Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol

Li Fan
Pei Cao
Jussara Almeida

Technical Report #1361

February 1998

Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol

Abstract

The sharing of caches among Web proxies is an important technique to reduce Web traffic and alleviate network bottlenecks. Nevertheless it is not widely deployed due to the overhead of existing protocols. In this paper we demonstrate the benefits of cache sharing, measure the overhead of the existing protocols, and propose a new protocol called "Summary Cache". In this new protocol, each proxy keeps a summary of the cache directory of each participating proxy, and checks these summaries for potential hits before sending any queries. Two factors contribute to our protocol's low overhead: the summaries are updated only periodically, and the directory representations are very economical, as low as 8 bits per entry. Using trace-driven simulations and a prototype implementation, we show that, compared to existing protocols such as the Internet Cache Protocol (ICP), Summary Cache reduces the number of inter-cache protocol messages *by a factor of 40 to 65*, reduces the bandwidth consumption *by over 50%*, eliminates *75% and 95%* of the protocol CPU overhead, all while maintaining almost the same cache hit ratio as ICP. Hence Summary Cache scales to a large number of proxies.

1 Introduction

As the tremendous growth of the World-Wide Web continues to strain the Internet, caching has been recognized as one of the most important techniques to reduce bandwidth consumption [26]. In particular, caching within Web proxies has been shown to be very effective [13, 30]. To gain the full benefits of caching, proxy caches behind a common bottleneck link should cooperate and serve each other's misses, thus further reducing bottleneck traffic. We call this cooperation "Web cache sharing."

Web cache sharing was first proposed in the context of the Harvest project [23, 11]. The Harvest group designed the Internet Cache Protocol (ICP) [15] that supports discovery and retrieval of documents from neighboring caches. Today, many institutions and many countries have established hierarchies of proxy caches that cooperate via ICP to reduce traffic to the Internet [22, 27, 39, 4, 13].

Nevertheless, the wide deployment of web cache sharing is currently hindered by the overhead of the ICP protocol. ICP discovers cache hits in other proxies by having the proxy multicast a query message to *all* other proxies whenever a cache miss occurs. Thus, as the number of proxies increases, both the communication and the CPU processing overhead increase quadratically. A number of alternative protocols have been proposed to address the problem, for example, a cache array routing protocol that partitions the URL space among proxies [43]. However, such solutions are often not appropriate for wide-area cache sharing, which is characterized by limited network bandwidth among proxies and non-uniform network distances between proxies and their users (for example, each proxy might be much closer to one user group than to others).

In this paper, we address the issue of scalable protocols for wide-area Web cache sharing. We first examine the benefits of Web cache sharing by analyzing a collection of Web access traces. We show that sharing cache contents among proxies significantly reduces traffic to Web servers, and that simple cache sharing, which imposes no coordination among cache replacements of proxies, suffices to obtain most of the benefits of fully coordinated caching. We also quantify the overhead of the ICP protocol by running a set of proxy benchmarks. The results show that even when the number of cooperating proxies is as low as four, ICP increases the inter-proxy traffic by a factor of 70 to 90, the number of network packets received by each proxy by 13% and higher, and the CPU overhead by over 15%. In the absence of inter-proxy cache hits (also called remote cache hits), the overhead can increase the average user latency by up to 11%.

We then propose a new cache sharing protocol called Summary Cache. Under this protocol, each proxy keeps a compact summary of the cache directory of every other proxy. When a cache miss occurs, a proxy first probes all the summaries to see if the request might be a cache hit in other proxies, and sends a query messages only to those proxies whose summaries show promising results. The summaries do not need to be accurate at all times. If a request is not a cache hit when the summary indicates so (a false hit), the penalty is a wasted query message. If the request is a cache hit when the summary indicates otherwise (a false miss), the penalty is a higher miss ratio.

We examine two key questions in the design of the protocol: the frequency of summary updates and the representation of summary. Using trace-driven simulations, we show that the update of summaries can be delayed until a fixed percentage (for example, 1%) of cached documents are new, and the hit ratio will degrade proportionally (For the 1% choice, the degradation is between 0.02% to 1.7% depending on the traces).

To reduce the memory requirements, we store each summary as a "Bloom filter" [5]. This is a computationally very efficient hash-based probabilistic scheme that can represent a set of keys (in our case, a cache directory) with minimal memory requirements while answering membership queries with 0 probability for false negatives and low probability for false positives. Trace-driven simulations show that with typical proxy configurations, for N cached documents represented within just N bytes, the percentage of false positives is 1% to 2%. In fact, the memory can be further reduced at the cost of an increased false positive ratio. (We describe Bloom filters in more detail below.)

Based on these results, we designed the Summary-Cache Enhanced ICP protocol and implemented a prototype within the Squid proxy. Using trace-driven simulations as well as experiments with benchmarks and trace-replays, we show that the new protocol reduces the number of inter-proxy messages *by a factor of 40 to over 65*, reduces the network bandwidth consumption (in terms of bytes transferred) *by over 50%*, and eliminates *75% to 95%* of the protocol CPU overhead. Compared with no cache sharing, our implementation experiments show that the protocol incurs little network and CPU overhead when there are no remote cache hits, and only increases network traffic by 7% and CPU overhead by 11% at the remote cache hit ratio of 10%. Yet, the protocol achieves a cache hit ratio similar to the ICP protocol most of the time.

The low overhead shown in the simulation and implementation results indicates that the Summary Cache Enhanced ICP protocol can scale to a large number of proxies. Thus, it has the potential to significantly increase the deployment of Web cache sharing and reduce Web traffic on the Internet. Toward this end, we are making our implementation publicly available [36] and are in the process of transferring it to the Squid users community.

2 Traces and Simulations

For our study we have collected five sets of traces of HTTP requests. The number of requests in each trace, the number of clients, and other statistics are listed in Table 1. In particular, Table 1 lists the "infinite" cache size for each trace, which is the total size in bytes of unique documents in the trace (i.e. the size of the "infinite" cache which incurs no cache replacement).

- **DEC** traces [29]: Digital Equipment Corporation Web Proxy server traces, servicing about 17,000 workstations. The trace is for a period of 25 days (Aug. 29 to Sep. 21, 1996). We partitioned the trace into three one-week traces and one half-week traces. Our simulator can only simulate the subtraces due to swap-space limitations. In this paper, we present the results on the trace of the week of Aug. 29 to Sep. 4, 1996. Results on other traces are very similar.

- **UCB** traces [21]: traces of HTTP requests gathered from the Home IP service offered by UC Berkeley to its students, faculty, and staff. The total trace is for a period of 18 days from Nov. 1 till Nov. 19, 1996, and is partitioned into four subtraces covering every four or five days. We present the results on the traces from Nov. 14 till Nov. 19. Though the trace originally records 2,468,890 requests, many of them have response data sizes of 0 or 1, and we decide to ignore those requests [1]. Again, we have run the simulations on other traces in the UCB collections, and the results are similar to what are presented here.

- **UPisa** traces [41]: traces of HTTP requests made by the users in Computer Science Department in Universita di Pisa, Italy, for a period of three months from January to March, 1997. Of the traces, we only simulate GET requests, and only those whose URLs do not include query strings, since most proxies do not cache query requests.

[1] The change may have contributed to the difference between our hit ratio results on UCB and those reported in [20].

Traces	DEC	UCB	UPisa	Questnet	NLANR
Time	8/29-9/4, 1996	9/14-9/19, 1996	Jan-March, 1997	1/15-1/21, 1998	12/22, 1997
Requests	3543968	1907762	2833624	2885285	1766409
Infinite Cache Size	2.88e+10	1.80e+10	2.07e+10	2.33e+10	1.37e+10
Maximum Hit Ratio	0.49	0.30	0.40	0.30	0.36
Maximum ByteHit Ratio	0.36	0.14	0.27	0.15	0.27
Client Population	10089	5780	2203	12	4
Client Groups	16	8	8	12	4

Table 1: Statistics about the traces. The hit ratio and byte hit ratio are achieved under infinite cache.

- **Questnet** traces [44]: 7-days worth of logs of HTTP requests seen by the parent proxies at Questnet, which is a regional network in Australia, from Jan. 15 to Jan. 21, 1998. The proxies are parent proxies serving about 12 child proxies in the regional network. We extract successful GET requests seen by the parent proxies. Thus, the trace is only a subset of user requests going to the ten proxies. Unfortuantely, the full set of user requests to the proxies are not available.

- **NLANR** traces [35]: one-day log (Dec. 22, 1997) of HTTP requests to four major parent proxy caches in the National Caching hierarchy by NLANR (National Lab of Applied Network Research). There are about eight proxies in the National caching hiearchy, but only four of them ("bo", "pb", "sd", and "uc") handle documents from the servers in .com, .net, .edu, and other major domains[2]. Thus, we decide to simulate requests to the four proxies only.

In our simulation of cache sharing, we partition the clients in DEC, UCB and UPisa into groups, assume that each group has its own proxy, and simulate the cache sharing among the proxies. This roughly corresponds to the scenario where each branch of a company or each department in a university has its own proxy cache, and the caches collaborate. We set the number of groups in DEC, UCB and UPisa traces to 16, 8 and 8, respectively. A client is put in a group if its clientID mod the group size equals the group ID. Though the simulation does not exactly correspond to reality, we believe it does bring insight on cache sharing protocols. Questnet traces contain HTTP requests coming from a set of child proxies in the regional network to the parent proxy. We assume that these are the requests going into the child proxies (since the child proxies send their cache misses to the parent proxy), and simulate cache shairng among the child proxies. Finally, NLANR traces contains actual HTTP requests going to the four major proxies, and we simulate the cache sharing among them.

