Acknowledgement

Preface

Index

Acknowledgement 1
Preface....2
Index 3
Index of Pictures & Tables 5
Abstract....7
1. Introduction....8
1.1 Background....8
1.2 Requirements 9
1.3 Problems 9
1.4 Motivation....10
1.5 Proxy....11
1.6 Squid 12
2. Existing system 13
2.1 Delay Pools in Squid....13
2.1.1 Token Bucket Algorithm 13
2.1.2 Token Bucket Hierarchy in Delay Pools 14
2.1.3 Initialization of Delay Pools 15
2.1.4 Delay Pool Stages 16
2.1.5 Drawbacks of Delay Pools in Squid 17
2.2 Dynamic Delay Pools 18
2.2.1 Bandwidth allocation within the pool....18
2.2.2 Transferring bandwidth among delay pools 19
2.3 What is missing?....20
3. Delay Pool Improvement....22
3.1 Upgrading to Newer Squid Version....22
3.2 Problem of Bandwidth Transfer among Delay Pools 22
3.3 Improving Bandwidth Management within a Single Pool....23
3.3.1 Introduction of Common Individual Restore Value 23
3.3.2 Individual Minimum Limit Removal....24
3.3.3 Poor Bandwidth Utilization at Off-peak Time....26
3.4 Support for the User Initiated Bandwidth Management System 28
4. Bandwidth Monitoring....30
4.1 The Importance of Bandwidth Monitoring in My Research....30
4.2 Existing Bandwidth Monitoring Tools 30
4.2.1 MRTG....30
4.2.2 Webalizer 31
4.3 Own Approaches....31
4.3.1 Squid Log Analyser 32
4.3.2 Bandwidth Monitoring using delayBytesIn() Function 34
4.4 Analysing Results 35
5. Inter-Proxy Bandwidth Negotiation....36
5.1 Necessity of Inter-Proxy Bandwidth Communication....36
5.2 Overall System....36
5.2.1 Understanding the Functionality....36

5.2.2 Overall Implementation 37
5.3 Local Bandwidth Detection of the Proxy 39
5.4 Bandwidth Usage Pattern Recording 40
5.4.1 Recent Usage Pattern 40
5.4.2 Five Minute Summary for 7 Days of the Week 41
5.5 Decision Making System 42
5.6 Inter-Proxy Communication 43
5.6.1 Message Handling System 43
5.6.2 Inter-Proxy Bandwidth Negotiation Protocol 45
5.7 Bandwidth Receiver End: Redirecting Local Traffic 48
5.8 Bandwidth Donor End: Allocation for the Requesters 49
6. Other Approaches 50
6.1 tc 50
7. Results & Further Works 51
7.1 Multi-disciplinary Decision Making 51
7.2 Inter-Proxy Communication via Multicast 51
8. Conclusion 52
References 53
Appendices 55
App 01: Delay Pools Setting in Squid 55

Index of Pictures & Tables

Fig 1-01: System Model....10

Fig 1-02: Centralized Bandwidth at LEARN....11

Fig 1-03: Role of a Proxy....11

Fig 2-01: Concept of Delay Pool....13

Fig 2-02: Delay Pool's Hierarchy in Squid....14

Seg 2-01: Delay Pool Parameters Setting....15

Fig 2-03: Delay Pool Initialization....16

Fig 2-04: Different Delay pool Status....17

Fig 2-05: Bandwidth Allocation within the Delay Pool....18

Fig 2-06: Bandwidth Transfer among Delay Pools....19

Fig 2-07: Possibilities of bandwidth transfer....21

Fig 3-02: Unfair Individual Restore Values in Original Dynamic Delay Pools....23

Tbl 3-01: Restore value calculation with time....25

Seg 3-01: Function to Set Restore Value of Delay Pools....29

Fig 4-01: MRTG Link Analysis – University of Moratuwa (1st February 2005 at 20:16)....31

Fig 4-02: Sample Result from Webalizer – University of Moratuwa, May 2004....31

Seg 4-01: Squid Log Analyzer Exception for Local Traffic*]....33**

Seg 4-02: Bandwidth Monitoring through Delay Pools....34

Fig 5-01: How Inter-Proxy Bandwidth Negotiation Works....37

Fig 5-02: Implementation of Inter-Proxy Bandwidth Negotiation....38

Fig 5-03: Flexibility of Local Bandwidth Detection System....40

Seg 5-01: Recent Usage Pattern Structure....41

Seg 5-02: Upgrading Upper Level of Usage Patter Records....41

Seg 5-03: Five Minutes Backup Structure per Day....42

Fig 5-04: Algorithm used in Decision Making System....43

Fig 5-05: Algorithm for Bandwidth Negotiation 45

Fig 5-06: Inter-Proxy Bandwidth Negotiation Protocol..... 46

Fig 5-07: Message Format..... 47

Fig 5-08: Redirection of Local Traffic via Multiple Uplinks 48

Abstract

1. Introduction

Using the Internet is becoming a necessity even in areas with a shortage of affordable bandwidth. Many organizations and ISPs in the South Asian Region are facing the problem of using a small amount of bandwidth equitably to meet the demands of their users. Increasing bandwidth though simple is a costly solution, which many organizations can not afford. At the same time, it is evident that in many organizations the existing bandwidth is not fully utilized, at times.

Our work allows a set of *co-operating organizations* (e.g. universities or departments) to share one or more *bandwidth-limited links*, using *Squid proxies* [1]. Each proxy server monitors the current demand for bandwidth by its users, compares the demand with its current bandwidth allocation, and negotiates dynamically with *parent* and *peer proxies* to either obtain additional bandwidth from other proxies or release unutilized bandwidth to others. In this manner, each organization is assured of a minimum bandwidth at all times, but can use additional bandwidth if it is available in the system.

