Appears in the Proceedings of the 3rd International WWW Caching Workshop, Manchester, England, June 1998

Using Network-level Support to Improve Cache Routing

Ulana Legedza and John Guttag

Software Devices and Systems Group
Laboratory for Computer Science
Massachusetts Institute of Technology
{ulana,guttag}@lcs.mit.edu
http://www.sds.lcs.mit.edu/

Abstract

We present the design of a new distributed Web caching infrastructure that uses network-level mechanisms to provide routing of Web requests to caches. Our work is motivated by the need to provide effective caching not only for the small number of extremely popular Web documents, but also for the large number of documents of only intermediate popularity. In currently existing and proposed systems, requests for less popular documents suffer from long latency stemming from the inefficiencies of routing through an application-level overlay network. We combat this problem by integrating cache routing into the network layer, which results in shorter search paths and fewer application-level checks.

Keywords: Web caching, cache routing, push-caching, network-level services.

1 Introduction

It is widely agreed that a distributed cooperative Web caching infrastructure is needed to provide scalable and timely access to Internet servers. Most proposed approaches construct a network of caches as an overlay on the Internet. A key challenge for such schemes is the design of an efficient algorithm for searching the network of caches for the desired document. While very popular documents can usually be found quickly, the search for less popular documents may follow a long and circuitous path of numerous failed checks. The impact of this is substantial since, as seen in [4], the hit rate on web caches is typically less than 50%, indicating a large number of requests for documents of only low to moderate popularity.

Our work is aimed at reducing the latency of searching for these *warm* documents, that is, documents that are actively being accessed, but are not popular enough to be cached at very many locations, if at all. At the same time, we want to avoid sacrificing the other important goals of: ease of configuration, balancing load across the caches, efficient use of storage and bandwidth resources, and low latency for fetching popular documents.

The key idea behind our approach is to do more work at the network level. In particular, we integrate the routing of requests to caches with the routing services already provided within the network. Although deployment of the approach will involve some changes and extensions to network routers, packet headers, and endpoint protocol processing, this effort is justified by the following benefits:

- Direct access to routing and topology information enables shorter cache search paths, especially for requests for warm documents.
- Replacing application-level routing decisions by network-level decisions leads to reduced overhead.
- Caches do not need to know about each other in order to cooperate. This greatly simplifies configuration of the system.

The paper is organized as follows. In the next section, we elaborate on the cache routing problem and describe our solution. We then go on to describe how different styles of cache management can be implemented within our cache routing framework. We next present related work, and then end with a summary.

2 Cache routing

2.1 The problem

Scalability and deployment concerns have led most designers of Web caching infrastructures to design schemes based on deploying a large number of Web caches scattered around the Internet. The main challenge in such approaches is being able to quickly find a cache containing the desired document. While this problem is similar to the general problem of routing in a network, it cannot be solved in the same way.

Conventional routing protocols scale because of the route aggregation made possible by hierarchical addressing. However, since documents with the same URL prefixes or server address prefixes will not necessarily be delivered to the same clients, there is no necessary relation among their cache locations. With no way to aggregate routes, the cache routing tables would be unmanageably large. In addition, they would have to be updated frequently.

These difficulties lead to doing cache routing by searching. Some sequence of caches is checked for the document, and if it isn't found in any of them, the request is forwarded to the home server. Additionally, some of the checked caches request the document for themselves, to help out future requests.

The question, of course, is how to make this search fast? The common wisdom is to organize Web caches into some sort of hierarchy to be traversed bottom-up by each Web request, with popular documents propagating down the hierarchy towards sources of high demand. This works well for requests for very popular documents, because these documents will propagate out to many leaf caches, and so will be found quickly. However, requests for warm documents are likely to suffer. Warm documents are cached in only a few places, if anywhere. So, requests for these documents will undergo numerous failed cache checks before either finding the document, or going to the home server after all. As caching systems scale to accommodate a growing Internet, this number of failed checks will be large, introducing three significant sources of latency:

1. repeated transitions between application and network layers,
2. repeated miss processing at every failed check, and
3. travel through a large number of network hops, because cache placement is independent of network topology.

