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

Li Fan, Pei Cao, and Jussara Almeida
Department of Computer Science
University of Wisconsin-Madison
{lfan,cao,jussara}@cs.wisc.edu

Andrei Z. Broder
Systems Research Center
Digital Equipment Corporation
broder@pa.dec.com

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 propose a new protocol called "Summary Cache"; each proxy keeps a summary of the URLs of cached documents of each participating proxy and checks these summaries for potential hits before sending any queries. Two factors contribute to the low overhead: the summaries are updated only periodically, and the summary representations are economical – as low as 8 bits per entry. Using trace-driven simulations and a prototype implementation, we show that compared to the existing Internet Cache Protocol (ICP), Summary Cache reduces the number of inter-cache messages by a factor of 25 to 60, reduces the bandwidth consumption by over 50%, and eliminates between 30% to 95% of the CPU overhead, while at the same time maintaining almost the same hit ratio as ICP. Hence Summary Cache enables cache sharing among a large number of proxies.

1 Introduction

Caching has been recognized as one of the most important techniques to reduce Internet bandwidth consumption caused by the tremendous growth of the World-Wide Web [17]. In particular, caching within Web proxies has been shown to be effective [7, 21]. To gain the full benefits of caching, proxy caches on the same side of a common bottleneck link should cooperate and serve each other's misses, a process we call "Web cache sharing."

Web cache sharing was first proposed in the context of the Harvest project [14, 6]. The Harvest group designed the Internet Cache Protocol (ICP) [31] 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 [13, 18, 2, 7].

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 processing overhead increase quadratically.

Several alternatives have been proposed to address this problem, for example, a cache array routing protocol that partitions the URL space among proxies [30]. 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.

In this paper, we address the issue of scalable protocols for wide-area Web cache sharing. We first 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, increases the number of network packets received by each proxy by over 13%, and increases the CPU overhead by over 15%. In the absence of inter-proxy cache hits (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 (that is, the list of URLs of cached documents) 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 total cache 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" [3]. This is a computationally efficient hash-based probabilistic scheme that can represent a set of keys (in our case, the collections of URLs of cached documents) with minimal memory requirements, while answering membership queries with zero 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 around 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 later.)

Based on these results, we design the Summary-Cache Enhanced ICP protocol and implement a prototype within the Squid proxy. Using trace-driven simulations and experiments with benchmarks and trace-replays, we show that the new protocol reduces the number of inter-proxy messages by a factor of 25 to 60, reduces the network bandwidth consumption (in terms of bytes transferred) by over 50%, and eliminates between 30% to 95% of the CPU overhead. Compared with no cache sharing, our experiments show that the protocol incurs little network traffic and increases CPU time only by 5% to 12% depending on the remote cache hit ratio. Yet, the protocol achieves a total cache hit ratio similar to that of the ICP protocol most of the time.

The results indicate that the Summary Cache Enhanced ICP protocol can scale to a large number of proxies. Thus, the protocol has the potential to significantly increase the deployment of Web cache sharing and reduce Web traffic on the Internet. Toward this end, we make our implementation publicly available [8]. In addition, a variant of our approach called Cache Digest is under beta test at the National Cache Hierarchy [29].

2 Traces and Simulations

For this study we have collected five sets of traces of HTTP requests (for more details, see [9]):

- Digital Equipment Corporation Web Proxy server traces (DEC) [20];
- traces of HTTP requests from the University of California at Berkeley Dial-IP service (UCB) [12];
- traces of HTTP requests made by users in the Computer Science Department, University of Pisa, Italy (UPisa);
- logs of HTTP GET requests seen by the parent proxies at Questnet, a regional network in Australia (Questnet);
- one-day log of HTTP requests to the four major parent proxies, "bo", "pb", "sd", and "uc" in the National Web Cache hierarchy by National Lab of Applied Network Research (NLANR) [28].

Table 1 lists various information about the traces, including duration of each trace, the number of requests and the number of clients. The "infinite" cache size is the total size in bytes of unique documents in a trace.

To simulate cache sharing, we partition the clients in DEC, UCB and UPisa into groups, assuming 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. Questnet traces contain HTTP GET requests coming from 12 child proxies in the regional network. 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 sharing among the child proxies. NLANR traces contain actual HTTP requests going to the four major proxies, and we simulate the cache sharing among them.