In all our simulations, we use LRU as the cache replacement algorithm, with the restriction that documents larger than 250KB is not cached. The policy is similar to what are used in actual proxies. We do not simulate expiring documents based in age or time-to-live. Rather, most of our traces come with the last-modified time of a document for every request, and if a user request hit on a document whose last-modified time is changed, we count it as a cache miss. Thus, in our simulations we ignore the cache consistency issues that arise in practice. There are many other protocols [11, 31, 25] that address the cache consistency issue in real life.

Most of our simulations assume a cache size that is 10% of the "infinite" cache size. Studies have shown that 10% of the "infinite" cache size typically achieves about 90% of the maximum cache hit ratio [46, 7, 32]. We also performed simulations with cache sizes being 5% of the inifinite cache size and the results are very similar.

3 Benefits of Cache Sharing

Recent studies [7, 20, 13] have shown that under infinite cache capacity, Web cache hit ratio appears to grow logarithmically with the size of the user population served by the cache. Clearly, the overlap of requests from different users reduces the number of cold misses, often a significant portion of all misses, since both first-time reference to documents and document modifications contribute to them.

To examine the benefits of cache sharing under finite cache sizes, we simulated the following schemes using the traces listed in the previous section:

- *No Cache Sharing*: proxies do not collaborate to serve each other's cache misses;

[2]This can be seen in the cache figuration files at http://ircache.nlanr.net/Cache/Configuration/

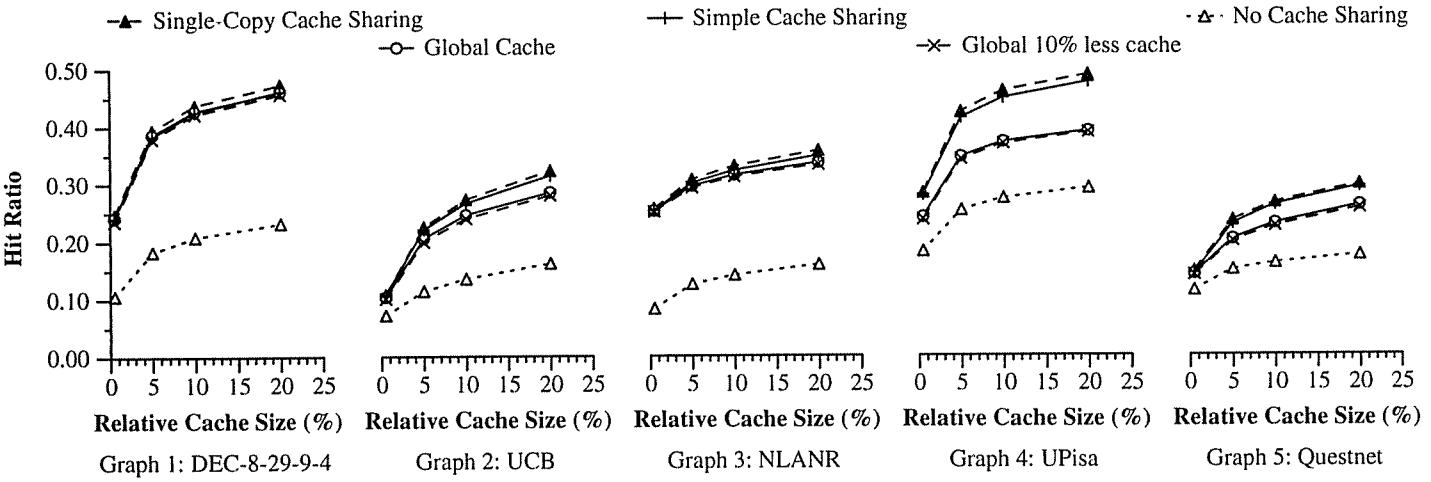

Figure 1: Cache hit ratios under different cooperative caching schemes. Results on byte hit ratios are similar.

- *Simple Cache Sharing*: proxies serve each other's cache misses. Once a proxy fetches a document from another proxy, it caches the document locally. Proxies do not coordinate cache replacements. This is the sharing implemented by the ICP protocol.
- *Single-Copy Cache Sharing*: proxies serve each other's cache misses, but a proxy does not cache documents fetched from another proxy. Rather, the other proxy marks the document as most-recently-accessed, and increases its caching priority. Compared with simple cache sharing, this scheme eliminates the storage of duplicate copies and increases the utilization of available cache space.
- *Global Cache*: proxies share cache contents and coordinate replacement so that they appear as one unified cache with global LRU replacement to the users. This is the fully coordinated form of cooperative caching. We simulate the scheme by assuming that all requests go to one cache whose size is the sum of all proxy cache sizes.

We examine these schemes in order to answer two questions: whether simple cache sharing significantly reduces traffic to Web servers, and whether the more tightly coordinating schemes lead to a significantly higher hit ratio.

Figure 1 shows the hit ratios under the different schemes considered when the cache size is set to 0.5%, 5%, 10%, and 20% of the size of the "infinite cache size" (the minimum cache size needed to completely avoid replacements) for each trace. The results on byte hit ratios are very similar, and we omit them due to space constraints.

Looking at Figure 1, we see that, first, all cache sharing schemes significantly improve the hit ratio over no cache sharing. The results amply confirm the benefit of cache sharing even with fairly small caches.

Second, the hit ratio under single-copy cache sharing and simple cache sharing are generally the same or even higher than the hit ratio under global cache. We believe the reason is that global LRU sometimes performs less well than group-wise LRU. In particular, in the global cache setting a burst of rapid successive requests from one user might disturb the working set of many users. In single-copy or simple cache sharing, each cache is dedicated to a particular user group, and traffic from each group competes for a separate cache space. Hence, the disruption is contained within a particular group.

Third, when comparing single-copy cache sharing with simple cache sharing, we see that the waste of space has only a minor effect. The reason is that a somewhat smaller effective cache does not make a significant difference in the hit ratio. To demonstrate this, we also run the simulation with a global cache 10% smaller than the original. As can be seen from Figure 1, the difference is very small.

Thus, despite its simplicity, the ICP-style simple cache sharing reaps most of the benefits of more elaborate cooperative caching. Simple cache-sharing does not do any load balancing by moving content from busy caches to less busy ones, and does not conserve space by keeping only one copy of the documents. However, if the resource planning for each proxy is done properly, there is no need to perform load-balancing and incur the overhead of more tightly coordinating schemes.

Finally, note that the results are obtained under the LRU replacement algorithm explained in Section 2. Different replacement algorithms [7] may give

different results. Also, separate simulations have confirmed that in case of severe load imbalance, the global cache will have a better cache hit ratio, and therefore it is important to allocate the cache size of each proxy in proportion to its user population size and anticipated use.

4 Overhead of ICP

The Internet Cache Protocol (ICP) [15] has been very successful at encouraging the practice of Web cache sharing around the world. It requires loose coordinations among the proxies, and is built on top of UDP for efficiency. It was designed by the Harvest research group [23] and supported by both the public-domain Squid [16] proxy software and some commercial products today. With the deployment of Squid proxies around the globe, ICP is widely used by international countries to reduce traffic over trans-Atlantic and trans-Pacific links.

Despite its success, ICP is not a scalable protocol. The problem is that ICP relies on queries to find out remote cache hits. Everytime one proxy has a cache miss, everyone else receives a query message and processes it. As the number of collaborating proxies increases, the overhead quickly becomes prohibitive.

To measure the overhead of ICP and its impact on proxy performance, we run experiments using a proxy benchmark designed by us [37]. (The benchmark has been submitted to SPEC as a candidate for the industry standard benchmark and is currently in-use at a number of proxy system vendors.) The benchmark consists of a collection of client processes that issue requests following patterns observed in real traces, including request size distribution and temporal locality, and a collection of server processes that delay the replies to emulate latencies in the Internet.

The experiments are performed on 10 Sun Sparc-20 workstations that are connected with 100Mb/s Ethernet. Four workstations act as four proxy systems, running Squid 1.1.14, and each having 75MB of cache space[3]. Another four workstations run 120 client processes, 30 processes on each workstation. The client processes on each workstation connect to one of the proxies. Client processes issue requests with no thinking time in between, and the requested document size follow the Pareto distribution with $\alpha = 1.1$ and $k = 3.0$ [8]. Finally, two workstations act as servers, each with 15 servers listing on different ports. The Web server forks off a process when handling an HTTP request, and the process waits for 1 second before sending the reply to simulate the network latency.

We experiment with two different cache hit ratios, 25% and 45%, as the overhead of ICP varies with the cache miss ratio in each proxy. In the benchmark, the client issues requests following the temporal locality patterns observed in [32, 7], and the inherent cache hit ratio in the request stream can be adjusted. In an experiment, each client process issues 200 requests, for a total of 24000 requests.