In Squid/ICP-based [16] systems such as the NLANR hierarchy in the U.S. [17], requests for Web documents are forwarded up a manually-configured cache hierarchy in search of a cached copy. In an attempt to keep from overloading caches at the root, caches query their siblings before passing requests upwards. This approach lengthens the duration of each check. As the hierarchy grows to accommodate more users and more documents, these long checks will lead to long latency for requests for warm documents. Also, as described in [20], in upper levels of the hierarchy, requests can be deflected far away from their original route to the home server. This lengthens the path traveled by requests that end up having to go to the home server.

The problem of long search paths (large number of hops) is addressed by Adaptive Web Caching [20], in which a mesh of caches is used, and distribution trees for each server are built into the mesh. The caches in the mesh are organized into overlapping multicast groups through which a request travels in search of a cached document. This scheme benefits from constructing different distribution trees for different servers (so that no one root node will be overloaded and requests will travel in the general direction of the document's home server) and from being robust and self-configuring. However, requests for warm objects still suffer because they will travel through many caches, and each check requires a query to and responses from a group of machines. The authors suggest dealing with this problem by limiting the number of caches a request can access, but this may greatly reduce the hit rate for all but the most popular pages.

2.2 Solution

The inefficiencies of current distributed Web caching systems stem from trying to do routing at the application layer, through an overlay network. Our approach avoids these inefficiencies by pushing the task of routing requests to caches down into the router/network level. In particular, we address each of the latency issues as follows:

1. In order to reduce the number of application-network transitions, we keep requests in the network layer until a cache containing the document is found. This means that cache presence checks are performed in the network layer.
2. In order to reduce the number of hops traveled by requests, we ensure that requests travel on the shortest path[1] to the home server until they encounter a router that knows of a cache that contains the desired document. That router then redirects the request to the appropriate cache. In effect, we are creating per-server distribution trees as in Adaptive Web Caching [20], except our trees better reflect the underlying network topology.
3. Because a request can experience many cache misses during its search for a document, it is important to keep miss processing short. Therefore, we do not delay requests in case of a miss; instead, we immediately forward them on toward the home server.

The idea is to diffuse server hot spots by placing *cache redirection pointers* into the network nodes that form the routing trees leading to the popular or overloaded servers. The pointers, usually placed traffic aggregation points, serve to redirect requests for cached documents to appropriate caches. To do so, each redirection pointer maps a URL to a cache address. Each participating router is associated with a nearby Web cache and stores redirection pointers for the documents stored at this cache. The router uses the pointers to identify and redirect requests for these documents to the cache. Other requests are simply forwarded on toward the home server. Several important benefits result from the redirection pointers lying directly in the path of the targeted traffic: 1) caching becomes transparent to

[1] By "shortest path," we mean the best path as determined by the routing protocol currently in use.