The simulation results reported here assume a cache size that is 10% of the "infinite" cache size. Results under other cache sizes are similar. The simulations all use LRU as the cache replacement algorithm, with the restriction that documents larger than 250KB are not cached. The policy is similar to what is used in actual proxies. We do not simulate expiring documents based on age or time-to-live. Rather, most traces come with the last-modified time or the size of a document for every request, and if a request hits on a document whose last-modified time or size is changed, we count it as a cache miss. In other words, we assume that

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	3,543,968	1,907,762	2,833,624	2,885,285	1,766,409
Infinite Cache Size	2.88e+10	1.80e+10	2.07e+10	2.33e+10	1.37e+10
Max. Hit Ratio	49%	30%	49%	30%	36%
Max. Byte Hit Ratio	36%	14%	27%	15%	27%
Client Population	10089	5780	2203	12	4
Client Groups	16	8	8	12	4
Hit Ratio w. Cache Sharing	43%	27%	45%	26%	32%
Hit Ratio w/o Cache Sharing	21%	13%	27%	16%	14%

Table 1: Statistics about the traces. The maximum cache hit ratio and byte hit ratio are achieved with the infinite cache. The other hit ratios are calculated assuming a cache size that is 10% of the infinite cache.

cache consistency mechanism is perfect. In practice, there are a variety of protocols [6, 22, 16] for Web cache consistency.

Benefits of Cache Sharing Using the traces, we have studied the benefits of cache sharing. Results show that sharing cache contents among proxies significantly reduces traffic to the Internet. Table 1 lists the cache hit ratios under cache sharing and no cache sharing, assuming a cache size that is 10% of the infinite cache size. Results also show that in most cases, ICP-style simple cache sharing suffices and more tightly coordinated schemes such as global replacement are not necessary. More details can be found in [9].

3 Overhead of ICP

Though the Internet Cache Protocol (ICP) [31] has been successful at encouraging Web cache sharing around the world, it is not a scalable protocol. It relies on query messages to find remote cache hits. Every time one proxy has a cache miss, everyone else receives and processes a query message. 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 the Wisconsin Proxy Benchmark 1.0 [1]. The benchmark is designed by us and has been submitted to SPEC as a candidate for the industry standard benchmark. It 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 Internet latencies.

The experiments are performed on 10 Sun Sparc-20 workstations connected with 100Mb/s Ethernet. Four workstations act as four proxy systems running Squid 1.1.14, and each has 75MB of cache space. 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 document sizes follow the Pareto distribution with $\alpha = 1.1$ and $k = 3.0$ [5]. Two workstations act as servers, each with 15 servers listening on different ports. Each server forks a new 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. The benchmark allows the cache hit ratio to be adjusted. In each experiment, a client process issues 200 requests, for a total of 24,000 requests.

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 overhead, the requests issued by the clients do not overlap; there is no remote cache hit among 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; otherwise the heavy-tailed document size distribution would lead to high variance. The relative differences between no-ICP and ICP are the same across different settings of seeds. We present results from one set of experiments here.

We measure the hit ratio in the caches, the average latency seen by the clients, the user and system CPU times consumed by the Squid proxy, 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

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%*	*9000%*	*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%*	*7300%*	*-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 5.1 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 over no-ICP for each measurement. Note that in these synthetic experiments there is no inter-proxy cache hit.

messages, the total network traffic seen by the proxies is increased by 8% to 13%. Protocol processing increases the user CPU time by 20% to 24%, and UDP processing increases the system CPU time by 7% to 10%. To the clients, the average latency of an HTTP request is increased by 8% to 12%. 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 cache administrators: there are clear benefits of cache sharing (as shown in Table 1), but the overhead of ICP is high. Furthermore, 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 protocol: *Summary Cache*.

4 Summary Cache

In the Summary Cache scheme, each proxy stores a summary of URLs of documents cached at every other proxy. When a user request misses in the local cache, the 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. Otherwise, the proxy sends the request directly to the Web server.

The key to the scalability of the scheme is that summaries do not have to be up to date or accurate. A summary does not have to be updated every time the cache directory is changed; rather, the update can occur upon regular time intervals or when a certain percentage of the cached documents are not reflected in the summary. A summary only needs to be inclusive (that is, depicting a superset of the documents stored in the cache) to avoid affecting the total cache hit ratio. That is, two kinds of errors are tolerated:

- *false misses*: the document requested is cached at some other proxy but its summary does not reflect the fact. In this case, a remote cache hit is lost, 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.

The errors affect the total cache hit ratio or the inter-proxy traffic, but do not affect the correctness of the caching scheme. For example, a false hit does not result in the wrong document being served. In general, we strive for low false misses, because false misses increase traffic to the Internet and the goal of cache sharing is to reduce traffic to the Internet.

A third kind of error, *remote stale hits*, occurs in both summary cache and ICP. A remote stale hit is when a document is cached at another proxy, but the cached copy is stale. Remote stale hits are not necessarily wasted efforts, because delta compressions can be used to transfer the new document [27]. However, it does contribute to the inter-proxy communication.

Two factors limit the scalability of summary cache: the network overhead (the inter-proxy traffic), and the memory required to store the summaries (for performance reasons, the summaries should be stored in DRAM, not on disk). The network overhead is determined by the frequency of summary updates and by the number of false hits and remote 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.