Using the benchmark, we compare two configurations: *no-ICP*, where proxies do not collaborate, and *ICP*, where proxies collaborate via ICP. Since we are only interested in the overheads, the requests issue by the clients do not overlap, and there is no remote cache hits among the proxies. This is the worst case scenario for ICP, and the results measure the overhead of the protocol. We use the same seeds in the random number generators for the no-ICP and ICP experiments to ensure comparable results, since otherwise the heavy-tailed document size distribution and our low request numbers lead to high variance. We present results from one set of experiments here. (We have done more and the results are similar.)

We measure the hit ratio in the caches, the average latency seen by the clients, the CPU time consumed by the Squid proxy in terms of user CPU time and system CPU time, and network traffic. Using netstat, we collect the number of UDP datagrams sent and received, the TCP packets sent and received, and the total number of IP packets handled by the Ethernet network interface. The third number is roughly the sum of the first two. The UDP traffic is incurred by the ICP query and reply messages. The TCP traffic include the HTTP traffic between the proxy and the servers, and between the proxy and the clients. The results are shown in Table 2.

The results show that ICP incurs considerable overhead even when the number of cooperating proxies is as low as four. The number of UDP messages is increased by a factor of 73 to 90. Due to the increase in the UDP messages, the total network traffic seen by the proxies are increased by 8% to 13%. Protocol processing increases the user-mode CPU by 20% to 24%, and UDP messages processing increases the system CPU time by 7% to 10%. Reflected to the clients, the average latency of an HTTP request is increased by 8% to 11%. The degradations occur despite the fact that the experiments are performed on a high-speed local area network.

The results highlight the dilemma faced by Web cache administrators. There are clear benefits of cache sharing, and yet the overhead of ICP is high. Further-

[3]The cache size is artificially small so that cache replacement occurs during the short duration of the experiments. For our final version we plan to experiment with more realistic cache sizes.

Exp 1	Hit Ratio	Client Latency	User CPU	System CPU	UDP Msgs	TCP Msgs	Total Packets
no ICP	25%	2.75 (5%)	94.42 (5%)	133.65 (6%)	615 (28%)	334K (8%)	355K(7%)
ICP	25%	3.07 (0.7%)	116.87 (5%)	146.50 (5%)	54774 (0%)	328K (4%)	402K (3%)
Overhead		*12%*	*24%*	*10%*	*90*	*2%*	*13%*
SC-ICP	25%	2.85 (1%)	95.07 (6%)	134.61 (6%)	1079 (0%)	330K (5%)	351K (5%)
Overhead		*4%*	*0.7%*	*0.7%*	*75%*	*-1%*	*-1%*
Exp 2	Hit Ratio	Client Latency	User CPU	System CPU	UDP Msgs	TCP Msgs	Total Packets
no ICP	45%	2.21 (1%)	80.83 (2%)	111.10 (2%)	540 (3%)	272K (3%)	290K (3%)
ICP	45%	2.39 (1%)	97.36 (1%)	118.59 (1%)	39968 (0%)	257K (2%)	314K (1%)
Overhead		*8%*	*20%*	*7%*	*73*	*-1%*	*8%*
SC-ICP	45%	2.25 (1%)	82.03 (3%)	111.87 (3%)	799 (5%)	269K (5%)	287K (5%)
Overhead		*2%*	*1%*	*1%*	*48%*	*-1%*	*-1%*

Table 2: Overhead of ICP in the four-proxy case. The SC-ICP protocol is introduced in Section 6 and will be explained later. The experiments are run three times, and the variance for each measurement is listed in the parenthesis. The overhead row lists the increase in percentage for each measurement. Note that in the synthetic experiments there is no inter-proxy cache hit.

more, most of the time the processing of query message is wasted because the document is not cached. Essentially, the effort spent on processing ICP is proportional to the total number of cache misses experienced by other proxies, instead of proportional to the number of actual remote cache hits.

To address the problem, we propose a new scalable cache sharing protocol *Summary Cache*.

5 Summary Cache

In the summary cache scheme, each proxy stores a summary of its directory of cached document in every other proxy. When a user request misses in the local cache, the local proxy checks the stored summaries to see if the requested document might be stored in other proxies. If it appears so, the proxy sends out requests to the relevant proxies to fetch the document. If it is not, the proxy sends the request directly to the Web server.

The key to the scalability of this scheme is that summaries do not have to be perfectly up to date or perfectly accurate. A proxy does not have to update the copy of its summary stored with other proxies upon every modification of its directory: it can wait until a certain percentage of its cached documents are not reflected in other proxies. In other words, two kinds of errors can be tolerated:

- *false misses*: the document requested is cached at some other proxy but its summary does not reflect it. In this case, a remote cache hit is not taken advantage of, and the total hit ratio within the collection of caches is reduced.
- *false hits*: the document requested is not cached at some other proxy but its summary indicates that it is. The proxy will send a query message to the other proxy, only to be notified that the document is not cached there. In this case, a query message is wasted.

In general we are striving for *inclusive* summaries, that is we are trying to avoid false misses as much as possible, even at the cost of extra false hits. It is important to note that false hits and false misses affect the total cache hit ratio or the inter-proxy traffic, but do not affect the correctness of the caching scheme (e.g. a false hit does not result in the wrong document being served).

Two factors limit the scalability of this scheme: the network overhead (the inter-proxy traffic), and the memory required to store the summaries (for performance reasons, the summaries should be stored in memory, not on disk). The network overhead is determined by the frequency of summary updates and by the number of false hits. The memory requirement is determined by the size of individual summaries and the number of cooperating proxies. Since the memory grows linearly with the number of proxies, it is important to keep the individual summaries small. Below, we first address the update frequencies, and then discuss various summary representations.

5.1 Impact of Update Delays

We investigate the following technique of updating summaries: a proxy broadcasts the summary changes to all other proxies whenever the percentage of cached documents that are "new" (that is, not reflected in the summary) reaches a threshold. The delays introduce false misses (documents newly stored are not

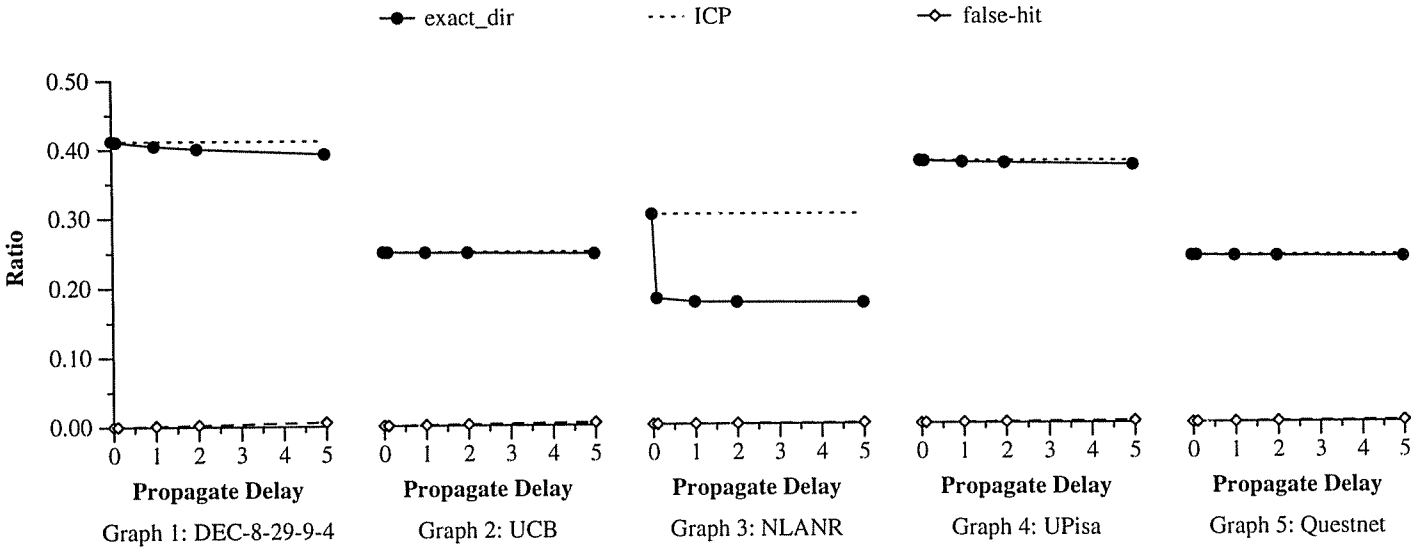

Figure 2: Impact of summary update delays on total cache hit ratios. The cache size is 10% of the "infinite" cache size.

visible in the summary) and (much less likely) false hits (documents deleted from the cache are still in the summary). Intuitively, the number of false misses should be roughly proportional to the number of documents that are not reflected in the summary.

Using the traces, we simulate the total cache hit ratio when the threshold is 0.1%, 1%, 2% and 5% of the cached documents. For the moment we ignore the issue of summary representations and assume that the summary is a copy of the cache directory. The results are shown in Figure 2. The top line in the figure is the hit ratio when no update delay is introduced. The second line shows the hit ratio as the update delay increases. The difference between the two lines is the false miss ratio. The bottom curve shows the false hit ratio, which almost overlaps the x-axis.

