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

Computer Networks and ISDN Systems 30 (1998) 2169–2177

COMPUTER NETWORKS and ISDN SYSTEMS

Adaptive web caching: towards a new global caching architecture

Scott Michel [a], Khoi Nguyen [a], Adam Rosenstein [a], Lixia Zhang [a,*], Sally Floyd [b], Van Jacobson [b]

[a] *UCLA Computer Science Department, 4531G Boelter Hall, Los Angeles, CA 90095, USA*
[b] *Lawrence Berkeley National Laboratories, Berkeley, CA, USA*

Abstract

An adaptive, highly scalable, and robust web caching system is needed to effectively handle the exponential growth and extreme dynamic environment of the World Wide Web. Our work presented last year sketched out the basic design of such a system. This sequel paper reports our progress over the past year. To assist caches making web query forwarding decisions, we sketch out the basic design of a URL routing framework. To assist fast searching within each cache group, we let neighbor caches share content information. Equipped with the URL routing table and neighbor cache contents, a cache in the revised design can now search the local group, and forward all missing queries quickly and efficiently, thus eliminating both the waiting delay and the overhead associated with multicast queries. The paper also presents a proposal for incremental deployment that provides a smooth transition from the currently deployed cache infrastructure to the new design.

Keywords: Web caching; Self-organizing; Multicast

1. Introduction

As Zhang, Floyd, and Jacobson argued [1], an adaptive, highly scalable, and robust web caching system is needed to effectively handle the exponential growth and extreme dynamic environment of the World Wide Web. This paper elaborates on the initial architecture for an *adaptive* web caching system outlined in [1]. When this web caching architecture was initially proposed, several other approaches to building a caching system for the World Wide Web were also introduced, including the Squid web-caching system, which has since been widely deployed and has emerged as the de facto standard for web caching. Although the Squid system has proven successful in accomplishing many of the goals of web caching, we believe our adaptive approach to the problem provides an effective, natural evolutionary step towards a truly scalable, robust, efficient, and demand-driven web-caching system.

The dominant web-cache infrastructure on the Internet today consists of a hierarchy of Squid object caches. In the Squid system, all cache servers are inter-connected in a *manually configured* hierarchical tree. Each cache server exchanges Internet Cache Protocol (ICP) queries and replies with its 'peers' (siblings and parent) in order to determine the most appropriate peer from which to retrieve a client's requested web content (either via a cached copy or directly from the origin server). Fueled in part by Squid's success in demonstrating the potential benefits of web caching on the Internet, a proliferation

* Corresponding author. E-mail: lixia@cs.ucla.edu.

0169-7552/98/$ – see front matter
PII: S0169-7552(98)00246-3

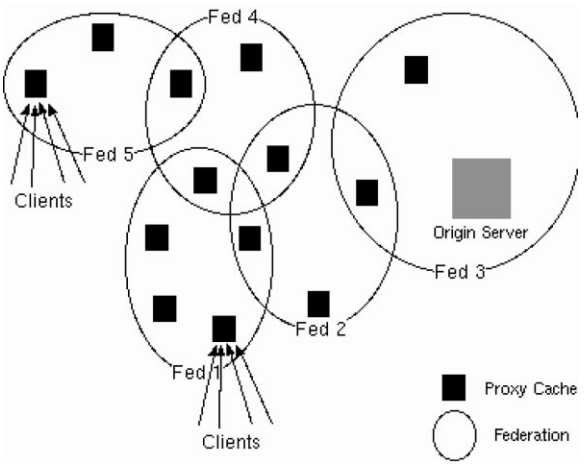

Fig. 1. Sets of proxy caches are organized into small overlapping groups, thus forming a mesh through which clients' web queries may be forwarded.