4.1 Impact of Update Delays

We investigate delaying the update of summaries until the percentage of cached documents that are "new" (that is, not reflected in the summaries) reaches a threshold. The threshold criteria is chosen because the number of false misses (and hence the degradation in total hit ratio) tends to be proportional to the number of documents that are not reflected in the summary. An alternative is to update summaries upon regular time intervals. The false miss ratio under this approach can be derived through converting the intervals to thresholds. That is, based on request rate and typical cache miss ratio, one can calculate how many new documents enter the cache during each time interval and their percentage in the cached documents.

Using various traces, we simulate the total cache hit ratio when the threshold is 0.1%, 1%, 2%, 5% and 10% 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 (i.e. the list of document URLs). The results are shown in Figure 1. 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 two curves show the ratio of remote stale hits and the ratio of false hits (the delay does introduce some false hits because documents deleted from the cache may still be present in the summary).

The results show that, except for the NLANR trace data, the degradation in total cache hit ratio increases almost 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). The remote stale hit ratio is hardly affected by the update delay. The false hit ratio is very small, though it does increase linearly with the threshold.

For the NLANR trace, it appears that some clients are simultaneously sending two requests for the exact same document to proxy "bo" and another proxy in the collection. If we only simulate the other three proxies, the results are similar to those of other traces. With "bo" included, we simulate 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.

The results demonstrate that in practice, a delay threshold of 1% to 10% for updating summaries results in a tolerable degradation of the cache hit ratios. For the five traces, the threshold values translate into roughly 300 to 3000 user requests between updates, and on average, an update frequency of roughly every 5 minutes to an hour. Thus, the bandwidth consumption of these updates can be very low.

4.2 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 [24]. Although DRAM prices continue to drop, we still need a careful design, since the memory requirement grows linearly with the number of proxies, and summaries take DRAM away from in-memory cache of hot documents.

We first investigate two naive summary representations: exact-directory and server-name. In the exact-directory approach, the summary is essentially the list of URLs of cached documents, with each URL represented by its 16-byte MD5 signature [26]. In the server-name approach, the summary is the collection of Web server names in the URLs of cached documents. Since on average, the ratio of different URLs to different Web server names is about 10 to 1 (observed from our traces), the server-name approach can cut down the memory requirement by a factor of 10.

We simulate these approaches using the traces and find 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 4.4). 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 traffic.

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."

4.3 Bloom Filters – the math

A Bloom filter is a method for representing a set $A = \{a_1, a_2, \ldots, a_n\}$ of n elements (also called keys) to support membership queries. It was invented by Burton Bloom in 1970 [3] and was proposed for use in the web context by Marais and Bharat [25] as a mechanism for identifying which pages have associated comments stored within a *CommonKnowledge* server.

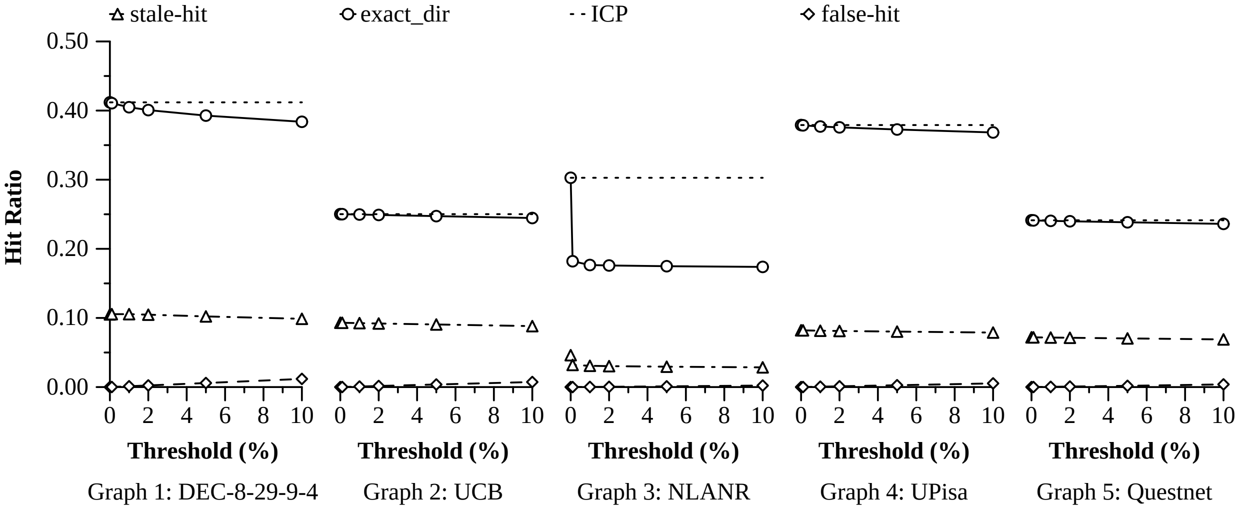

Figure 1: Impact of summary update delays on total cache hit ratio, remote stale hit ratio, and false hit ratio. The cache size is 10% of the "infinite" cache size.