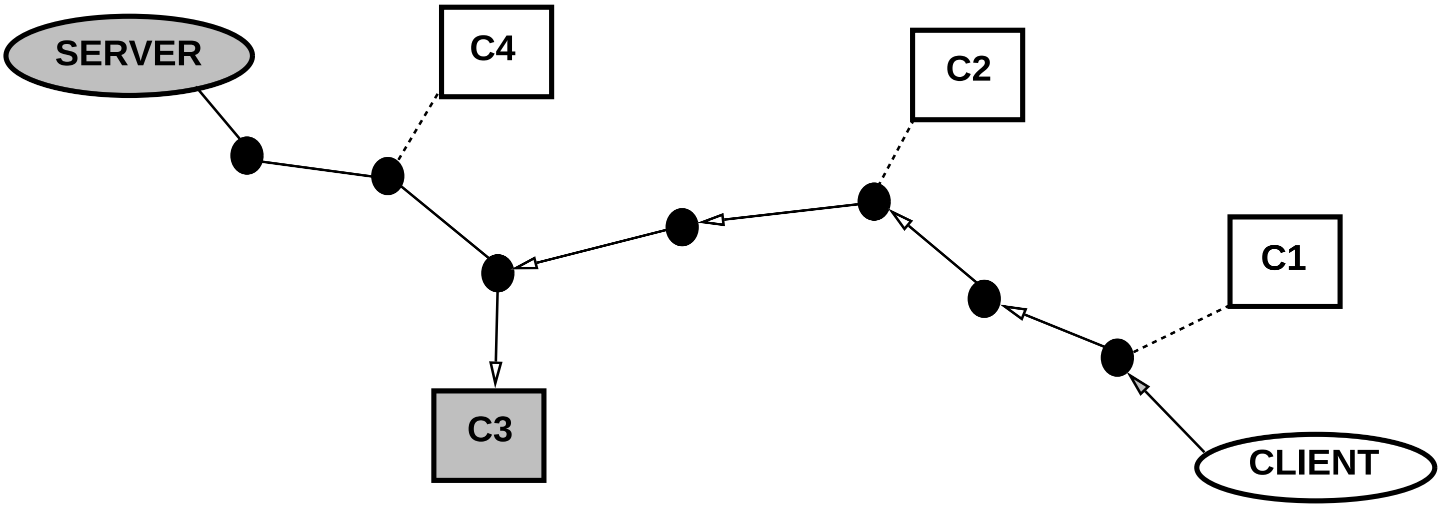

Figure 1: Cache routing example.

clients and caches: clients do not have to choose proxies, and caches do not need to be manually configured with the addresses of their neighbors, and 2) requests for items that are not cached anywhere are not redirected away from their original shortest path. Also, requests for warm documents experience at most one application-level check, and miss processing is short.

Consider the following example. In figure 1, the black nodes are the routers on the shortest network path between the client and the server. C1 to C4 are Web caches. Each dotted line connects a router to the cache that is closest to it. The arrows indicate the path traveled by a request made by the client for a document stored at the server and cached in C3. The request travels directly to C3 without stopping at C1 or C2, thus avoiding two application-level checks and two rounds of miss processing. A request for an uncached document would travel directly to the server, without stopping at any cache, and without being deflected away from the shortest path.

In our current design, cache redirection pointers are implemented as follows. Each participating router maintains a URL-indexed table of cache addresses (redirection pointers) indicating if and where a particular document is (or should be) cached at a nearby cache. To make indexing by URL easier, we index the table by the MD5 hash of the URL string. To avoid re-computation at every node, this hash is carried along in the request packet header. As each Web request packet travels through the Internet towards its URL's home server, routers along the way perform a table lookup using the URL hash in the packet. The request is redirected to a cache only if an entry is found.

The size of each router's table is controlled by the fact that the URLs of documents not cached nearby do not have table entries. Also, Bloom filter-based techniques described in [3] can be used to make the table quite compact – only several MB in size. Table lookup takes about as much time as a routing table lookup, so this slows down the routing of Web request packets by roughly a factor of two in the case of a miss.

When a redirected request arrives at a cache, the cache either services the request locally or first retrieves the document from the home server. To avoid redirection loops, routers check the source address of each request before redirecting it. Requests with source addresses that match the target cache address are not redirected, but sent on toward the home server.

Though our cache routing scheme is simple, some complications do arise in the task of deciding which documents to cache where and in maintaining this information in the appropriate routers. We address these issues in section 3.

2.3 Protocol issues

So far, our discussion of cache routing has implicitly assumed that the initial request sent out by a client contains the desired URL (and/or a hash of it), and routers along the way can use this field to route the request packet appropriately. In fact, this is not what actually happens. Because HTTP is layered on top of TCP, the first packet sent out by a client is a TCP connection setup packet (SYN), which contains no HTTP-level data. No information on the desired URL is sent out until after a connection is established. This presents a problem for cache routing, because the URL is needed beforehand, in order to select a cache with which to set up a connection.

To avoid the overhead of setting up a connection with each cache that is checked, Squid-based caches use the Internet

Cache Protocol (ICP) [16]. ICP is a connectionless protocol designed to be used for querying caches for particular documents. Since each ICP query specifies a URL, our scheme could easily route ICP query packets to appropriate caches. However, using ICP requires an extra round-trip to resolve each URL to a cache IP address. A better solution is to combine the ICP query and TCP SYN into one message: i.e., include the URL or a hash of it in the SYN packet. The most reasonable way to do this is to define an IP option [11] for this purpose – i.e., to hold the URL hash in the packet header for routers to access. Despite the bureaucratic obstacle to introducing and accepting new IP options, we believe this to be the correct long-term solution. It will require little change to browsers – simply setting an option field before sending out each SYN packet.