of research in possible improvements to the system has recently emerged. It has been proposed that improvements to the current manual configuration of the Squid hierarchy would eliminate the heavy burden placed on system administrators who must coordinate with each other, make the system less prone to both human error and misconfiguration by system administrators, and improve the scalability of the system as well as the system's ability to adapt to changing network conditions. In addition, research has suggested that Squid's single, static hierarchy should be enhanced to address the potential problem of inefficient routing of web requests. This problem can arise since all cache misses must first be fetched by a root node, which may be significantly farther from the origin server than the client's proxy cache. Moreover, the fact that the static hierarchy is not necessarily well suited to the unpredictable, roving nature of web hot-spots is also subject to improvement. Squid can be further optimized to avoid the inordinate amount of delay associated with the 3-way handshake protocol for exchanging ICP messages. Lastly, the possibility that upper nodes in the Squid hierarchy may become excessively loaded and degrade both the system's performance and ability to scale offers another opportunity for enhancement. Thus, these challenges associated with improving the Squid system suggest that the system must evolve towards a more scalable, adaptive, efficient, and self-configuring web-caching system in order to effectively support the phenomenal growth in demand for web content on the Internet.

We believe our adaptive web caching system provides an effective evolutionary step towards this goal. As originally presented in [1], the general architecture of the envisioned adaptive web caching system would be comprised of many cache servers that self-organize into a mesh of overlapping multicast groups and adapt as necessary to changing conditions (see Fig. 1). This mesh of overlapping groups forms a scalable, implicit hierarchy that is used to efficiently diffuse popular web content towards the demand. (The reader is referred to [1] for further details of the general architecture.) There are two important research issues associated with this approach: (1) how to self-organize web caches to establish the different paths of communication among the cache groups in the mesh/hierarchy; and (2) how to forward requests and other information along the most appropriate path. The high-level focus of our work, then, is (1) to design a reasonable architecture for an adaptive, scalable web caching system; and (2) to determine the most effective direction of evolution from the current web caching infrastructure to our proposed system.

This paper extends our original adaptive web caching design [1] by first proposing how the current web-caching infrastructure may evolve towards an adaptive web-caching system and by discussing the system and its associated research issues, including those mentioned above, in greater detail than before. The paper is organized as follows: We first discuss in Section 2 a realistic evolutionary path from the currently deployed Squid caching infrastructure to our proposed adaptive system. In Section 3, we discuss how caches may share information about their contents, and in Section 4, we present our recent progress in our protocol for forwarding requests through our mesh of cache groups. We then describe the issues and mechanisms of self-organization in Section 5, and finally, we conclude our paper in Section 6.

2. Time scales

Incremental deployment is critical in introducing or modifying any global-scale system; it is infeasible

to envision the one-step deployment or modification of such large systems. A good plan for incremental deployment is especially important for an adaptive caching system since multicast, an important element in our system architecture, is yet to be ubiquitously supported, and some of the research goals are on a longer time scale than others (e.g., self-organization is a longer term research topic). In addition, an adaptive caching system would be built on top of the existing web caching infrastructure, and this requires gradual, incremental changes to the system.

As a result, we envision a deployment of adaptive web caching in three distinct phases, in which the evolution from one stage to the next would occur in tandem with the evolution of the Internet itself as needed functionality for adaptive web caching becomes widely supported in the Internet. The three deployment phases would be (1) the introduction of adaptive web caching for distributing information and web requests among the cache mesh, without multicast and self-organization; (2) the addition of multicast as a mechanism for both efficient group communication and as a building block for the self-organization of groups, which will be introduced in the third phase; and finally (3) the introduction of self-organization into the adaptive web caching system to make it completely adaptive and robust.

