Using Dynamic Delay Pools for Bandwidth Management

Gihan Dias [gihan@cse.mrt.ac.lk]
Chamara Gunaratne [gunarapc@cse.mrt.ac.lk]
Dept. of Computer Science and Engineering, University of Moratuwa

Abstract

This paper presents a method of improving the bandwidth management controls in the Squid cache server, thus allowing the majority of users to obtain a significantly better performance in a bandwidth-constrained environment. In many countries, the unavailability of sufficient bandwidth results in very slow access to the Internet. This situation is exacerbated by a small number of users who use a disproportionate share of the available bandwidth for file downloads, etc., starving other users of bandwidth.

Therefore some means of controlling and managing the bandwidth available to users is necessary. Most ISPs provide web access through cache/proxy servers. As all web accesses are channeled through the cache server, using the same server for bandwidth management is expedient. Squid is a popular cache server which contains a bandwidth management feature called "delay pools". However, the existing implementation of delay pools did not provide satisfactory performance as bandwidth allocations are static and do not adapt to varying usage patterns.

Our solution adjusts the bandwidth allowed per each user and group of users to provide an optimum level of access to all users. During off-peak hours, each user can use essentially the full available bandwidth, while during peak hours light interactive users get better performance at the expense of heavy and non-interactive users. By implementing these modifications, we are now able to service a larger number of users at a lower cost per user.

1.0 Introduction.

The Lanka Education and Research Network [LEARN] provides Internet and e-mail facilities to the universities and research institutions in Sri Lanka. However, as in many places, demand for Internet and web services is substantially higher than the available capacity.

Currently over 10 institutions with a combined staff and student population of over 20,000 are served by approx. 2Mb/s bandwidth. Before implementing this solution, accessing even simple web pages often took over 30 seconds, making the web almost unusable. Being a publicly funded organization, LEARN could not afford to purchase sufficient bandwidth to meet the complete demand. Therefore, we needed to improve the utilization of the currently available bandwidth.

When examining how the link bandwidth is used, we found that there are two main classes of users. One group comprises of heavy users of resources and typically has several open browser windows with pages being rendered simultaneously and also may have one or more file downloads in operation. The other group consists mainly of interactive users who open one browser window and wait until it is fully rendered before proceeding further. Even though the first group is numerically much smaller, they tend to utilize a far larger share of the available bandwidth to the detriment of the second group.

Therefore, to ensure reasonable and fair levels of access to all users, we decided to prevent a few users from utilizing a disproportionate proportion of the available resources.

2.0 Cache Servers

One common means of reducing the load on links is a cache server that temporarily locally stores web pages accessed by users [1]. When a user requests a new web page, the request is processed through the cache server and if it is available locally, it is provided from the cache rather than

from the remote web site. This way we can save the cost of fetching the same data over and over.

Many ISPs use cache servers to process the web accesses of their users. A major side-effect of this is that all web accesses have to go through the cache server and this provides a focal point for monitoring and controlling how users utilize the available resources.

3.0 Deploying delay pools.

The bandwidth management component in Squid [2] is called "Delay Pools" and is written by David Luyer [3]. Using this program, we can manipulate the bandwidth available to the user by changing two parameters.

Using the Token Bucket algorithm [4], delay pools allows us to define buckets, or "pool", at 3 separate levels:

- Aggregate – applies to the whole pool,
- Network – applies to each Class C network within the pool and
- Individual – applies to each individual IP address.

Each pool has two parameters: the *max* value and the *restore* value. The *max* value identifies the maximum size of the token bucket and the *restore* value is the rate at which the tokens are issued into the bucket.

When this feature is used, the time-averaged bandwidth available to each user is limited by the rate at which tokens are issued to the bucket (the *restore* rate). However, if the user has not requested any new data for some time and has accumulated tokens in the bucket, when a new page is requested the data is delivered in a burst, up to the number of tokens in the bucket.

