World Wide Web 2 (1999) 47–67

A performance study of the Squid proxy on HTTP/1.0

Alex Rousskov [a] and Valery Soloviev [b]

[a] *National Laboratory for Applied Network Research, 1850 Table Mesa Drive, SCD, Room 22c, Boulder, CO 80307, USA*
E-mail: rousskov@nlanr.net

[b] *Inktomi Corporation, 1900 South Norfolk Street, Suite 310, San Mateo, CA 94403, USA*
E-mail: soloviev@inktomi.com

This paper presents a performance study of the state-of-the-art caching proxy called Squid. We instrumented Squid to measure per request network and disk activities and conducted a series of experiments on large Web caches. We have discovered many interesting and consistent patterns across a wide variety of environments. Our data and analysis are essential for understanding, modeling, benchmarking, and tuning performance of a proxy server.

1. Introduction

The World Wide Web has clearly become *the* environment for global information distribution, exchange, and sharing. Caching proxies are playing an important role in handling Web traffic. Web caching is one of the key methods for coping with the exponential growth of the Web.

Caching proxies are usually installed where many clients are accessing the Internet through a single path. During the first client request for a Web object, a proxy cache stores a copy of the response. Subsequent request for the same object may then be served from the local storage rather than from a remote server. Caching saves network bandwidth on outbound connections and may reduce client response time.

Local copies of the Web objects are usually stored on disk. The storage capacity of a typical proxy is several gigabytes. While savings in bandwidth are almost guaranteed, the improvement in response time is not. Indeed, every request that goes through a proxy encounters extra protocol overheads and processing delays. These delays may actually increase total response time compared to a direct (no proxying) transfer. Caching proxies must employ special algorithms to reduce processing delays. Also, when a caching proxy becomes a bottleneck, the clients immediately experience untolerable "network" delays. Thus, a proxy must handle intense traffic without performance degradation. Sophisticated techniques have been developed to improve proxy performance under heavy load.

However, little is known about performance of real proxies. At first, caching proxies were developed as public domain software by Web enthusiasts. These pioneers did not have time and means for experimenting with alternative designs and performance solutions. Thus, they had to rely on "common sense" and extensive borrowing of design and performance ideas from other applications. Only now, public proxy software is mature enough to shift the focus from adding essential features to enhancing the performance. The public domain leader is a state-of-the-art caching proxy called Squid [Wessels 1998].

There is also a growing number of commercial products that are squeezing out public domain software from caching solutions market. Commercial products are appealing to enterprises handling large volumes of Web traffic (the interesting case). Some commercial products claim their performance superiority to public domain software but present limited interest to the research community because of their closed design and proprietary protocols. Interestingly, many commercial proxies are derived from Squid or its predecessor Harvest.

2. Research contributions

We present performance analysis essential in understanding the work of a state-of-the-art caching proxy. Our instrumented version of Squid measures per request network and disk activities. These detailed measurements allows for in-depth studying of major proxy components. We identify and quantify network and disk performance degradation during high load periods. We also demonstrate that various classes of requests have different impact on proxy resources, and optimization decisions must take that into account.

We study the performance of caching proxies in a variety of Unix environments and are able to identify many common performance patterns. We also show how the level of caching hierarchy affects proxy performance.

Our measurements and observations can be used for:

- performance modeling of a proxy server,
- benchmarking of proxies,
- performance tuning of existing proxies,
- identifying potential bottlenecks,
- evaluating current design decisions and future enhancements.

Our analysis is unique because we cover a variety of hardware, Unix operating systems, caching hierarchy levels, and workloads rather than concentrating on a single

configuration. To our knowledge, our data collection is the only source of comprehensive proxy performance statistics available publicly [Rousskov and Soloviev 1998]. The collection has already been used in several studies on Web caching [Tewari *et al.* 1998].

Finally, our study is not without a few shortcomings. For example, the measurements are done with HTTP/1.0 traffic only. For details and discussion, please refer to section 7.

3. Terminology

Web caching, as any modern invention, uses special terminology to name objects of interest. Unfortunately, many terms are often misused or misinterpreted. In this section, we give our interpretation of commonly used Web caching terms. Terms are grouped logically rather than alphabetically. HTTP document gives precise definitions for some of the terms [Fielding *et al.* 1998]. Note that throughout the discussion we use terms *object*, *document*, and *file* interchangeably, usually meaning an origin server *response*.

client An originator of a request for a particular Web object.

origin server A Web server that keeps the original copy of a document.

proxy cache HTTP defines proxy as "An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients."

server Origin server or a proxy. With cooperative caching, a proxy may fetch misses from another proxy. The latter proxy effectively becomes an "origin" server for the first proxy.

uncachable object An object that should not be cached or served from the cache by a proxy for whatever reasons. For example, a "force-reload" request from a Web browser (the client forbids caching), or a "dynamic" document fetched from an origin server (the server forbids caching). Finally, a caching policy of the proxy itself may not admit certain documents for caching. For example, very big documents are often uncachable. Statistics not included in this paper shows that about 20–25% of Web objects coming through proxies are uncachable.

negative caching Storing an indication that a *client request* for a particular object resulted in error. If the same object is requested again within a short period of time, the same error reply will be sent to a client without verifying the current situation.

hit A client request satisfied with information already available in the cache. The origin server may be contacted to verify the freshness of the information.

200 hit A hit with a 200 reply code. Occurs when a proxy reply contains object content and no errors were detected. This is a most common type of a hit.

IMS hit (304 hit) A hit with a 304 reply code. Occurs when a client requests an object with modification date later than a certain time. The proxy checks that the object has not been modified after that time. Thus, only a small notification message is sent back to the client. This hit class should not be confused with returning of stale responses to "reload" client requests. Correctly configured proxy honors "reload" requests and fetches the object directly from the origin server.

disk hit 200 hits that were resolved by reading content from disk.

memory hit 200 hits that were resolved by sending content from memory. The requested document was in the "hot memory" buffer and, thus, no disk activity was necessary.

negative hit A hit for a *negatively cached* document.

miss A client request that is not a hit.

swap request *Swap-In* or *Swap-Out* request. May result in more than one disk I/O. Not all HTTP requests result in swap requests. Moreover, one HTTP request may result in many swap requests, depending on the HTTP request properties and cache condition.

swap-in request An internal request to load an object from disk into memory.

swap-out request An internal request to store an object from memory to disk.

hit ratio (DHR) Document Hit Ratio is a ratio of the number of *hits* to the total number of requests received by a proxy within a given time interval.

byte HR (BHR) A ratio of the total size of all *hits* to the total size of all replies sent by a proxy within a given time interval.

swap-in ratio A ratio of the number of *Swap-In* requests to the total number of *swap* requests performed within a given time interval.

4. Methodology

4.1. The patch

Performance data was collected using a *patched* version of Squid caching proxy [Rousskov 1997]. The patch enabled Squid to log detailed *per request* measurements of network and disk I/O activities. For each processing stage, the patch recorded the following three measurements.

- *The start of the activity*: A timestamp when the activity is about to begin with microsecond[1] resolution.
- *The first delay*: For example, the time it took to read the first page of a file or to receive the first block of data over the network. The delay is measured from the start of the activity with microsecond resolution.
- *Total delay*: The time it took to complete the activity with microsecond resolution. For example, the total time to read a document from a disk or receive a document from a server. All delays that happened after the start of the activity are included.

[1] To record the start of "client start" and swap activities, the patch re-uses standard log fields that have millisecond resolution.

Some fields may contain null values if the corresponding request did not participate in a certain activity.

For each HTTP GET request, the patch logged information on the following request stages.

Client connect: Accepting a connection from a client and reading the client request. Client-side delays are not measured. Network interface card queuing time prior to the `accept()` system call is not measured.

Proxy connect: Establishing a connection with a server and then sending a request for a document. DNS lookup delays are not included.

Server reply: Receiving a reply from a server and closing the connection.

Proxy reply: Sending a reply to a client and closing the connection.

Swap-In: Swapping a document from a disk into memory. Includes a delay for opening a file. Note that we can measure file system performance, not raw disk I/Os.

Swap-Out: Swapping a document out from memory to disk. Includes a delay for creating a file.

It is important to note that the patch itself did not increase the proxy load. Time measurements were performed using Squid internal time, without extra system calls. Total memory consumption did not change since measurements were kept in a compact form, for pending requests only; a negligible addition especially compared to all static information that had to be maintained by Squid. Overhead from adding extra bytes to standard log entries is considered to be negligible because the cost of I/O for small log entries does not depend on their actual size. The total disk space requirement was increased. However, it was not a problem for the participants since logs were usually kept on a separate disk with enough space.[2]

4.2. Experiment framework

For each participating proxy, we collected at least one day worth of logs. When log files were collected, we ran several analyzing scripts to extract useful statistics. For a 24 hour log, we performed about 200 measurements. A measurement is a trace or distribution of a single performance parameter like "client connect delay trace" or "distribution of network transfer size". Related measurements were grouped in *plots*. For example, the "File Size Distribution" plot contains size distributions for various classes of documents.

To produce meaningful results, we used *percentiles* and distributions whenever possible. The *median* (50th percentile) was used as an estimate of an average value (instead of the mean which is often unreliable with data that includes large isolated peaks).

Most of the measurements were grouped using 20 minute slots. These slots allowed us to detect spikes in proxy performance while still having enough entries to produce a meaningful median. That is, significantly longer slots would make the graphs too "smooth," and shorter slots may not have enough measurements to calculate reliable statistics.

On some proxies, there was virtually no load at night and early in the morning. We filtered out those periods as neither interesting for this study nor statistically representative.

In a few instances, we have discussed unusual patterns with the proxy administrator to isolate interesting performance phenomena from anomalies caused by configuration bugs, network outages, etc.

5. Participating proxies

Seven Squid proxies participated in our study[3] (table 1). The proxies represented all levels of caching hierarchy from *leaf* university proxies to *top-level* proxies serving large country-wide networks to the *root* proxy of NLANR international hierarchy. Information about Squid configuration and general traffic characteristics are summarized in tables 2 and 3.

Squid uses a two-level cache. Very popular and recently requested objects are kept in a small "hot memory" buffer. The majority of the documents reside in disk cache. High and low watermarks control the buffer management algorithms. In short, when the high watermark is reached, Squid starts removing objects more rapidly. When the current level is below the low watermark, a more aggressive caching mode is enabled. In practice, utilization levels should fluctuate between the two watermarks.

Proxies cache objects according to an algorithm similar to LRU-Threshold. The threshold parameter is usually set to 4 or 8 MB, admitting the majority of cachable objects. CGI and other "dynamic" documents are usually not cached.

The reader is referred to Squid documentation [Wessels 1998] for other numerous implementation details. Clearly, some Squid-specific hacks could influence the collected data. However, we believe that most performance patterns are typical for many other caching proxies currently in use.

6. Results

We have collected 18 days worth of logs. Most participating proxies submitted one–three day logs while **sv** was profiled for more than a week. Most experiments were conducted in October 1997. All logs were analyzed. We selected days when the load was high and Squid was running without problems for several days. In this section, we compare *plots* from different proxies or different days of one proxy. These comparisons reveal consistent patterns and identified performance anomalies.

[2] The latter also explains the negligible effect of logging, if done properly, on Squid performance.

[3] After the experiments were launched, the number of participants actually increased to 11 proxies.

Table 1
Proxy configuration.

Proxy	Country	Type	Machine	RAM MB	#disks	Storage GB	OS
sv	USA	root	DEC Alphaserver 1000, 266 Mhz	512	7	26	DEC UNIX V3.2D-1 (Rev. 41)
surfnet	Netherlands	top level	IBM RS6000 model 960	256	12	6	AIX v4.2
ruu	Netherlands	leaf	Sun Sparcstation 4	160	3	6	SunOS 4.1.4 sun4m
uninett	Norway	top level	SGI Challenger S 200 Mhz MIPS R4400	256	3	9	IRIX 5.3
uit	Norway	leaf	SGI Challenger 200 Mhz MIPS R4400	256	4	13	IRIX 5.3
mexcom	Mexico	inter-mediate	Intel Pentium 133 MHz	128	3	7.5	FreeBSD 2.2
adfa	Australia	leaf	Sparc 20, 2×150 MHz	284	4	8	SunOS 5.5

Table 2
Squid configuration.

Proxy	Squid version	Hot memory buffer		#disks for cache	Cache capacity	
		MB	low–high%		GB	low–high%
sv	1.1.17	20	85–95	5	16.0	75–95
surfnet	1.1.15	24	85–90	6	6.2	90–95
ruu	1.1.15	32	75–90	2	5.6	90–95
uninett	1.1.16	32	90–90	2	6.4	90–95
uit	1.1.16	32	75–75	2	3.8	90–95
mexcom	1.1.17	48	75–90	2	1.8	90–95
adfa	1.1.17	32	75–90	2	5.8	85–95

Table 3
Daily traffic.