One potential interim solution is to route SYN packets to caches based only on the home server IP address contained in the packet header, not based on the complete URL. That is, each router would store a table indicating which servers have documents cached nearby, and use it to redirect SYN packets headed toward these servers to appropriate caches. While some requests would be incorrectly redirected to caches not containing the desired document, many requests would still be able to benefit from the ability to bypass caches.

Another interesting interim solution is implemented in DynaCache [6], a network-level Web cache. It performs URL-based redirection through TCP-level connection spoofing.[2] Since SYN packets do not contain URLs, DynaCache does not intercept these but forwards them on to the home server. Once the client-server connection has been established, DynaCache snoops on the connection and intercepts the client packet that contains the "GET URL" request. It looks up the URL in a table, and if the document is present in its cache, it transfers the document to the client by pretending to be the server. Also pretending to be the client, DynaCache uses the already-established connection to the server to forward HTTP cookie information and to request freshness information on the URL. On a miss, DynaCache simply records the desired URL and forwards the request on to the home server. This keeps the miss penalty small, and allows DynaCache to keep track of changes in document popularity.

Though it is clever, there are two drawbacks to the DynaCache approach. First, it requires a connection to the home server on every document access. Second, the spoofing-based scheme relies on routing symmetry and stability: the forward and return paths between client and server must both go through the link on which DynaCache sits, and they must do so for the entire duration of each document transfer. The well-documented problems of route flapping and routing asymmetry [10, 7] may, therefore, cause this scheme to be impractical. In contrast, our proposed approach of including the URL in the connection setup packet avoids these difficulties.

[2] Although it describes some implementation details, the DynaCache technical literature does not provide a complete description of how their URL-based redirection scheme is implemented. We believe that spoofing is used because it seems to be the only way to accomplish URL-based redirection today and because it is consistent with other advertised characteristics of their approach.

3 Cache management

In most Web caching systems, caches observe all the Web request traffic passing through a particular link or network region, and thus can make a well-informed decision about which documents to cache. In our scheme, caches only see requests for documents that have already been flagged for caching. This means that document popularity must be determined in another way. Also, the appropriate redirection pointers must be installed to reroute requests. In this section, we describe how this can be done either with a variant of the traditional approach, where caches determine document popularity, or with server-controlled push-caching. We also discuss the important issues of pointer removal, cache discovery, and deployment.

3.1 Cache-controlled document selection

Our scheme can easily be made to support traditional cache-controlled document selection. Each router simply records the URLs of the requests passing through it and forwards this information on to its associated Web cache. In this way, the cache sees all Web request traffic (without slowing it down), and so can determine which documents are worth caching. To enable caching of a document found to be popular, the cache sends a cache redirection pointer to the appropriate router to be installed in its table. Future requests for the document will be forwarded to the cache. This approach is similar to that implemented in DynaCache [6].

3.2 Push-caching

Although cache-controlled document selection is fairly effective, there are reasons for wanting to give servers control over placement of their documents. For instance, a server of only small to medium size may be willing to purchase cache space for its popular documents during peak periods, since its documents may not be popular enough on a large-scale to merit selection by any cache itself. Or, a server may want to set up a mirror of popular documents in anticipation of a peak period. In either case, the server would have to be able to determine which documents to cache and where.

The idea of having a server decide when and where to cache its documents was introduced as *push-caching* in [5]. In the original design, each server tracked geographical access information by mapping client IP addresses to zip codes and then to longitude/latitude coordinates, and mirrored its data accordingly. Redirection of requests to an appropriate mirror was performed at the home server. In the rest of this section, we present a network-based approach that improves upon the original push-caching scheme in several ways. First, we collect more accurate topology information to help choose a better location for the mirror. Second, our scheme allows the server to discover potential caches easily and dynamically, without having to be pre-configured with all possible cache addresses. Also, our redirection pointers intercept requests and redirect them to the mirror automatically, avoiding the latency otherwise required to find out about possible mirrors from the home server.