It is unlikely that multicast will be ubiquitously supported on the Internet in the near-term, and thus we cannot assume that adaptive web caching will initially be able to leverage the efficiencies and useful properties of multicast. At the same time, even if the long-term research goal of enabling cache servers to self-organize themselves into reasonable overlapping groups in an adaptive and fully distributed manner was fully realized, it would not be likely to be deployed in the current web caching infrastructure in the near term. An initial deployment of adaptive web caching, then, would take place in a mesh of manually organized cache servers that communicate with each other largely using unicast communications. It should be noted that although multicast serves a more important purpose than only as an efficient group communication mechanism in our system, the use of unicast suffices in an initial deployment of adaptive web caching with cache groups that are manually configured and static. That is, in an initial deployment of adaptive web caching, the routing of information among the cache mesh would be adaptive, but the underlying communication infrastructure among the web caches would remain static. In this short-term system, there are two important research issues that we will address: (1) how web caches share and make use of information about the contents of other nearby caches; and (2) how to best route requests through the manually configured cache mesh (including how to handle 'cold' page requests). Both of these issues are discussed in more detail in the next section.

Once multicast becomes widely supported on the Internet, the second phase of deployment of adaptive web caching may be initiated. Adding multicast to our system provides two primary benefits. First, it allows members of a group to communicate and share information efficiently; and second, it provides the necessary functionality for a scalable, fully distributed algorithm to self-organize the cache servers into overlapping multicast groups. Such an algorithm requires that each cache server be able to freely join or leave cache groups without having to inform anyone in the group. Multicast groups provide this capability. In addition, multicast provides negotiation-free group discovery, an important aspect of auto-configuration [1].

Thus, the primary research issue for this intermediate-term system is how to best leverage multicast both in the intermediate-term as well as in the long-term final system. Note that the use of multicast in our design is strictly within certain local scope, thus multicast-enabled self-organization can be deployed in a piecemeal way, without needing to wait till multicast becomes ubiquitous through the Internet.

Finally, an adaptive web caching system can only be fully deployed after the achievement of a long-term research goal of developing a scalable, robust, adaptive, and fully distributed protocol for self-orga-

[1] In the 1997 white paper on adaptive web caching, it was suggested that, in addition to the above proposed uses for multicast, multicast would also be used as an information discovery vehicle. We have since decided that cache group members would inform each other of their cache contents, using multicast if possible for efficiency, in advance to avoid the delays associated with duplex suppression for multicast queries. Thus, multicasting requests for data to neighboring web caches is no longer required in our system.

nizing an arbitrary number of cache servers into overlapping multicast groups based on some vector of metrics (e.g., delay and bandwidth between group members, demand for web content, cache server load, and possibly policy constraints). The addition of self-organization to an adaptive web caching system can make the whole system scalable, adaptive, and robust. Some of the fundamental underlying research issues here will be (1) how to efficiently self-organize the groups to provide good communication paths among the caches in the cache mesh; (2) how to organize caches separated by some political or resource-constrained boundary (e.g., transoceanic links, network boundaries between secure organizations); and (3) whether an implicit hierarchy (cache mesh) is adequate or if an explicit hierarchy (a second level cache structure overlaid over a lower level cache mesh) is needed. These research issues along with others associated with self-organization are discussed in greater detail in Section 5.

3. Information about the contents of nearby caches

There are two distinct issues in applying adaptive algorithms for the exchange of information among caches in a cache mesh. First, within a region or local area, caches receiving a request for data check to see if some nearby cache already has the requested contents. A second step dynamically routes the request to another cache or caches within the cache mesh after the first step has failed. In this section we address the first step, of caches availing themselves of the contents of nearby caches.

In our earliest thoughts of a adaptive web caching architecture, as in the current Squid caching infrastructure, caches used explicit probing to query neighboring caches about their cache contents. In the Squid architecture, for example, a Squid cache sends a query to its sibling caches and waits for a positive acknowledgment that the requested content is cached within the sibling group. Several recent proposals have proposed alternative mechanisms that would reduce the unnecessary delay of this explicit probing behavior. One such approach is [2], where *Bloom filter* hash code sets are periodically exchanged between siblings. This allows each sibling to make an informed judgment about whether or not a requested URL is cached elsewhere in the group [2].