Proxy	Documents	Unique documents	Unique clients	Unique servers
sv	991,085	542,568	95	39,144
surfnet	356,322	223,992	29	12,383
ruu	151,881	75,564	518	4,548
uninett	212,419	145,729	90	9,690
uit	147,927	65,152	378	3,759
mexcom	53,202	41,592	21	2,541
adfa	248,104	120,999	798	7,308

The number of possible comparisons was very large. We had to automate the process of generating the graphs and grouping the results. It was not technically feasible to present all our plots for all proxies in this paper. Thus, we selected three proxies for a base line presentation: **sv**, **surfnet**, and **ruu**. The three selected proxies represented the three levels of caching hierarchy and were the busiest proxies in our collection.

The entire set of plots is publicly available along with many interesting comparisons [Rousskov and Soloviev 1998].

This section presents a series of comparisons. For each comparison, we include three plots, one per selected proxy. It is essential to note that other proxies express similar performance *patterns* unless noted otherwise. As we shall see, most patterns are shared by all proxies despite the differences in absolute numbers. We use the same Y-scale plots within a comparison whenever possible. However, on some graphs we had to use different Y-scales to show important details. To enhance the presentation, some distributions are shown with *y*-axis starting above 0. Unadjusted graphs are available on the Web [Rousskov and Soloviev 1998]. All time-of-day measurements are in Coordinated Universal Time (UTC), *not* local time.

6.1. Traffic patterns

In this section, we group measurements that do not depend much on the hardware or software used for proxying. These measurements describe *environmental factors* such as distribution of file sizes or traffic intensity. Environmental factors are important for estimating system requirements and tuning configuration parameters. Relevant in-depth studies can be found elsewhere [Arlitt and Williamson 1996; Barford and Crovella 1998].

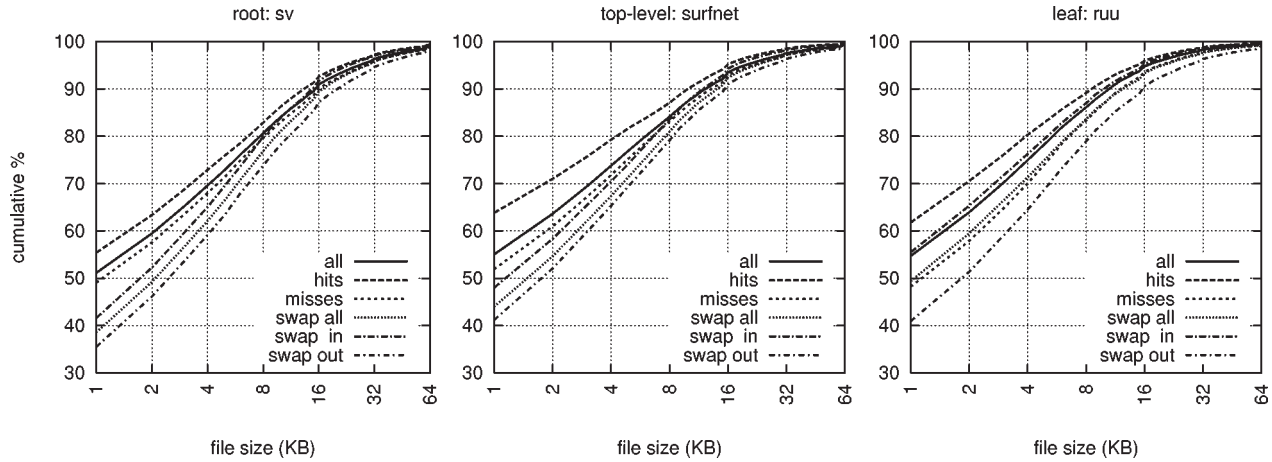

Figure 1. Transfer sizes.

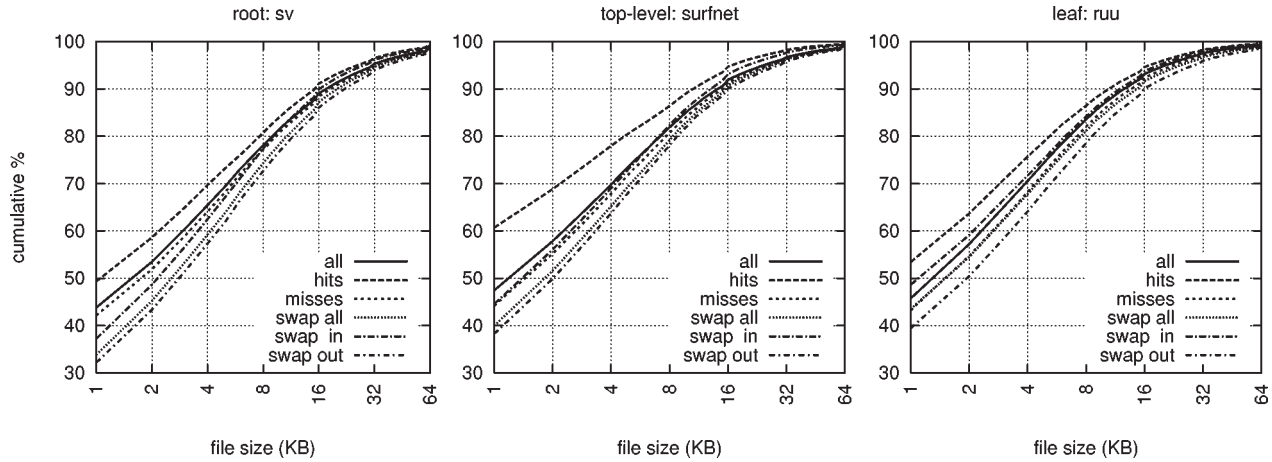

Figure 2. File sizes.

6.1.1. Transfer size distribution

We start our presentation by plotting the distributions of transfer sizes[4] for various categories (figure 1). To show the real traffic rather than file size distribution, we count *every* access to a document. Thus, a document that had 3 accesses would be counted once as a miss (if the document was not cached before) and twice as a hit (if the document was still in the cache). The "all" curve counts both hits and misses. The "swap all" curve shows the size distribution for all cached documents counting both swap-in and swap-out requests.

Transfer sizes are essential in estimating network and disk bandwidth requirements. Figure 1 shows that more than 50% of all network transfers are smaller than 1 KB. About half of the disk transfers are smaller than 2 KB. Note that all network and disk transfers are counted. Squid uses 8 KB pages for these transfers.

Misses are larger than hits (have more large transfers) because large documents are not popular (big unpopular files are always counted at least once as misses but rarely counted as hits) and every access to a document is counted (small popular files are counted many times for hits and only once for misses). Note that participating proxies do cache large files (up to 8 MB). However, 99% of transfers are smaller than 64 KB. Thus, the existence of an upper limit for cachable objects has a negligible impact on figure 1.

Swap-in requests are larger than hits because of a significant number of IMS hits that are very small but are never retrieved from the disk. Swap-out transfers are probably larger than misses because the number of relatively small but *uncachable* documents is high. Further measurements are needed to support the last statement.

6.1.2. File size distribution

Disk capacity requirements depend on what files are *cached*, not on what files are transferred or swapped. The "File sizes" graph (figure 2) shows the distribution of file sizes for hits, misses, and swap requests. That is, all misses contribute to the "miss" curve, all swap-ins to the "swap in" curve, etc. To show file size distribution rather than traffic, we count only the first access to a document. A file size rarely changes significantly from one access to another

[4] Squid logs the total number of bytes transfered, including HTTP headers. Also, HTTP headers are stored together with documents.

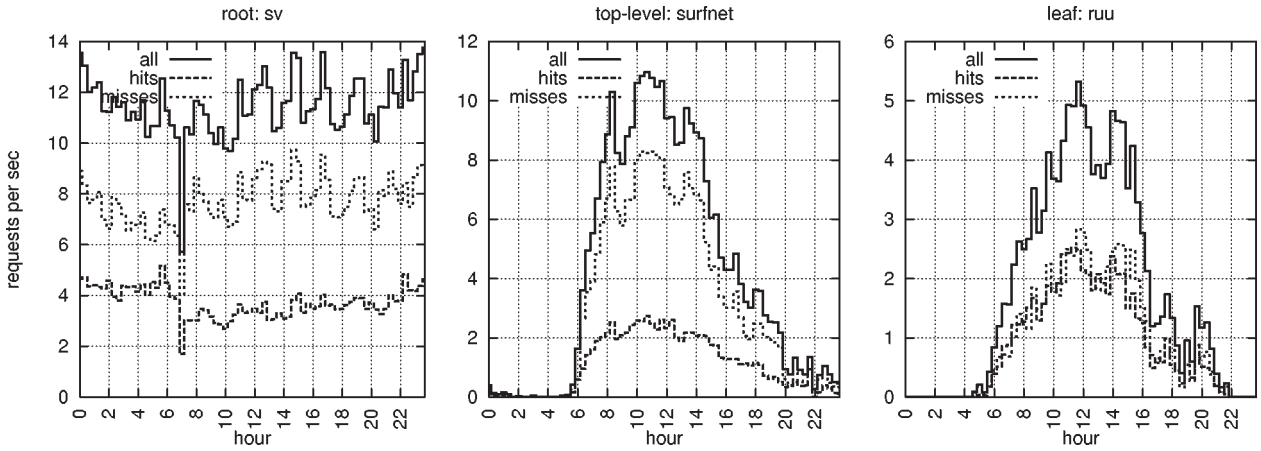

Figure 3. Traffic intensity.

and does not change at all for hits. Thus, counting only the first access to a document is valid.

The distribution of file sizes is important for estimating memory requirements for Squid. A common approach is to estimate the average size of a document by considering all entries in an access log file (i.e., calculating the average *transfer* size). Thus, a single popular document may be counted many times. Also, non-cachable documents that are never stored in a hash table will be counted. Using transfer sizes instead of file sizes may lead to underestimation of memory requirements and severe degradation in performance due to swapping.

On average, proxies swap-out larger objects than they swap-in. This asymmetry is, however, inevitable. Proxies have to cache large objects to get a decent Byte Hit Ratio. The latter is created by relatively *few* popular *large* objects.

6.1.3. Proxy traffic intensity

The number of client requests received by a proxy (figure 3) determines the stress on each proxy component such as the network or disk storage subsystem. As we shall see, performance patterns always follow traffic intensity.

We have detected two types of proxy load. Root proxy (**sv**) experiences relatively small variations in load during a 24-hour period. Other proxies have bell-shaped curves with highest load during the day and lowest load at night. It is possible to utilize a proxy during idle hours for improving its performance during peak load. For example, an alternative caching policy can use idle time to refresh the content of a cache based on last day statistics [Rousskov and Soloviev 1997]. Prefetching of documents into the cache is also a possibility.

Root proxy serves international traffic for all time zones. Consequently, it does not have idle periods. One may speculate that there may be a tendency to "constantly loaded" top level servers as the share of international Web traffic grows and traffic becomes more uniformly distributed.

It is important to analyze the performance of a proxy throughout the day. The presence of idle periods is important to algorithms that rely on lazy garbage collection or do other "off-line" activities [Rousskov and Soloviev 1997]. Peak periods often correspond to excessive load and should be studied with care; real proxies may be *over*loaded and show extreme performance patterns that should not happen in normal conditions.

6.2. Aggregate performance

In this section, we present measurements that show the *aggregate* proxy performance. That is, performance that depends on several proxy components at once. Contributions of individual components are studied in the following sections.

6.2.1. Concurrent requests

The number of concurrent requests (figure 4) reflects the ability of a proxy to handle traffic.[5] Less powerful or overloaded proxies have larger average request response time. And larger request response time corresponds to more concurrent requests pending in the system. For any proxy, the request response time increases with load. The actual increase depends on the request type and is hard to predict.

The concurrency level has a theoretical lower bound of the number of concurrent *clients* or clients submitting requests at the same time. In reality, the number of concurrent requests being processed well exceeds that limit. That is, the number of concurrent requests assuming zero proxying delay (unrealistic best-case scenario) would be significantly lower than the measured value.

The ratio of concurrent hits to misses is quite different from the same ratio for traffic intensity. Misses (middle curve on all graphs) are much slower than hits (lowest curve) because they are retrieved from other servers rather than directly from proxy. Recall that hits rarely contact other servers (only when the cached document becomes stale). The ratio of concurrent misses to hits present in the system may be as high as 7 : 1, which is much higher than the ratio of misses to hits in the traffic.

[5] It also reflects the difference between client and server transfer speed if any.