The approach works as follows. A server, upon detecting that it is busy, and perhaps soon to be overloaded, performs traffic analysis to determine which documents to cache where. It first triggers the packets traveling to it to collect the IP addresses of the participating routers they travel through, as well as the information routers have about the caches found near them. The server then uses this information to reconstruct a routing tree leading to it. Using this tree, the server figures out where most of the traffic is concentrated, and where caching would be most useful. The server then sends a message to the appropriate caches to enable caching of its documents there. Upon agreeing to cache the documents, the cache installs the appropriate cache redirection pointers in nearby routers.

The specifics of the scheme are as follows. A server detects that it is busy by observing an increase in its request rate. While busy, the server periodically monitors the source of its high load by enabling path-tracing on all new connections for a period of time. During this interval, the server responds to document requests with an ACK packet that has a special trace-enable flag set. This prompts the client to send out its next packet (an ACK, or request for data) as a path-tracing packet, that collects the IP addresses of all routers it passes through, in addition to the IP addresses of the cache (if any) associated with each router. The server collects this information for many connections into a tree data structure resembling the routing tree leading to it. After the monitoring period is over, the server traverses the tree to determine the network regions generating a large fraction of its traffic, and chooses routers at the roots of such regions to serve as redirection points for its traffic. The server then sends a reliable message containing the URLs of its most popular documents to the caches associated with the chosen routers. Upon agreeing to cache a set of documents, each cache installs the appropriate redirection pointers in nearby routers.

Because cache size is limited, the documents pushed by one server may be evicted to make room for more popular documents from another server. This may cause a server to repeatedly and frequently request cache space from a particular cache. To avoid this, the server employs a backoff strategy for each cache it contacts during a busy period.

3.3 Removing pointers

No matter which document selection scheme is used, it will be necessary to disable some redirection pointers when patterns of demand change. To remove stale redirection pointers, we rely on two mechanisms. First, routers employ an LRU (or other appropriate) policy for table entries, so some entries for documents that are no longer popular will be evicted because of space constraints. Second, a cache may choose to delete the redirection pointers for documents that it has evicted from its own storage. In order to enable this, every time a router performs a redirection, it appends its own IP address to the packet header. Then, when a cache receives a request for a recently evicted page, it can send a pointer removal message to the appropriate router.

3.4 Cache discovery and fault tolerance

In order for routers to discover the caches around them (and vice versa), caches send out periodic broadcasts (of limited radius or scope) and routers respond by sending an ACK to the cache. Routers not receiving any broadcast messages do not participate in cache redirection. If a router receives more than one cache announcement message, it can choose to contact a subset of the caches based on a cache characteristic advertised in the message (e.g., distance from router, size, load). This cache discovery scheme makes it easy to deploy new caches because it does not require manual configuration of routers with cache addresses. In addition, it enables routers to dynamically adapt to changing conditions by selecting different caches.

In order to keep from redirecting requests to a cache that is no longer responding, these broadcasts also serve as keep-alive messages. A router does not redirect requests to any cache it has not heard from recently.

3.5 Deployment issues

Deploying a scheme of this kind is not a trivial matter. Our design requires changes and extensions to network routers, packet headers, and endpoint protocol processing. Instead of treating all packets alike, routers must perform different processing for document request (connection setup) packets and for data transfer and acknowledgement packets.

Incorporating such a scheme into the current Internet would require replacing a large number of routers (though not all). Instead of attempting that, we are using Active Network [14] technology as a deployment mechanism. An active router is one that can be dynamically extended to provide many different types of services beyond conventional IP routing and forwarding. This extensibility makes it possible to deploy new services while avoiding the difficulty of modifying existing routers. In an ANTS-style [18] active network, a packet specifies a particular service routine to be loaded and run on its behalf on each active router it encounters. This routine can perform arbitrary (but short) computation, store information in soft-state, and create and send packets back out into the network. The ANTS system takes care of obtaining the code for this routine from its own code cache or from previous nodes in the packet's path. The default routine performs conventional IP forwarding. We are currently implementing our caching scheme on the ANTS network in our lab, and plan to deploy it on the nationwide (U.S.) ABone.