We have designed and implemented a *Dynamic Bandwidth Negotiation Protocol* which allows co-operating Squid proxies to notify when they have unutilized bandwidth. Proxies which require more bandwidth can then request and obtain an allocation from a proxy which has an excess allocation. We are currently running this system in test mode at the *University of Moratuwa*.

1.1 Background

With the development of the Internet, more and more *bandwidth-hungry applications* have come into use. We have identified the following major types of applications with respect to patterns of bandwidth usage: [2]

1. Interactive web browsing
2. File downloading
3. Constant bandwidth applications (e.g. video conferencing, VoIP)

Interactive web browsers download a page and peruse it and if they feel it is interesting, they click on another link. They wait until the page is downloaded. And peruse that page and continue the same procedure. Even we have many of this type of users; they do not consume much bandwidth, since they spend more time on page reading. The amount they download is small. Therefore equitable systems should allow downloading their content quickly.

The other controversial part of this group is the *file downloaders*. What they generally do is to start a couple of file downloads in the background. After that they do not look at them and continue with what ever other work they are doing. Some times they are not even present in front of their computers. They are generally not bothered how long it will take to download the files. They start their work with the downloaded file when they see the completion of the download.

What we believe is downloading should not be motivated during peak hours (this does not mean downloading is prohibited). It is a good idea to ask them to do download whatever it is, off peak time (over night). There are many *offline downloading applications* [3] on which users can place a download request. Then that server does the job at off-peak time and notifies the user generally via email.

We identified another group of users who do not fall in to any of the above groups. They have combinational characteristics of interactive users and downloaders. We consider them to be *Multibrowsers* since they keep many web browsers opened simultaneously. However, they work actively only with one or a couple of pages at a time.

Bandwidth usage and its distribution might be different for the organizations that have high bandwidth. With the availability of higher bandwidth the usage of Internet shifts to real time applications, online games, video conferencing which needs high guaranteed bandwidth [4]. *Bandwidth management* at proxy level is a bottleneck that leads to performance degrading in such a network.

However our application domain is in bandwidth limited countries and organizations. Most of South-Asian and Latin-American countries still suffer from limited bandwidth. They are unable to cater to all existing web traffic. Further this problem will last at least for another four to five years due to their economical, political and technical barriers [5].

1.2 Requirements

All Internet applications provide good performance when sufficient bandwidth is available. However, in many areas of the world, especially in South-Asia, the available bandwidth is a severe limitation on applications. As the cost of an Internet connection is dependent on its bandwidth, many organizations in the region cannot obtain sufficient bandwidth to provide all the users a satisfactory level of service during times of peak demand.

We have established that at certain times a site does not use all the bandwidth allocated to it. However at the same time, another site may be short of bandwidth.

Therefore, we decided to develop a system, to cater to a number of co-operating organizations, who are *internally well-connected* (e.g. departments within a university or a set of campuses). It can negotiate amongst the organizations to make optimum use of the total subscribed bandwidth. This system could also be used by service providers to implement a bandwidth-on-demand service for their customers.

We have previously implemented a system for bandwidth distribution among local users using *dynamic delay pools* on Squid proxy server [2]. We then needed to extend the system to negotiate bandwidth among peer proxies to gain the maximum out of existing total bandwidth. This is an important issue to a co-operative organization like the *Sri Lankan University System*.

1.3 Problems

Let's consider the following model [Fig 1-01]:

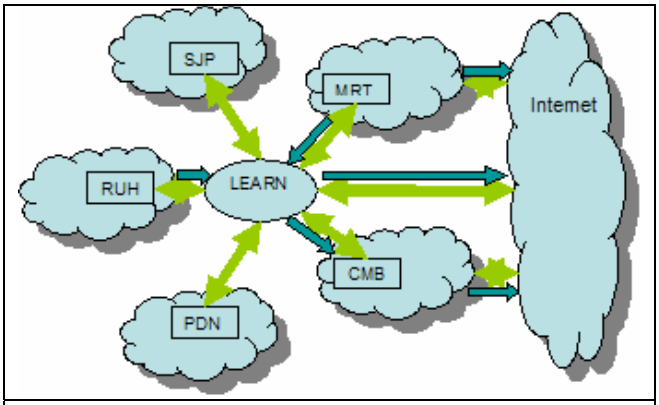


Fig 1-01: System Model

Here the entire network system (e.g. Sri Lankan University System) consists of several independent local components (e.g. universities). Individual organizations share a central internet connection, but some organizations also have their own independent Internet connection. Web access constitutes a major part of Internet usage, and our objective is to provide *optimum use of the Internet connections for web access* using a *proxy-based bandwidth management system*. This approach was chosen instead of *network-layer bandwidth management*, as the use of the *open-source Squid proxy* allowed us much greater control over the system than if router-based controls were used.

LEARN currently uses *dynamic delay pools* at the central web proxy to allocate bandwidth to sites. However the current system does not support multiple uplinks, i.e. when one site (e.g. MRT) has excess bandwidth, this cannot be used by any other site.

Further, we needed a system for sites to *reserve bandwidth in advance* for activities such as web-based video.

This paper presents our system which allows a number of independent sites to dynamically share the available bandwidth with one another. We have, in addition, developed a system, described in [6], which allows individual users to *request specific bandwidth* requirements, and for the system to *price bandwidth dynamically* based on the current demand.

1.4 Motivation

The very first basic motivation was the real requirement of this sort of system at *Lanka Educational And Research Network; LEARN*. It is the major ISP for the University system in Sri Lanka. Initially LEARN had the entire bandwidth centralized to LEARN [Fig 1-02]. Therefore our initial proposal was based on this assumption. It discusses bandwidth negotiation among multiple links at a single centralized site.