The results show that except NLANR, the degradation in total cache hit ratio grows more or less linearly with the update threshold. At the threshold of 1%, the relative reductions in hit ratio are 0.2% (UCB), 0.1% (UPisa), 0.3% (Questnet), and 1.7% (DEC). DEC traces seem to be a bit more sensitive to update delays; we suspect it might be because of the large amount of Pointcast traffic in them. The false hit ratio also seems to increase linearly with the delay threshold, but is always smaller than the threshold.

For NLANR, it appears that some clients are simultaneously sending two requests for the exact same document to proxy "bo" and another proxy in the NLANR collection. If we only simulate the other three proxies in NLANR, the results are similar to those of other traces. With "bo" included, we also simulated the delay being 2 and 10 user requests, and the hit ratio drops from 30.7% to 26.1% and 20.2%, respectively. The hit ratio at the threshold of 0.1%, which roughly corresponds to 200 user requests, is 18.4%. Thus, we believe that the sharp drop in hit ratio is due to the anomaly in the NLANR trace. Unfortunately, we cannot pin down the offending clients because the client IDs are not consistent across NLANR traces [35].

The results demonstrate that in practice, a summary update delay threshold of 1% to 5% results in a tolerable degradation of the cache hit ratios. For the five traces, the threshold values translate into roughly 300 to 1500 user requests between updates, and on average, an update frequency of roughly every 5 to 25 minutes. Thus, the bandwidth consumption of these updates can be very low.

5.2 Summary Representations

The second issue affecting scalability is the size of the summary. Summaries need to be stored in the main memory not only because memory lookups are much faster, but also because disk arms are typically the bottlenecks in proxy caches [33]. Although DRAM prices continue to drop, we still need a careful design, since the memory requirement grows linearly with the number of proxies. Summaries also take DRAM away from the in-memory cache of hot documents, affecting the proxy performance. Thus, it is important to keep the summaries small. Fortunately, summaries only have to be inclusive (that is, depicting a superset of the documents stored in the cache) to avoid affecting the cache hit ratio.

We first investigated two naive summary representations: exact-directory and server-name. In the

exact-directory approach, the summary is essentially the cache directory, with each URL represented by its 16-byte MD5 signature [19]. In the server-name approach, the summary is the list of the server name component of the URLs in cache. Since on average, the ratio of different URLs to different server names is about 10 to 1 (observed from our traces), the server-name approach can cut down the memory by a factor of 10.

We simulated these approaches using the traces and found that neither of them is satisfactory. The results are in Figure 6, along with those on another summary representation (Figure 6 is discussed in detail in Section 5.2). The exact-directory approach consumes too much memory. In practice, proxies typically have 8GB to 20GB of cache space. If we assume 16 proxies of 8GB each and an average file size of 8KB, the exact-directory summary would consume $(16-1)*16*(8GB/8KB) = 240MB$ of main memory *per proxy*. The server-name approach, though consuming less memory, generates too many false hits that significantly increase the network messages.

The requirements on an ideal summary representation are small size and low false hit ratio. After a few other tries, we found a solution in an old technique called "Bloom filters."

5.3 Bloom Filters – the math

A Bloom filter is a method for representing a set $A = \{a_1, a_2, \ldots, a_n\}$ of n keys to support membership queries. It was invented by Burton Bloom in 1970 [5] and was proposed for use in the web context by Marais and Bharat [34] as a mechanism for identifying which pages have associated comments stored within a *CommonKnowledge* server.

The idea is to allocate a vector v of m bits, initially set to 0, and then choose k independent hash functions, $h_1, h_2, \ldots, h_k$, with range $\{1, \ldots, m\}$. For each key $a \in A$ the bits $h_1(a)$, $h_2(a)$, $\ldots$, $h_k(a)$ of v are set to 1. (A particular bit might be set to 1 multiple times.) Given a query key b we check the bits $h_1(b)$, $h_2(b)$, $\ldots$, $h_k(b)$. If any of them is 0, then certainly b is not in the set A. Otherwise we conjecture that b is in the set although there is a certain probability that we are wrong. This is called a "false positive" or, for historical reasons, a "false drop." The parameters k and m should be chosen such that the probability of a false positive is acceptable. (Note that in our context a false positive is not the only cause of a false hit. The latter can happen also because the summary is out of date.)

Observe that after inserting n keys into a table of size m, the probability that a particular bit is still 0

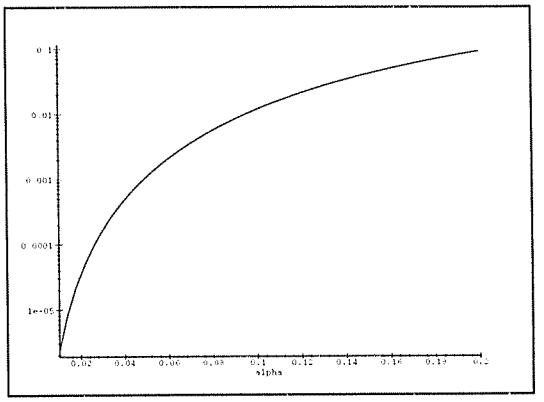

Figure 3: Probability of false positives (log scale). The curve above is for 4 hash functions. The curve below is for the optimum (integral) number of hash functions.

is exactly

$$\left(1 - \frac{1}{m}\right)^{kn}.$$

Hence the probability of a false positive in this situation is

$$\left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{kn/m}\right)^k.$$

The right hand side is minimized for $k = \ln 2 \times m/n$, in which case it becomes

$$\left(\frac{1}{2}\right)^k = (0.6185)^{m/n}.$$

In fact k must be an integer and in practice we might chose a value less than optimal to reduce computational overhead.

The graph in Figure 3 shows the probability of a false positive as a function of the number of bits allocated for each entry, that is, the ratio $\alpha = n/m$. The curve above is for the case of 4 hash functions. The curve below is for the optimum number of hash functions. The scale is logarithmic so the straight line observed corresponds to an exponential decrease. It is clear that Bloom filters require very little storage per key at the slight risk of some false positives. For instance for a bit array 10 times larger than the number of entries, the probability of a false positive is 1.2% for 4 hash functions, and 0.9% for the optimum case of 5 hash functions. The probability of false positives can be easily decreased by allocating more memory.

As discussed so far Bloom filters support insertions but not deletions. In our context each proxy

maintains a local Bloom filter that represents its own directory so deletions must be supported. This is done by maintaining for each location ℓ in the bit array a count $c(\ell)$ of the number of keys that hashed to ℓ under any of the hash functions. All the counts are initially 0. When a new key a is inserted or deleted the counts $c(h_1(a))$, $c(h_2(a))$, ..., $c(h_k(a))$ are incremented or decremented accordingly. If a count becomes 0 the corresponding bit is turned off. Hence the local Bloom filter always reflects correctly the current directory. Since we must allocate memory for the counters it is important to know how large they can become.

The asymptotic expected maximum count after inserting n keys with k hash functions into a bit array of size m is (see [19, p. 72])

$$\Gamma^{-1}(m)\left(1+\frac{\ln(kn/m)}{\ln\Gamma^{-1}(m)}+O\left(\frac{1}{\ln^2\Gamma^{-1}(m)}\right)\right),$$

and the probability that any count is greater or equal i is

$$\Pr(\max(c)\geq i)\leq m\binom{nk}{i}\frac{1}{m^i}\leq m\left(\frac{enk}{im}\right)^i.$$

As already mentioned the optimum value for k (over reals) is $\ln 2m/n$ so assuming that the number of hash functions is less than $\ln 2m/n$ we can further bound

$$\Pr(\max(c)\geq i)\leq m\left(\frac{e\ln 2}{i}\right)^i.$$

Hence taking $i=16$ we obtain that

$$\Pr(\max(c)\geq 16)\leq 1.37\times 10^{-15}\times m.$$

In other words if we allow 4 bits per count, the probability of overflow for practical values of m during the initial insertion in the table is minuscule.

In practice we must take into account that the hash functions are not truly random, and that we keep doing insertions and deletions. Nevertheless, it seems that 4 bits per count would be amply sufficient. Furthermore if the count ever exceeds 15, we can simply let it stay at 15; after many deletions this might lead to a situation where the Bloom filter allows a false negative (the count becomes 0 when it shouldn't be), but the probability of such a chain of events is so low that it is much more likely that the proxy server would be rebooted in the meantime and the entire structure reconstructed.

5.4 Bloom Filters as Summaries

Bloom filters provide a straightforward mechanism to build summaries. A proxy builds a Bloom filter from the list of URLs of cached documents, and sends the bit array plus the specification of the hash functions used to other proxies. When updating the summary, the proxy simply specifies which bits in the bit array are flipped. Each proxy maintains a local copy of the bloom filter, and updates it as documents are added to and replaced from the cache. As explained, to update the local filter, a proxy maintains an array of counters, each counter remembering the number of times the corresponding bit is set to 1. When a document is added into the cache, the counters for the corresponding bits are incremented; when it is deleted from the cache, the counters are decremented. When a counter increases from 0 to 1 or drops from 1 to 0, a record is added to a list which will be sent to other proxies at the next summary update.

The advantage of Bloom filters is that they provide a tradeoff between the memory requirement and the false positive ratio (which induces false hits). Thus, if proxies want to devote less memory to the summaries, they can do so at a slight increase of inter-proxy traffic.