4 Related Work

In this section, we describe how our work relates to other approaches to cache routing.

A recurring problem in many Web caching systems is the long latency incurred by requests for objects that are cached in very few places, if anywhere. Section 2 described why this is the case for multicast-based Adaptive Web Caching [20] and hierarchical approaches like Squid/NLANR [17]. Here we provide several more examples.

Like Squid/NLANR, Povey and Harrison [12] construct a manually-configured hierarchy that must be traversed by all requests. Their scheme is promising in that it reduces load on top-level caches by keeping only location pointers in the hierarchy, but it suffers from the same inefficiencies for less popular documents.

Panigrahy [9] describes a theoretically-based technique for constructing per-server distribution trees. This technique exhibits excellent load balancing properties. However, since server load is not taken into account when determining tree size, the trees for small and/or lightly-loaded servers may be too big, resulting in many useless application-level checks. Also, since trees are random, the client-server paths through them are likely to be much longer than the regular routing path from client to server.

Wang and Crowcroft [15] describe a preliminary plan to put cache routing tables in caches to specify, for each page or server, where to look next if the local cache does not hold the document. A default route for some documents would help keep table size reasonable. Our approach can be considered to be a network-supported implementation of this very general idea. Our tables are located in routers (separate from the caches), and the default route is the path to the home server.

Recent work on SummaryCache [3] shows that the list of URLs stored in a cache can be represented in a very compact table. As a consequence, caches can easily store information about the contents of a number of other caches, and redirect requests to appropriate caches using only local lookup operations. While this approach will certainly be useful for improving cache routing, it cannot, on its own, scale to accommodate all caches in the Internet. Since the information is represented in individual per-cache tables, a large number of lookup operations would be needed to find all but the most popular documents. Combining SummaryCache with network-level request redirection seems to be a more promising approach. We plan to investigate the benefits of such a scheme in future work.

A number of recent products provide network-level support for cache routing. The most common approach is to have a switch or router redirect all incoming Web traffic to its associated Web cache or cache cluster [13, 19, 8]. While this helps make caching transparent, it unnecessarily sends requests for warm documents to every cache in their paths. Dynacache [6] succeeds in avoiding this problem, but exhibits several other drawbacks described earlier in section 2. In a different approach, products such as the ArrowPoint Content-Smart Switch [1] can perform URL-based request redirection by acting as TCP-level routers (TCP forwarders) [2]. This approach is reasonable as a point solution to the problem of choosing a machine in a cache cluster, but is not an appropriate tool with which to build a global cache hierarchy. The result would be much too heavyweight, requiring: 1) routers to act as reliable connection endpoints (maintaining connection state and performing flow control), and 2) requests and data to be forwarded through a potentially long series of TCP connections.

5 Summary

Our work is motivated by the need to provide effective caching not only for the small number of extremely popular Web documents, but also for the large number of documents of only intermediate popularity. In all the currently existing and proposed systems with which we are familiar, requests for these warm documents suffer from long latency stemming from the inefficiencies of routing through an application-level overlay network. We combat this problem by integrating cache routing into the network layer in a way that results in shorter search paths, fewer application-level checks, and shorter miss processing.

In our scheme, per-document redirection pointers located in routers serve to direct requests for popular documents to nearby caches. Router support for path tracing allows servers to gather the information needed to make intelligent choices on pointer placement. The result is that requests for warm documents are allowed to travel quickly to the home server or to a cache near a traffic aggregation point. This approach exhibits the benefits of having caches in routers without actually having to put them there. Also, configuration of the system is simplified by the fact that caches do not need to know about each other in order to cooperate.

The key technology enabling our approach is a collection of router mechanisms that export network-level information to the application – in this case, Web caching. In some sense, this dependence on network-level support is a liability because it makes it difficult to implement our approach in the current network infrastructure. On the other hand, the benefits of our scheme demonstrate the need for such network-level support, and our work can serve to suggest new services that should be incorporated into future routers.

Acknowledgments

We thank David Wetherall for many helpful discussions about network caching and for his work on ANTS. We also thank Frans Kaashoek, Vanu Bose, and Hariharan S. Rahul for their insightful comments on the design of our approach.