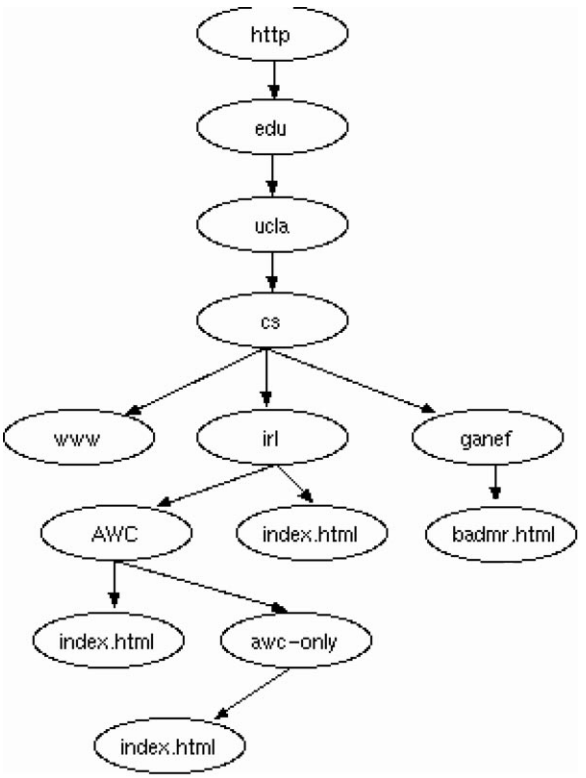

Fig. 2. Decomposition of URLS into scheme, network location, and path components. Multiple URLS from UCLA's Computer Science Department build this tree of shared components.

Our current approach to adaptive web caching is that caches exchange their entire content state with the other members of their cache groups, eliminating the delay and unnecessary use of resources of explicit cache probing. URL hash coding is employed as a compression mechanism. Each URL whose contents occur in a single cache is decomposed into *scheme* (e.g., 'html'), *network location* (DNS host name), and following *path* components. Repeated URL decompositions are organized as a tree (see Fig. 2). A depth-first traversal of the tree yields a string of hash codes, which are subsequently exchanged.

[2] A completely different issue is that of neighboring of co-located caches under common administrative control. In this case, hash codes can be used not only to disseminate information about cache contents, but in fact to allocate different portions of the URL address space to different caches within the cache group [3].

An immediate concern is the size of the state message as caches accumulate a large number of URLs. However because this content state is exchanged only among the members of each cache group, we expect the cost of such exchanges to be well within acceptable level as long as the grouping is formed properly, taking into account the distance among members and the bandwidth availability within each group. Furthermore, the size of the content state message scales in the number of the URL tree nodes. There exists a tradeoff between the reduced delay that results from more complete information about other caches' URL decomposition trees, and the additionally imposed load of larger state exchange messages. Another important issue is the effectiveness of the hashing scheme and the number of false positives. This is a piece of ongoing research. Additional concerns include cache update synchronization, as [4] found in routers exchanging RIP protocol messages. As [4] proposes, synchronization can be broken by randomizing the exchange intervals.

4. Forwarding requests

In this section, we consider adaptive approaches for the forwarding of requests within the cache mesh. These forwarding mechanisms are used after a cache has decided that none of its fellow group members have the requested contents. In some cases, it is sufficient for a cache to forward requests to a fixed set of 'parent' caches in a configured hierarchy. This is a reasonable approach for the current web caching infrastructure, where one of the first-order limitations is that of congested trans-oceanic links connecting clients in other continents with busy web servers situated in North America. A fixed hierarchy for forwarding requests is also sufficient for those extremely hot web pages that are quickly disseminated to web caches throughout the global web caching infrastructure. However, as the web caching infrastructure increases in both size and content diversity, the importance of adaptive algorithms increases for those requests that cannot be satisfied locally and must be forwarded. A cache must decide whether to forward the request towards the origin server, a nearby replicated server, or a closer large cache judged likely to have the desired contents.

