SBFilter : A Fast URL Filter Engine for Internet Access Management

Kartik Gopalan and Tzi-cker Chiueh
Computer Science Department
State University of New York at Stony Brook
Stony Brook, NY 11794-4400
{kartik, chiueh}@cs.sunysb.edu

Abstract:

*Internet access management refers to the capability to control the accesses to selected Web sites according to a pre-defined policy. Access requests to web sites or files other than what is explicitly allowed are denied. In this paper, we describe the design and implementation of **SBFilter**, a fast URL filter engine used for controlling access to internet resources. The key feature of **SBFilter** is that it supports a fine-grained access control based on the source host addresses and/or subnets, and the destination Web sites' file system sub-directories. Therefore, SBFilter is particularly useful for Internet Service Provider's (ISP) main proxy servers, where individual subscribers may choose different access control policies tailored to their own needs. To reduce the performance overhead associated with fined-grained access control, **SBFilter** uses an intelligent two-level caching scheme to speed up URL request filtering. The access control is transparent to client machines and adds little to the proxy server's overall performance overhead. SBFilter can be operated in a two modes - coulped and decoupled - depending on whether the proxy server and SBFilter run on the same or different machines. The two modes provide a convenient method of controlling the tradeoff between flexibility and performance of the filter engine.*

Keywords : Filtering, URL, access control, World Wide Web, proxy, caching, web server.

1. Introduction

A critical software technology to the effective use of the Web as a productivity enhancing tool is an access filter that focuses users to the part of the Web space that is both high quality and relevant to the context in question. Such a filter mechanism is useful in classroom settings to actively direct students' web accesses to the set of pages custom built for a given subject area, or passively prevent students from accessing improper web sites. In corporate environments, such a filter tool is essential to strike a balance between exploiting the wealth of information over the Web to facilitate business processes and losing productivity due to aimless or personal browsing [4].

Accesses to the global Internet usually go through a proxy web server which sits between the client machines within a subnet and the web sites outside. The proxy server forwards the URL requests from client machines to external web sites, receives their responses and hands them back to its client machines. Thus the proxy server provides an ideal location where web requests can be examined to determine if a particular request should be allowed access to the target web sites [5]. Earlier solutions assumed that the proxy server institutes a single access control policy that all requests have to observe. While this approach may be acceptable for a corporate proxy server, it is too restrictive for ISPs, because they are serving a large number of distinct subscribers, each potentially with a different access

control requirement. It is conceivable that ultimately each household that accesses the Internet through an ISP may require a customized access control policy for each member in the family.

This paper describes the design, implementation, and evaluation of **SBFilter**, one of the first URL access filter that supports fine-grained access control. Given the URL of a web access, SBFilter checks whether the client machine is on the Allow list and/or Disallow list of the URL. Only when the client is in the Allow list and not in the Disallow list will the filter let the access through. In all other cases, the access is denied. Typically each entry of the Allow and Disallow list is a client IP address. The semantics is that all accesses from clients in the Allow (Disallow list) of the target URL are allowed (disallowed). Because each entry is specified in terms of URL, this filter model can apply to email and FTP as well. A simple generalization of this basic filter model is that each entity in the external network can be associated with different Allow and Disallow lists, and the associations as well as the Allow and Disallow lists themselves can be modified at run time by authorized personnel.

To be effective, SBFilter has to be able to intercept all web requests from the machines whose access to the Internet is to be managed. A natural choice is the Internet access provider (ISP) to which an education institution or a corporate organization is connected. When web accesses arrive at the ISP, SBFilter consults with the accessed URL's associated Allow and Disallow lists using the client's IP address and determines whether the accesses should be denied or not. Given the industry trend that individual ISPs strive to differentiate market segments by offering special value-added services, such an access management feature appears to be a crucial ingredient to the service portfolio of future ISPs.

An important design goal of Internet access management systems is transparency, i.e., the existence of access filters should be completely invisible to end users. This calls for an efficient filter engine architecture that can make the admission/rejection decision for each incoming web access without increasing user perceived delays. SBFilter is both fast and scalable, and is sufficiently modular that it could inter-operate with existing proxy servers with minimal modifications to provide internet access management. SBFilter improves the request filtering performance through *filter caching* and through multiple filter processes running on one or multiple processors.