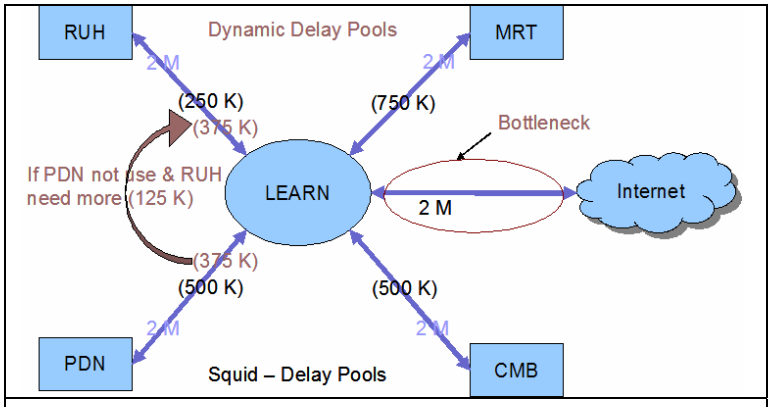


Fig 1-02: Centralized Bandwidth at LEARN

However with the elapsing of time, Universities started to get their own Internet links directly to their local sites. Therefore we had to generalize our objectives of the system to cater to organizations having multiple uplinks decentralized and distributed among several local sites.

Further the Department of Computer Science & Engineering, University of Moratuwa has done a lot of researches in this specific topic. One of these research outcomes identified that *most of our Internet traffic is web traffic*. Our Internet *links are fully saturated at peak hours*. But the usage falls almost zero even, at off-peak hours [7]. We use several techniques to optimize our existing available bandwidth [2, 6, 8].

One of these successful researches; *"Dynamic Delay Pools"* [2] which is briefly described later in this thesis; was highly effective. Therefore we realized the value of improved expansion of this research, to have a better outcome.

Not only our universities, but also most of the ISPs in the country and other countries in south-Asia and Latin-America face almost similar problems [9]. Our proposed solution can be used at all these places, effectively.

1.5 Proxy

The proposed solution in this thesis for the above problem has is been implemented at *proxy level*. Therefore it is important to brief what proxy is and its role in a network, with a *limited international bandwidth.*

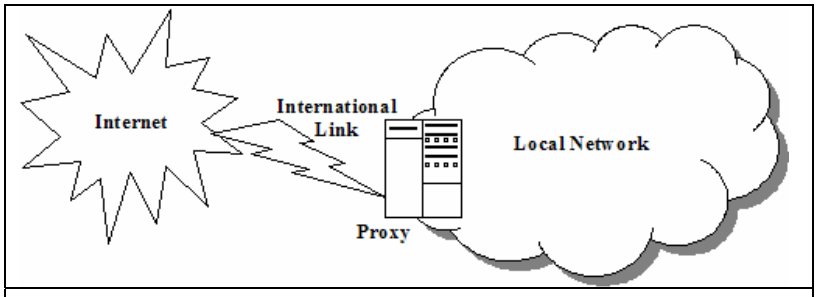


Fig 1-03: Role of a Proxy

Proxy is a server that is sited in front of a local network as the *gateway to the external world* [Fig 1-02]. It monitors all incoming and outgoing packets. If it detects unwanted packets at network, transport or application layer, *denies* the access or just *drops* the packets. In this way proxy can restrict unnecessary traffic on the limited international link [10].

We do *traffic shaping* at the proxy in our application. This is the most appropriate place, since all the traffic should pass through this point to access the International link. Generally in our model

each organization is expected to have own proxies. These proxies are aware of their local traffic. Therefore they can track the times of bandwidth scarcity and excess. Then, they can separately communicate to share their total bandwidth more effectively. That is what you see in [Fig 1-01] above and the details of the functional model is described later in this thesis [Section 5].

1.6 Squid

Squid Cache [11] proxy server is one of the most popular open source web proxies in the world. It provides user authentication, web caching, access control lists, access logging, content filtering, web acceleration and many other features [12].

Among all of them our interest is *user bandwidth management*. Squid uses *delay pools* for bandwidth management. Next section of the thesis [Section 2.1] describes the concept of delay pools in Squid. It is just a static configuration in squid configuration file [13]. It could not adopt the dynamical changes of the bandwidth usage of local users. *Dynamic Delay Pools* [describe in Section 2.2] solved this problem and managed locally existing bandwidth among all users in a fair way.

Dynamic delay pools are not sufficient to cater to the problem another level up. At that level we require the possibility of *managing multiple uplinks* and proper *communication among neighbour proxies*. Our development of *Inter-Proxy Bandwidth Negotiation Protocol* [Section 5] addresses this problem. It comes-up with an extension to the original Squid implementation utilizing dynamic delay pool concept.

2. Existing system

Even bandwidth management is not a major issue for developed countries these days; bandwidth scarceness was one of the main concerns in the past. Therefore many researchers have already addressed this problem. These approaches were from different points of view [14, 15, 16].

One of these approaches is to manage bandwidth at the *proxy level* [17]. In this model they assume all the traffic (at least the majority of the traffic) passes through proxies. This will help not only for *bandwidth management*, but also for web caching, user authorization, virus scanning, logging, user tracking, traffic analyzing, traffic controlling, access controlling, web acceleration and so on [18]. Squid Proxy is one of the open source project obeys this model.

Squid uses *Access Control Lists* (ACL) [19] for user identification. It has the capability of building ACLs based on source and destination IPs, protocols, source and destination ports, user authentication and the domains accessing too. However in our concern we are currently interested only on ACLs based on source IPs.

Squid uses *Delay Pools* [20] for bandwidth management. Delay pools work in conjunction with access control lists for ACL based bandwidth allocation.

2.1 Delay Pools in Squid

Delay pool uses *Token Bucket Algorithm* [21] in bandwidth management. First, it is important to understand how *Token Bucket Algorithm* works in *Squid Delay Pools*.