We experimented with three configurations for Bloom filter based summaries: the number of bits being 8, 16, and 32 times the average number of documents in the cache (the ratio is also called a "load" factor). The average number of documents is calculated by dividing the cache size by 8K (the average document size). All three configurations use four hash functions[4]. The hash functions are build from the MD5 signature of the URL.

The performance of these summary representations, the exact-directory approach, and the server-name approach are shown in Figures 4 through 8. In Figure 4 we show the total cache hit ratios and in Figure 5 we show the false hit ratios. *Note that the y-axis in Figure 5 is in log scale.* The Bloom filter based summaries have virtually the same cache hit ratio as the exact-directory approach, and have slightly higher false hit ratio when the bit array is small. Server-name has a much higher false hit ratio. It has a higher cache hit ratio, probably because its many false hits help to avoid false misses.

Figure 6 shows the total number of inter-proxy network messages, including the number of summary updates and the number of query messages (which includes both remote cache hits and false hits). *Note that the y-axis in Figure 6 is in log scale.* For comparison we also list the number of messages incurred by ICP in each trace. All messages are assumed to be uni-cast messages. The figure normalizes the num-

[4]The number of hash functions is not the optimal choice for each configuration, but suffices to demonstrate the performance of Bloom filters. For the final version we plan to provide results for the optimal number of hash functions as well.

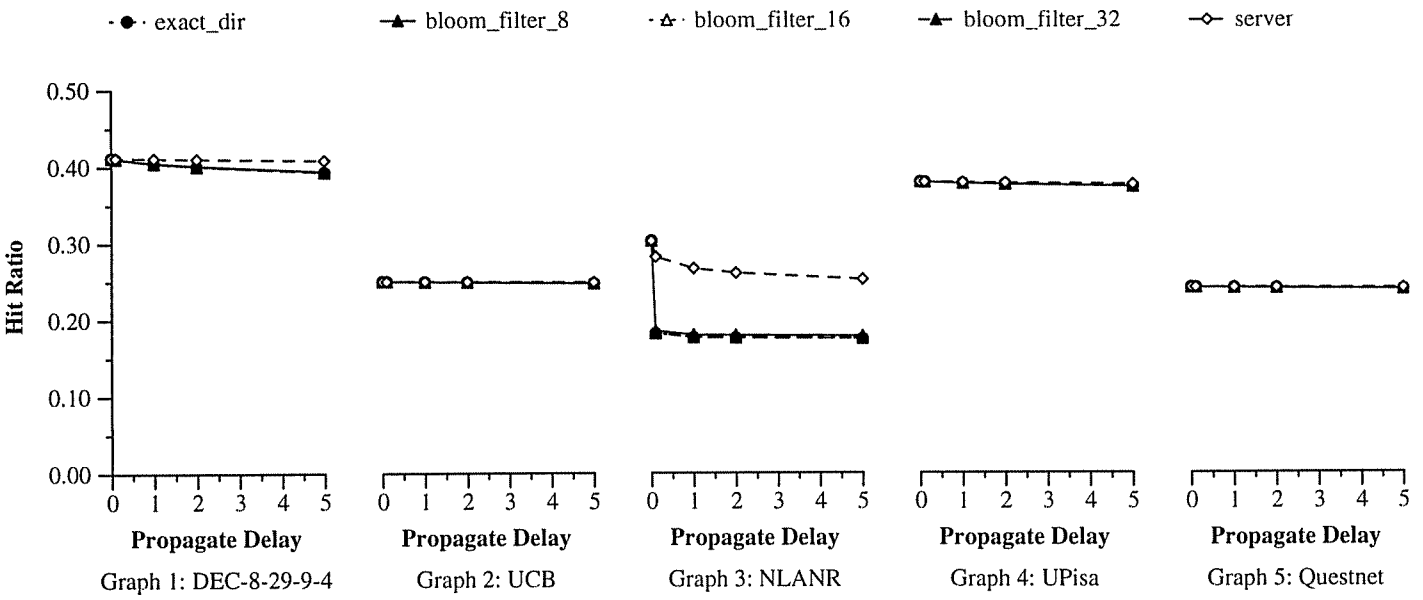

Figure 4: Total hit ratio under different summary representations.

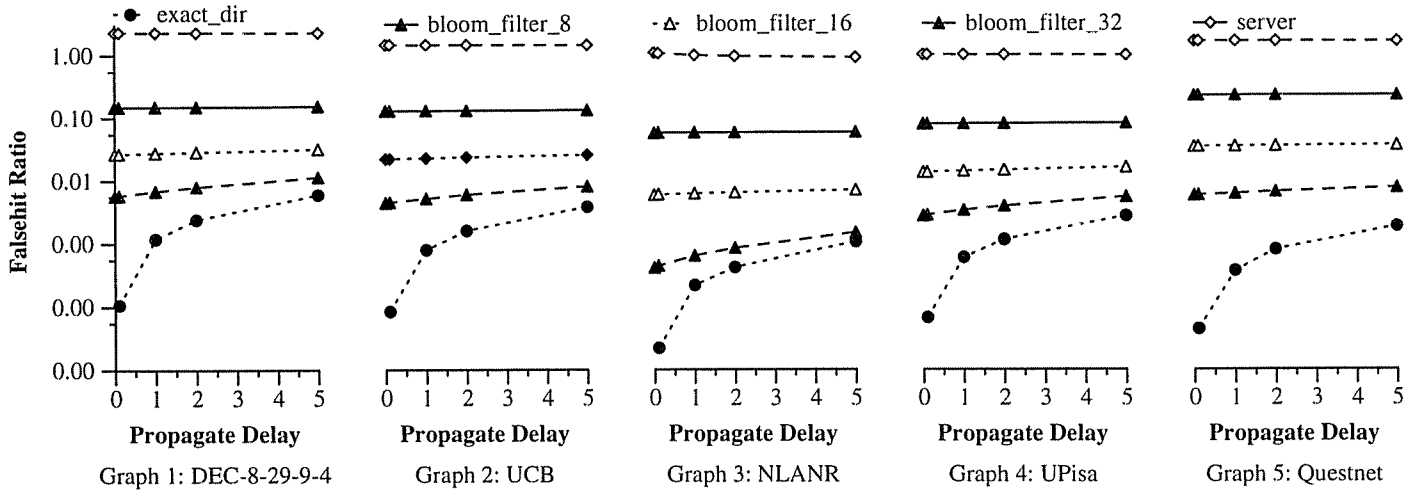

Figure 5: Ratio of false hits under different summary representations. Note that the y-axis is in log scale.

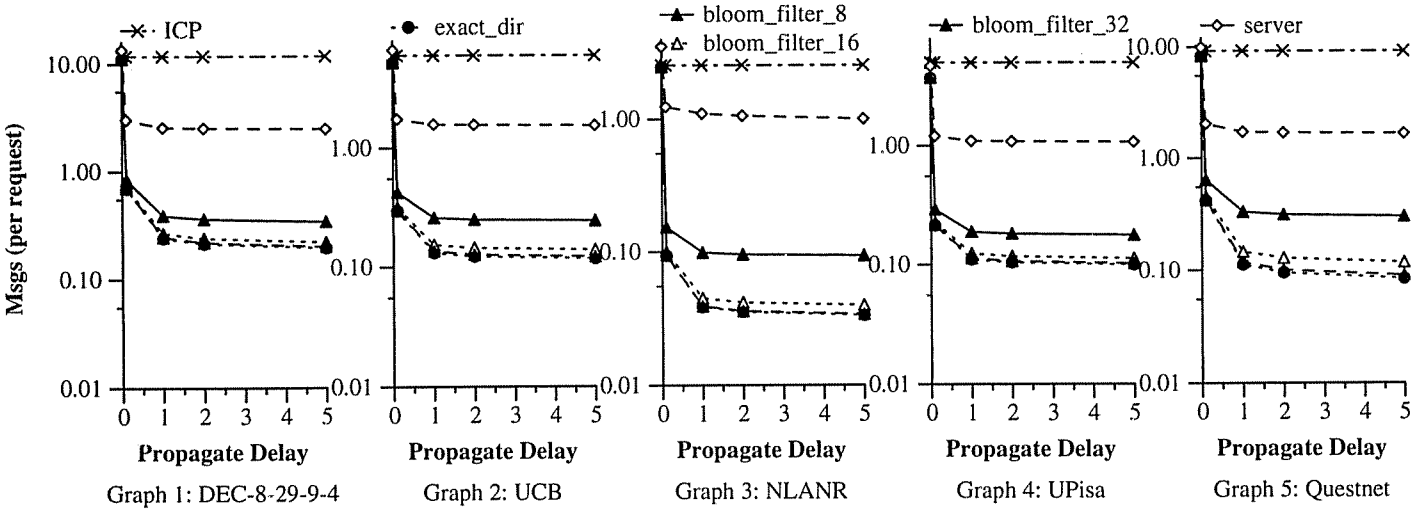

Figure 6: Number of network messages per user request under different summary forms. Note that the y-axis is in log scale.