SBFilter successfully strikes a good balance between convenience and freedom of speech and information. Arbitrarily imposing a global filtering rule set upon all the customers of an ISP may be unconstitutional since it may infringe upon the right to information of certain customers. SBFilter provides a convenient tool with which to regulate internet access on a basis which is tailored to individual's right to information. Of course, the ISPs that use SBFilter to provide access filtering service can always provide a default Allow and Disallow list to customers that choose not to be involved in customizing filter rules.

2.Related work

Various commercial filtering products abound. Most of these rely on a global access control list of undesirable web sites that gets updated periodically. These filtering products run on the client machine and block access to the undesirable sites specified in the access control list. For instance CyberPatrol[3] and SurfWatch[9] utilize this method. However, the use of a global access control list might is not suitable where the access control needs to be tailored more to specific user requirements.

NetNanny [7] screens all web sites, newsgroups, and text messages that are accessed from a machine. It screens not only material coming from the Internet, but also information the client machine sends back

to the Internet. However, this approach is not suitable for a corporate environment which has multiple client machines trying to access the Internet since one would have to individually configure and maintain each machine. Network Safety Inc. [8] and VicomSoft [10] provide products for group based access control for fixed durations. Site accesses can be blocked for an entire LAN or a group of users. However, the access control lists are global.

Another approach to provide internet access control is through rating of various sites [11]. This approach helps administrators in assessing the 'goodness' of a site. However opinions about effective ratings may differ from one group to another. [6] describes a technique for customizing the ratings of sites.

Another very important design issue of access request filtering is the increase in the user perceived delay, especially when the number and variety of filter rules increase. To the best of our knowledge, none of the above works report performance data on their filter design in the literature.

3. Design of SBFilter

In this section, we describe the design of SBFilter, which runs on an off-the-shelf PC and can be easily integrated with the proxy server. The first design goal of SBFilter design is to support fast URL filtering that does not become a bottleneck for the proxy server performance. The second goal is to make the functioning of the filter engine independent from that of the proxy server to the maximum possible extent. Thus the same filter engine can work with different proxy servers with very minor modifications to the proxy server code. Thirdly, the filter engine should allow administrators to specify filtering rules that have fine grained access control based on the sources and targets of URL requests. Finally, the existence of the filter engine should be completely transparent to end users.

3.1. Software Architecture

To achieve the goal of keeping SBFilter separate from the proxy server, we implement the filter as an independent set of processes. The organization is shown in Figure 1.

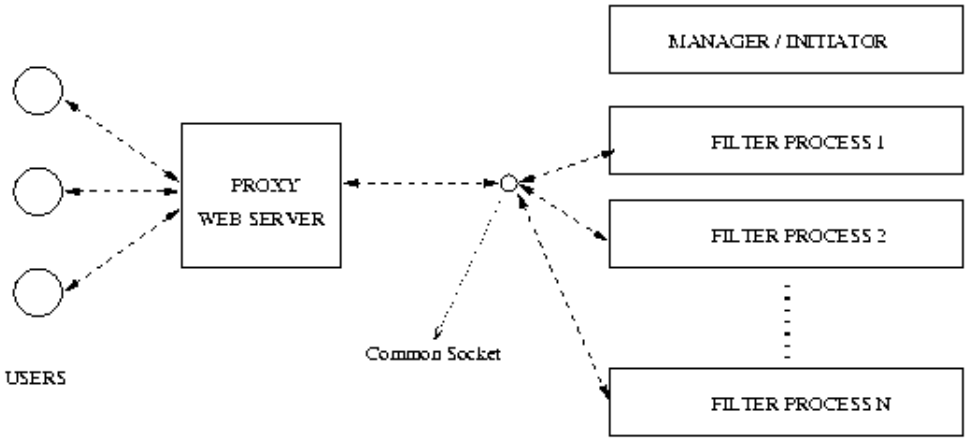


Figure 1. Components of SBFilter

The filter consists of an "Initiator/Manager process" and N "Server processes," which are spawned by the initiator at the time of the filter startup. N is determined by the administrator according to the expected filtering load to be handled. The Server processes listen on a common socket for requests for

filtering web accesses. The proxy server and SBFilter can be running on the same machine (coupled mode) of separate machines (decoupled mode).