As articulately argued in [5], design principles for the global web caching infrastructure include minimizing both the number of cache hops to access the data and the delay in searching for a request that cannot be satisfied by the caches. For requests that are not sufficiently hot and thus not pervasively distributed among the web caching infrastructure, the number of cache hops to access the data should be minimized. This may be accomplished by forwarding the request to a cache or caches significantly closer to the origin or replicated server, thus significantly increasing the likelihood of reaching a cache that already has the requested data. In certain cases, minimizing the number of cache hops to access the data can also be accomplished by forwarding the request to a cache known to be significantly more likely to have the data, regardless of that cache's distance to the origin or replicated server. For a request that is not sent directly to the origin server, the delay in the case of a miss is *only* minimized if that request was routed to a cache closer towards the origin/replicated server. Two other design principals set forth in [5] are to 'share data among many caches', and 'cache data close to clients'. Adaptive web caching is true to this advice by complementing the origin-ward query forwarding with a *demand-ward* data replication scheme.

We propose that web caches maintain a *URL routing table*, and use this URL routing table to decide where to forward a request in the web caching infrastructure. The URL routing table bears a resemblance to the IP routing table: caches would maintain a URL prefix table just as routers maintain an IP network prefix table. This URL routing mechanism can be deployed either in an infrastructure based on multicast groups of web caches or in the current unicast-based web caching infrastructure. The URL routing table's primary keys are URL prefixes, with which are associated one or more identifiers to the next-hop caches or cache groups. The URL prefix may be a distinct, complete URL, or, more typically, an aggregation of URLs designated by their common *scheme*, *network location* and *path* components. For URLs that do not match to anything more specific in the URL routing table, a default entry may exist in the table, similar to the default forwarding entry maintained in many IP routing tables. This allows the familiar tradeoff between minimizing overhead and optimiz-

ing request routing (by decreasing or increasing the level of detail in the URL routing table).

The information for the URL routing table is learned from caches within the participating caching infrastructure, transparent to web clients and servers. The initial information for the URL routing table may be filled in a number of possible ways. For example, *source-based* entries can result from a cache's proximity to a specific server or replicated server, or from that cache's more general proximity to an entire set of servers (e.g. an aggregated entry could refer to all the servers from '*.berkeley.edu', or all the servers from the same country.) The information for the source-based entries includes the URL prefix, an identifier for the cache, and a metric reflecting that cache's average measured delay in seconds to retrieve a request from a matching URL. To maintain these entries the caches may use their own experience of sending requests directly to those servers to assure their existence and to estimate the distance in seconds from those servers. New source-based entries in a cache's URL routing table can result from receiving location advertisement of a nearby busy origin server. When origin servers are unknown to the cache system, Web requests are sent directly to the servers identified in the URL's. Thus read-once pages are served as fast as possible without going through the caching infrastructure. However whenever the volume of requests reaches a certain level, a busy server (or a nearby cache on the server's behalf) can actively advertise its location to nearby caches, resulting in further requests and data dissemination to go through the caching system. However caches must also build in self-protection mechanism to avoid idle web servers from aggressively pushing out their own reachability information, thus loading up the URL table with useless entries.

A second information source for the URL routing table is derived from *content-based entries*. Content-based entries are learned from caches which accumulate URLs matching a certain URL prefix, either as the result of administrative policy or their own request history. An example scenario where content-based entries are necessary is the Japan cache in the United States [6], created to cache entries from servers with domain names '*.jp'. Caches distinguish between source-based and content-based entries in the URL routing table. Because forwarding decisions made on content-based entries are less likely to head towards the origin server, these forwarding decisions are more likely to increase request delay in the event of a cache miss. However, content-based entries often will cover a broad range of URLs (e.g. '*.jp') and will thus allow such queries to take advantage of a great degree of aggregation. While content-based caching is different from a true mirror, such large-scale aggregation points provide a similar benefit in answering requests from neighboring cache groups. Such aggregation points may also look similar to the cache-banks that use URL-hashing to designate each cache for serving a certain range of URLs, however with the advantage of having control over the exact cached content types and volumes.

