ps.gz

[4] A. Chankunthod, P. Danzig, C. Neerdaels, M. Schwartz, K. Worrell, “A Hierarchical Internet Object Cache,” Proceedings of USENIX 1996, San Diego, CA (January 22-26, 1996),
http://catarina.usc.edu/danzig/cache.ps

[5] J. Cooperstock,., S. Kotsopoulos, “Why use a fishing line when you have a net? an Adaptive Multicast data Distribution protocol,” Usenix '96 Proceedings.
http://www.ecf.utoronto.ca/afdp/

[6] S. Gadde, M. Rabinovich, J. Chase, “Reduce, Reuse, Recycle: An Approach to Building Large Internet Caches,” Workshop on Hot Topics in Operating Systems (HotOS) (May 1997),
http://www.research.att.com/~misha/crisp/distrProxy/hotos.ps

[7] J. Gwertzman, M. Seltzer, “The Case for Geographical Push-Caching,” Proc. Fifth Annual Workshop on Hot Operating Systems, Orcas Island, WA (May 1995),
http://www.eecs.harvard.edu/~vino/web/hotos.ps

[8] A. Heddaya, S. Mirdad, D. Yates, “Diffusion Based Caching along Routing Paths,” Proceedings of the 2nd NLANR Web Cache Workshop, Boulder, Colorado (June 9-10, 1997),
http://www.cs.bu.edu/faculty/heddaya/Papers-NonTR/webcache-wkp.ps.Z

[9] LSAM proxy release, v0.7, March 1998
http://www.isi.edu/lsam/proxy/

[10] NCSA, Overview of Webcast,
http://wwwdoctest.ncsa.uiuc.edu/SDG/Software/XMosaic/CCI/webcast-doc.html

[11] NLANR global internet cache web pages,
http://ircache.nlanr.net

[12] *sd* tool
ftp://ftp.cs.lbl.gov/

[13] J. Touch, “Defining `High Speed' Protocols: Five Challenges & an Example That Survives the Challenges,” IEEE JSAC., special issue on Applications Enabling Gigabit Networks, Vol. 13, No. 5, June 1995, pp. 828-835.
http://www.isi.edu/~touch/pubs/jsac95.html

[14] Z. Wang, “Cachemesh: A Distributed Cache System for the World Wide Web,” Proceedings of the 2nd NLANR Web Caching Workshop, Boulder, Colorado (June 1997),
http://www.bell-labs.com/user/zhwang/papers/cache.html

[15] D. Wessels, K. Claffy, ICP and the Squid Web Cache (August 13, 1997),
http://www.nlanr.net/%7ewessels/Papers/icp-squid.ps

[16] L. Zhang, S.Floyd, V. Jacobson, “Adaptive Web Caching,” Proceedings of the 2nd NLANR Web Cache Workshop, Boulder, Colorado (June 9-10, 1997),
http://ircache.nlanr.net/Cache/Workshop97/Papers/Floyd/floyd.ps