Whenever the proxy server needs to check the access permissions of a URL access request from a host, it first establishes a connection with one of the SBFilter servers listening on the common socket. The IP address of the host that requested the URL and the URL itself are encapsulated and passed as a Filter Request to the filter server with whom connection has been established. The filter process performs request filtering according to the scheme described in the next subsection and determines whether the given host is to be allowed access to the target URL. This decision is then sent back to the proxy server over the connection that already exists. Once this is done the connection is closed and the filter Server goes back to listening on the common socket for more requests. Whenever a Server filter dies due to unknown reason, the Initiator/Manager process detects this event and starts another Server process.

The above design has several advantages. First, the only interaction of the proxy server with SBFilter is that of sending a Filtering Request to SBFilter and receiving a response. This way the SBFilter design is completely independent of the proxy server. Secondly, in decoupled mode a single filter engine can be used for filtering requests from multiple proxy servers. Different proxy servers within the same organization can send Filter Requests for processing to a single filter engine server over the network. Finally, the independent design allows SBFilter and proxy server to scale according to their own computation requirements. An alternative design would be to build the filtering functionality into the proxy server itself in order to achieve higher performance. However, this would take away all the above mentioned flexibility. Further the proxy server's code would end up becoming heavier and the filtering code less portable among different proxies.

3.2. Filtering Strategy

Every URL of interest has a record corresponding to it in a disk resident database. This database can be organized as a hash table or B-tree or any other convenient data structure that provides fast search, insertion and deletion. Each *URL Record*in database has two lists associated with it - an *Allow List* and a *Disallow List.* The allow (disallow) list contains a list of IP addresses of hosts that are allowed (disallowed) access to the corresponding URL.

Figure 2 illustrates the organisation of data structures in SBFilter. A *two-level cache* to the disk resident database is created in a shared memory region to avoid as much disk I/O as possible. This shared memory region is accessible to every Server filter process and is initialized by the Initiator/Manager process. The second-level shared memory cache consists of URL Records that have been used recently for validating accesses to this site. The cache is maintained in the form of a hash table. An LRU policy is used for replacing URL Records when shared memory space becomes low. The first-level cache maintained in shared memory is a User Record hash table that stores User Records, one corresponding to each host who is currently using the web through the proxy and needs validation before accessing any web site. This *per host user record* contains two small tables or lookup buffers - *allowed lookup buffer (ALB)* and *disallowed lookup buffer (DLB).* ALB (DLB) contains a few most recent URLs that were allowed (disallowed) to be visited by this host in the recent past. Here we use an approximate LRU policy to replace entries in the lookup buffer with new ones. A timeout policy periodically flushes these entries.

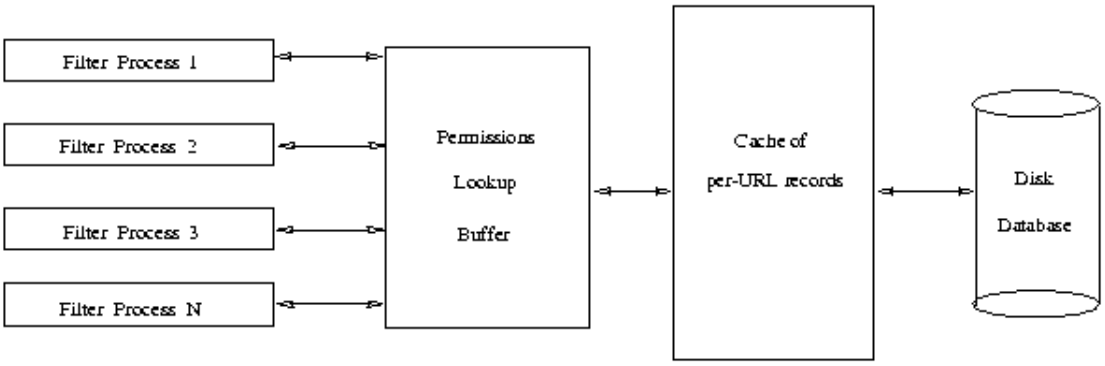


Figure 2. Organization of Data Structures in SBFilter

With the above data structures in place, filtering proceeds as follows. Given the IP address of a host and a URL being accessed by that user, the filter engine first checks if the corresponding User Record exists in the shared memory. If it doesn't, a new User Record is created and placed in the User Record hash table. Then the ALB and DLB of the User Record are searched for the presence of the URL that is being accessed by the host. If the URL is found in ALB(DLB), the site access is ALLOWED (FORBIDDEN) to the host.