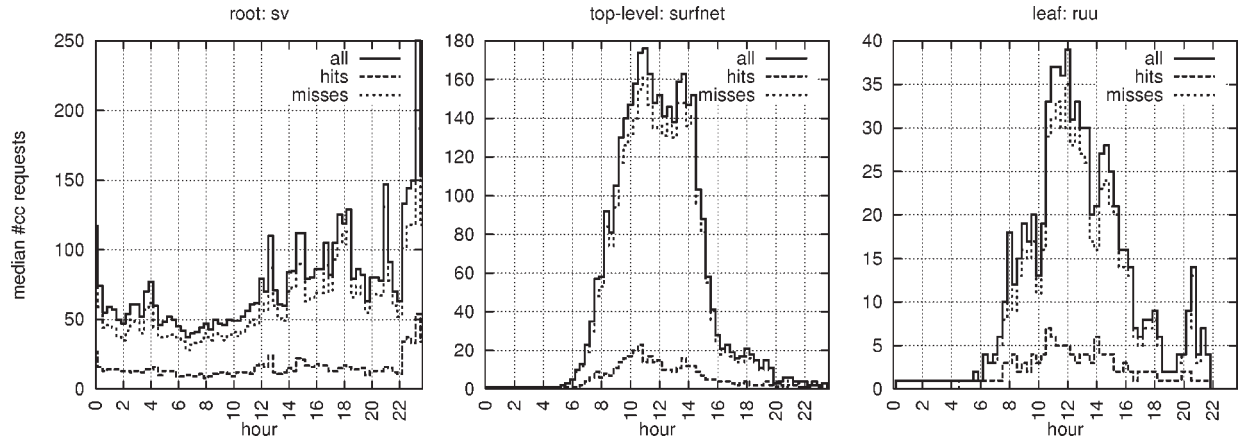

Figure 4. Concurrent requests.

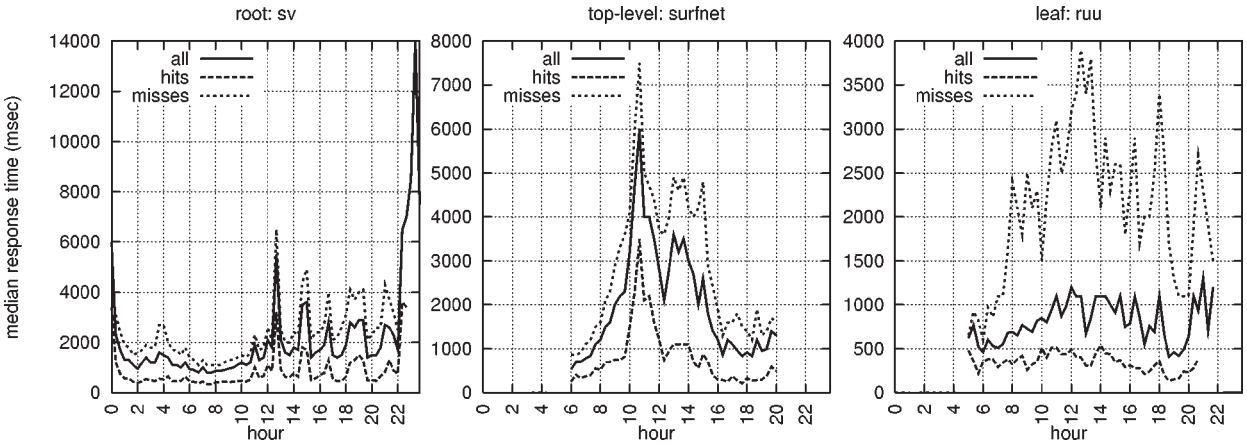

Figure 5. Request response time.

Slow speed makes misses the major consumer of proxy resources. It may seem that slow, virtually idle TCP connections would have little if any impact on proxy performance. That is not true for several reasons. A connection (idle or not) does consume memory. A proxy must keep various in-memory metadata about the request being processed and, perhaps, some I/O buffers for the connection. Moreover, `select(2)`-based implementations must check "idle" TCP sockets for readiness all the time. A large number of potentially ready file descriptors makes select loops very expensive [Banga and Mogul 1998] and jeopardizes the performance of "busy" connections (e.g., hits). The situation becomes even worse when a slow server connection leaks data in small portions. A proxy spends two–three I/Os to process every small chunk of data, increasing delays for other connections to be processed.

Identifying the major source of resource consumption is essential for optimization purposes. For example, one could increase the Squid I/O page size for hit requests to reduce the number of I/Os per hit. Reduction in disk I/Os may improve hit response time while preserving total memory requirement because the number of concurrent hits is much smaller than misses and their memory impact is negligible. On the other hand, a similar increase for misses may significantly increase the total memory requirement.

Another important observation is at least 100% increase in the number of concurrent requests on the root proxy during peak load. **Sv** works here at the limit of its resources. Even a small increase in traffic intensity may lead to severe performance degradation. This behavior is typical for several other proxies we have studied. Ideally, such a proxy requires an upgrade.

6.2.2. Request response time

Decreasing response time is a major function of a caching proxy. Our results (figure 5) show that the median response time of a hit may be five times smaller than of a miss. Thus, a proxy does decrease the response time for some requests. The average improvement (i.e., savings in the average response time compared to the all-misses case) depends on the Hit Ratio. **Sv** proxy improved the response time of an average request by 28%. Leaf proxies can achieve a 60% improvement. It is not clear, however, how the combination of fast hits and slow misses affects user perception of QoS. User perception is hard to estimate quantitively, especially when performance data from the client-side is not available.

Ideally, a separate study should compare the response time of proxied versus direct traffic. However, it is important to note that the presence of proxies is often determined

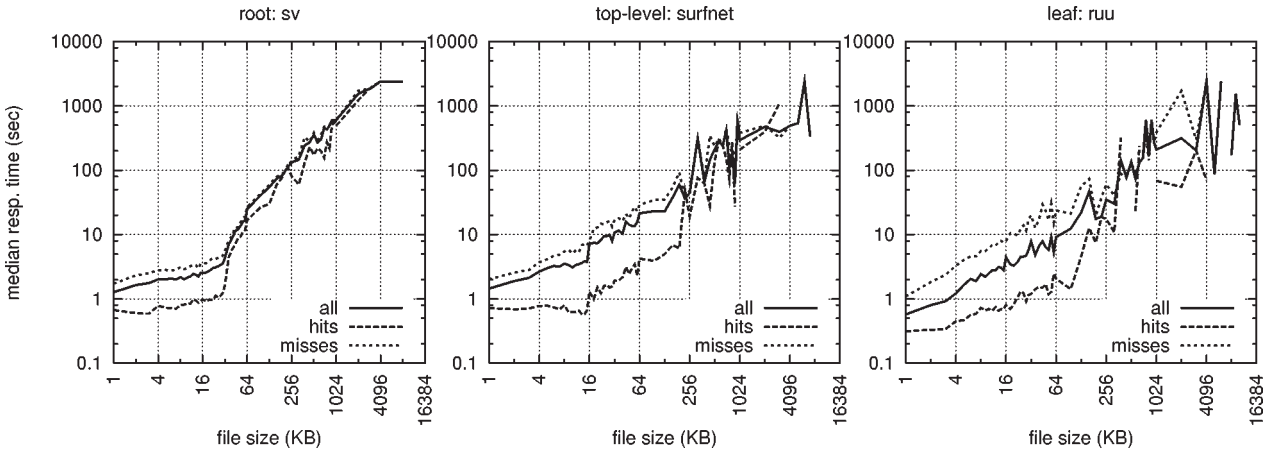

Figure 6. Request response time vs. transfer size.

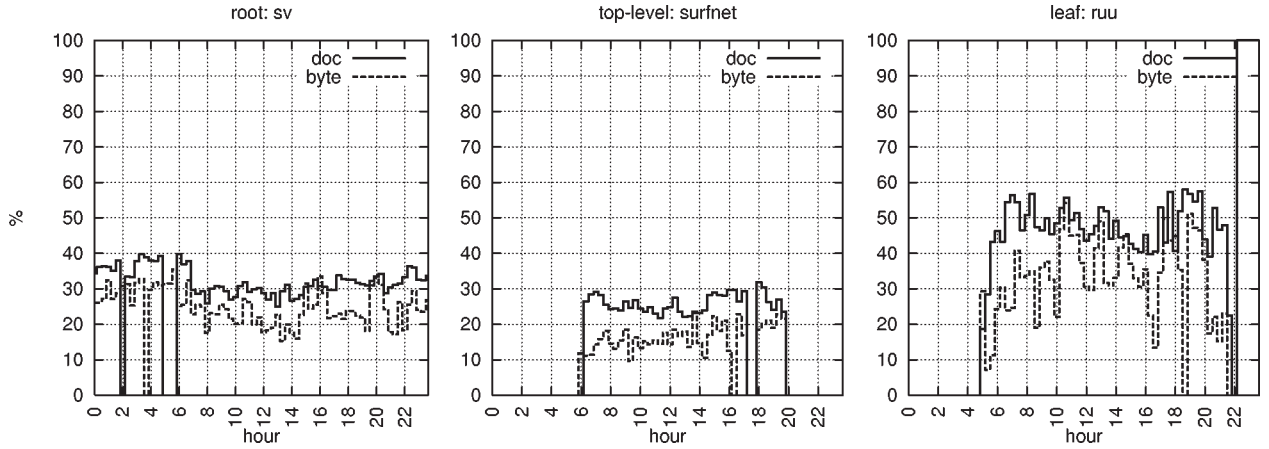

Figure 7. Hit ratios.

by factors other than response time. For example, network bandwidth costs or firewalls may mandate proxying. Thus, comparison with direct traffic is certainly interesting but not essential for proxy performance analysis.

Both hit and miss response times increase sharply during peak load. Further measurements will identify network and proxy components responsible for this effect.

6.2.3. Request response time vs. transfer size

The response time reduction for hits compared to misses depends on the document size. Interestingly, most savings in response time come from hits smaller than the size of TCP socket output buffer (32 KB for **sv** and 16 KB for **surfnet**), see figure 6. The effect of TCP buffer is not visible on the **ruu** proxy. **Ruu** was the only proxy where we did not observe this effect.

The graphs suggest that increasing the TCP buffer for hits may be worthwhile. A low level analysis of TCP stack traces would be required to make the final conclusion. The actual improvement (if any) may depend on the OS and other factors. Also note that the number of files that exceed the TCP buffer size is relatively small and may affect the precision of response time measurements.

Large hits do not improve the response time much. However, they are responsible for a higher Byte Hit Ratio. Consequently, we have a fundamental tradeoff between improving the response time (by limiting the maximum cachable document size and, thus, caching more smaller objects) and saving more bandwidth (by caching more large objects). There is no general algorithm for solving the tradeoff because the optimal solution depends, among other parameters, on a *cost model*, and cost models do vary a lot.

6.3. Hits analysis

This section concentrates on analysis of hit requests. Hits make deployment of proxies worthwhile. By increasing the number of hits and optimizing the hit response time one can significantly improve QoS. Hits have a major impact on proxy performance.

6.3.1. Hit ratios

Document Hit Ratio (figure 7) affects the savings in request response time as well as the proportion between swap-in and swap-out requests. Byte Hit Ratio characterizes savings in network bandwidth and determines disk bandwidth requirements. DHR is usually 7–9% higher than BHR.

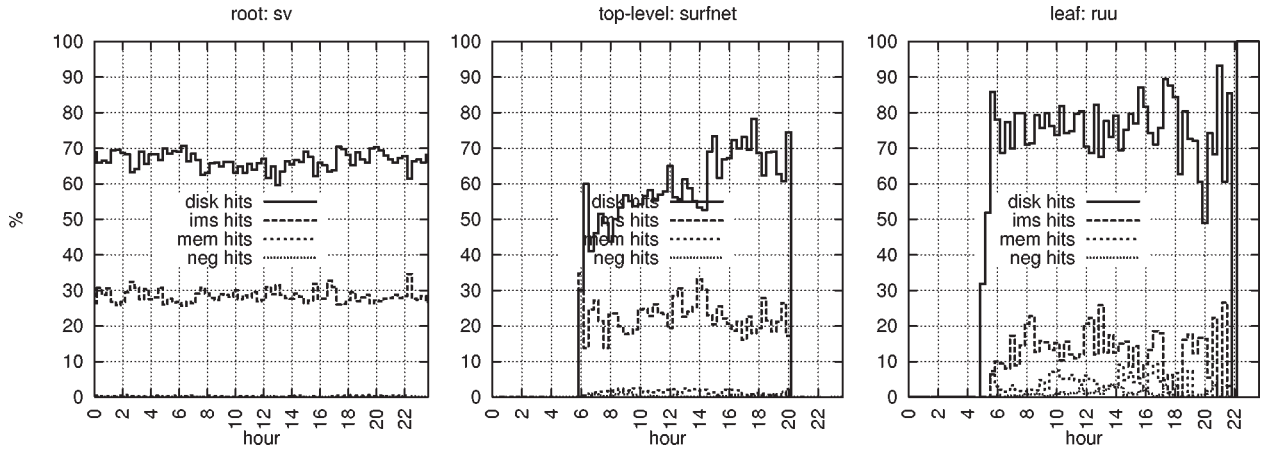

Figure 8. Hits classification.