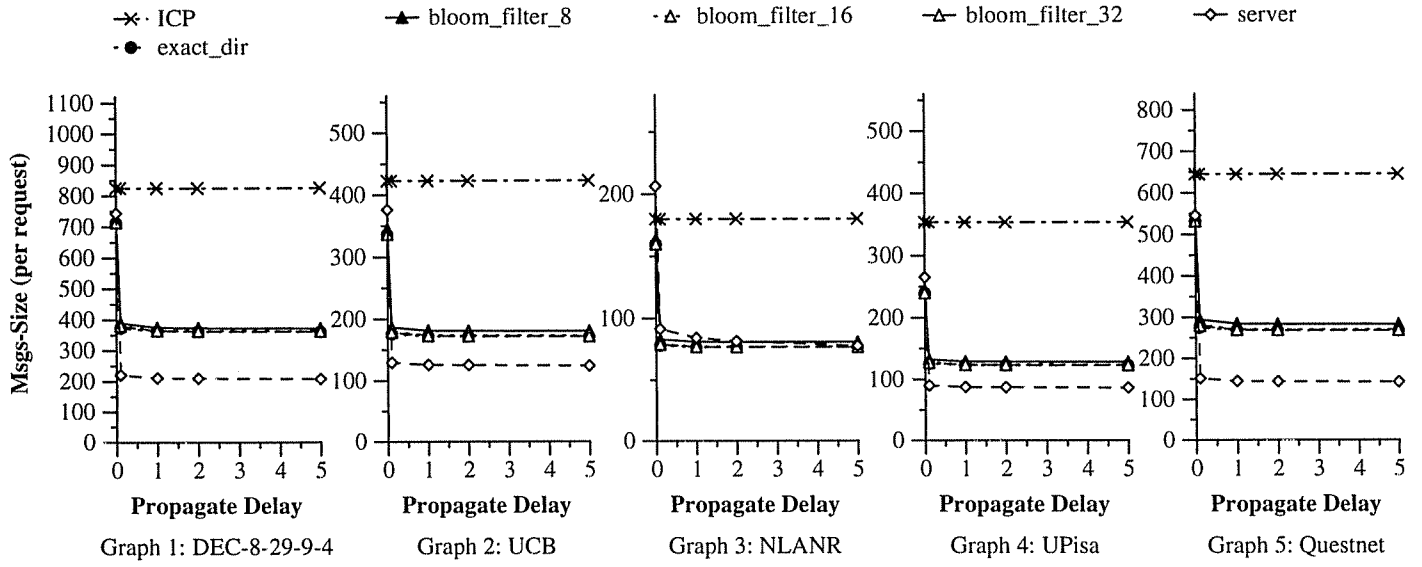

Figure 7: Bytes of network messages per user request under different summary forms.

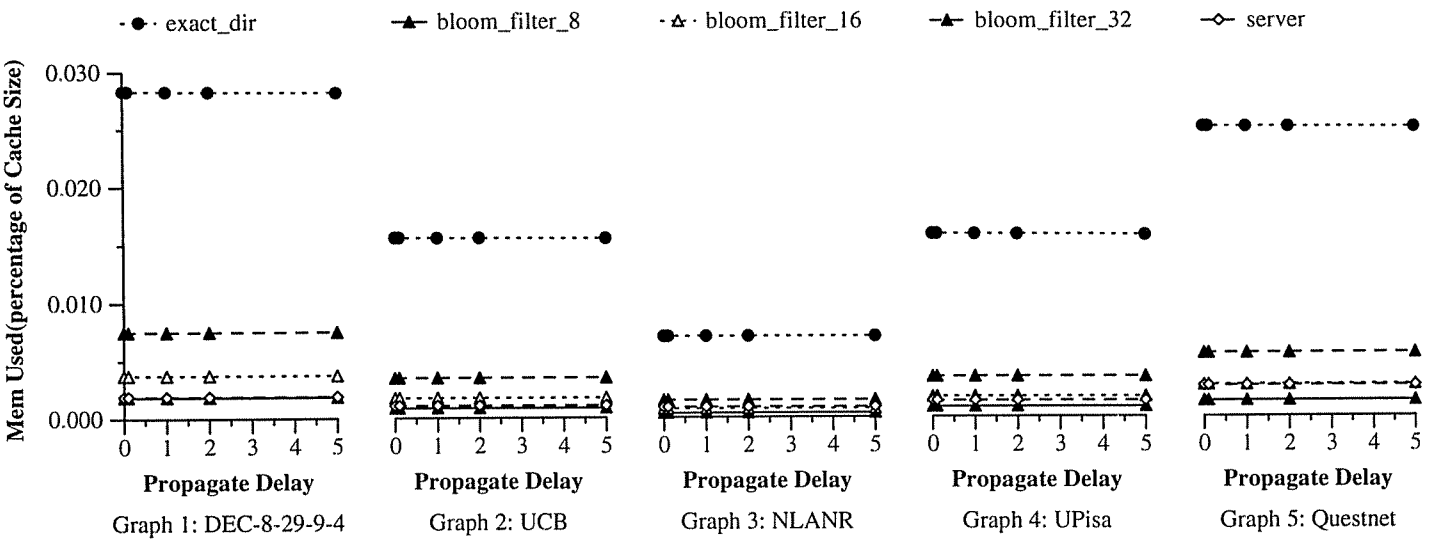

Figure 8: Memory requirement of different summary representations.

ber of messages by the number of HTTP requests in each trace. Clearly, both the exact-directory and the Bloom filter based summaries perform well. There is a clear tradeoff between bit array size and the number of messages, as expected. The server-name approach and ICP generate much more messages. Compared to ICP, the Bloom filter based summaries reduce the number of messages by a factor of 40 to 65.

Figure 7 shows the estimated total size of inter-proxy network messages in bytes. We estimate the size because update messages tend to be larger than query messages. The average size of query messages in both ICP and other approaches is assumed to be 20 bytes of header and 50 bytes of average URL. The size of summary updates in exact-directory and server-name is assumed to be 20 bytes of header and 16 bytes per change. The size of summary updates in Bloom filter based summaries is estimated at 32 bytes of header (see Section 6) plus 4 bytes per bit-flip. The results show that in terms of message bytes, Bloom filter based summaries improves over ICP by 55% to 64%. In other words, summary cache uses occasional burst of large messages to avoid continuous stream of small messages. Looking at the CPU overhead and network interface packets in Tables 2 and 3 (in which SC-ICP stands for the summary cache approach), we can see clearly that it is a good tradeoff.

Finally, Figure 8 shows the memory per proxy of the summary cache approaches, in terms of percentage of cache size. The three Bloom filter configurations consume much less memory than exact-directory, and yet perform similarly to it in all other aspects. The Bloom filter summary at the load factor of 8 has a similar memory requirement to the server-name approach, and much fewer false hits and network messages.

Considering all the results, we see that Bloom filter summaries provide the best performance in terms of low network overhead and low memory requirements. This approach is simple and easy to implement. The only drawback is the MD5 calculation, which must be performed every time a document is added to or deleted from the cache. However, faster hashing methods are available, for instance hash functions can be based on polynomial arithmetic as in Rabin's fingerprinting method (See [40, 6]), or a simple hash function (e.g. [19, p. 48]) can be used to generate, say 32 bits, and further bits can be obtained by taking random linear transformations of these 32 bits viewed as an integer. We are still exploring whether any degradation happens if we use the faster approaches. One potential disadvantage is that these faster functions are efficiently invertible (that is, one can easily build an URL that hashes to a particular location), a fact that might be used by malicious users to nefarious purposes.

5.5 Recommended Configurations

Combining the above results, we recommend the following configuration for the summary cache approach. A proxy should broadcast the summary changes every time 1% to 5% of cached documents are new, depending on the available bandwidth between proxies. The summary should be in the form of a Bloom filter. The default load factor (size of array in bits divided by the average number of pages) is 16. The proxies can lower or raise it depending on their memory and network traffic concerns; the data in section 5.3 and the figures in the section above can be used as

references. The hash functions are bits taken from the MD5 signature of the URL, and if more bits are needed, we can compute the MD5 signature of the URL concatenated with itself any number of times.

5.6 Scalability

Although our simulations are done for 4 to 16 proxies, we can easily extrapolate the results: For example, assume that 100 proxies each with 8GB of cache would like to cooperate. Each proxy stores on average about 1M pages. The Bloom filter memory needed to represent 1M pages is 2MB at load factor 16. Each proxy needs about 200 MB to represent all the summaries plus another 8 MB to represent its own summary counts. The messages in this system consist of update messages, false hits, and remote hits. The threshold of 1% corresponds to 10K requests between updates, each update consisting of 99 messages, and thus the number of update messages per request is less than 0.01. The false hit ratios are around 4.7% for the load factor of 16 with 10 hash functions. (The probability of a false positive is less than 0.00047 for each summary, but there are 100 of them.) Thus, not counting the messages introduced by remote cache hits, the overhead introduced by the protocol is under 0.06 messages per request for 100 proxies. Of these messages only the update message is large, of the order of several hundreds KB. Fortunately, the update messages are broadcasts and can be transferred via a non-reliable multicast scheme (see Section 6). Our simulations predict that, while keeping the overhead low, this scheme reduces the hit ratio by less than 2% compared to the theoretical hit ratio of ICP.

We are working on larger simulations to verify these "back of the envelope" calculations. However, based on the existing results, we are confident that Summary Cache scales well.

6 Summary-Cache Enhanced ICP

Based on the simulation results, we propose the following Summary-Cache Enhanced Internet Cache Protocol as an optimization of ICP. The protocol has been implemented in a prototype build on top of Squid 1.1.14 and the prototype is available for public domain [38].