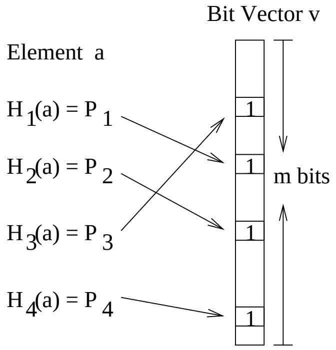

Figure 2: A Bloom Filter with 4 hash functions.

The idea (illustrated in Figure 2) is to allocate a vector v of m bits, initially all set to 0, and then choose k independent hash functions, $h_1, h_2, \ldots, h_k$, each with range $\{1, \ldots, m\}$. For each element $a \in A$, the bits at positions $h_1(a)$, $h_2(a)$, $\ldots$, $h_k(a)$ in v are set to 1. (A particular bit might be set to 1 multiple times.) Given a query for b we check the bits at positions $h_1(b)$, $h_2(b)$, $\ldots$, $h_k(b)$. 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." The parameters k and m should be chosen such that the probability of a false positive (and hence a false hit) is acceptable.

The salient feature of Bloom filters is that there is a clear tradeoff between m and the probability of a false positive. Observe that after inserting n keys into a table of size m, the probability that a particular bit is still 0 is exactly $(1 - 1/m)^{kn}$. 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 $1/2^k = (0.6185)^{m/n}$. Thus, under optimal k, the probability of a false positive reduces exponentially as m increases. In practice k must be an integer and 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 $\alpha = m/n$. 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 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.

Since in our context each proxy maintains a local Bloom filter to represent its own cached documents, changes of set A must be supported. This is done by maintaining for each location ℓ in the bit array a count $c(\ell)$ of the number of times that the bit is set to 1 (that is, the number of elements that hashed to ℓ under any of the hash functions). All the counts are initially 0. When a key a (in our case, the URL of a document) is inserted or deleted, the counts $c(h_1(a))$, $c(h_2(a))$, $\ldots$, $c(h_k(a))$ are incremented or decremented accordingly. When a count changes from 0 to 1, the corresponding

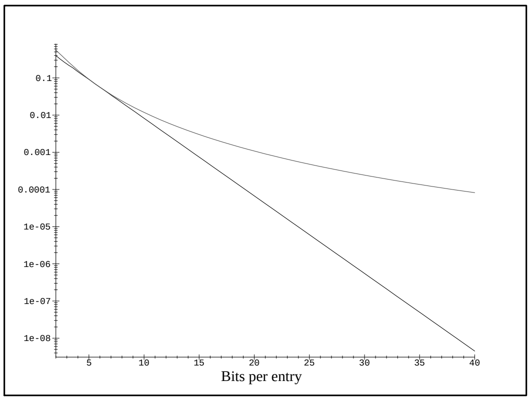

Figure 3: Probability of false positives (log scale). The top curve is for 4 hash functions. The bottom curve is for the optimum (integral) number of hash functions.

bit is turned on. When a count changes from 1 to 0 the corresponding bit is turned off. Hence the local Bloom filter always reflects correctly the current directory. In practice, allocating 4 bits per count is amply sufficient (for analysis, see [9]).

4.4 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 to other proxies. When updating the summary, the proxy can either specify which bits in the bit array are flipped, or send the whole array, whichever is smaller.

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 experiment 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; the number of hash functions is not the optimal choice for each configuration, but suffices to demonstrate the performance of Bloom filters. The hash functions are built by first calculating the MD5 signature [26] of the URL, which yields 128 bits, and then taking four groups of 32 bits from it. MD5 is a cryptographic message digest algorithm that hashes arbitrary length strings to 128 bits [26]. We select it because of its well-known properties and relatively fast implementation.

The performance of these summary representations, exact-directory, and server-name are shown in Figures 4 through 7. 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 remote cache hits, false hits and remote stale 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 number of messages by the number of HTTP requests in each trace. Both exact-directory and Bloom filter based summaries perform well, and server-name and ICP generate many more messages. For Bloom filters, there is a tradeoff between bit array size and the number of messages, as expected. However, once the false hit ratio is small enough, false hits are no longer a dominant contributor to inter-proxy messages. Rather, remote cache hits and remote stale hits become dominant. Thus, the difference in terms of network messages between load factor 16 and load factor 32 is small. Compared to ICP, Bloom filter based summaries reduce the number of messages by a factor of 25 to 60.