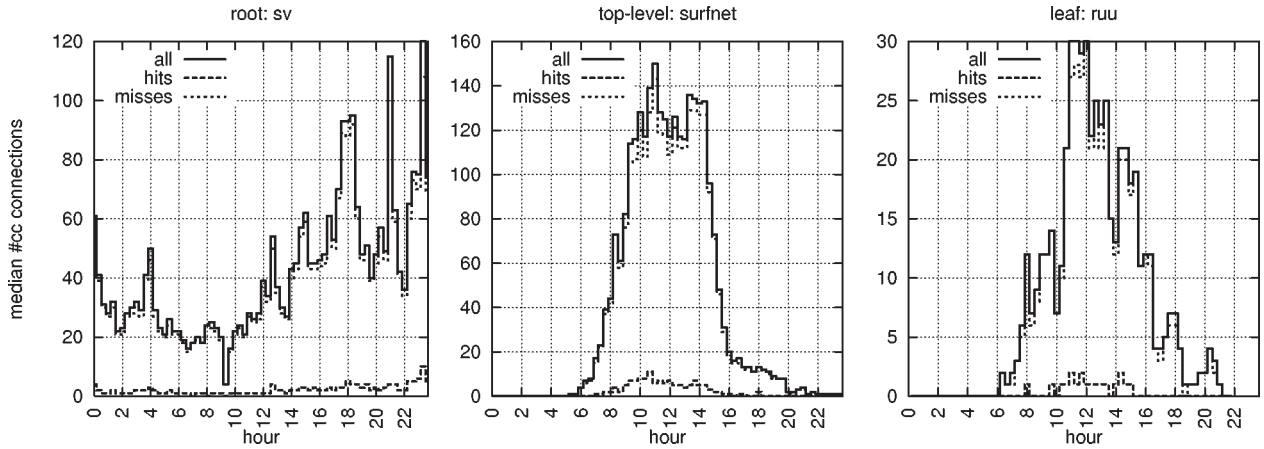

Figure 9. Concurrent outgoing connections.

Interestingly, neither ratio depends on the traffic intensity. One may expect that the number of hits increases with the number of client requests because of the constant *reference locality* principle. This pattern has been observed [Gribble and Brewer 1997] and used in some simulation studies [Cao and Irani 1997; Duska *et al.* 1997]. However, this is not the case in the studied environments. Apparently, peak hours user population has access patterns different from light load hours (or the population itself is different). This phenomenon requires further investigation.

Hit ratios may depend on the time of day which, again, implies that user population and/or access patterns vary with time. New models and further experiments are needed to study this variation.

6.3.2. Hits classification

By profiling stages of individual requests, we are able to identify several classes of hits (figure 8). For all hit classes, the original server could be contacted to check the freshness of a cached object. The algorithms that determine when freshness should be verified are often complex and based on such factors as document expiration and last modification times and even specific URLs.

Note that classes do not intersect. For example, an IMS hit is a positive reply (not a negative hit) when the document body is not retrieved from disk or memory (not a disk or memory hit). There are also some rare cases not covered by our classification.

To illustrate the relative importance of each class, we plot the percentage of *all* hits a class represents. Clearly, disk and IMS hits are responsible for more than 90% of all hits.

The average share of IMS hits is large and increases with caching hierarchy level: about 14% on leaf, 23% on top level, and 28% on root proxy. Optimizations must take into account the portion of IMS hits, and, thus, may require different techniques depending on the hierarchy level.

Low percentage of memory hits indicates that keeping documents in the "hot" memory buffer does not improve the performance of Squid. There is a belief that hits for the same object often come in "bursts". Thus, it may make sense to keep new documents in the memory buffer for a short time in case they will be requested again soon. However, our measurements (not presented here, see "memory hit analysis" in [Rousskov and Soloviev 1998]) show that the number of such hits is usually below 5% of all *memory* hits.[6] Thus, for studied hot buffer sizes (typical for

[6] Very low memory hit ratio was a surprise for many Squid developers and researchers. After completion of this study, Professor Pei Cao from the

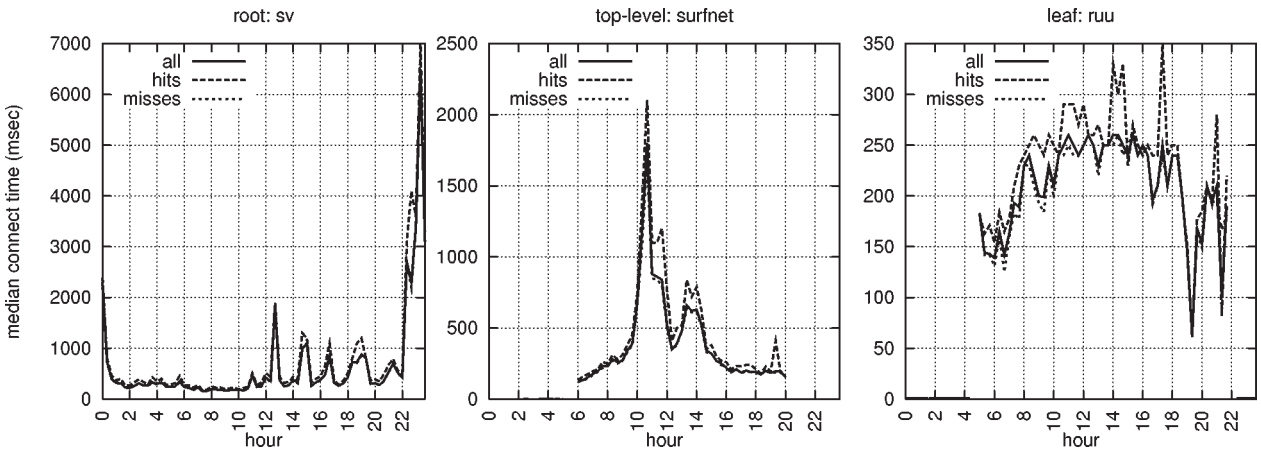

Figure 10. Proxy connect time.

proxies), new objects could be swapped to disk as soon as possible to free buffer space for incoming traffic or other alternative buffering policies should be considered [Rousskov and Soloviev 1999].

Negative caching was always considered an important feature of Squid. Our measurements show that the percentage of negative hits is small. However, invalid requests (their results are negatively cached), unlike most other requests, may have very large response times. Thus, additional experiments are needed to show if response time reduction for invalid requests is worth caching them.

6.4. Outbound network

All misses and some hits request data from other proxies or origin servers. This section studies the performance of the outbound network that has a crucial influence on request response time and proxy resources consumption.

6.4.1. Concurrent outgoing connections

The total number of outgoing concurrent connections in figure 9 follows the traffic intensity pattern. The number of outgoing connections for miss requests is close to the number of concurrent misses (figure 3). This happens because every miss uses an outgoing connection to retrieve data.

On the contrary, the number of outgoing connections for hits is noticeably lower than the number of concurrent hit requests. Several factors affect this. First, only a few hit requests require an outgoing connection to verify the freshness of a document. Second, the origin server confirms such requests with a small message without actually reading the document. Thus, network and server delays are minimal, and a fast response time for hits further decreases the number of outgoing hit connections.

6.4.2. Proxy connect time

Proxy connect time (figure 10) is the time it takes to send an HTTP request to an origin server or other proxy. Misses (lowest curve) send requests to retrieve documents. When a client requests an "old" cached object, a proxy may also send an `If-Modified-Since` request that can result in a hit (highest curve).

Note that DNS lookup activity is not included in proxy connect time. Also, for this graph, we ignore requests that did not make an outgoing connection.

Interestingly, it takes longer for a hit to send an IMS request than for a miss to send a regular request. This effect is especially visible on leaf proxies, but can be also detected on higher levels of caching hierarchy. We speculate that there may be two reasons for this anomaly.

First, hit documents may come from origin servers that are farther, on average, in network topology than servers that deliver misses. For example, a leaf proxy in Europe would retrieve a lot of hits from a few popular servers in the USA using a slow transatlantic link. On the other hand, a hit ratio for local documents may be lower due to the higher document diversity.

Second, hits are served mostly by *popular* servers. Popular servers are often overloaded and, thus, have a high response time. Misses are served, on average, from less loaded servers.

6.4.3. Server reply time

Server reply time (figure 11) is the time it takes to receive a reply from an origin server or proxy. Clearly, server reply time is susceptible to load and network congestion. Note that hit replies are transmitted much faster because they are all small IMS acknowledgment regardless of the file size.

On average, hit replies are 40% faster than misses on **sv** proxy, 60% faster on **surfnet**, and 70% on **ruu**. The difference in relative hit reply times on the three proxies is probably due to their position in the caching hierarchy (similar reasons were discussed in the previous subsection). More analysis of hierarchical relationships is needed to support this speculation.

Server reply time for hits does not change with object size. The only possible server reply for a hit is a positive IMS reply that carries no content. Thus, the length of an

University of Wisconsin-Madison and her students discovered a Squid bug that lead to inefficient buffer management.

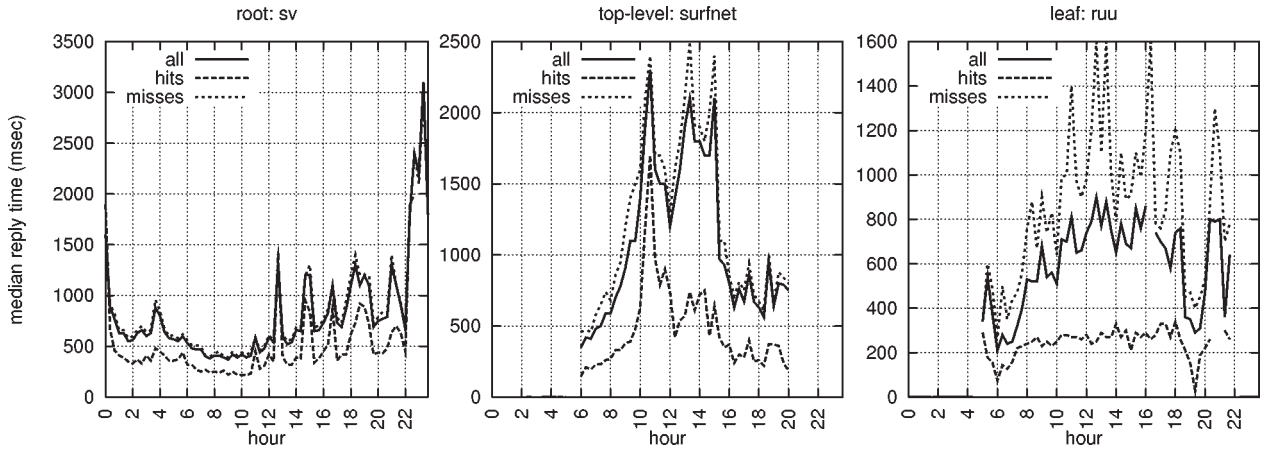

Figure 11. Server reply time.

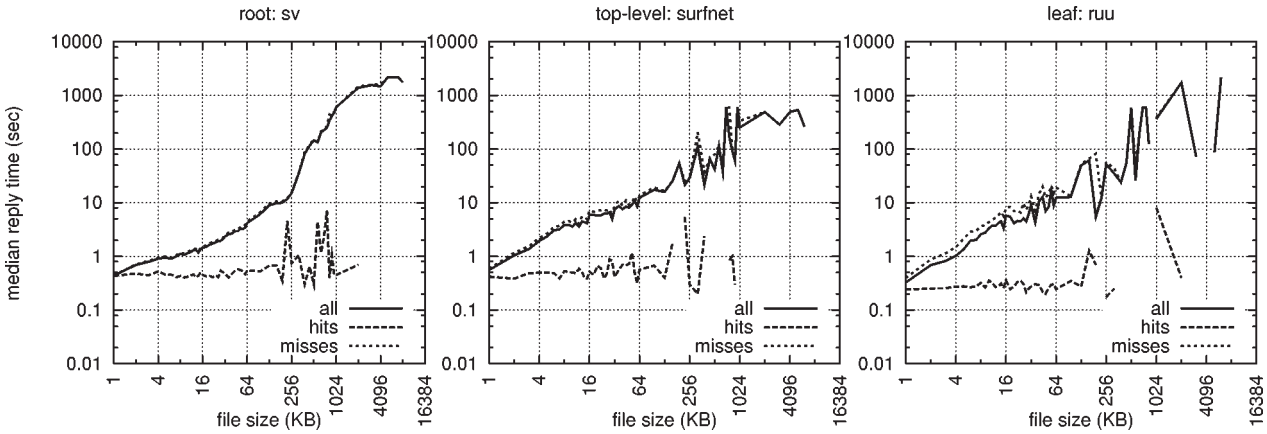

Figure 12. Server reply time vs. object size.