If the URL is not found in either of the lookup buffers in User Record, the filter engine forms a set of prefixes from the given URL pathname. For instance if the URL is `http://www.yahoo.com/a/b/c.html`, the set of prefixes will be `www.yahoo.com, www.yahoo.com/a, www.yahoo.com/a/b` , and `www.yahoo.com/a/b/c.html.` For each of the prefixes in the set SBFilter checks whether the corresponding URL Record of the prefix is in shared memory cache or not. If it is not in cache, it is brought in by reading the record from the disk-resident database. Then the Allow List and Disallow List of this URL Record of the prefix are searched for the presence of this host's IP address. In case the IP address of the access request's source host is in the Disallow list or it is not in the Allow list, the access is rejected. Otherwise SBFilter moves on to the next prefix. Only when the access is not forbidden by any of the prefixes is the access allowed. Thus access to web sites can be controlled at the level of the web sites' file system sub-directories. Access is allowed to lower level sub-directories or files only if access is allowed for the parent directory.

For web sites that do not have an explicit entry in the URL database, a default permission record exists that specifies which hosts or subnets should be allowed/denied access to the rest of the web resources. A host is allowed access unless it is explicitly disallowed in the default permissions record.

Some ISPs dynamically assign IP addresses to the customers at login time. Thus the same customer may end up having different IP addresses in each session. To address the issue of uniquely identifying the customer for filtering purpose in such a situation, one can easily adapt the above filtering strategy to use the unique customer ID at the ISP instead of the dynamically assigned IP address.

3.3 Specifying Rules for Filtering

The administrator enters rules into the database by specifying them in a simple text format and informing the filter engine to perform the update of rules on the database. To allow a site the, the rule is specified as

allow host <host_ip> site <url> or

allow subnet <subnet_ip> <number_of_subnet_mask_bits> site <url>

To disallow a site, we replace the *allow* keyword in the above rule by *disallow*. This scheme provides fine levels of granularity in access control by allowing the administrator to allow/disallow permissions to sites for individual hosts or to entire subnets that use the services of proxy server. Fine grain access control to web sites' file system can be provided by specifying the URL at the level of files or sub-directories. For instance the following rule

disallow host aaa.bbb.ccc.ddd site www.badsite.org or

disallows the host/subnet *aaa.bbb.ccc.ddd* from accessing the entire site *www.badsite.com* including any of the files or sub-directories at that site.

3.4. Concurrency Control

Two concurrency control design issues in SBFilter are (a) what happens when multiple server filters simultaneously try to modify the contents of the shared memory and (b) what happens when one filter process is modifying the shared memory cache while another is reading it. In the first case, we use semaphores to ensure that only one server filter writes to the shared memory at a time so that shared linked lists in hash table buckets are uncorrupted. For the second case, we organize the write in such a manner that any read that occurs concurrently with another read/write goes through safely without having to block on any semaphore. This makes the common case of reads fast while maintaining the correctness.

3.5. Decoupled Mode Operation of SBFilter

To make SBFilter scalable with the load, separate instances of filter engine can be run on multiple PCs connected by a network and served by a common DNS server. The IP addresses of all these PCs are mapped to a common hostname (say `sbfilter.ecsl.cs.sumysb.edu`) by inserting an entry in the mapping table of the DNS server. Thus any host name lookup on `sbfilter.ecsl.cs.sunysb.edu` returns a list of IP addresses of the mapped PCs running the filter engine. The order of these IP addresses rotate in a cyclic fashion. A proxy server, which has an access request to be validated, first does a `gethostbyname()` on `sbfilter.ecsl.cs.sunysb.edu` and receives an IP address, which is the first in the corresponding set of IP addresses. It then sends the filter request to the IP address obtained above. An instance of the filter engine sitting on that machine processes the request and returns a reply. Successive requests are distributed among multiple filter engines, because the first member of the IP address set is rotating on each `gethostbyname()` call. In order to avoid the hostname lookup overhead on each access request, the list of IP addresses returned by the first `gethostbyname()` call in the proxy server is cached in the proxy server and used for uniformly distributing subsequent requests.