This work was supported by DARPA and by seed funding from Sun Microsystems Inc., and monitored by the Office of Naval Research under contract No. N66001-96-C-85221.

References

[1] Arrowpoint Communications. CS-100 data sheet. http://www.arrowpoint.com.

[2] Peter Danzig and Karl L. Swartz. Transparent, scalable, fail-safe web caching. Technical Report 3033, Network Appliance, Inc., 1998.

[3] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. In *Proceedings of SIGCOMM '98*, September 1998.

[4] National Laboratory for Applied Network Research (NLANR). Hierarchical caching system usage statistics. http://ircache.nlanr.net/Cache/Statistics/Hierarchy/, 1998.

[5] James Gwertzman and Margo Seltzer. The case for geographical push-caching. In *HotOS-V*, 1995. http://www.eecs.harvard.edu/˜vino/web/hotos.ps.

[6] Abdelsalam Heddaya. DynaCache: Weaving caching into the internet. Presented at the 3rd International WWW Caching Workshop, Manchester, England, June 1998.

[7] Craig Labovitz, G. Robert Malan, and Farnam Jahanian. Internet routing instability. In *Proceedings of SIGCOMM '97*, September 1998.

[8] Katia Obraczka, Peter Danzig, Solos Arthachinda, and Muhammad Yousuf. Scalable, highly available web caching. Technical Report 97-662, University of Southern California Computer Science Department, December 1997.

[9] Rina Panigrahy. Relieving hot spots on the World Wide Web. Master's thesis, MIT EECS, June 1997.

[10] Vern Paxson. End-to-end routing behavior in the internet. In *Proceedings of SIGCOMM '96*, August 1998.

[11] Jon Postel. Internet protocol. Technical Report Internet RFC 791, USC/Information Sciences Institute, 1981.

[12] Dean Povey and John Harrison. A distributed Internet cache. In *20th Australian Computer Science Conference*, February 1997. http://www.psy.uq.edu.au:8080/˜dean/project/.

[13] Cisco Systems. Cisco Cache Engine. http://www.cisco.com/warp/public/751/cache/cds_ov.htm, September 1997.

[14] David Tennenhouse, Jonathan Smith, David Sincoskie, David Wetherall, and Gary Minden. A survey of active network research. *IEEE Communications Magazine*, January 1997.

[15] Zheng Wang and Jon Crowcroft. Cachemesh: A distributed cache system for the World Wide Web. NLANR Web cache workshop http://ircache.nlanr.net/Cache/Workshop97/agenda.html, June 1997.

[16] Duane Wessels. Squid and ICP: Past, present, and future. AUUG Conference, September 1997. http://www.nlanr.net/%7ewessels/Papers/auug-keynote.ps.gz.

[17] Duane Wessels and Kim Claffy. Evolution of the NLANR cache hierarchy: Global configuration challenges. http://www.nlanr.net/Papers/Cache96, November 1996.

[18] David Wetherall, John Guttag, and David Tennenhouse. ANTS: A toolkit for building and dynamically deploying network protocols. In *Proceedings of IEEE OPENARCH*, San Francisco, CA, April 1998.

[19] Bert Williams. Transparent web caching solutions. Presented at the 3rd International WWW Caching Workshop, Manchester, England, June 1998.

[20] Lixia Zhang, Sally Floyd, and Van Jacobson. Adaptive Web caching. NLANR Web cache workshop http://ircache.nlanr.net/Cache/Workshop97/agenda.html, June 1997.

Vitae

Ulana Legedza is a Ph.D. candidate at the MIT Laboratory for Computer Science. Her research interests are in computer systems, networking, and parallel computing. Her current focus is on the design of network-level support for application-specific routing functions. She received the B.S.E. degree in Computer Science from Princeton University in 1992, and the M.S. degree in Computer Science from MIT in 1995.

John Guttag is Associate Department Head, Computer Science, of MIT's Electrical Engineering and Computer Science Department, and head of the Laboratory for Computer Science's Software Devices and Systems Group. The group does research in networking, distributed computing, computer and communications security, and wireless communications. Professor Guttag has also done research, published, and lectured in the areas of software engineering, mechanical theorem proving, hardware verification, and compilation.