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 5.1) plus 4 bytes per bit-flip. The results show that in terms of message bytes, Bloom filter based summaries improve over ICP by 55% to 64%. In other words, Summary Cache uses occasional bursts of large messages to avoid continuous stream of small messages. Looking at the CPU overhead and network interface packets in Tables 2 and 4 (in which SC-ICP stands for the summary cache approach), we can see that this is a good tradeoff.

Table 3 shows the memory requirement per proxy of the summary representations, in terms of percentage of the proxy cache size, for DEC (16 proxies), and NLANR (4 proxies). (More data are available in [9].) The three

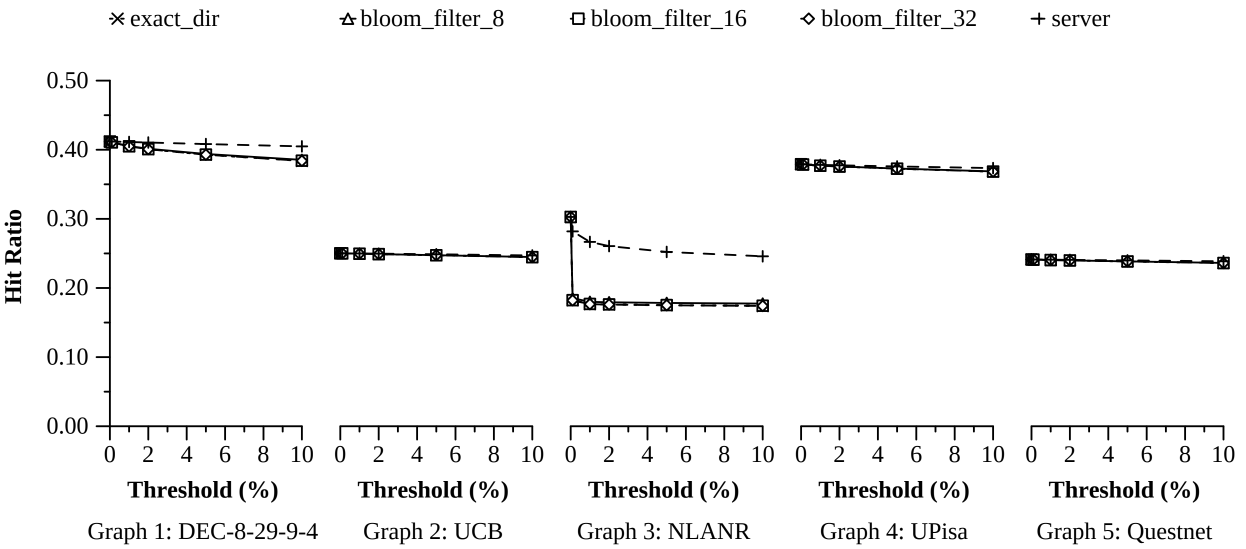

Figure 4: Total hit ratio under different summary representations.

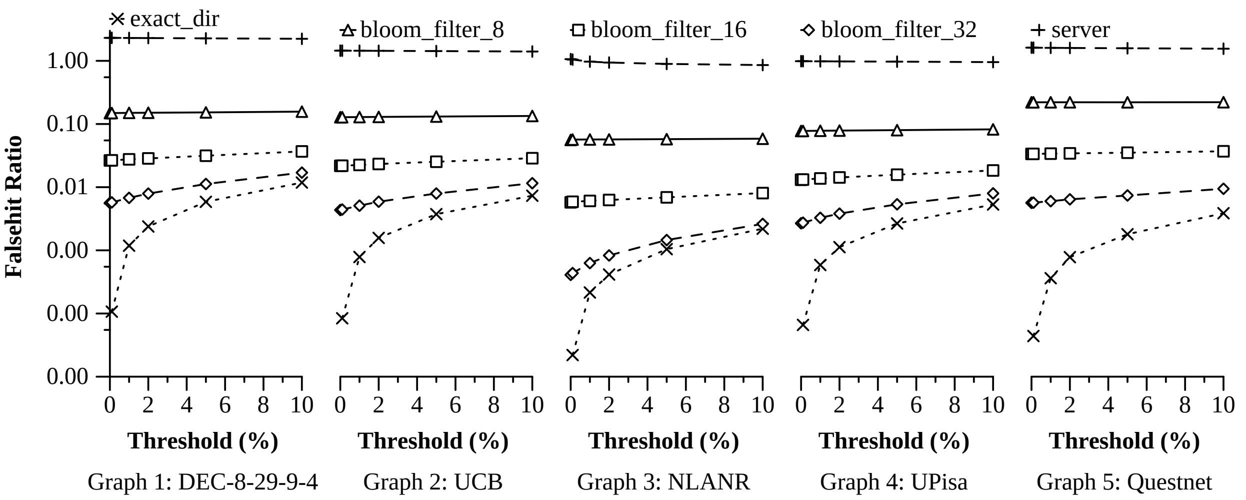

Figure 5: Ratio of false hits under different summary representations. Note that the y-axis is in log scale.