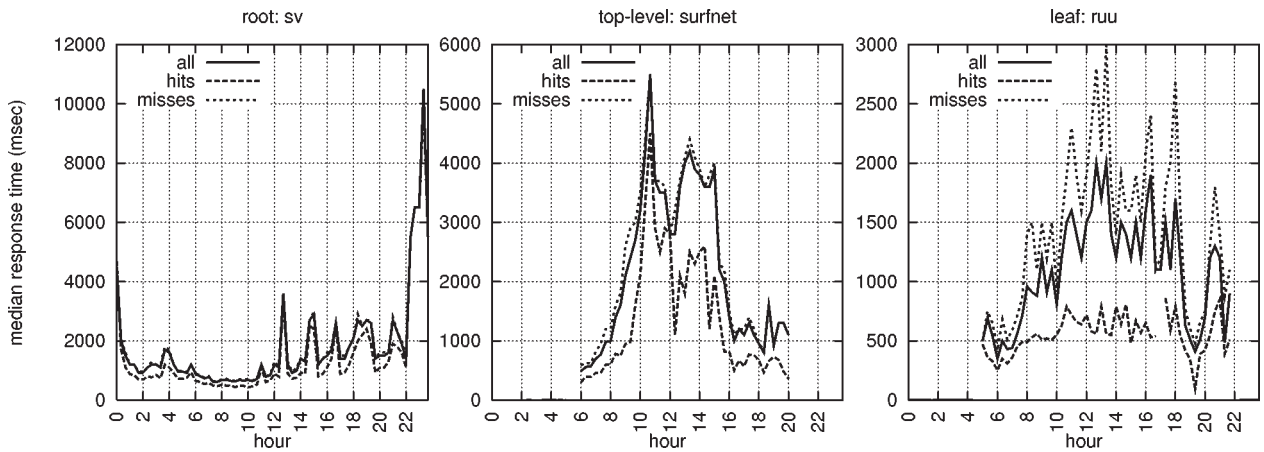

Figure 13. Server response time.

IMS reply message does not depend on the actual document size. In an ideal world, the "hit" curve in figure 12 would be a straight line on the graph.

Although not shown here, the IMS reply time is actually the same as the miss reply time for very small documents. This suggests that network congestion dominates in server reply time. We do not have detailed server-side measurements required to support this statement, but similar observations were reported elsewhere [Manley and Seltzer 1997].

6.4.4. Server response time

Hits are slower than misses during the connect phase. The opposite is true during the reply phase. Server response time, depicted in figure 13, measures the *total* network delay for communication with a server (i.e., proxy connect

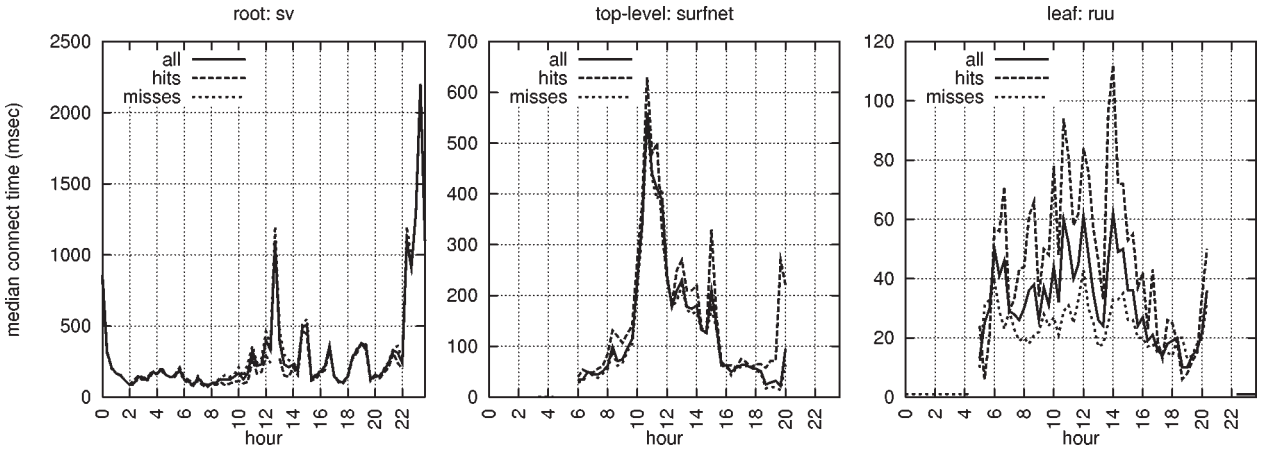

Figure 14. Client connect time.

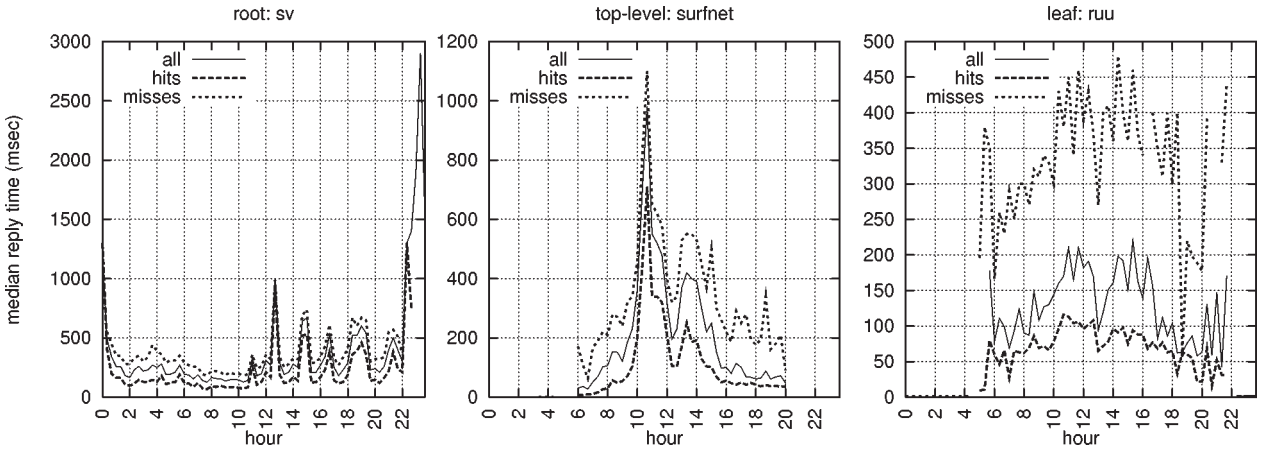

Figure 15. Proxy reply time.

time plus server reply time). In spite of a slower connect time, the total server response time for hits is lower. Thus, even when hits have to verify the freshness of an object, they save time compared to misses that have to fetch data over the network.

Another important observation is that hierarchy level changes per request resource consumption of misses and hits (proportional to the median response time). The difference in hit and miss response times on root proxy is usually less than on top-level proxy, with leaf proxy having the largest difference. Such cooperation caused variation makes it harder to optimize caching algorithms.

6.5. Inbound network

This section studies the performance of an inbound network. All clients are using the proxy's inbound network connections to request a document and receive a reply. The major function of a proxy is to decrease the total response time by reducing the number of outbound network transmissions. However, if inbound connections are congested, savings in the total response time may be marginal.

6.5.1. Client connect time

Client connect time (figure 14) is the delay from the `accept()` system call on a proxy until receiving a parseable HTTP request. The network connection remains open after the client request is received because it is reused to send the reply. Recall that we cannot measure the actual client-side delays or network card queuing time of the client request prior to the `accept()` system call.

It may seem that client connect time cannot depend on the result of the request (hit or miss) because the result is not known during connect phase. This is not the case. On all participating proxies, we have detected a small but persistent difference between client connect time for hits and misses. On most proxies, it takes longer for hits to connect. The root proxy, however, has an opposite pattern.

The longer connect time for hits can be explained by the *source* of a (future) hit request. There are two sources for hits: clients and neighbor proxies (siblings). Sibling caches, which might be farther away than normal clients, contribute more to the hit statistics than to the miss statistics: By definition, siblings only fetch hits from each other. Thus, clients connect times are shorter than connect times for hit requests originated from siblings. This effect is especially noticeable on leaf proxies where hit connect time

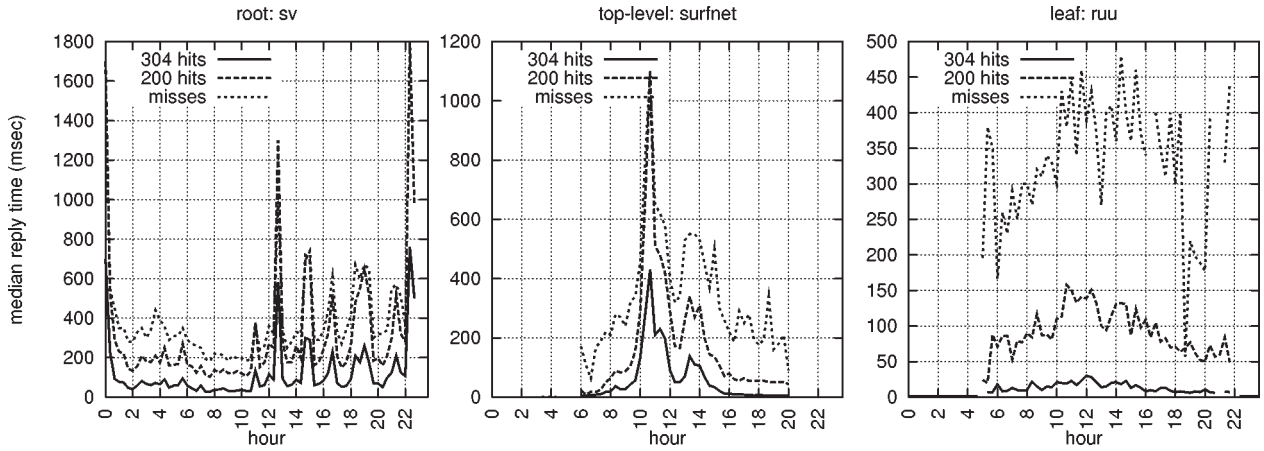

Figure 16. Proxy reply time (detailed).

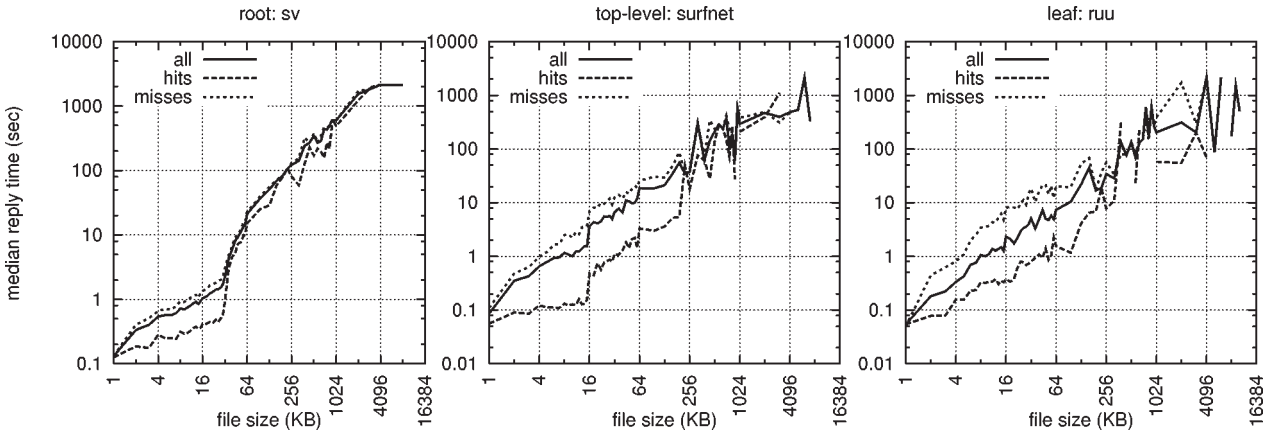

Figure 17. Proxy reply time vs. reply size.

is often twice as long as miss connect time. Additional experiments may be needed to verify this theory.

On the root proxy, the opposite holds. Major **sv** siblings are connected via fast **vBNS** links which makes the hit connect phase shorter than misses.

Client connect time is load dependent. This can be caused by outbound network congestion or Squid performance. Following plots may help in identifying the real cause.

6.5.2. Proxy reply time

Proxy reply time (figure 15) is the time it takes to send a reply to a client after the document was retrieved from the cache or another server. Note that due to pipelining, a reply process may start prior to receiving the last byte from the disk or another server.

It takes Squid less time to reply with a hit than with a miss. One apparent reason is that large percentage of hits are IMS hits that have a very small content. To further investigate proxy reply time, we isolated IMS hits from 200 hits that transfer document content in reply.

Note that proxy reply time (figure 16) varies with time. Increase in proxy reply time may be affected by three factors:

- degradation of outbound connections (slows delivery of a large miss),
- proxy performance degradation (affects misses and 200 hits),
- degradation of inbound connections (affects all replies).

The behavior of the “304 hits” line is important. Neither outbound connections nor proxy performance should affect 304 replies much. If the reply time for 304 hits goes up, then *inbound* connections are likely to be congested.