In this manner, an interactive user who generally waits for a page to be rendered and does not use several browser windows simultaneously gets a reasonable level of access while users who try to use several connections at once are limited to the *restore* rate. This limitation can also be set at the *aggregate* level, to define the values for a group of users.

When we deployed delay pools in the LEARN network, we found that though it functioned properly and enabled fair access to users, it had a major weakness. The delay pools parameters are statically defined in the Squid configuration file and cannot be changed dynamically to suit the changing conditions in the network.

Bandwidth usage patterns change considerably at different times of the day, and an individual *restore* value that gives fair access to all users during peak hours is unduly restrictive during slack times, confining users to a small bandwidth and generally leaving a large proportion of the available bandwidth unused. Though it is possible to vary the delay pool configuration by using time based controls this does not respond to the actual demand at a given time and can be very cumbersome to implement in even a moderately complex environment.

Since the Squid source code was available for modification, the most attractive solution to this problem was to modify the delay-pools code to dynamically reconfigure the parameters to share the available bandwidth fairly among the current users, based on the principles of link-sharing described by Floyd and Jacobson [5].

4.0 Modifications to the Squid source code

Before arriving at the present implementation, we tested several different modifications and changes to the Squid code. Initially, we tried to keep track of the users in a delay pool and allocate bandwidth to each user based on the number of active users. However, this proved to be inefficient, as not all users connected to a particular pool would be browsing the Internet at the same time. Some would be reading data already rendered and others would be waiting for requests delayed due to congestion in other parts of the Internet. Therefore not all users would be utilizing the fixed amount of bandwidth assigned to them.

Another implementation that we tested was to track each individual's usage independently and dynamically allocate each user a separate bandwidth value up to a certain fixed ceiling. However, in this case the fairness criteria were invalidated as heavy users were quickly assigned a large amount

of bandwidth, to the detriment of more casual users.

The current algorithm that we have implemented [Fig. 1] trades sophistication for simplicity and speed of execution. Speed of execution is a very important factor as we dynamically evaluate and adjust the values once per second and an inefficient algorithm would pose a heavy workload for the system running the program. This algorithm comprises of 2 stages:

1. Allocation of bandwidth to users per delay pool
2. Transfer of unused bandwidth from one pool to another.

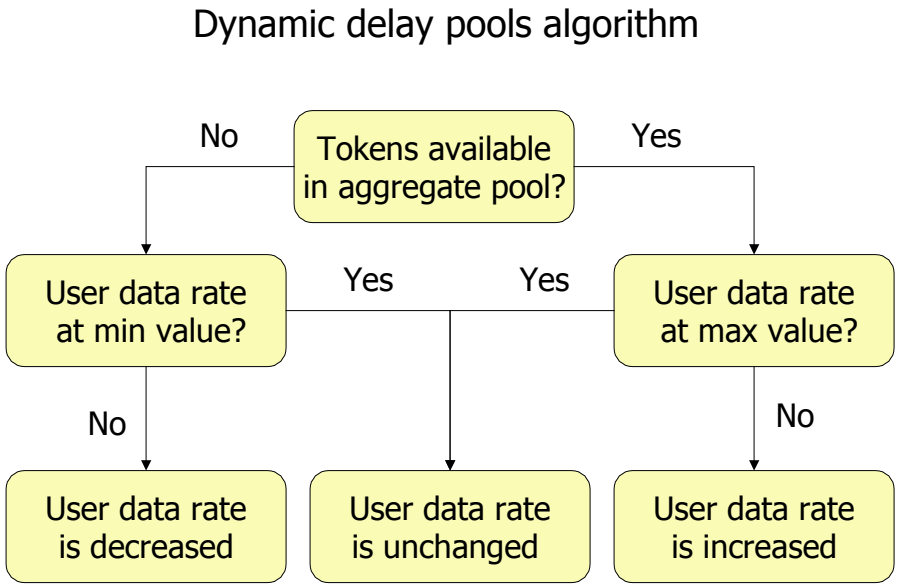


Figure 1

The algorithm uses the *aggregate* number of tokens in the delay pool as the criteria for whether the user's link bandwidth is fully utilized and skips the *network* settings entirely. If the number of tokens exceeds the cutoff value [presently 10% of the aggregate delay pool size], indicating that there is some capacity being underutilized, each individual user's bandwidth allocation is increased by a fraction. If the number of tokens is less than the cutoff value, indicating that the requests are backing up and link bandwidth is fully utilized, individual user's bandwidth allocation is reduced by a fraction. This is shown in the flow chart below:

Usually the link bandwidth is used both for the *restore* rate (at which tokens are issued to the pool) and the maximum size of the pool. The aim of the algorithm is to keep the number of tokens in the pool as close to zero as possible. Practically, we have set a cutoff value that is 10% of the token restore rate and when the available tokens drop below this value the user's bandwidth is reduced. A value that is larger than zero is used, as it is difficult to precisely control the number of tokens in the pool.

Depending upon the number of tokens in the aggregate delay pool, the bandwidth an individual user receives can range between a small minimum value, e.g. 100 Bytes per second, and the rate at which tokens are issued to the aggregate delay pool. The small minimum value is set to ensure that even at peak load times users get some access and practically this limit is not usually reached. Setting the maximum value to the rate at which tokens are issued to the aggregate pool ensures that even if there is only one user at that instant in the whole delay pool, that user can use all available resources.

Transferring bandwidth among delay pools

Using a single delay pool allows bandwidth to be shared fairly among users. However, in many instances, we want some users (e.g., researchers) to be allowed more bandwidth than others. This can

easily be achieved by having several delay pools with different parameters.

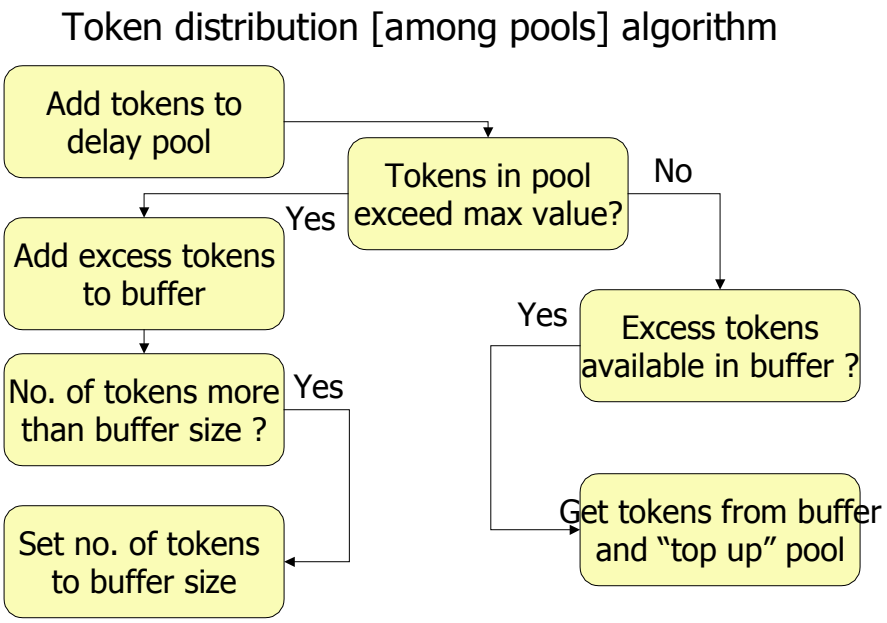


Figure 2

However, we found that when several delay pools operate on the same server, some pools do not use all their allocated bandwidth while others are saturated.

The Squid code was therefore further modified to allow bandwidth assigned to a particular delay pool, which is not being utilized, to be transferred to another delay pool [Fig. 2].