2.1.1 Token Bucket Algorithm

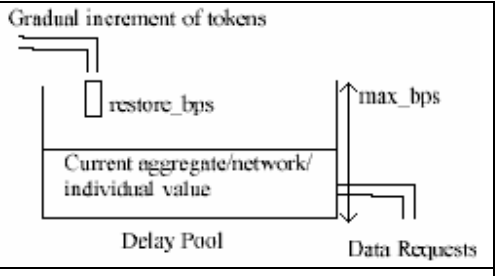


Fig 2-01: Concept of Delay Pool

A Delay Pool can be considered to be a bucket with two independent incoming and outgoing taps [Fig 2-01]. There is a gradual increment of *bucket level* in the incoming rate of the tap. We called *restore_bps* for this rate in Squid.
In addition to that there is a data outflow tap. This is the real data reception for the actual user. It is not at a constant rate. User can get any amount of data at once provided that he/she has enough tokens in his/her bucket.

For further elucidation we can consider the token bucket as the water tank in our house. It always keeps on filling gradually at the rate of the incoming water rate independently from the way we use it. If we have water inside the tank we can have any amount of water from the outlet. Obviously, we can have even a higher speed than the incoming water rate. However if the water level of the tank reaches zero, then the outgoing water speed also will be restricted to the incoming water speed. If we do not use water for a long time the tank will keep on filling and will

reach the full capacity of the tank. There after it will overflow. *Max_bps* in *token bucket* corresponds to this capacity of the tank.

2.1.2 Token Bucket Hierarchy in Delay Pools

Squid allows grouping all the existing users as required in different groups called *pools*. For example in our University we use groups called general students, academic staff, research students, administrative staff, executives and special groups. Each of these groups can have their own *token buckets*. They can have independent *restore rates* depending on their priorities.

Individuals within each group also have their own *individual buckets*. These buckets assure the fairness among users within each user groups. Otherwise *multibrowsors* and *downloaders* might override the *interactive users* in the same group at peak hours.

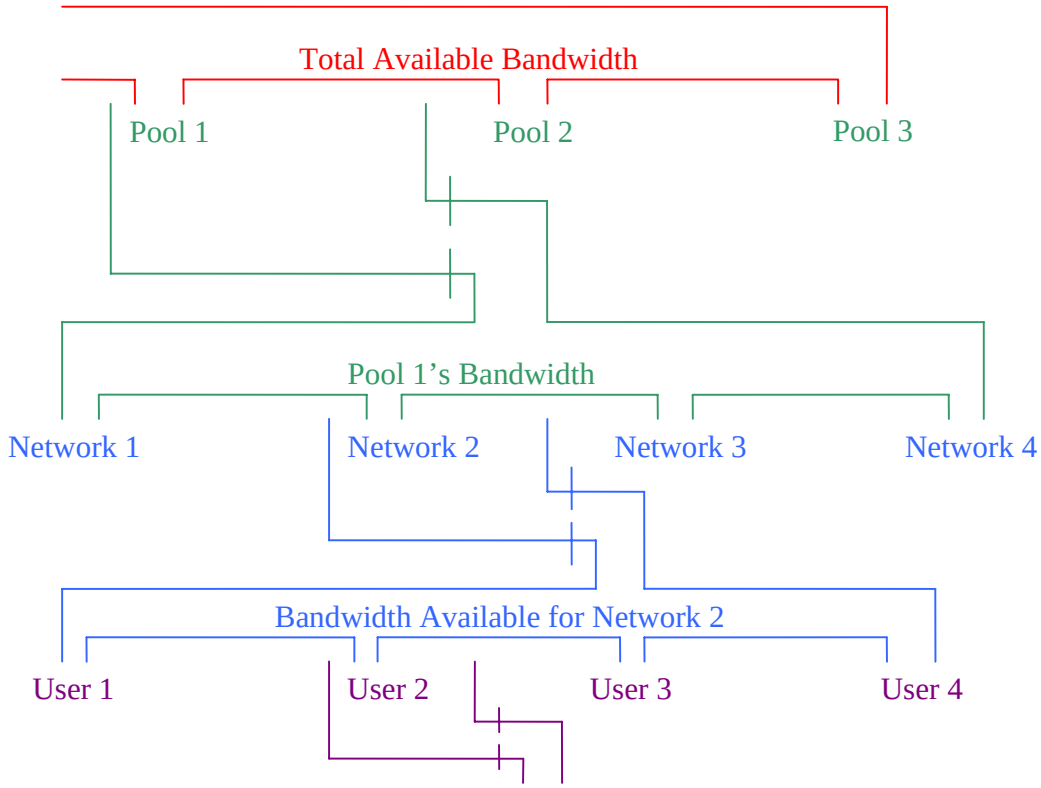


Fig 2-02: Delay Pool's Hierarchy in Squid

Squid provides another level of bandwidth distribution in between pools and individuals. i.e. *network*. It assigns token buckets for each class C network within a particular delay pool. The following diagram [Fig 2-02] illustrates all these levels of delay pools.

In our university, system we do not need to consider user's networks as far as we properly identified the user groups using their individual IP addresses (such as student or staff). Therefore we ignored network level of bandwidth management in our development. We simply disabled this

level in the configuration. By this method we achieved bandwidth fairness among all the users in a particular user group independent of the network they are connected from.

2.1.3 Initialization of Delay Pools

Initially we should decide on the parameters we should allocate for the delay pools and the individual users within a pool. These values mainly depend on the available bandwidth, user groups of the organization, levels of priorities for each user group and individuals and many other factors unique to organizations.

The best way of explaining this is to have an example. Let us consider the university system. If we consider only the student and staff groups for lucidity, staff member should have priority over students. This does not mean that we allocate rest of the bandwidth to the students after assigning required amount of bandwidth to the staff.