200 hits and misses depend on the performance of the proxy, and misses depend on outbound connections. Note that 200 hits and misses suffer from congestion more than 304 hits because of a larger size that may require several network I/Os. We proceed by taking reply size into account (figure 17).

6.6. Disk storage subsystem

Most hits are retrieved from the disk. The performance of the disk storage subsystem has direct impact on request response time improvement achieved by a proxy. In this section, we present disk performance measurements. Squid does not support raw disk I/O and relies on Unix file system performance.

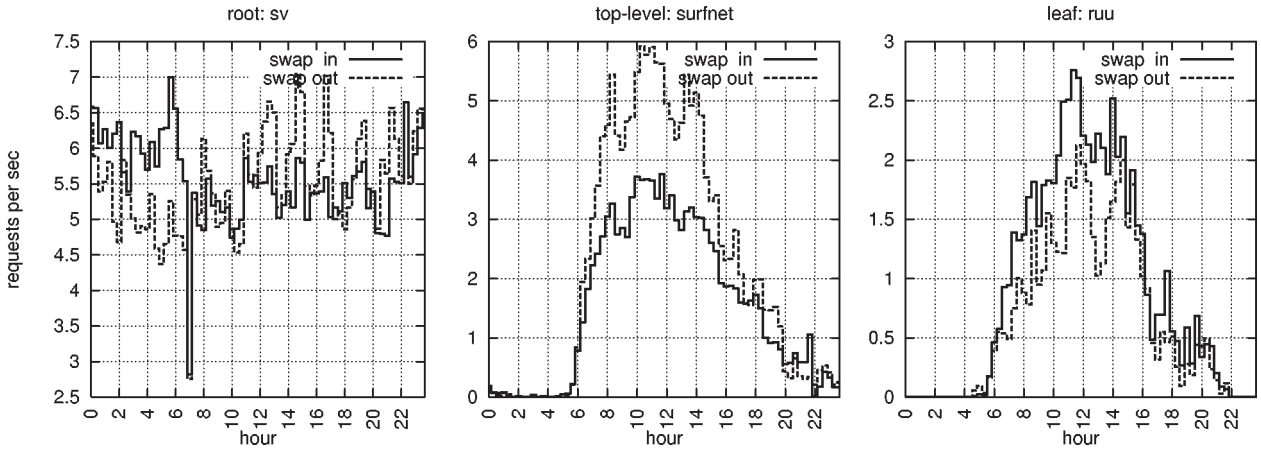

Figure 18. Disk traffic intensity.

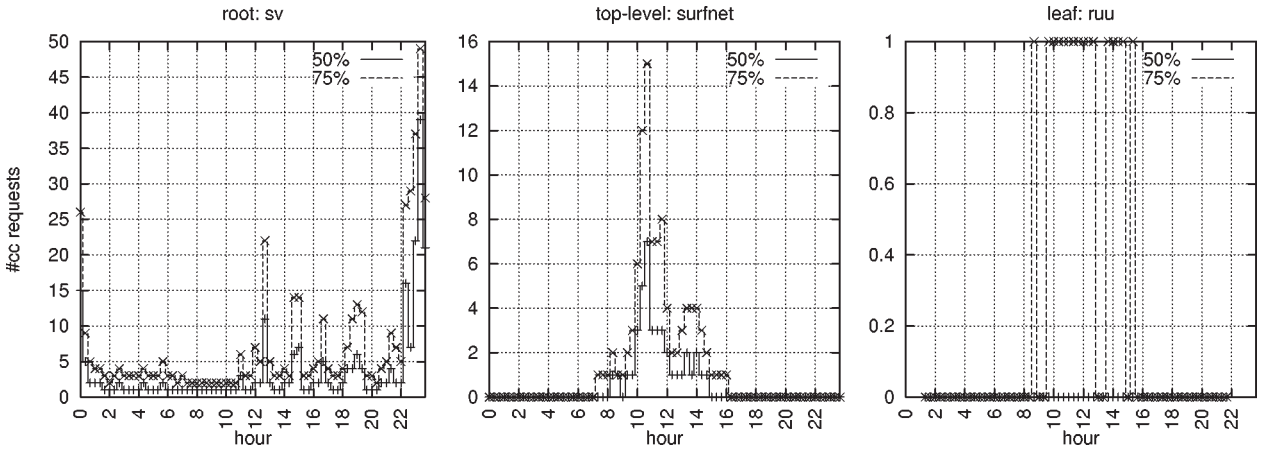

Figure 19. Concurrent disk requests.

Note that our measurements do not distinguish individual disks. Thus, the results in this section analyze the performance of the disk storage subsystem or file system *as a whole*. We will often use a shorter word "disk" to name the disk storage subsystem.

6.6.1. Disk traffic intensity

Figure 18 shows the number of swap requests per second for three proxies. In spite of hit ratios usually being lower than 50%, the rate of swap-out requests may be lower than the rate of swap-ins since not all misses are cachable. The *rate* of swap-in and swap-out requests is determined by traffic intensity, Document Hit Ratio, and caching policy.[7] The rate of swap-in requests is usually higher on leaf proxies. The opposite is true for top level caches that have lower hit rates than leaves. On the root proxy, swap-in (swap-out) requests dominate half of the day! Moreover, the *Swap-In Ratio* significantly changes with time on all participating proxies. These changes make performance tuning harder because *static* optimizations may not work.

The rate of incoming swap requests determines the stress on the disk storage subsystem. Squid treats swap requests equally regardless of their direction. However, swap-in and swap-out requests have different priorities if QoS factors and Squid memory requirements are considered. Indeed, swap-in requests contribute to the response time of hits. Thus, by reducing their response time we can improve QoS. A better response time can be achieved, for example, by giving a higher priority to swap-in requests. However, delaying swap-out requests may increase the total memory requirement because incoming documents will not be swapped out to disk. Knowing the swap request rate helps in solving this tradeoff.

6.6.2. Concurrent disk requests

Figure 19 depicts the number of concurrent swap requests present in a proxy server. We count the number of requests in the system using 2 msec intervals and calculate the median based on 20 minute grouping. Small 2 msec intervals assure that we count the number of concurrent requests rather than the total number of requests per [large] interval. Note that this is not a "disk request per second" graph.

We plot the 50th and 75th percentiles. The 50th percentile is the same as median. Our measurements cannot

[7] Note that object sizes do not affect the rate of swap requests, but may affect the number of concurrent swap-ins and swap-outs.

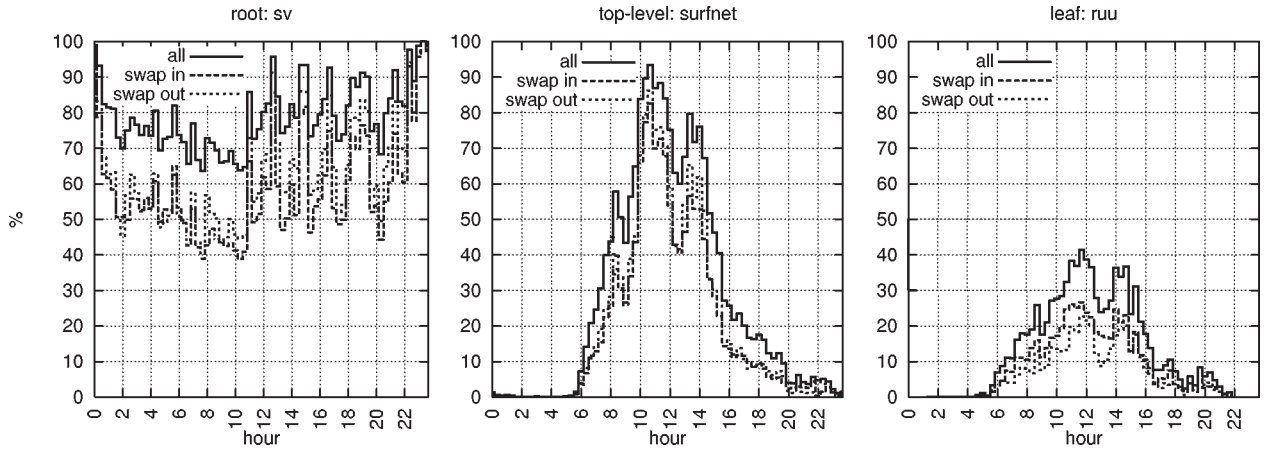

Figure 20. Disk utilization.

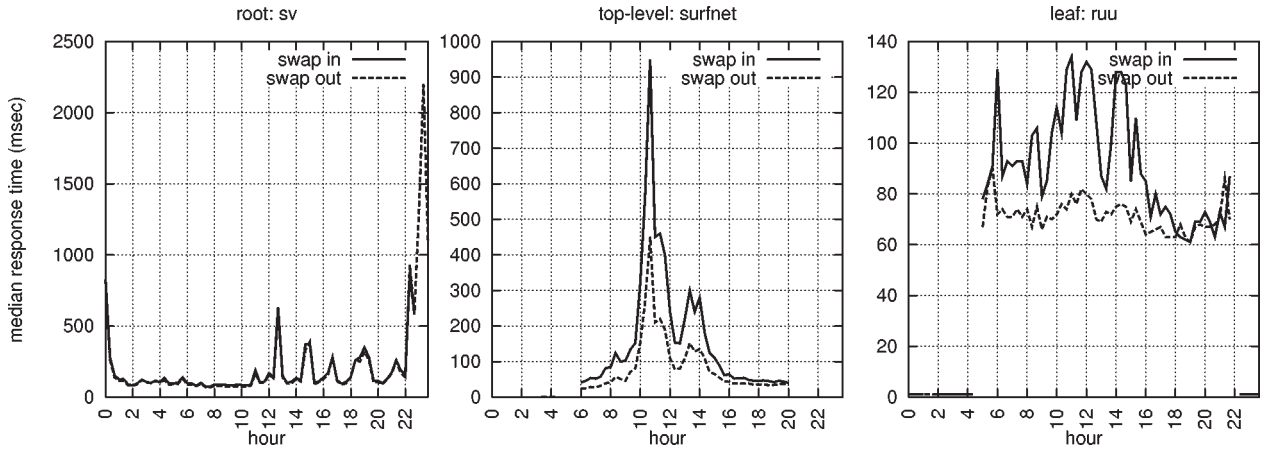

Figure 21. Disk response time.

distinguish individual disks. Thus, we actually measure the total number of concurrent requests in the disk storage subsystem. However, this number directly affects the length of queues to individual disks.

The number of concurrent swap requests increases sharply during peak load. The increase is not proportional to the incoming swap requests rate. This is a direct effect of the large queuing time of swap requests.

6.6.3. Disk utilization

Disk utilization in figure 20 is measured in the percentage of times where there was at least one active swap request. The measurements are done using 2 msec intervals. Actual disk utilization is represented by the "all" curve. Curves for swap-in and swap-out requests are given to compare the contribution of each class towards disk utilization.

Again, we do not measure *per disk* utilization. The graph represents the utilization of the disk storage subsystem as a whole. In other words, if there is always one disk I/O in the system, then utilization is 100% regardless of the number of physical disks installed.

Disk utilization often reaches 90% on root and top level proxies. Participating leaf proxies have at most 40% utilization because the incoming traffic is lighter.

6.6.4. Disk response time

Disk response time (figure 21) is the total time it takes to swap in/out a document from/into the disk cache. Note drastic increases in response time that correspond to a higher number of concurrent swap requests. These peaks sharply increase the total response time for hits.

On most participating proxies, swap-out requests were somewhat faster than swap-in requests (in spite of a larger average size). One may attribute this to file system level caching of disk write requests. An OS may postpone scheduling of a write request until the time when it can be completed faster. Such optimization is transparent to Squid and is perceived as a faster response time for writes. Of course, read requests cannot be postponed. Low level disk scheduling differs among operating systems. These differences may explain faster swap-out requests on some proxies (not shown here). However, we do not have enough data to support this claim.