Approach	DEC	NLANR
exact_dir	2.8%	0.70%
server_name	0.19%	0.08%
bloom_filter_8	0.19%	0.038%
bloom_filter_16	0.38%	0.075%
bloom_filter_32	0.75%	0.15%

Table 3: Storage requirement, in terms of percentage of proxy cache size, of the summary representations.

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 or smaller memory requirement to the server-name approach, and yet has many fewer false hits and network messages. Table 3 also shows that for all approaches, the memory requirement grows linearly with the number of proxies.

Considering all the results, we see that Bloom filter summaries provide the best performance in terms of low network overhead and low memory requirements. Thus, we recommend the following configuration for the summary cache approach. The update threshold should be between 1% and 10% to avoid significant reduction of total cache hit ratio. If a time-based update approach is chosen, the time interval should be chosen such that the percentage of new documents is between 1% and

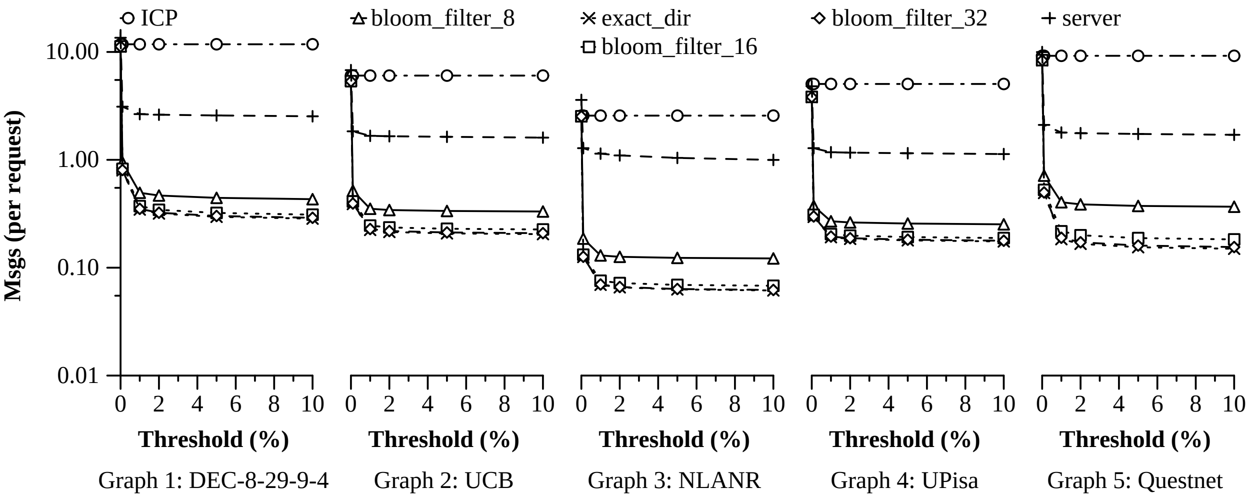

Figure 6: Number of network messages per user request under different summary forms. Note that the y-axis is in log scale.

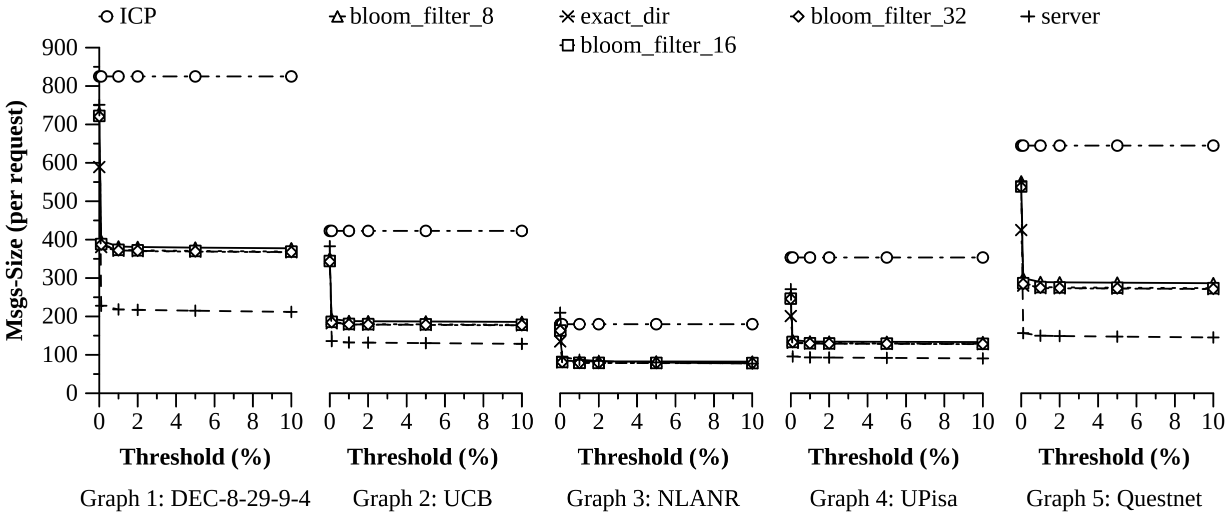

Figure 7: Bytes of network messages per user request under different summary forms.