Therefore *Squid configuration* file does (*squid.conf*) initial bandwidth allocation for user groups. Generally in a university we have a lesser number of staff, to students. For keeping values simple let us allocate 50% of the available bandwidth to the staff group (which has lesser number of users) while allocating other 50% to the student group. This guarantees the staff priority over students. This is achieved by setting up the *aggregate restore* value of the *delay pools* for students and staff user groups. This setting is done in *delay_parameters* tag in configuration file. In the following configuration segment [Seg 2-01] staff and student delay pools have *25000* aggregate restore. This means 25000 bytes per second (i.e. ~= 1 Mbps) for each user group. This is a setting for 2Mbps Internet link. Please refer [App 01] for a detailed explanation about delay parameters settings.

```
#delay_parameters pool aggregate[res/max] network[res/max] individual[res/max]
delay_parameters 1 25000/50000 -1/-1 600/150000   # staff
delay_parameters 2 25000/50000 -1/-1 300/150000   # students
```

Seg 2-01: Delay Pool Parameters Setting

We have realized by experience it is acceptable to set delay parameters even slightly higher than the existing bandwidth while using *static delay pools* which originally came with Squid. It allows much better utilization of available bandwidth.

In *squid.conf* there are 3 major value pairs for *delay_parameters* tag. 1st pair refers to the *aggregate bucket* of the *delay pool.* 2nd pair refers to the *network* level and 3rd one refers to the *individual* user within the group. At each pair of any level 1st value is the *restore* value and 2nd value is the *maximum* bucket level allowed at any instance.

The network level of delay parameters guarantees different (*class C*) networks are getting fair bandwidth distribution within a single pool. In this way we can assure each department (if they have different class C IP blocks) guaranteed bandwidth for their staff (assuming the staff delay pool). However by setting this value pair as *-1/-1* we can disable this level of bandwidth allocation.

The last pair of delay parameters refers each individual within the delay pool and it guarantees to receive *individual restore* value even at the peek hours. In the above example staff members get 600 bytes per second while students having half the number. The last value of the delay parameters tag is the *individual maximum* bucket level (i.e. 150000 in above example). This is the upper limit for each individual within the pool. No matter what the restore rate of an individual is, they can receive data equal to the number of bytes available in their buckets. If they do not use them bucket starts to fill at the rate of *individual restore*. Finally the restored bytes will discard when bucket reaches the *individual maximum*.

Squid should set an initial level of buckets at the start of the squid process (In other word initial running of the proxy). This can be set to zero, 100% or any other level as required by the system administrator, through the squid configuration file (*delay_initial_bucket_level* tag). Default setting is 50% [Fig 2-03]

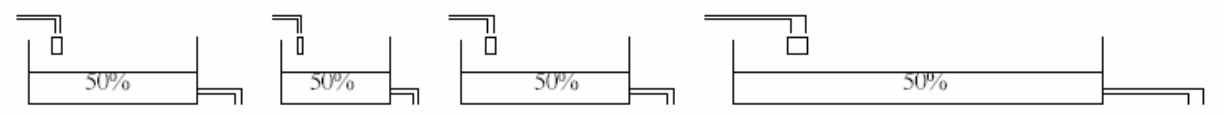
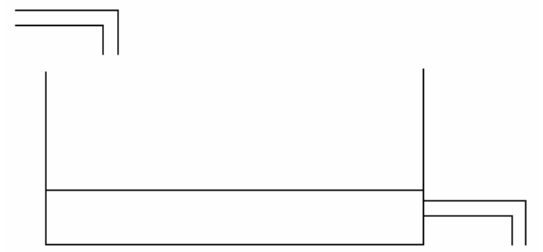


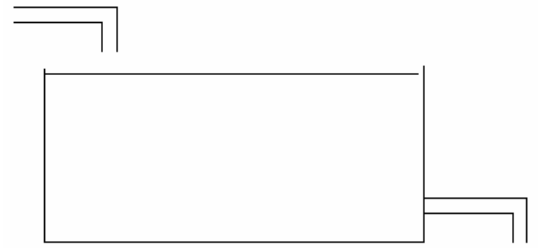
Fig 2-03: Delay Pool Initialization

2.1.4 Delay Pool Stages

With the initial setting up of the bucket levels delay pools run as a continuous process. All the delay pools are filled at its corresponding restore speed by just adding *restore value* to the *current level* of each delay pool bucket at the end of each second. With retrieval of data the current bucket level reduces accordingly. According to this behaviour there are 3 major possible statuses for any of the delay pool.



(a): Steady Operation



(b): Excess Bandwidth

(c): Bandwidth Deficient

Fig 2-04: Different Delay pool Status

1. *Steady operation* – In this case the *restore rate* (average rate at which the tokens are allocated in the pool) is equal to the average data *outflow rate*. So buckets always have a certain amount of tokens up to which, user can request data at burst. This is the general behaviour of the delay pools for average users. If the users retrieve data at their allocated speed this situation will be maintained unceasingly [Fig 2-04 (a)].

2. *Excess bandwidth* – If the users just went off from the computers after clearing all their downloads, there is no way of reducing their bucket levels. However still their buckets would fill at the restore rate. After a certain length of time the buckets would start to overflow after achieving their maximum bucket level. In the original squid delay pool implementation there is no method of using this wasted bandwidth [Fig 2-04 (b)].

3. *Bandwidth deficient* – When the average token allocation rate is less than data outflow rate the bucket runs out of tokens. Then user is limited, at his/her restore rate [Fig 2-04 (c)]. This is his/her guaranteed bandwidth. If the user has not used his/her bandwidth a while, the collected bandwidth he can get accumulated. These accumulated data can be retrieved at once. However there is no way of receiving any additional bandwidth at any instance.

However in the original implementation delay pools do not negotiate among themselves to dynamically allocate the above *Excess bandwidth* to the *Bandwidth deficient* pools, wasting existing bandwidth while demand is still there.