We are exploring a range of possible approaches for caches to exchange URL prefixes in order to build URL routing tables based on these source-based and content-based entries. One simple approach is to use a distance-vector-style routing algorithm among the web caches, based on the rough measured metric of 'seconds' between caches and their neighboring caches. A second approach to building the URL routing table is to use a link-state routing analog. A link-state approach means that routing information is exchanged globally, to scale well one must restrict the global URL routing updates to a subset (e.g., at most several hundred) of large caches. This approach reflects an inherent underlying organization of the web caching infrastructure into larger regional caches that participate in the global URL routing table, and smaller local caches that tend to serve their local community. Each cache constructs its own URL routing table based on its knowledge of the cache topology, combined with the source-based and content-based entries distributed among the large web caches indicating which caches are close (measured in seconds of delay) to which URL prefixes.

5. Self-organization

We are still in the early stage of developing a Cache Group Management Protocol (CGMP) to let autonomous Web caches organize themselves into multiple, overlapping multicast groups, forming a transport substrate for URL requests and data flow.

We envision that each popular page will grow its own cache tree upon the foundation provided by this mesh of overlapping cache groups. Cache trees for 'flash spots' may come and go highly dynamically, but the cache groups upon which they are built will remain relatively stable. Groups will gradually and continuously adjust themselves according to dynamic changes in the operational environment such as changes in network topology due to cache servers entering and leaving the population, links coming up or going down, and changes in traffic load and user demand levels. We have considered several possible mechanisms for self-configuration of cache groups. These included approaches based on reaction-diffusion models found in biology as well as more traditional negotiation techniques. Further examination of scaling issues, however, led us to abandon those approaches in favor of a new voting scheme with which we have had success in preliminary simulation studies. Given the increasingly complex World Wide Web, a distributed caching protocol must scale effortlessly. To meet this goal we envision a global cache system capable of utilizing new cache servers whenever and wherever they may appear.

In order to best achieve such a scale-up, our design allows cache servers to be added by any party, at any time, in any location. To facilitate this vision, adaptive web caching defines cache servers to be *autonomous* entities which observe and react to their changing environments independent of outside control. A continuing increase in web access demand will necessitate a corresponding continual addition of cache servers to the global cache mesh. Self-organization is a necessity in this kind of decentralized environment, both in terms of adaptive web caching protocols and in terms of the policy authorities controlling adaptive web caching servers.

The fundamental design goals of CGMP include making the entire cache topology group-wise connected, making the grouping follow and match the topological constraints, and grouping caches to minimize the number of 'hops' a URL request must travel upon cache fault. Cache groups that cross transoceanic links should keep the group size minimal in order to minimize the intra-group communication cost, while caches with low-delay, high bandwidth connectivity to one another should group themselves together. Furthermore, a group should cover a contiguous neighborhood in network topology (that is, not fragmented by intersections with other groups). The last point implies that whether a cache joins a group G should not be determined by whether its location is close to a single member of G, but by taking into account the locations of all members in G. Lastly, the CGMP should not require manual configuration and, consequentially, must learn of the existence of cache groups which it may potentially join without resorting to negotiation. This suggests that multicast group discovery would be particularly advantageous.

In the proposed voting scheme, each cache server periodically sends voting messages on a well-known multicast address, one for each group of which it is a member. Simultaneously, each cache server individually decides which cache groups to join based on the voting announcements received from other cache servers. The overlapping of cache groups is achieved by requiring that caches join more than one group, but with no two cache servers joining the same two groups. Each voting message contains the group ID for the cache group and a voting signal. Recipients of a voting message from member M of group G evaluate the voting signal relative to their distance from M, and then treat this value as the partial support from M for their membership in G. The sum of all the partial supports a cache received from members of G indicates its appropriateness for membership in group G. If the support for membership in G surpasses a threshold, a cache wishing to join G (an 'applicant') may simply begin operating as a member of G. If a member of G notices a decline in voting support below a different threshold, it will exit group G (it has been 'voted out'). In addition to sending periodic voting messages to all other caches, members and applicants of a group G also multicast 'voting requests' to the multicast group G. These requests call for some relative increase or decrease in the voting signal. Group members average the requests, and alter the form and distance of their voting signals in response. In this way, each member indirectly responds to every applicant, as members attempt to balance the strength of the voting signal to be 'strong enough' to support distant members and applicants as it can, while not being 'too strong' for those members that are close-by. This approach is robust because dropped

votes merely appear to weaken support for group membership temporarily.