10%. The proxy can either broadcast the changes (or the entire bit array if it is smaller), or let other proxies fetch the updates from it. The summary should be in the form of a Bloom filter. A load factor between 8 and 16 works well, though proxies can lower or raise it depending on their memory and network traffic concerns. Based on the load factor, four or more hash functions should be used. The data provided here and in [9] can be used as references in making the decisions. For hash functions, we recommend taking disjoint groups of bits from the 128-bit MD5 signature of the URL. If more bits are needed, one can calculate the MD5 signature of the URL concatenated with itself. In practice, the computational overhead of MD5 is negligible compared with the user and system CPU overhead incurred by caching documents (see Section 5.2).

4.5 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 Web 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 counters. The interproxy messages consist of update messages, false hits, remote cache hits and remote stale hits. The threshold of 1% corresponds to 10K requests between updates, each update consisting of 99 messages, and 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 and remote stale hits (which are relatively stable across the number of proxies), 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, on the order of several hundreds KB. Fortunately, update messages can be transferred via a non-reliable multicast scheme. Our simulations predict that, while keeping the overhead low, this scheme reduces the total hit ratio by less than 2% compared to the theoretical hit ratio of ICP.

Though none of the traces are large enough to enable meaningful simulation of 100 proxies, we have performed simulations with larger number of proxies and the results verify these "back of the envelope" calculations. Thus, we are confident that Summary Cache scales well.

5 Implementation and Experiments

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 built on top of Squid 1.1.14 and the prototype is publicly available [8]. A variant of our approach is also implemented in Squid 1.2b20 [29].

5.1 Summary-Cache Enhanced ICP

Our implementation assumes small delay thresholds and updates summaries via sending the differences. We add a new opcode in ICP version 2 [31], 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 hash functions for 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 taking the modulo of 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 specifies only which bits should be flipped, then loss of previous update messages would have cascading effects. The design enables the messages to be sent via a unreliable multicast protocol. Furthermore, every update message carries the header, enabling receivers to verify various information.

We modify Squid 1.1.14 to implement the above protocol. The default load factor in the implementation is 8, and the default number of hash functions is 4. An additional bit array is added to the data structure for each neighbor. The array 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 IP packet, we deviate from the recommendation in Section 4.4 by sending updates whenever there are enough changes to fill an IP packet. The implementation leverages Squid's built-in support to detect failure and recovery of neighbor proxies, and reinitializes a failed neighbor's bit array when it recovers.

5.2 Performance Experiments

We run two experiments with the prototype. The first experiment repeats the test in Section 3 and the results are included in Table 2 in Section 3, under the title "SC-ICP." The improved protocol reduces the UDP traffic by a factor of 50, and has network traffic, CPU times and client latencies similar to those of No-ICP.

Our second experiment replays the first 24,000 requests from the UPisa trace. We use a collection of 80 client processes running on 4 workstations, and have each client process emulate a set of real-life clients through issuing their Web requests. Client processes on the same workstation connect to the same proxy server. 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 3. Different from the synthetic benchmark, the trace contains a noticeable number of remote hits. The results are listed in Table 4.

The results show that the enhanced ICP protocol reduces the network traffic and CPU overhead significantly, while only slightly decreasing the total hit ratio. The enhanced ICP protocol lowers the client latency slightly compared to the No-ICP case, even though it increases the CPU time by about 12%. The reduction in client latency is due to the remote cache hits. Separate experiments show that most of the CPU time increase is due to servicing remote hits, and the CPU time increase due to MD5 calculation is less than 5%. We have experimented with other ways of replaying the trace, and the results are similar [9].

Exp	Hit Ratio	Client Latency	User CPU	System CPU	UDP Traffic	TCP Traffic	Total Packets
no ICP	16.94	6.22(0.4%)	81.72(0.1%)	115.63(0.1%)	4718(1%)	242K(0.1%)	259K(0.1%)
ICP	19.3	6.31(0.5%)	116.81(0.1%)	137.12(0.1%)	72761(0%)	245K(0.1%)	325K(0.2%)
Overhead		*1.42%*	*43%*	*19%*	*1400%*	*1%*	*25%*
SC-ICP	19.0	6.07 (0.1%)	91.53(0.4%)	121.75(0.5%)	5765(2%)	244K(0.1%)	262K(0.1%)
Overhead		*-2.4%*	*12%*	*5%*	*22%*	*1%*	*1%*

Table 4: Performance of ICP and Summary-Cache for UPisa trace.