2.1.5 Drawbacks of Delay Pools in Squid

Squid delay pools allow bandwidth allocation only at the configuration. These settings affect only at the start of squid process. Therefore we should know the average bandwidth usage at each user group in advance. The static delay pools do not have the ability of adopting the dynamic behaviour. This inability causes huge performance degradation in our application domain. As an example in our University students access Internet early morning before lectures. Then they come to the labs whenever they have intervals. Sometimes sudden cancellation of lectures results

increment of usage in certain labs. Therefore it is really difficult to determine a suitable bandwidth usage pattern to this type of user group.

2.2 Dynamic Delay Pools

Mr. Chamara Gunaratne addresses this issue in squid delay pools and he has come up with an improvement called *Dynamic Delay Pools*. In dynamic delay pools also initial configuration sets user groups and their permitted bandwidths as same as in ordinary delay pool implementation. However in operation it detects any excess bandwidth at a particular user group and collects it in another bucket as *excess_bytes*. Whenever there is a user group which (delay pool) needs more bandwidth the delay bucket gets some more bandwidth from *execess_bytes.*

The Link-sharing and Resource Management Model for Packet Network [21] was used to modify squid to dynamically transfer bandwidth among delay pools [2]. The algorithm that has been implemented trades sophistication for simplicity and speed of execution. Otherwise it would pose a heavy workload for the system running the program. This algorithm comprises 2 stages:

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

2.2.1 Bandwidth allocation within the pool

In the first case, if the number of tokens of a particular delay pool exceeds the *upper cut-off value*, (i.e. link is unutilized), each individual user's bandwidth allocation is increased by a fraction. If the number of tokens is less than the *lower cut-off value*, (i.e. link is fully utilized), individual user's bandwidth allocation is reduced by a fraction [Fig 2-05].

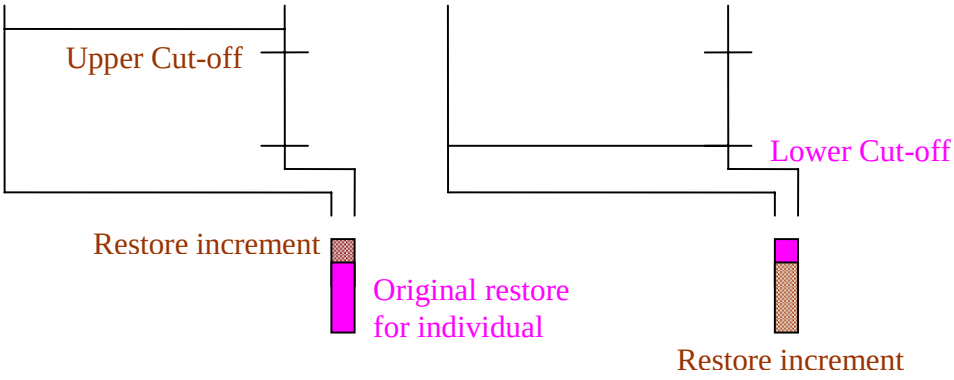


Fig 2-05: Bandwidth Allocation within the Delay Pool

The aim of the algorithm is to keep the number of tokens in the pool as close as possible to the middle level of the bucket. Depending on 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 but in practice bandwidth is not dropped to this level under normal conditions.. 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 bandwidth.

Let us assume the following example to get a practical sense of the above situation. The delay pool for the students in the University is allocated with a certain restore value for the delay pool (Say 400kb/s). The Delay pool bucket for the user group fills up at this speed. This bandwidth should be shared among the number whatever it is, of existing users in the group even at the peak time. Let us say 400 students access simultaneously at the peak time. Then the possible restore value for individual is 400kbps/400 = 1kbps. However what we know is that there are not so many users at off-peak hours. In original delay pool implementation still individual has to stay with individual's fixed restore value, since it can not be change according to the dynamic behaviour of the users. Therefore the user group bucket reaches its maximum and just overflows, wasting the bandwidth.

Dynamic bandwidth allocation within delay pool detects the increment of bucket levels at the absence of users within the delay pool. When bucket level rises beyond a certain maximum level, the system automatically increased individual restore value of the individuals of the user in the group [Fig 2-05]. In this way the entire bandwidth of the user group is utilize among existing users. Therefore ultimately if there is only one user in the user group he/she can enjoy the entire user group bandwidth.

Similarly, when the bucket level drops below a certain minimum level, system starts to drop individual's restore values gradually. In this way system maintains the equilibrium.

2.2.2 Transferring bandwidth among delay pools

Using a single delay pool allows bandwidth to be shared fairly among users within a single group. However, in many instances, it is important for certain users to be allowed more bandwidth than others. This can easily be achieved by having several delay pools with different parameters. However, when several delay pools operate on the same server, some pools do not use all their allocated bandwidth while others are saturated.