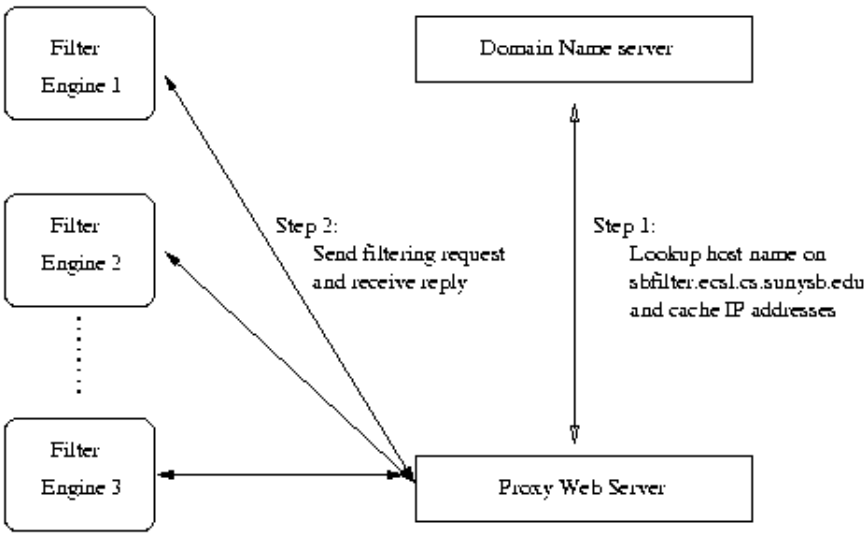


Figure 3. Operating SBFilter in Decoupled Mode

4. Implementation

SBFilter has been implemented such that it can run with any proxy server with minimal modification to the proxy server code. The main filter engine is a completely independent program which communicates with the proxy server using UNIX domain sockets if the proxy is running on the same machine or through TCP/IP sockets if the proxy is running on a different machine.

We added a new module to the Apache[1] web server, which can be configured to run as a proxy server. The module contains about 170 lines of code that enable the proxy to query the filter engine for validating client requests for any URL access. Currently the filter engine runs on Linux, which is one of the popular operating systems among the ISPs, and FreeBSD. The distribution can be downloaded via ftp from ftp://sequoia.ecsl.cs.sunysb.edu/pub/sbfilter.tar.gz.

5. Performance Analysis

This section presents the results of a performance study on the SBFilter prototype. The aim of these performance experiments was to stress test the filter engine by studying its performance under simultaneous access from a large number of hosts. First we describe the micro-benchmark results obtained by measuring the latency of various delay components of the filter engine. Next we measure the factors affecting the overall throughput of the filter engine and examine SBFilter's performance when running in decoupled mode. Finally we compare the relative overheads due to SBFilter and proxy server.

5.1 Measurements Setup

All the measurements were conducted by running SBFilter on Intel PentiumPro processor 200 MHz machines running FreeBSD Unix. All machines used for decoupled mode measurements had similar configuration in hardware and software. All time stamps were taken with standard Unix `gettimeofday()` facility. A 10 Mbps Ethernet was used for decoupled mode measurements of SBFilter.

Each machine used had a 64-Mbyte memory.

Since we did not have access to a large enough number of real host machine or subnets that we could use to stress the filter engine to its limits, we conducted our measurements using a synthetic database of site records. The synthetic URL database is built using 10000 rules, whose format is described in Section 3.3. There are totally 1000 distinct URLS and 10 different subnets, each subnet having 1000 source host machines that access the proxy server for internet access. The first-level cache for each user consisted of 10 cached entries of site permissions in each of allowed and disallowed lookup buffers. The second-level cache of per site permission records consisted of a hash table whose buckets could grow up to occupy maximum of 1MB of shared memory space among them.

In order to feed the filter engine with filtering requests, we used actual web access traces collected by our laboratory's proxy server. The number of distinct real source hosts in the trace was much smaller than the number of source host machine we wanted to simulate (10x1000). Hence we extracted the access substream from each real source host in the trace, made multiple copies of each stream and replaced the real host's IP address and real URL accessed by a synthetic IP address and synthetic URL respectively. The resulting streams of web accesses were multiplexed and fed to SBFilter to evaluate its performance. The workload derived this way preserved the access locality characteristic from a single user, but not across users.

5.2. Latency Measurements

The major components of processing a filtering request in the filter engine are (a) the overhead of communicating via sockets, including the wait overhead and data read/write overhead, and (b) processing time to determine access permissions. Part (b) has three significant components: (1) the search time at first-level cache, (2) search time at second-level cache, if missing in the first-level cache and (3) fetching a site record from the disk resident URL database. We present the delay breakdowns in the following two tables - first when a URL request filtering is completed by accessing only one of the two levels of cache and second when the request misses in both levels of cache and the disk needs to be accessed to retrieve the URL record. SBFilter ran in coupled mode for these measurements.