6.6.5. Disk response time anatomy

To further understand the performance of the disk storage subsystem, we plot disk response time versus file size (figure 22). Squid swaps files using 8 KB I/O pages. For each swap direction, we plot the total request response time

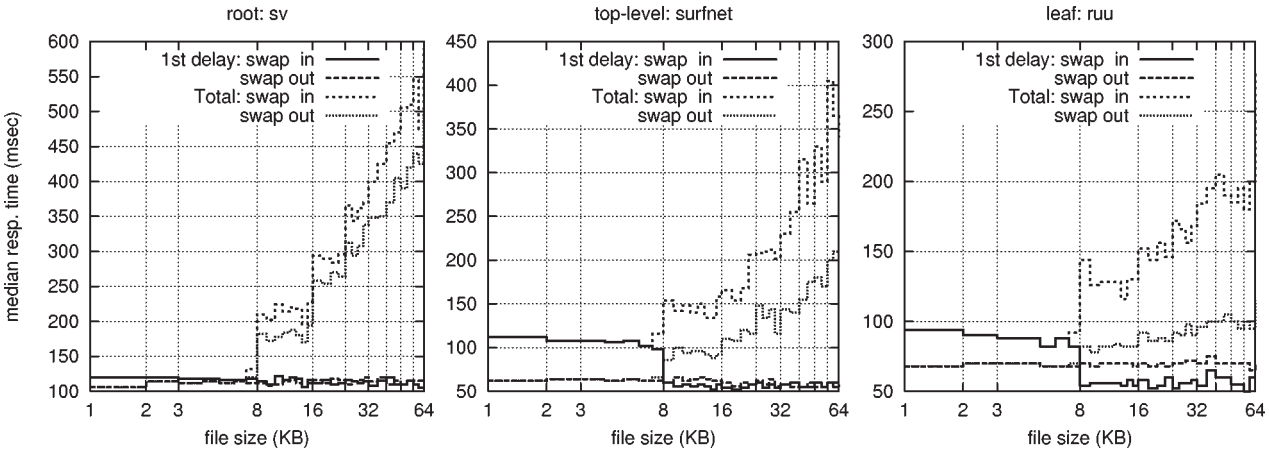

Figure 22. Disk response time anatomy.

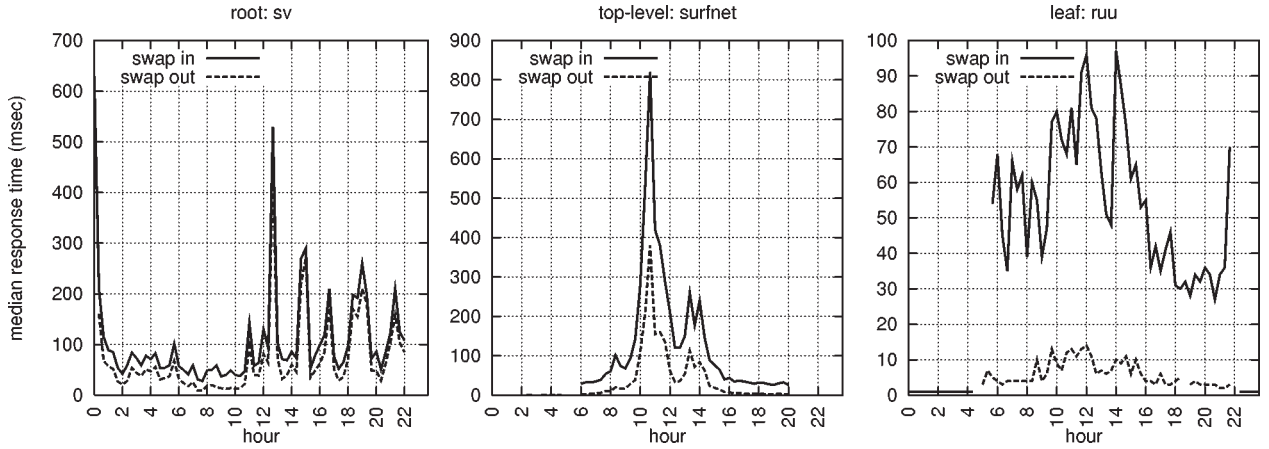

Figure 23. Second I/O duration.

and the time it takes to swap the first page. The "total" and "1st delay" curves for files smaller than 8 KB are the same.

Since various *per I/O* delays dominate the disk transfer time, the *size* of an I/O is not very important (the *number* of I/Os is). This fact explains the step-like shape of the "total" curves: the times to read files up to 8 KB are approximately the same.

The first delay must be the same for any file size. However, when we first ran our patch, we discovered that it was not the case. The first delay *dropped* for **surfnet** and **ruu** proxies. Our first suspicion was that our measurements were incorrect. After many hours of searching for a bug, we found it in ... Squid. Squid was doing one extra I/O for the *last* page of every document. Thus, the first delay for multi-page documents did not include this extra I/O and had smaller response times. After our bug report [Rousskov 1997], the bug was fixed in Squid version 1.1.17. All participating proxies but **surfnet** and **ruu** fixed the bug before running our experiments.[8]

The first disk delay always includes OS overhead on opening a file. Consecutive I/Os for the same file (if any) do not have this overhead. Thus, one may compute both I/O duration and OS overhead using the following simple model (*assuming* that all overheads do not depend on the file size for small files):

$$\begin{aligned} \text{1st\_Delay} &= \text{Overhead} + \text{I/O}, \\ \text{Total (8 KB)} &= \text{1st\_Delay}, \\ \text{Total (16 KB)} &= \text{1st\_Delay} + \text{I/O}, \\ &\Downarrow \\ \text{I/O} &= \text{Total (16 KB)} - \text{Total (8 KB)}, \\ \text{Overhead} &= \text{Total (8 KB)} - \text{I/O}. \end{aligned}$$

The model is based on very reasonable assumptions. However, we have discovered that, for many proxies, the model does not have a positive solution for the *overhead* component (i.e., computed overhead is negative). One of the main reasons for our model failure is probably *queuing delays* for individual I/Os in Squid. Squid attempts to process all ready swap requests in an interleaved fashion, one I/O page of one swap request at a time. Thus, when the number of ready requests is large, a single disk request may wait for all others to be completed. The number of

[8] Our measurements on **sv** proxy (not presented in this study) show that the influence of that bug on disk response was insignificant, probably because the extra I/O was not going all the way to the disk but was intercepted by the OS on EOF condition.

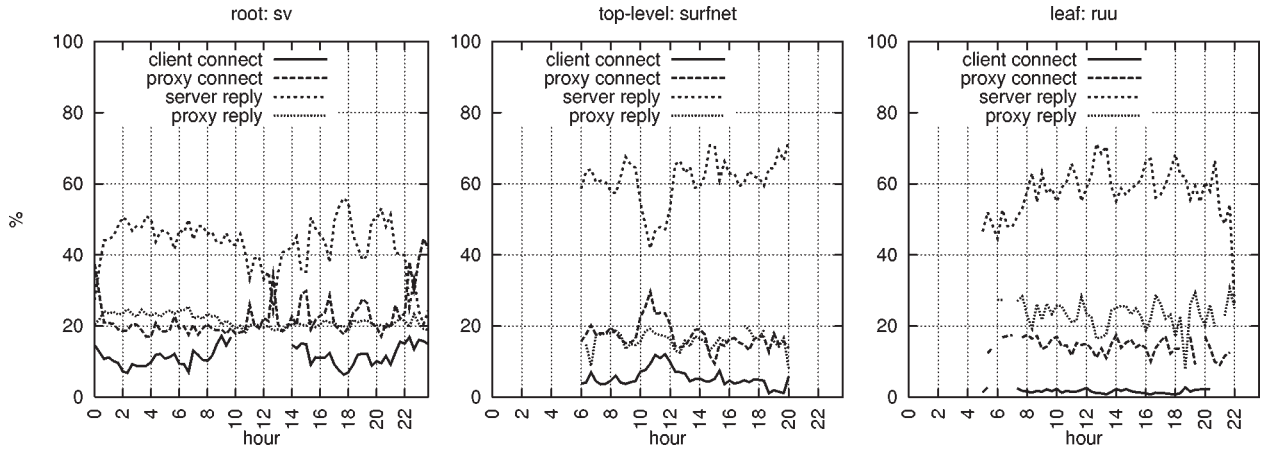

Figure 24. Proxy response time components (misses).

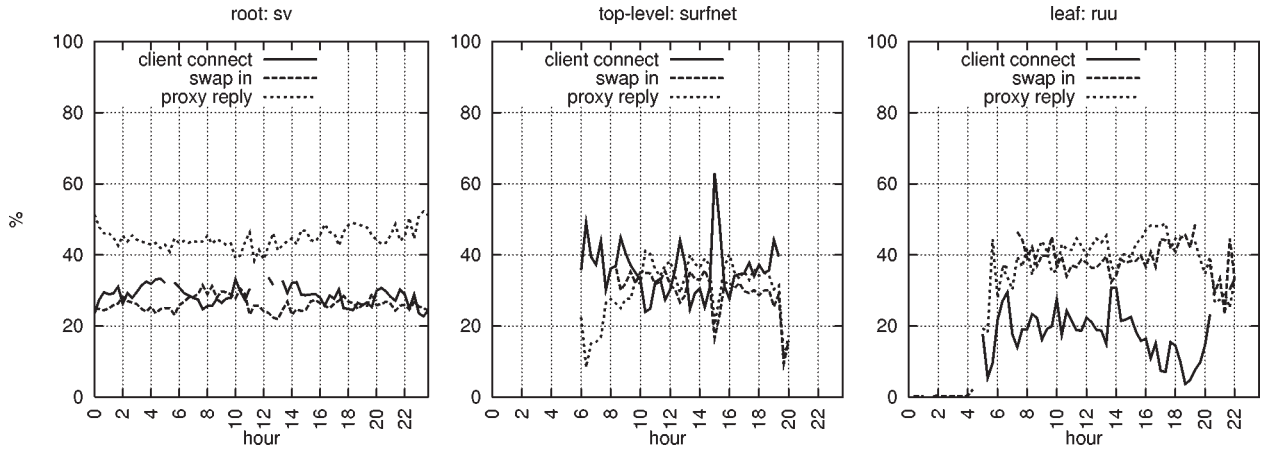

Figure 25. Proxy response time components (hits).

ready requests is very unstable, and large "random" delays it causes may affect our model. We are currently working on other ways of OS overhead estimation.

One way of detecting Squid queuing delays is to analyze the second I/O time for swap requests. Note that the second I/O does not include OS overhead for opening a file but includes Squid queuing delays. We measure the duration of the first I/O and total response time. For two page files (16 KB), the difference between the two gives the second I/O duration.

The duration of the second I/O on figure 23 experiences the same peaks as the total swap response time. This observation supports the hypothesis that Squid queuing delays dominate disk response time during peak loads.

Squid enqueues a disk request *after* opening a file. Thus, one page documents are served in two steps. First, a file is opened, and a page is scheduled for an I/O. Second, an actual I/O is performed. Each stage may experience a queuing delay. Thus, even one page documents may suffer from *two* queuing delays.

There is usually about 80% of one page swap requests. Squid was designed to handle multi-page documents in a fair fashion: Every swap request is split into several small steps, and steps execution is interleaved. We have demonstrated that this interleaving may lead to huge extra queuing delays for one page documents. Using measurements collected during this study, we currently investigate ways of improving disk response time while preserving fairness.

6.7. Proxy response time components

We conclude our study by analyzing the relative impact of request processing stages. Such analysis is essential for performance optimization since it helps in identifying performance bottlenecks.

For misses (figure 24), we distinguish four major stages: *client connect*, *proxy connect*, *server reply*, and *proxy reply*. The graphs show the relative contribution of each stage towards total delay. Total delay is calculated as a sum of all stages.[9]

For performance optimization, both the *absolute level* of contribution and its *change* with time are important. The former tells us what contributes to the total delay the most. The latter may help in identifying performance problems that arise during peak load. For example, the server reply component dominates in miss response time (50–60%).

[9] Median total delay may differ from median response time because of pipelining and such non-accounted activities as DNS lookups.

However, its relative contribution *decreases* during peak loads. On **sv**, the server reply time component becomes even less important than the proxy connect stage (which sharply increases its relative value during peak loads). In other words, it actually takes longer to send a small request to a server than to receive a (potentially large) reply.

The contribution of client connect stage is usually 10% or less, but it increases by more than twice during peak load. Interestingly, proxy reply time is responsible for the same portion of the total delay regardless of the load.

Our analysis implies that connect times are most susceptible to traffic intensity. Using *persistent connections* (HTTP/1.1) is a promising way to reduce the impact of load on response time.[10]

For hits (figure 25), we distinguish three major stages: *client connect*, *swap-in*, and *proxy reply*. For the purpose of this analysis, we consider 200 hits only. We also ignore communication with other servers because the majority of 200 hits (70–80%) do not verify their freshness.

The curves for hits are more stable than for misses. On **sv** proxy reply time dominates, and swap in stage takes approximately the same time as client connect. On **surfnet**, all three components are equally important. Client connect time is the least stable component for both proxies. This pattern is consistent with misses.

An important observation is that disk delays contribute about 30% towards the total response time. Network performance is often beyond control of a caching proxy. On the contrary, disk I/O performance is isolated from external factors and can be optimized.

Note that we cannot account for pipelining effects that may change relative contributions for large requests. However, more than 80% of requests cannot be pipelined due to small document sizes. We believe that our estimations are close to actual performance.

7. Shortcomings of this study

It must be noted that the performance data collected is based on the HTTP/1.0 traffic and HTTP/1.0 proxy. HTTP/1.1 has several important features that are designed specifically to improve client-proxy interaction. Persistent connections, request pipelining, explicit caching control, and other additions will improve proxy performance when implemented and used widely. However, many observations and conclusions in this paper do not depend on the connection model. Moreover, the actual deployment of HTTP/1.1 takes time. For example, only about 20–25% of connections to NLANR caches are persistent.[11] Up-to-date information about HTTP deployment is available from the W3C Web Characterization Activity site [WWW Consortium 1998].