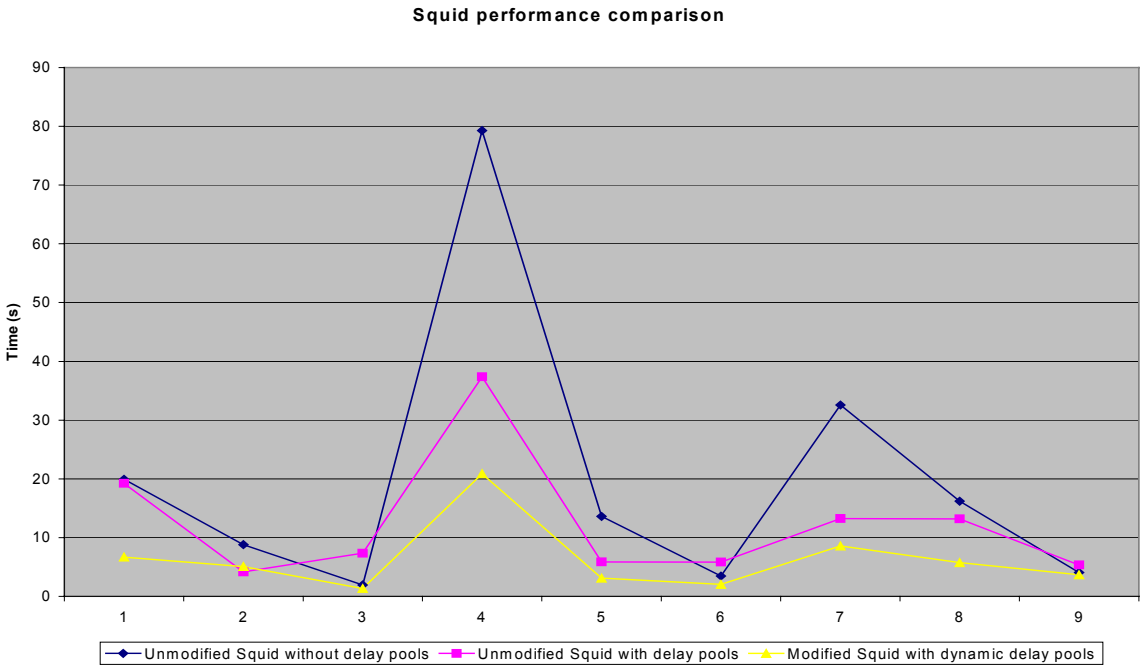
This modification checks the level of tokens in each delay pool after adding the token allocation available to that particular pool. If the level of tokens is greater than the maximum allowable value, then the excess tokens are transferred to a buffer and if a delay pool whose level of tokens is less than maximum permissible value is encountered, the tokens in the buffer are used to "top up" the pool up to it's max value [or until the tokens in the buffer are exhausted]. Accordingly, excess tokens that would be unutilized by lightly loaded delay pools are distributed among the pools that are using their token allocation fully. The maximum size of the buffer is set to a few seconds worth of tokens.

5.0 The modified cache in operation

Considering that the modifications implemented were quite simple, it was perhaps a bit surprising that they worked well. When running as the University of Moratuwa's main cache server, the modified cache proved to be quite adept at maximizing the bandwidth utilization at all times. Generally, the *current* values were negative (indicating a bandwidth over-usage) for only very short periods of time before the controls kicked in and reduced the users' allocations. We found that the ability of a few users to monopolize the resources to be almost negligible as no matter how many browser windows or downloads may be open, all users in the delay pool get the same allocation.

Usually even during peak hours [noon lunch break], the individual allocation per user ranged between 400 ~ 600 Bytes per second. When considering that in addition, each user can use up to 100kB *before* this limitation applies, users obtained acceptable response times. During slack times, values several times greater were allocated, up to the whole capacity of the link. Sharing resources between delay pools has enabled us to setup groups of users with [e.g. postgraduate research students] high resource allocations without worrying that the resources may not be fully utilized.