We added a new opcode in ICP version 2 [45], ICP_OP_DIRUPDATE (= 20), which stands for directory update messages. In an update message, an additional header follows the regular ICP header and consists of: 16 bits of `Function_Num`, 16 bits of `Function_Bits`, 32 bits of `BitArray_Size_InBits`, and 32 bits of `Number_of_Updates`. The header completely specifies the hashing functions used to probe the filter. There are `Function_Num` of hashing functions. The functions are calculated by first taking bits 0 to M-1, M to 2M-1, 2M to 3M-1, etc. out of the MD5 signature of the URL, where M is `Function_Bits`, and then modular the bits by `BitArray_Size_InBits`. If 128 bits are not enough, more bits are generated by computing the MD5 signature of the URL concatenated with itself.

The header is followed by a list of 32-bit integers. The most significant bit in an integer specifies whether the bit should be set to 0 or 1, and the rest of the bits specify the index of the bit that needs to be changed. The design is due to the concern that if the message only specifies which bits should be flipped, then loss of previous update messages would have cascading effects. It makes the scheme more robust toward update message loss, and enables the messages to be sent via a unreliable multicast protocol. Furthermore, every update message carries the header, which specifies the hash functions so that the receiver can verify the information. The design limits the hash table size to be less than 2 billion, which for the time being is large enough.

We modified the Squid 1.1.4 software to implement the above protocol. An additional bit array structure is added to the data structure for each neighbor. The structure is initialized when the first summary update message is received from the neighbor. The proxy also allocates an array of byte counters for maintaining the local copy of the bloom filter, and an integer array to remember the filter changes. The update messages are sent via the outgoing ICP connection to all neighbors. Since ICP uses UDP, in order for the message to fit in one ethernet IP packet, we deviate from the above design by sending update messages whenever there are enough filter changes to fill an IP packet. This is a workaround for the fact that Squid currently uses UDP for outgoing ICP connection, and we are working on better solutions. The code as of now does not yet implement retransmissions or recovery from other types of errors. In addition, we have not done any performance tuning on the prototype. We are working on these problems.

7 Experiments

We run two experiments with the summary-cache enhanced ICP prototype. The first experiment repeats the test in Section 4 and the results are included in Table 2 in Section 4, under the title "SC-ICP." Clearly, the improved protocol reduces the UDP traffic by a factor of 50, and results in network traffic, CPU times and client latencies similar to the no-ICP

Exp	Hit Ratio	Client Latency	User CPU	System CPU	UDP Traffic	TCP Traffic	Total Packets
no ICP	10.65	4.53(0.4%)	77.19(0.1%)	115.43(0.4%)	626(1%)	221K(0%)	237K(0%)
ICP	23.5	4.88(0.7%)	114.58(0.1%)	140.19(0.1%)	64582(0%)	236K(0%)	317K(0%)
Overhead		*8%*	*48%*	*21%*	*102*	*7%*	*34%*
SC-ICP	21.15	4.50(0.1%)	89.51(0.4%)	124.25(0.5%)	4941(2%)	233K(0%)	253K(0%)
Overhead		*0.7%*	*16%*	*7%*	*7*	*5%*	*7%*

Table 3: Overhead of the ICP processing for UPisa trace.

case.

Our second experiment takes the first 24,000 requests from the UPisa trace, and let the client processes issue requests from the trace. The experiment does not try to replay the trace faithfully, but rather as a test of the correctness of the implementation. We have a total of 80 client processes running on 4 workstations, issuing requests round-robin from the trace file. Each request's URL carries the size of the request in the trace file, and the server replies with the specified number of bytes. The rest of the configuration is similar to the experiments in Section 4. Different from the synthetic benchmark, the trace results in fairly high remote hit ratio. The results are listed in Table 3.

The results also show that the enhanced ICP protocol reduces the network traffic and CPU overhead significantly, with only slight impact on total hit ratios. The hit ratio degradation is higher than simulation indicates. We suspect that it is because the requests are issued round-robin, thus requests from the same user in the trace are issued by different clients. We are working on a different way to replay the trace to verify the conjecture.

In addition, the enhanced ICP protocol lowers the client latency slightly compared to the No-ICP case, and yet increases the CPU time by about 10%. The reduction in client latency is due to the remote cache hits. Comparing with the results of no remote hits in the first experiment, the numbers seem to indicate that the CPU time increase is due to serving remote hits. We are inspecting the code to find out exactly where the CPU time increase is from. We are also in the process of gathering more workstations and experiments with larger number of proxies to verify the scalability of the protocol. In addition, we are experimenting with more traces and more faithfully replay of the traces.

The existing results do indicate that the summary-cache enhanced ICP solves the overhead problem of ICP, requires minimal changes, and should be deployed as soon as possible. Toward this end, we are actively pushing the adoption of the new protocol in the Squid user community.

8 Related Work

Web caching is an active research area. There are many studies on Web client access characteristics [9, 2, 13, 30, 20], web caching algorithms [46, 32, 7] as well as Web cache consistency [25, 28, 31, 12]. Our study does not address caching algorithms or cache consistency maintanence, but overlaps some of client traffic studies in our investigation of the benefits of Web cache sharing.

There have also been a lot of studies on Web cache hierarchies and cache sharing. Hierarchical Web caching is first proposed in the Harvest project [23, 11], which also introduces the ICP protocol. Currently, the Squid proxy server implements version 2 of the ICP protocol [45], upon which our Summary-Cached enhanced ICP is based. Adaptive Web caching [47] proposes a multicast-based adaptive caching infrastructure for document dissemination in the Web. In particular, the scheme seeks to position the documents at the right caches along the routes to the servers. Our study does not address the positioning issues. Rather, we note that our study is complimentary in the sense that the summary cache approach can be used as a mechanism for communicating caches' contents.

Though we did not simulate the scenario, Summary-Cache enhanced ICP can be used between parent and child proxies. Hierarchical Web caching includes not only cooperation among neighboring (sibling) proxies, but also parent and child proxies. The difference between a sibling proxy and a parent proxy is that a proxy cannot ask a sibling proxy to fetch a document from the server for it, but can ask a parent proxy. Though our simulations only involve the cooperation among sibling proxies, the summary-cache approach can be used to propogate information about the parent cache's content to the child proxies, and eliminate the ICP queries from the child proxies to the parent. Our inspection of the Questnet traces shows that the child-to-parent ICP queries can be a significant portion (over 2/3) of the messages that the parent has to process.

Recently, there have been a number of new cache sharing approaches proposed in the literature. The

directory server approach [18] use a central server to keep track of the cache directories of all proxies, and all proxies query the server for cache hits in other proxies. The drawback of the approach is that the central server can easily become a bottleneck with the query and update messages from the proxies. Another approach is the Cache Array Routing Protocol [43], which divides URL-space among an array of loosely coupled proxy servers, and lets each proxy cache only the documents whose URLs are hashed to it. An advantage of the approach is that it eliminates duplicate copies of documents. However, it is not clear how well the approach performs for wide-area cache sharing, where proxies maybe distributed over a regional network. The Relais project [24] also proposes using local directories to facilitate finding documents in other caches, and updating the directories asynchronously. The idea is similar to summary cache. However, the project does not seem to address the problem of the linearly growing memory requirements of the local directories. From the publications on Relais that we can find and read [3], it is also not clear to us whether the project addresses the issue of directory update frequencies. Finally, proxies built out of tightly-coupled clustered workstations also use various hashing and partitioning approaches to utilize the memory and disks in the cluster [17], but the approaches are not appropriate in wide-area networks.

In the operating system context, there have been a lot of studies on cooperative file caching [10, 1] and the global memory system (GMS) [14]. The underlying assumption in these systems is that the high-speed local area networks are faster than disks, and workstations should use each other's idle memory to cache file pages or virtual memory pages to avoid traffic to disks. In this aspect, the problem is quite different from Web cache sharing. On the other hand, in both context there is the issue of how tightly coordinated the caches should be. Most cooperative file caching and GMS systems try to emulate the global LRU replacement algorithm, sometimes also using hints in doing so [42]. It is interesting to note that we arrive at quite different conclusions on whether global replacement algorithm is necessary [14]. The reason is that in the OS context, the global replacement algorithm is used for stealing memory from idle workstations (i.e. load-balancing the caches), while in Web cache sharing, every proxy is busy all the time. Thus, while simple cache sharing performs poorly in the OS context, it suffices for Web proxy cache sharing as long as each proxy's resource configuration is appropriate for its load. Finally, note that the technique of Bloom filter based summary cache is not restricted to the Web proxy caching context, but can be used whereever the knowledge of other caches' contents is beneficial, for example, in caching and load-balancing in clustered servers.

9 Conclusions and Future Work

We propose Summary-Cache enhanced ICP, a scalable wide-area Web cache sharing protocol. Using trace-driven simulations and measurements, we demonstrate the benefits of Web proxy cache sharing, illustrate the overhead of the current cache sharing protocol ICP, and propose the summary cache approach to reduce the protocol overhead. We study two key questions in the summary cache approach: the delay in updating summaries, and the representation of summaries. Our solution, Bloom filter based summaries with update delay thresholds, consumes low memory and network overhead, and yet achieves hit ratio similar to the ICP protocol. Trace-driven simulations show that the new protocol reduces the number of inter-proxy protocol messages by *a factor of 40 to 65*, reduces the bandwidth consumption *by over 50%*, and yet keeps similar cache hit ratios as ICP. Simulation and analysis further demonstrate the scalability of the protocol.