No decisions regarding group membership are made on hard thresholds, but rather they are made with tolerance to varying support and randomized times to act. If a lossy link prevents an applicant from receiving many votes from some members, then the apparent vote will be too low to join —an appropriate observation given the applicant's impaired connectivity. Likewise, applicants appearing to fit well into the chosen scale upon which the group is operating will be able to determine this by simply observing the group's votes, and may join the group's multicast group and start issuing its own votes without ever contacting a member. Thus, the adaptive web caching group management protocol does not need reliable multicast nor direct negotiations, it is tolerant to changing topologies and transient failures, it does not rely on any hand configuration, and it generates little protocol overhead.

Using a well-known multicast address to bootstrap auto-configuration allows for transparent neighborhood discovery and configuration-free initialization. The associated overhead can be controlled by adjusting both the frequency of the voting messages and the maximum distance to which the messages travel. This distance limiting is achieved by group members setting the TTL limit in their voting announcements to be not much farther than the farthest existing member of the group. For an unconnected cache server to achieve connectivity it must send 'vote solicitation requests' on the global well-known multicast address with increasing TTL values until connected caches respond by increasing the TTL on their voting announcements sufficiently. Because the TTL value controls the number of router hops a message travels, a trans-oceanic link is counted the same way as a FDDI hop. Thus an increasing TTL value assures that the request eventually reaches other existing caches. On the other hand, caches take into account measured delays and connectivity between each other in considering the appropriateness of group membership, so that a group crossing a trans-oceanic link will most likely result in the minimal size of two caches (one at each end), and a group crossing a campus FDDI ring will include all the caches connected to the ring. A special case that is yet to be incorporated into our design is handling of satellite channels —although they have a long propagation delay, their broadcast nature suggests the grouping of all their connected caches.

Preliminary simulation results have shown the voting scheme to be stable, resistant to node failures, adaptive to changing topology, and successful in forming and maintaining the group at any desired size. However, the simulations performed so far are based on small topologies and are very coarse-grained. We are moving onto larger-scale simulation tests to examine the proposed scheme in more realistic network settings. Because engineering of self-organized systems is, in general, an area of open research, we relegate full deployment of CGMP towards the end of the proposed incremental deployment of adaptive web caching. In the interim, we propose a gradual adjustment of currently deployed Squid cache hierarchies into adaptive web caching groups.

Initially, adaptive web caching can be built on top of existing Squid hierarchies. In this environment new adaptive web caching servers will need to determine their 'sharing neighbors' and their place in the Squid hierarchy. This initial self-organization within the Squid hierarchy is driven by the adaptive web caching server's hand-configured knowledge about its own cache capacity and network quality. The use of multicast to identify parents, siblings and children is a well-established mechanism which takes advantage of multicast's unique features. This sort of self-organization will be introduced into the system when the 'birth rate' of adaptive web caching servers warrants it.

As adaptive web caching grows, it can break away from the traditional Squid-type hierarchy to a form that better suits the task of demand-driven data diffusion. Aggregation of client requests (to improve hit rates) will be achieved by replacing Squid leaf caches with groups of topologically nearby caches. As URL routing is introduced, adaptive web caching will transition from the explicit Squid hierarchy to an implicit hierarchy. This configuration can be achieved through the use of multi-scale groupings. Groups will vary in size, with upper-hierarchical-layer Squid servers joining/forming wide-area groups. These wide-area groups form an implicit hierarchy because they will serve the same purpose as network backbones with respect to request forwarding. As the most expedient cache-group link

to get from a local cache group to any number of distant servers, the wide-area cache groups will see more traffic, and thus cache more objects than the local-area groups. They should also experience the highest hit-ratios in servicing requests.