Our results indicate that the summary-cache enhanced ICP solves the overhead problem of ICP, requires minimal changes, and enables scalable Web cache sharing over a wide-area network.

6 Related Work

Web caching is an active research area. There are many studies on Web client access characteristics, web caching algorithms [32, 23, 4], and Web cache consistency [16, 22, 19]. Our study does not address caching algorithms or cache consistency maintenance, but leverages the existing results.

Recently there have been several new proposals on Web cache sharing protocols. The Cache Array Routing Protocol [30] divides the 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 it performs for wide-area cache sharing, where proxies may be distributed over a regional network. The Relais project [15] suggests using local directories to facilitate finding documents in other caches, and updating the directories asynchronously. However, existing publications on this project do not seem to address the issues of update frequency and memory consumption. Finally, proxies built on top of a tightly-coupled cluster of workstations also use various hashing and partitioning schemes to utilize the memory and disks in the cluster [10], but the relevant methods are not appropriate for wide-area networks.

Our study is partially motivated by an existing proposal called directory server [11]. The approach uses a central server to keep track of the cache directories of all proxies, and lets 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. The advantage is that little communication is needed between sibling proxies.

Many studies also focus on Web cache hierarchies and cache sharing. Hierarchical Web caching was first proposed within the Harvest project [14, 6]. The “Adaptive Web caching” proposed in [33] offers a multicast-based adaptive caching infrastructure for document dissemination on the Web.

Though we do not address the issue in this paper, summary cache can be easily applied to cache hierarchies. That is, it can be used as a mechanism to communicate the contents of parent caches to child proxies, and eliminate most of ICP queries to the parent caches. 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 a parent proxy has to process. Applying summary cache can result in significant reduction of the queries and associated overheads.

7 Conclusions and Future Work

We propose the 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 current cache sharing protocols, and show that the summary cache approach substantially reduces the overhead. We study two key aspects of this approach: the effects of delayed updates, and the succinct representation of summaries. Our solution, Bloom filter based summaries with update delay thresholds, has low demand on memory and bandwidth, and yet achieves a hit ratio similar to that of the original ICP protocol. In particular, trace-driven simulations show that, compared to ICP, the new protocol reduces the number of inter-proxy protocol messages by *a factor of 25 to 60*, reduces the bandwidth consumption *by over 50%*, while incurring almost no degradation in the cache hit ratios. Simulation and analysis further demonstrate the scalability of the protocol.

We have built a prototype implementation in Squid 1.1.14. Synthetic and trace-replay experiments show that, in addition to the network traffic reduction, the new protocol reduces the CPU overhead between *75% to 95%* and improves the client latency. The prototype implementation is publicly available [8].

Much future work remains. We plan to investigate the impact of the protocol on parent-child proxy cooperations, and the optimal hierarchy configuration for a given workload. We also plan to study the application of summary cache to various Web cache consistency protocols. Last, summary cache can be used in individual proxy implementation to speed up cache lookup, and we will quantify the effect through modifying a proxy implementation.

Acknowledgement Luigi Rizzo and Julianne Weekers provided the UPisa and Questnet traces that made this study possible. We are indebted to the Wisconsin Wind Tunnel research group for providing the resources for our simulation. The research benefited from discussions with Jeff Mogul, Monika Henzinger and Bart Miller. Finally, the anonymous referees provided valuable comments that greatly improved the paper.

References

[1] Jussara Almeida and Pei Cao. Wisconsin proxy benchmark 1.0. *http://www.cs.wisc.edu/ cao/wpb1.0.html*, 1997.

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

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] Li Fan, Pei Cao, and Jussara Almeida. A prototype implementation of summary-cache enhanced icp in squid 1.1.14. *http://www.cs.wisc.edu/ cao/sc-icp.html*, February 1998.

[9] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. Technical report, Technical Report 1361, Computer Science Department, University of Wisconsin-Madison, February 1998. URL http://www.cs.wisc.edu/ cao/papers/summarycache.html.

[10] 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.

[11] 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/.

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

[13] 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/.

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

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

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

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[27] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta encoding and data compression for http. In *Proceedings of ACM SIGCOMM'97*, August 1997. Available from http://www.research.att.com/ douglis/.

[28] National Lab of Applied Network Research. Sanitized access log. *Available at ftp://ircache.nlanr.net/Traces/*, July 1997. Configuration files for the proxies are at http://ircache.nlanr.net/Cache/Configuration/.

[29] Alex Rousskov. Cache digest. *http://squid.nlanr.net/Squid/CacheDigest/*, April 1998.

[30] 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.

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

[32] 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 SIGCOMM'96*, August 1996. URL http://ei.cs.vt.edu/ succeed/96sigcomm/.

[33] 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.ps.