Regarding the load on the server, on our test setup, a Dell Power Edge server with dual Pentium III 500MHz CPUs and 512MB of RAM running RedHat Linux 7.1, CPU and memory utilization generally averaged around 8% ~ 10%.



The statistics below are from the figures we obtained from the University of Moratuwa Squid cache. Though the values fluctuate within a wide range the overall impression indicates that the modifications have had a noticeable impact on the performance. A set of representative web pages and graphics files were accessed 10 times (with Squid configured not to cache files from these sites) under normal usage conditions and the median access times were calculated for 3 situations:

1. Unmodified Squid without delay pools
2. Unmodified Squid with delay pools configured [individual restore = 1500Bps]
3. Modified Squid with dynamic delay pools

As the above graph shows, for most sites enabling delay pools gave better performance and the dynamic delay pool modifications increased performance further [note that under loaded conditions, dynamic delay pools have only a marginal advantage over static pools, but yield much better performance under moderately loaded conditions.

The table below [Table 1] gives the percentage decrease in access times with respect to the access times when running the Squid cache server without delay pools

The measurements taken over some time show a very wide variation in values. In some cases enabling delay pools has brought about a large improvement while in some cases the negative is true. However it can be observed that this has happened only for sites with very small access times. The most probable reason for this is that the processing overhead caused by running the delay pools code exceeds the advantage gained where accessing sites with low access times is concerned. Even though there is a wide variation in values and in some cases anomalies, it can be observed that enabling delay pools causes an improvement in access times and that enabling dynamic delay pools causes a further improvement.

Web site	Unmodified SQUID with delay pools [restore = 1500Bps] [%]	Modified SQUID with dynamic delay pools enabled [%]
1 - www.linuxgazette.com	3.51	66.42
2 - www.linuxgazette.com/../lglogo.png	52.81	42.47
3 - www.linuxgazette.com/../tm.gif	**-281.09**	31.35
4 - www.extremetech.com	52.86	73.68
5- www.lankacom.net	56.96	77.25
6 - www.google.com	**-68.98**	41.16
7 - www.yahoo.com	59.35	73.73
8 - www.academicinfo.com	18.54	64.49
9 - www.lacnet.org	**-30.88**	10.17

Table 1

6.0 Conclusion and future work

The present implementation of the modified cache server can be regarded as a limited success as though it does a fine job of ensuring fair and efficient access to users and maximum utilization of resources, the bandwidth management capabilities are severely limited

As far as the users, especially interactive users, are concerned the subjective impressions are stronger than what an objective analysis of the data can indicate. Timeouts while waiting for a web page are quite rare and the rendering speed of a web page for interactive users has increased dramatically. Usually around 15 ~ 20 seconds after clicking a link a readable web page is rendered in the user's browser, which is a significant improvement of the situation that existed previously.

Although the current system allows bandwidth allocated to a single server to be distributed near-optimally, we still face the situation of some caches on our network not using their bandwidth allocation while others are overloaded. We propose to extend the system to allow bandwidth to be exchanged among a set of caches.

This system does not prevent link congestion, especially when large files are being downloaded. It also has no facilities for real-time traffic. We are currently working on a congestion avoidance protocol [6], which will allow both real-time and interactive traffic to optimally share a link.

References

[1] Caching 101 – Internet caching resource center, available at http://www.caching.com/caching101.htm

[2] Squid web proxy cache server, available at http://www.squid-cache.org/

[3] SQUID Frequently Asked Questions: SQUID version 2 available at http://www.squid-cache.org/Doc/FAQ/FAQ-19.html#ss19.8

[4] Computer Networks, 3rd Edition by Andrew S. Tanenbaum [p381-p384]

[5] Link-sharing and Resource Management Models for Packet Networks by Sally Floyd and Van Jacobson [*IEEE/ACM Transactions on Networking, Vol. 3 No. 4, August 1995*]

[6] Link cost optimization through link utilization maximization by Gihan Dias and Shantha Fernando [*IEEE Sri Lanka, 9th Annual Conference*]