6. Conclusion

Extending Zhang, Floyd, and Jacobson's original adaptive web caching proposal in [1], we have discussed both our recent progress in designing our adaptive caching system as well as some of the important research issues associated with our work. In particular, we have presented our research towards building an adaptive mesh of overlapping multicast groups of cache servers via a self-organizing mechanism. We have also discussed how we intend to forward web requests and content through the cache mesh such that the web content can efficiently meet its demand.

We believe our proposed adaptive, self-configuring web caching system is needed to adequately accommodate the current exponential growth in demand for web content. Our design has specifically been designed to scale with the total traffic volume as well as with the number of users, hosts, and networks in the Internet. At the same time, our system is designed to remain robust and efficient in the face of the heterogeneous and dynamic network conditions that typically characterize the Internet. Although our system is being designed specifically for the World Wide Web, we believe that many of the research issues that we are addressing can be more generally applied to other Internet systems which face issues of scale and which can benefit from caching.

References

[1] L. Zhang, S. Floyd, V. Jacobson, Adaptive web caching, Research proposal, February 1997, http://irl.cs.ucla.edu/AWC/proposal.ps

[2] L. Fan, P. Cao, J. Almeida, A. Broder, Summary cache: a scalable wide-area Web cache sharing protocol, Technical Report 1361, Computer Sciences Dept., Univ. of Wisconsin-Madison.

[3] K. Ross, Hash routing for collections of shared Web caches, IEEE Networks 11 (6) (1997) 37–44.

[4] S. Floyd, V. Jacobson, The synchronization of periodic routing messages, IEEE/ACM Trans. Networking 2 (2) (1994) 122–136.

[5] R. Tewari, M. Dahlin, H. Vin, J. Kay, Beyond hierarchies: design considerations for distributed caching on the Internet, Technical Report TR98-04, Dept. of Computer Sciences, Univ. of Texas at Austin, February 1998.

[6] M. Nabeshima, The Japan cache project, NLANR Cache Workshop, http://ircache.nlanr.net/Cache/Workshop97/Papers/Nabeshima/nabeshima.html

Scott Michel is currently a graduate student at the University of California, Berkeley. He plans to draw his Ph.D. dissertation material from the Content Routing Protocol.

Khoi Nguyen is a graduate of the Computer Science Program at the University of California, Berkeley, Mr. Nguyen entered the computer science graduate program at the University of California, Los Angeles in 1997. He plans to draw his Ph.D. dissertation material from portions of the AWC project.

Adam Rosenstein is a graduate of the Computer Science Program at the University of Florida, Mr. Rosenstein entered the computer science graduate program at the University of California, Los Angeles in 1996. He plans to draw his Ph.D. dissertation material from the Cache Group Management Protocol.

Lixia Zhang was born in Shen-Yang, China. She received her B.S. degree in 1976 from Heilongjiang University in Harbin, China, her M.S. in Electrical Engineering in 1981, and her Ph.D. in Computer Science in 1989 from Cal State, L.A. and M.I.T., respectively. She has worked at Xerox Palo Alto Research Center in Palo Alto, California for over six years. Dr. Zhang joined the University of California, Los Angeles Computer Science faculty in November of 1995.

Sally Floyd received her B.A. degree in 1971 from the University of California, Berkeley, and her M.S. and Ph.D. in Computer Science from the University of California, Berkeley in 1987 and 1989, respectively. She is a staff scientist in the Network Research Group of the Information and Computing Sciences Division at Lawrence Berkeley National Laboratory in Berkeley, California.

Van Jacobson is currently the group leader of the Network Research Group of the Information and Computing Sciences Division at Lawrence Berkeley National Laboratory in Berkeley, California.