# of filter processes	Communication Latency	Processing Latency	First-Level Cache Search Latency	Second-Level Cache Search latency
1	201	60(level 1 hit) 359(level 1 miss)	7	205 (on level 1 miss)
5	2869	63(level 1 hit) 556(level 1 miss)	7	230 (on level 1 miss)
10	5837	63(level 1 hit) 650(level 1 miss)	7	243 (on level 1 miss)

Table 1 : Latency breakdown when no disk access is required (all values in microseconds)

# of filter processes	Communication Latency	Processing Latency	First-Level Cache Search Latency	Second-Level Cache Search latency	Disk I/O Latency
1	230	810	7	221	180 (from file system buffer cache) (34998) (from disk)
5	3213	861	7	179	281 (39885)
10	6357	1116	7	270	340 (40934)

Table 2: Latency breakdown when disk access is required (all values in microseconds)

From the above table, it is apparent that overhead of communicating with the proxy server dominates the request processing cost. When the number of filter processes increases, the cost of communicating with the proxy server also goes up since now there are more processes competing for communication resources. As the number of filter processes increases, the request processing time also goes up when the request misses in the first-level cache and the second-level cache needs to be touched. This is because in this case SBFilter needs to populate the first-level cache in shared memory with the information retrieved from the second-level cache. This requires that the filter process compete with other filter processes to acquire shared memory lock so that it can modify the shared cache. An increasing number of filter processes also increases the probability of lock contention and thus time spent waiting for locks. As expected, a request's processing time is higher when disk I/O is involved in filtering the request.

The first-level cache's search time is constant and is about 7 microseconds. The second-level cache search time depends on the number of entries in its allowed/disallowed lists. On the average, the URL records in the synthetic database contained five subnet addresses. The search times for the second-level cache do not show any marked variations. The disk I/O overhead also follows the trend of communication overhead, showing an increase as the number of filter processes increases. Typically the disk I/O requests get serviced by the file system's buffer cache in the kernel and do not need to go all the way to the disk.

5.3 Throughput Measurements

We measured the throughput of SBFilter in terms of numbers of filter request it can handle per second, under different workload and system parameters.

The variation in the throughput of the filter engine in coupled mode as the number of filter processes increases is given below. The default workload fed to the filter engine was as described in section 5.1 with requests multiplexed from multiple synthetic streams of user requests. Further the filter rules in permissions database were configured to allow all requests, i.e., all requests were 'benign' requests. The measurements were performed with a warmed up (loaded) second-level cache, so as to avoid too many disk accesses typical in the start-up phase.

# of processes	Filtering requests handled per second
1	1080
2	936
5	903
10	688
15	405

Table 3: Throughput of the filter engine in coupled mode with varying number of processes

First of all we observe that the overall throughput is sufficiently higher than the speed at which the proxy server can handle requests. Hence the filter engine will not become a bottleneck in the overall proxy server performance. With the increase in the number of filter processes, the overall throughput of the filter engine goes down. This is because when the cache has warmed up sufficiently, all processing in the filter processes is CPU bound. That is, since there is rarely any disk I/O involved, more filtering processes on a single processor actually hurts the overall filtering performance due to concurrency control and context switching. However, the advantage of having multiple filter processes to serve the filtering requests becomes apparent when a process misses in both levels of caches and needs to sleep on disk I/O while fetching the permissions from the disk database. At this time, other filter processes can handle the stream of incoming requests thus maintaining the desired throughput. Having more processes is useful when access pattern is not expected to display sufficient locality. Thus number of filter processes is a tunable parameter depending on the locality characteristics of the access pattern.

We also measured the throughput of coupled mode SBFilter when varying the percentage of 'disallowed' requests fed to the filter engine. 'Disallowed' requests usually miss at one or both levels of the cache since once disallowed, a site is has a low probability of being accessed again by the same user. On the other hand 'allowed' requests usually get serviced at one of the two levels of the cache. Hence the efficiency of caching depends on exploiting the fact that most of the requests are 'allowed' (or benign) requests. Table 4 shows the results.