Besides locating volunteers for this study, our major problem was a "one shot" approach during trace collection. For various reasons that we could not control, it was impossible to run more than one or two experiments on most of participating proxies. Thus, we had to *predict* future points of interest and program profiling patch accordingly. We did manage to get a very interesting and informative sample. However, several important measurements were not performed or not properly recorded. They include:

- OS level measurements reported by such widely used Unix tools as *netstat*, *vmstat*, *iostat*, etc. TCP stack performance analyzed with *tcpdump* or similar software.
- ICP traffic measurements [Wessels and Claffy 1997]. Clearly, ICP overheads may significantly affect network traffic and delays.
- DNS lookup delays.
- Squid internal scheduling delays such as "select loop" delay.

We consider the following research areas very important, but out of the scope of this study:

- Comparison of direct traffic delays with those of proxied traffic.
- Analysis of proxy software other than Squid and non-Unix operating systems.
- Long term analysis of historical trends and so on.

We encourage other researches in the area of caching to develop these topics for a complete picture and answer a few questions we failed to answer or confirm our speculations.

Finally, the volume of collected information and number of possible comparisons made any significant manual analysis impractical. We had to rely on our automated tools to collect statistics and plot the graphs. Clearly, in some cases, the presentation quality suffered.

8. Related work

Caching in general has been studied intensively in operating systems and databases. Complex hierarchical caches can be found in any modern CPU, and good performance of a database engine is impossible without a tuned buffer manager. However, attempts to migrate classic caching techniques to Web applications usually fail. Specifics of Web traffic are well known. Highly variable object sizes, delay sensitivity, highly skewed popularity distributions with heavy tails, short and long range dependencies, huge number of objects, etc., created challenges that traditional algorithms could not overcome. For example, the classic LRU policy that works well in most database applications is a poor choice for a Web cache because LRU does not account for object sizes (LRU-threshold should be used). A lot of other examples of inapplicability of traditional caching techniques can be found elsewhere.

Web caching research has concentrated primarily on caching policies and caching hierarchies [Abrams *et al.*

[10] The current Squid versions (Squid 2.x) support persistent connections.
[11] Measured with Squid 2.x, after this study was completed.

Table 4
Comparison with the study at DEC.

This study	Study at DEC
Per request measurements We use detailed *per request* measurements of network and disk delays. This allows for performance analysis based on request categories (e.g., misses, IMS hits, or small swap-in requests). We have demonstrated that request classes significantly differ in their performance characteristics. Aggregate statistics are often meaningless and can even be misleading.	**System state snapshots** The authors effectively utilized several Unix tools to collect *snapshots* of system activities. These snapshots contain important information on low level system performance that is not available in Squid. For example, actual network queues and raw disk delays can be measured.
Our approach lacks low level OS measurements though. It is often desirable to combine Squid level measurements with OS statistics to describe certain effects precisely. Organizational reasons (our patch was run in environments we have no control over) prevented us from performing such measurements.	A drawback of this approach is that connection between Squid actions and system performance is often lost. This loss, on top of the rapidly changing system states, makes it very hard to summarize the collected data in a meaningful form. The results may look messy and inconclusive.
Fixed caching software (Squid); Variety of environments We concentrated our research on one state-of-the-art proxy in various environments. It was essential for us to show that most of our observations hold regardless of proxy hardware, operating system, configuration, etc. Also, note that Squid software is extremely popular (e.g., more than 70% of European caching proxies surveyed were running Squid [European Caching Task Force 1996])	**Squid and CERN proxies; Fixed environment** The authors studied two proxies in a fixed environment to test alternative design decisions. They were able to show several important differences in performance caused by proxies' architecture. However, it is not clear how applicable their findings will be to other environments, especially to other levels of hierarchy.
Cooperative caches, several hierarchy levels All studied proxies were actively involved in cooperation with other caches. This cooperation is typical for modern proxies. We have demonstrated that cooperation does affect proxy performance. Participating proxies represented all levels of caching hierarchy.	**Single isolated cache** The authors concentrated on the performance of a leaf proxy. They did not study the effects of hierarchical caching.
Compact time range We base our analysis on measurements collected within a short time interval (about 5 weeks). Compact time range makes comparison between different proxies more realistic. However, we did not analyze trends in Web traffic and their impact on proxy performance.	**Six months time range** Web traffic handled by the studied proxy almost doubled from the time the CERN proxy was tested to the last experiments with Squid. This makes the performance comparison questionable. On the other hand, the long time range allowed for interesting observations based on historical trends.

1995; Bestavros *et al.* 1995; Cao and Irani 1997; Rizzo and Vicisano 1998]. Surprisingly little attention was paid to performance analysis. Most of the work in this area is currently done by administrators of large caching proxies that study their systems in order to detect potential problems and improve performance. However, these efforts are badly documented, and optimization recommendations are often based on anecdotal evidence.

One of the pioneer works in performance analysis of caching proxies was benchmarking of Harvest Object Cache [Chankhunthod *et al.* 1996]. The authors used a simulated workload generated by several local clients to compare Harvest performance with CERN. This work contains many important insights into Harvest architecture. However, performance measurements were made in a small, simulated environment.

The most related research publication is a performance study of two caching proxies conducted on Digital's Palo Alto Gateway [Maltzahn and Richardson 1997]. The authors analyzed performance of CERN and Squid proxies on real workloads. They measured proxy resource consumptions using *snapshots* of system activities. Our study is orthogonal to that work (table 4).

There were also several studies on the performance on origin Web servers [Luotonen *et al.* 1996; Tatarinov *et al.* 1997]. However, the results obtained on a Web server cannot be directly applied to a proxy. Modern Web servers rely mostly on the file system buffers to keep popular documents in memory (e.g., using `mmap()` interface). This approach is not applicable to caching proxies that cache most of the traffic on disk. However, a few CPU and network related observations (e.g., single process design versus a forked process per request approach) are relevant for proxy performance optimization.

A recent development in Web caching is the introduction of performance benchmarks. Major caching vendors and research groups are working on testing existing caching proxies [Almeida and Cao 1998; Rousskov and Wessels 1998]. There are all reasons to believe that interesting and powerful benchmarks will soon become standard tools in Web caching research.

9. Conclusions

We have presented a performance study of Squid caching proxy. Our approach, based on measuring *per request* network and disk activities, allowed for an in-depth analysis of major proxy subsystems. By carefully classifying requests, we were able to identify and quantify degradation of network and disk storage subsystems during high load periods. Many common performance patterns were detected across various proxy environments. By studying proxies on different levels of caching hierarchy, we analyzed the impact

of cooperative caching on proxies. Our data and analysis are essential in understanding, modeling, and enhancing performance of a proxy server.

We believe that per request profiling tools should be incorporated in caching proxies because they provide developers with essential information that is not available by other means. In fact, the success of our project was based, in part, on the lack of detailed performance statistics for Squid: The vacuum in this important area encouraged cache administrators to participate in our experiments.

Several performance observations presented in this paper suggest significant improvements in Squid architecture. We are investigating alternatives in this direction [Rousskov and Soloviev 1999]. We are also considering studying the impact of a large number of ICP requests on cooperative proxies (ICP requests were ignored for the purpose of this study).

Acknowledgements

We are very thankful to all cache admins who managed to squeeze time to run our experiments on their proxies *and* discuss the results: Henny Bekker (Utrecht University, the Netherlands), Edwin Culp (MexCom, Mexico), Brian Denehy (ADFA, Australia), Lars Slettjord (University of Tromsö, Norway), Ton Verschuren (SURFNet, the Netherlands), and Duane Wessels (NLANR, the USA). The discussions with them were very helpful. Special thanks to Duane Wessels and K. Claffy for commenting on drafts of this paper.

We encourage cache administrators to actively participate in the research experiments like this study. Our experience shows that such collaboration benefits both researchers and practitioners. Together, we can make the Internet a better place to live.

References

Abrams, M., C.R. Standridge, G. Abdulla, S. Williams, and E.A. Fox (1995), "Caching Proxies: Limitations and Potentials," In *Proceedings of the Fourth International WWW Conference*, Boston, MA.
`http://www.w3j.com/1/abrams.155/paper/155.html`

Almeida, J. and P. Cao (1998), "Wisconsin Proxy Benchmark."
`http://www.cs.wisc.edu/~cao/wpb1.0.html`

Arlitt, M. and C. Williamson (1996), "Web Server Workload Characterization: The Search for Invariants," In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems.*
`ftp://ftp.cs.usask.ca/pub/discus/paper.96-3.ps.Z`

Banga, G. and J.C. Mogul (1998), "Scalable Kernel Performance for Internet Servers under Realistic Loads," Technical Report TR-98-6, Digital Equipment Corporation.
`http://www.research.digital.com/wrl/techreports/abstracts/98.6.html`

Barford, P. and M. Crovella (1998), "Generating Representative Web Workloads for Network and Server Performance Evaluation," In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, Madison, WI.

Bestavros, A., R. Carter, M. Crovella *et al.* (1995), "Application Level Document Caching in the Internet," In *The Second International Workshop on Services in Distributed and Networked Environments*.
`http://www.cs.bu.edu/~best/res/papers/sdne95.ps`

Cao, P. and S. Irani (1997), "Cost-Aware WWW Proxy Caching Algorithms," In *Proceedings of the USENIX Symposium on Internet Technology and Systems*.
`http://www.cs.wisc.edu/~cao/publications.html`

Chankhunthod, A., P. Danzig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell (1996), "A Hierarchical Internet Object Cache," In *Proceedings of the USENIX Technical Conference*, San Diego, CA.
`http://excalibur.usc.edu/cache-html/cache.html`

Duska, B.M., D. Marwood, and M.J. Feely (1997), "The Measured Access Characteristics of WWW Client Proxy Caches," In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*.
`http://www.cs.ubc.ca/spider/marwood/Projects/SPA/`

European Caching Task Force (1996), "Survey Results."
`http://w3cache.icm.edu.pl/survey/results/`

Fielding, R., J. Gettys, J.C. Mogul *et al.* (1998), "Hypertext Transfer Protocol – HTTP/1.1," Internet Draft.
`draft-ietf-http-v11-spec-rev-06`

Gribble, S.D. and E.A. Brewer (1997), "System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace," In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*.
`http://www.cs.berkeley.edu/~gribble/papers/sys_trace.ps.gz`

Luotonen, A., H.F. Nielsen, and T. Berners-Lee (1996), "CERN httpd 3.0A."
`http://www.w3.org/pub/WWW/Daemon/`

Maltzahn, C. and K.J. Richardson (1997), "Performance Issues of Enterprise Level Web Proxies," In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, ACM Press, Seattle, WA.
`http://www.cs.Colorado.edu/carlosm/sigmetrics.ps.gz`

Manley, S. and M. Seltzer (1997), "Web Facts and Fantasy," In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*.
`http://www.eecs.harvard.edu/~margo/papers/`

Rizzo, L. and L. Vicisano (1998), "Replacement policies for a proxy cache," Technical Report RN/98/13, Department of Computer Science, University College London.
`http://www.iet.unipi.it/~luigi/research.html`

Rousskov, A. (1997), "Performance Profiling Patch."
`http://www.cs.ndsu.nodak.edu/~rousskov/research/cache/squid/profiling/`

Rousskov, A. and V. Soloviev (1997), "Static Caching for Proxy Servers," In *Second Web Caching Workshop*.
`http://ircache.nlanr.net/Cache/Workshop97/`

Rousskov, A. and V. Soloviev (1998), "Squid Profiling Statistics."
`http://www.cs.ndsu.nodak.edu/~rousskov/research/cache/squid/profiling/stats/`

Rousskov, A. and V. Soloviev (1999), "Caching Policies for Reducing Disk I/O," In *Communication Networks and Distributed Systems Modeling and Simulation Conference*.
`http://www.cs.ndsu.nodak.edu/~research/cache/papers/save_io/`

Rousskov, A. and D. Wessels (1998), "Web Polygraph – A High-Performance Proxy Benchmark."
`http://ircache.nlanr.net/Polygraph/`

Tatarinov, I., A. Rousskov, and V. Soloviev (1997), "Static Caching in Web Servers," In *Proceedings of the IEEE Conference on Computer Communications and Networks*.

Tewari, R., M. Hahlin, H.M. Vin, and J.S. Kay (1998), "Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet," Technical Report TR98-04, Department of Computer Science, University of Texas at Austin.

http://www.cs.utexas.edu/users/tewari/papers.html

Wessels, D. (1998), "Squid Internet Object Cache Documentation." http://squid.nlanr.net/Squid/

Wessels, D. and K. Claffy (1997), "Internet Cache Protocol (ICP), Version 2," RFC 2186. http://squid.nlanr.net/Squid/rfc2186.txt

WWW Consortium (1998), "Web Characterization Activity Site." http://www.w3.org/WCA/