We have implemented a prototype of the new protocol in Squid 1.1.14. Synthetic and trace-replay experiments with the prototype show that in addition to the network traffic reduction, the new protocol reduces the protocol CPU overhead by *75% and 95%* and improves the client latency.

There are many limitations in our current study. We need to perform simulations and experiments with larger numbers of proxies, larger caches, and longer traces. We are working on acquiring the necessary workstations for doing this. The prototype needs performance tuning. More faithful and longer trace-replay experiments are needed to understand the performance of the protocol in practice.

We are also planning to investigate how the protocol performs for parent-child proxy cooperations. We plan to use simulations and experiments to understand the appropriate replacement coordination among the parent and child proxies, and to incorporate it in the enhanced ICP protocol.

References

[1] Thomas E. Anderson, Michael D. Dahlin, Jeanna M. Neefe, David A. Patterson, Drew S. Roselli, and Randolph Y. Wang. Serverless network file systems. In *Proceedings of 15th ACM Symposium on Operating Systems Principles*, December 1995.

[2] M. Arlitt and C. Williamson. Web server workload characterization. In *Proceedings of the 1996*

ACM SIGMETRICS International Conference on Measurement and Modelling of Computer Systems, May 1996.

[3] Aline Baggio and Guillaume Pierre. Oleron: Supporting information sharing in large-scale mobile environments. In *Proceedings of the ERSADS Workshop*, March 1997. Available from http://www-sor.inria.fr/projects/relais/.

[4] Kirby Beck. Tennessee cache box project. In *the 2nd Web Caching Workshop, Boulder, Colorado*, June 1997. http://ircache.nlanr.net/Cache/Workshop97/.

[5] Burton Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of ACM*, pages 13(7):422–426, July 1970.

[6] Andrei Z. Broder. Some applications of Rabin's fingerprinting method. In Renato Capocelli, Alfredo De Santis, and Ugo Vaccaro, editors, *Sequences II: Methods in Communications, Security, and Computer Science*, pages 143–152. Springer-Verlag, 1993.

[7] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, December 1997.

[8] M. Crovella and A. Bestavros. Self-similiarity in world wide web traffic: Evidence and possible causes. In *Proc of the 1996 Sigmetrics Conference on Measurment and Modeling of Computer systems Philadelphia*, May 1996.

[9] Carlos R. Cunba, Azer Bestavros, and Mark E. Crovella. Characteristics of WWW client-based traces. Technical report, BU-CS-96-010, Boston University, October 1995.

[10] Michael D. Dahlin, Randolph Y. Wang, Thomas E. Anderson, and David A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation*, pages 267–280, November 1994.

[11] P. B. Danzig, R. S. Hall, and M. F. Schwartz. A case for caching file objects inside internetworks. In *Proceedings of SIGCOMM '93*, pages 239–248, 1993.

[12] Fred Douglis, Anja Feldmann, Balachander Krishnamurthy, and Jeffrey Mogul. Rate of change and other metrics: A live study of the world wide web. In *Proceedings of USENIX Symposium on Internet Technology and Systems*, December 1997.

[13] Bradley M. Duska, David Marwood, and Michael J. Feeley. The measured access characteristics of world-wide-web client proxy caches. In *Proceedings of USENIX Symposium on Internet Technology and Systems*, December 1997.

[14] Michael J. Feeley, William E. Morgan, Frederic H. Pighin, Anna R. Karlin, Henry M. Levy, and Chandramohan A. Thekkath. Implementing global memory management in a workstation cluster. In *To appear in Proceedings of 15th ACM Symposium on Operating Systems Principles*, December 1995.

[15] National Lab for Applied Network Research. Icp working group. *http://ircache.nlanr.net/Cache/ICP/*, 1998.

[16] National Lab for Applied Network Research. Squid internet object cache. *http://squid.nlanr.net/Squid/*, 1998.

[17] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, and Paul Gauthier. Cluster-based scalable network service. In *Proceedings of SOSP'16*, October 1997.

[18] S. Gadde, M. Rabinovich, and J. Chase. Reduce, reuse, recycle: An approach to building large internet caches. In *Proceedings of the Sixth Workshop on Hot Topics in Operating Systems (HotOS VI)*, May 1997. Available from http://www.research.att.com/ misha/.

[19] G. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures*. Addison-Wesley, 1991.

[20] Steve Gribble and Eric Brewer. System design issues for internet middleware service: Deduction from a large client trace. In *Proceedings of USENIX Symposium on Internet Technology and Systems*, December 1997.

[21] Steven Gribble and Eric Brewer. Ucb home IP HTTP traces. *Available at http://www.cs.berkeley.edu/ gribble/traces/index.html*, June 1997.

[22] Christian Grimm. The dfn cache service in B-WiN. In *the 2nd Web Caching Workshop, Boulder, Colorado*, June 1997. http://www-cache.dfn.de/CacheEN/.

[23] The Harvest Group. Harvest information discovery and access system. *http://excalibur.usc.edu/*, 1994.

[24] The Relais Group. Relais: cooperative caches for the world-wide web. *http://www-sor.inria.fr/projects/relais/*, 1998.

[25] James Gwertzman and Margo Seltzer. World-wide web cache consistency. In *Proceedings of the 1996 USENIX Technical Conference, San Diego, CA*, January 1996.

[26] Van Jacobson. How to kill the internet. In *SIGCOMM'95 Middleware Workshop*, August 1995. URL ftp://ftp.ee.lhl.gov/talks/vj-webflame.ps.Z.

[27] Jaeyeon. Nation-wide caching project in korea. In *the 2nd Web Caching Workshop, Boulder, Colorado*, June 1997. http://ircache.nlanr.net/Cache/Workshop97/.

[28] Balachander Krishnamurthy and Craig E. Ellis. Study of piggyback cache validation for proxy caches in the world wide web. In *Proceedings of USENIX Symposium on Internet Technology and Systems*, December 1997.

[29] T. M. Kroeger, J. Mogul, and C. Maltzahn. Digital's web proxy traces. *Available at URL: ftp://ftp.digital.com/pub/DEC/traces/proxy/webtraces.html*, August 1996.

[30] Thomas M. Kroeger, Darrell D. E. Long, and Jeffrey C. Mogul. Exploring the bounds of web latency reduction from caching and prefetching. In *Proceedings of USENIX Symposium on Internet Technology and Systems*, December 1997.

[31] Chengjie Liu and Pei Cao. Maintaining strong cache consistency for the world-wide web. In *The 17th International Conference on Distributed Computing Systems*, May 1997.

[32] P. Lorenzetti, L. Rizzo, and L. Vicisano. Replacement policies for a proxy cache. Technical report, Universita di Pisa, Italy, October 1996. URL http://www.iet.unipi.it/ luigi/caching.ps.gz.

[33] Carlos Maltzahn, Kathy Richardson, and Dirk Grunwald. Performance issues of enterprise level web proxies. In *Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modelling of Computer Systems*, pages 13–23, June 1997.

[34] J. Marais and K. Bharat. Supporting cooperative and personal surfing with a desktop assistant. In *Proceedings of ACM UIST'97*, October 1997. Available on-line at ftp://ftp.digital.com/pub/DEC/SRC /publications/marais/uist97paper.pdf.

[35] National Lab of Applied Network Research. Sanitized access log. *Available at ftp://ircache.nlanr.net/Traces/*, July 1997.

[36] Reference omitted for Anonymous Review.

[37] Reference omitted for Anonymous Review.

[38] Reference omitted for Anonymous Review.

[39] Pietsch. Caching in the washington state k-20 network. In *the 2nd Web Caching Workshop, Boulder, Colorado*, June 1997. http://ircache.nlanr.net/Cache/Workshop97/.

[40] Michael O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[41] Luigi Rizzo. Web proxy traces. *Available at URL: http://info.iet.unipi.it/ luigi/proxy-traces/*, May 1997.

[42] P. Sarkar and J. Hartman. Efficient cooperative caching using hints. In *Proceedings of the USENIX Conference on Operating System Design and Implementations*, October 1996.

[43] Vinod Valloppillil and Keith W. Ross. Cache array routing protocol v1.0. *http://ircache.nlanr.net/Cache/ICP/draft-vinod-carp-v1-02.txt*, 1997.

[44] Julianne Weekers. Personal communication. January 1998.

[45] Duane Wessels and Kim Claffy. Internet cache protocol (ICP), version 2. *http://ds.internic.net/rfc/rfc2186.txt*, 1998.

[46] S. Williams, M. Abrams, C.R. Stanbridge, G. Abdulla, and E.A. Fox. Removal policies in network caches for world-wide web documents. In *Proceedings of the ACM Sigcomm96*, August 1996. URL http://ei.cs.vt.edu/ succeed/96sigcomm/.

[47] Lixia Zhang, Sally Floyd, and Van Jacobson. Adaptive web caching. In *the 2nd Web Caching Workshop, Boulder, Colorado*, June 1997. http://ircache.nlanr.net/Cache/Workshop97/Papers/Floyd/floyd.