Disallowed requests	Filtering requests handled per second
0%	1086
10%	994
20%	623
30%	417
40%	367
50%	184

Table 4: Variation of throughput with percentage of 'disallowed' requests

We varied the number of hosts requiring permissions check at a time in order to measure its effect on the throughput of the filter engine in coupled mode. This was done by limiting the number of separate host streams we considered for sending requests to the filter engine. 10 percent of the requests sent were disallowed requests. The results are shown in Table 5. The decrease in the throughput with the increase

in number of hosts can be attributed to decreasing locality of reference in the filter's cache. Higher the number of hosts implies that larger number of different web sites will be accessed leading to lower hit ratio. This variation in hit ratio at the two levels of cache is also shown in Table 5.

# of users	Filtering requests handled per second	First-Level Cache Hit Ratio	Second-Level Cache Hit Ratio
1000	1610	65.2	86.2
2000	1302	62.6	86.3
5000	1075	61.8	88.5
10000	982	60.2	86.6

Table 5: Variation of throughput and cache hit ratios with number of hosts

We measured the throughput when the filter engine was run in the decoupled mode described in Section 3.5. Filtering requests were pumped to the each instance of SBFilter from 4 machines simultaneously. The different instances of filter engine itself were run on separate machines and used TCP sockets for receiving and responding to filtering requests. The variation in throughput with the number of instances of filter engine is shown in Table 6.

Number of filter engines (on separate machines)	Filtering requests handled per second
1	109
2	217
3	306

Table 6: Variation of throughput with number of instances of filter engine in decoupled mode.

It can be seen that in decoupled mode, the throughput provided by SBFilter increases with the number of instances of SBFilter running simultaneously. As compared to coupled mode, the lower throughput in decoupled mode is attributed to the massive network communication overhead encountered using TCP/IP sockets. As per our measurement, it took from 0.8 to 2.0 ms round trip time for echoing just 1 byte of data over a TCP/IP connection on a 10 Mbps LAN, which is much higher than SBFilter's average processing latency. The advantage of the decoupled mode approach lies in that a number of filter engines which reside physically on separate machines from the proxy server, can be used by different proxy servers to control accesses from a common set of host machines. This flexibility can become more useful when the filter rule database (and hence the filter engine) is to be put on a machine more secure than the proxy server itself. Thus one can easily achieve a convenient tradeoff between performance and flexibility by using coupled or decoupled mode of operating SBFilter.

5.4. Relative Overhead Comparison of SBFilter and Proxy Server

In order to verify that the overhead due to SBFilter is small as compared to the overhead due to proxy server, we measured the average response times observed by a client program trying to fetch a 1K byte HTML file from a web server 1000 times. The web server, the proxy server and the client program were running on different machines in the same subnet. The same subnet was chosen in order to minimize the