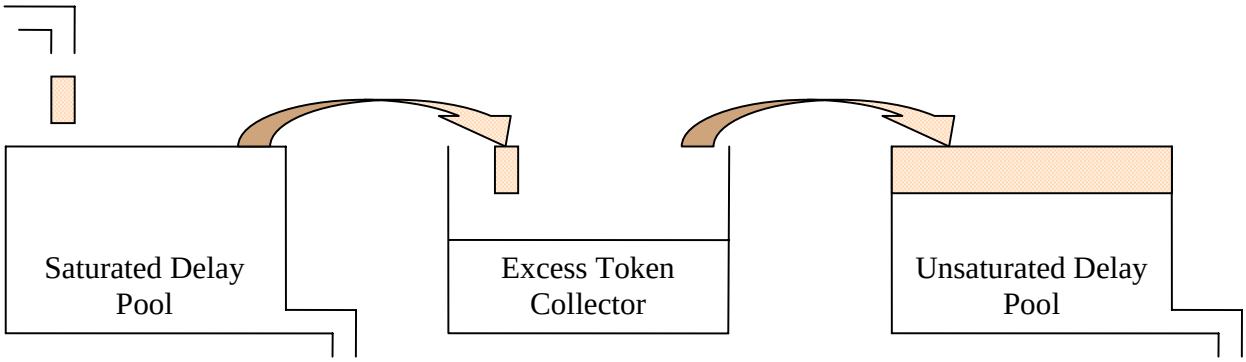


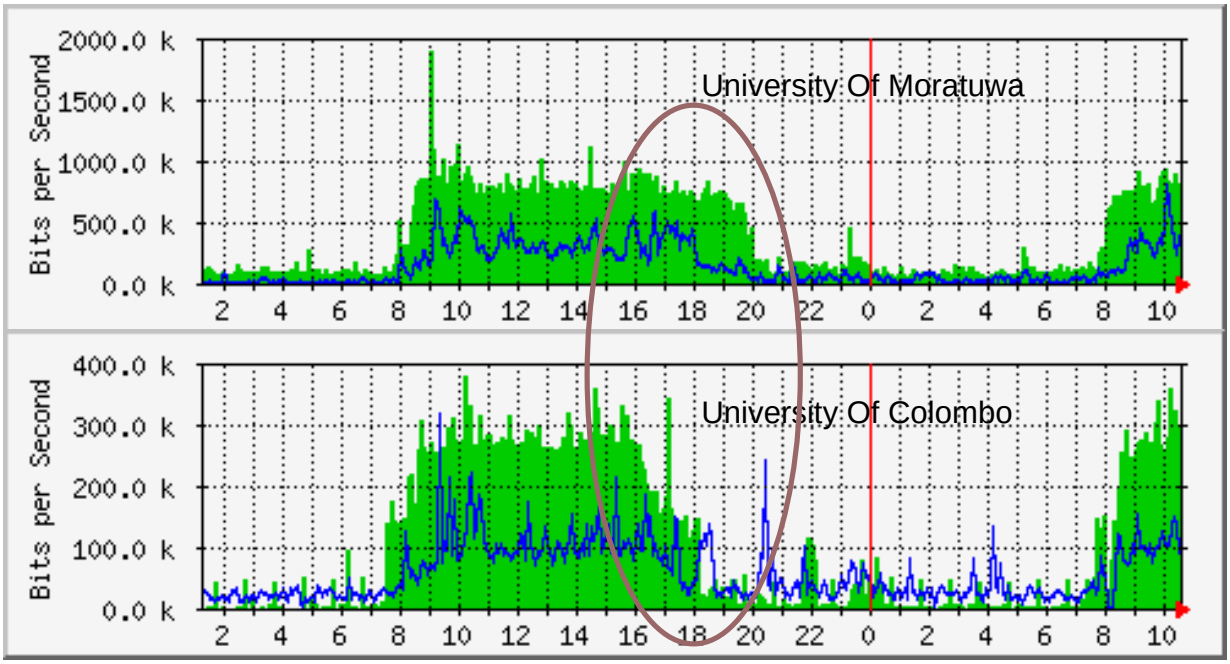
Fig 2-06: Bandwidth Transfer among Delay Pools

This problem has been overcome in *Dynamic Delay Pools* by allowing transfer tokens among delay pools [Fig 2-06]. 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 the maximum permissible value is encountered, the tokens in the buffer are used to "top up" the pool to its maximum 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.

2.3 What is missing?

Dynamic Delay Pools allow bandwidth reallocation among users of a single proxy. But for medium and large scale organizations who have limited bandwidth, this is not sufficient. They may obtain internet connections from more than one provider. i.e. There is more than one up link by which traffic can be directed. These links may not be under the control of a central authority.

For example [Fig 1-01] the University system in Sri Lanka has one central Internet provider (LEARN). The central bandwidth is allocated to each Individual Universities in pre-defined proportions. Currently Squid proxy with dynamic delay pools, is deployed at LEARN to allocate bandwidth among universities. In addition to this some universities (e.g. MRT, CMB) have their own Internet connections, since the central link is not enough for them. Even though these links are utilized by their own universities (Let us say it is CMB) at peak hours, these might be under utilization at other times. There might be another university (assume MRT) running out of its bandwidth at the same time [Fig 2-07]. As a co-operative organization CMB does not hesitate to offer its additional bandwidth to MRT. Further there is a possibility of lending the bandwidth this time which will allow CMB to get some additional bandwidth from MRT at an inverse situation.



Source: http://www.ac.lk/mrtg/e1.html

Fig 2-07: Possibilities of bandwidth transfer

The Dynamic Delay Pools can not handle bandwidth negotiation among multiple links. Therefore we came up with the next level of improvement called *Inter-Proxy Bandwidth Negotiation Protocol* which is explained in Section 5 of this thesis.

3. Delay Pool Improvement

Dynamic Delay Pool is a very effective concept. However there were problems in the original implementation. In my research I had to fix these problems. Further I tested the system with different parameter settings and configurations to achieve better performance.

3.1 Upgrading to Newer Squid Version

Original implementation of *Dynamic Delay Pools* was on *Squid 2.4*, an older version. There were many other features, performance improvements and bugs fixings on the next Squid version *Squid 2.5* [23]. I upgraded Dynamic Delay Pools to support Squid 2.5 at the start of my research. This helped me a lot, in the proper understanding, of the concept behind delay pools.

Squid released its next *Squid version 3.0*, midway during my research. This is a complete migration of the implementation from *C* to *C++* [24]. Therefore Delay Pool's modification needs considerable effort to incorporate dynamic behaviour in to it. I thought not to spend much time again on system compatibility upgrading, but spending more time on testing and implementing our new concept of *Inter-Proxy Bandwidth Negotiation Protocol*.