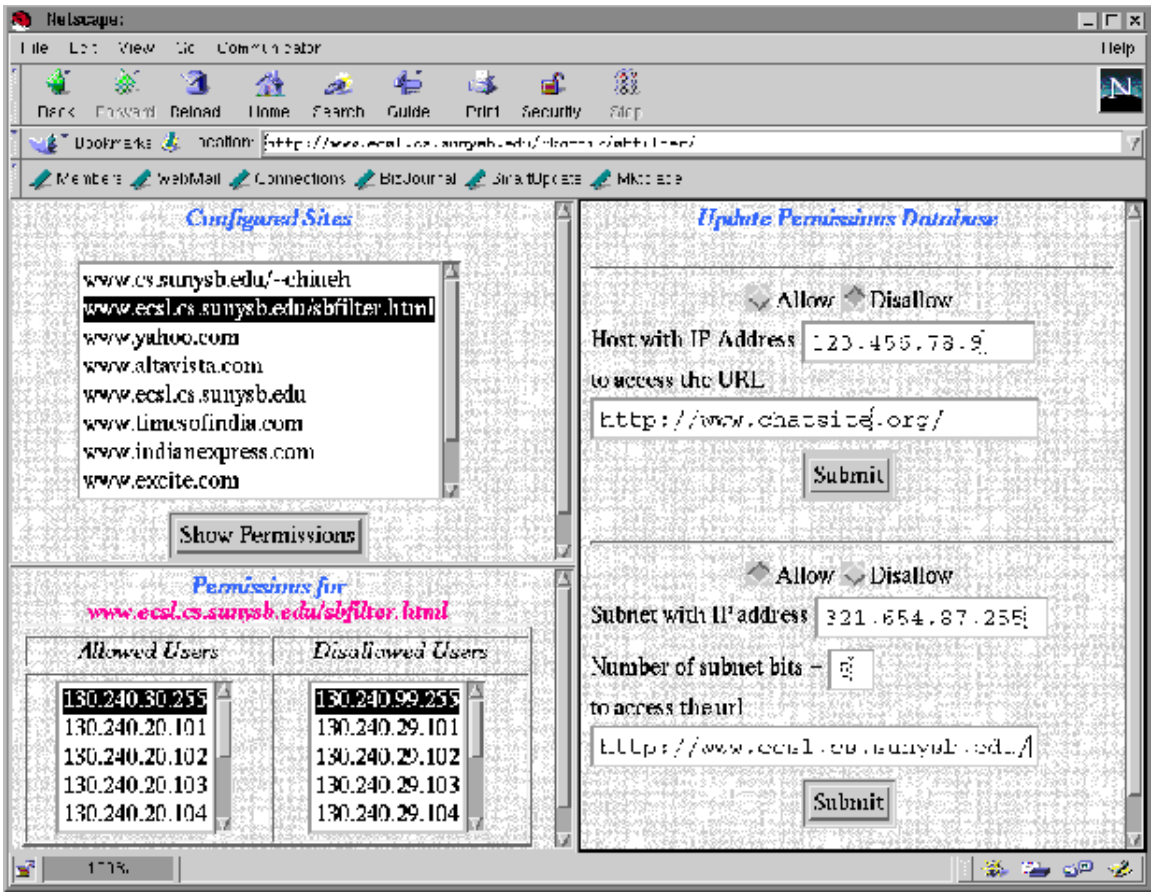
variations in delays due to network communication. The average response time at the client to fetch the HTML file directly from web server was 15.4 milliseconds. The average response time to fetch the HTML file through the proxy server, but without the filtering by SBFilter, was 28.1 milliseconds. The average response time to fetch the HTML file through the proxy server, with filtering by SBFilter enabled, was 30.3 milliseconds. From these figures we can see that the overhead due to proxy server alone is 12.7 milliseconds (28.1 - 15.4) and the overhead due to SBFilter alone is 2.2 milliseconds (30.3 - 28.1). This is just 17% of pure proxy server overhead on the same subnet. If the client and the proxy server are on different subnets, the proportional overhead due to SBFilter will be even lower.

6. Conclusion

In this paper, we presented the design and implementation of SBFilter - a fast URL filter engine - for controlling internet accesses. Internet access management is emerging as a crucial service when a growing number of users with a diversity of interests are using the Internet and the need to tailor the access control policy to specific requirements becomes more essential than ever. SBFilter successfully strikes a good balance between convenience to control and freedom of speech and information. SBFilter uses an intelligent two-level caching scheme for fast URL request filtering. The access control rules can be specified using a simple text format and used to update the filter rule database while the proxy server is running. Being logically separate from the proxy server itself, the presence of the filter engine is completely transparent to the end user. It can also run in a coupled or decoupled mode to provide different tradeoffs between flexibility and performance.

7. Future Work

A web based GUI interface is being developed to enable easy online administration of the permission database of the filter engine. The following image shows a sample view of software for administration of the permissions database.



8. References

[1] Apache Server Project, http://www.apache.org

[2] CERN httpd, http://www.w3.org/hypertext/WWW/Daemon/

[3] CyberPatrol Software, "Keeping Internet Access manageable", http://www.cyberpatrol.com/cpwp.htm

[4] Hayes, Mary, "Working Online, or Wasting Time?", Information Week , May 1, 1995, pp. 38-51.

[5] "Internet Access Control Using Proxy Servers", The DataBus - Vol. 36, No. 1 December, 1995-January, 1996, http://www.cedpa-k12.org/databus-issues/v36n1/proxy.html.

[6] Brenda S. Baker and Eric Grosse, "Local Control Over Filtered WWW Accesses", Fourth International World Wide Web Conference, December 1995, Boston, Massachusetts, USA, http://www.w3.org/Conferences/WWW4/Papers/117/

[7] NetNanny, http://www.netnanny.com.

[8] Network Safety Inc., "Supervised Internet access control", http://www.safety.net/supacc.html.

[9] SurfWatch Software, "SurfWatch", http://www.surfwatch.com.

[10] VicomSoft Software, http://www.vicomsoft.com.

[11] World Wide Web Consortium, "W3C Content Selection: PICS", http://www.w3.org/pub/WWW/PICS/, September 11, 1995.