3.2 Problem of Bandwidth Transfer among Delay Pools

As I explained early in this thesis [2.2] there are two major levels of bandwidth transfer in delay pools. There was a problem in bandwidth transfer among delay pools. i.e. user groups. I wanted to visualize bandwidth transfer among user groups in an administrative interface. This is really useful in analysing user behaviour in different user groups. At the same time such data helps in troubleshooting network problems and web based hackers and virus attacks on the computers on the network.

I kept track of the last ten seconds of bandwidth transfers from and to all the delay pools. This transfer happens at two points in *Dynamic Delay Pools*

1. *Aggregate Restore* – This is the gradual token update for all delay pools
2. *Excess Bytes* – If there is any excess or scarceness of tokens for a delay pool, it is balanced with the use of excess bytes.

We need not be concerned about the first point above, since that is the ordinary behaviour of the delay pool. The second one is the method used in dynamic delay pools to get any *excess tokens* from other pools when it runs out of tokens. On the other hand delay pools top-ups *excess_bytes* whenever a pool has more than enough tokens.

I managed to display the bytes going in and out from each delay pool on the *cache manager interface* [25]. This data represents what the delay pools used highly are, at the moment and what pools are idle.

3.3 Improving Bandwidth Management within a Single Pool

3.3.1 Introduction of Common Individual Restore Value

We kept *individual restore values* for each user within a single delay pool. i.e. We had individually managed restore values for each and every user within a single group. But we realised in our observations that all these restore values for a particular group reaches out a single value in the long run **[***Fig Need – old cache manager delaypool page***]**. Then that value changes in accordance to the available bandwidth and user demand.

At the same time there was another problem in the individual bandwidth management system. When the squid starts or reloads, there are no users in any of the delay pools. Users are added as requests come to the *proxy server*. These initial users start from the minimum restore values and gradually increase their restore values. The users that do new bandwidth requests will allocate the minimum restore rate while others enjoy a higher restore rate. After a certain period of time, individual bandwidth requests, reach the token limit for the entire pool. Then it starts to dynamically reduce and increase according to the group bandwidth consumption. In this way, there is an obvious unfair bandwidth distribution among early and newly joined users even within a single delay pool [Fig 3-01].

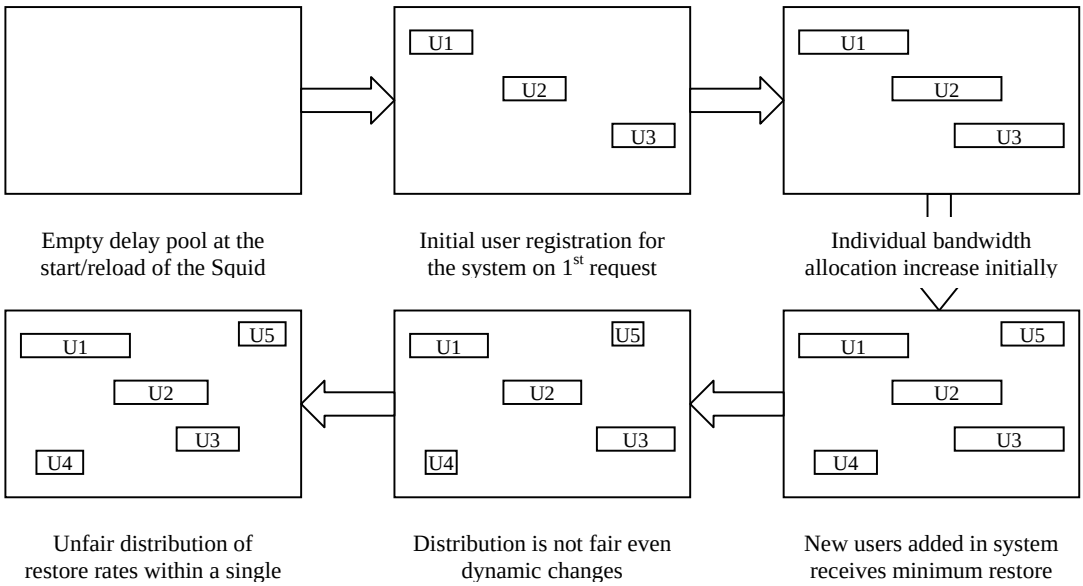


Fig 3-02: Unfair Individual Restore Values in Original Dynamic Delay Pools

We came up with the idea of a *common restore value* to overcome the above two problems. Here we have a single common individual restore value for each delay pool. This common value is increased whenever the pool token bucket has enough excess spare tokens and vice versa. All the individual users within the delay pool get this common restore value at any particular time, no matter how early or late they joined the user group **[***Fig Need – new cache manager delaypool page***]**. At the same time this implementation reduces the requirement of processing

for the delay pools, since we eliminated individual restore value calculation for each and every proxy user.

```
Pool: 4
        Class: 3
        Last 10 seconds Excess (Not in order):
                0, 313487, -60046, 0, 0, 0, 0, 0, 10689, 0,

        Aggregate:
                Max: 96000, Restore: 36000, Current: 76891

        Network:
                Disabled.

        Individual:
                Max: 100000, Rate: 183
                Current [Network 40]: 50:100000 14:100000 252:100000 23:100000
                Current [Network 17]: 22:-2 29:100000 26:89667 25:-2 24:-1
                Current [Network 19]: 81:100000 5:100000 10:100000 1:100000
                Current [Network 21]: 200:-3 199:0 193:-2 197:100000 208:1161
```

3.3.2 Individual Minimum Limit Removal

There was a minimum limit to the *individual restore value* in the original dynamic delay pool system. This value was decided by considering entire usage of the proxy at the peak time. Therefore this was a very small value. In addition, this minimum limit is set at the squid configuration. Therefore we had to analyse maximum possible users in the system periodically and whenever we add or remove users and user groups to the system configuration had to change accordingly. Further tracking the usage and number of maximum possible users, in the squid proxy, in an environment like a university, is extremely difficult. On the other hand these values change rapidly from time to time.

Time	Formulae	Restore value
0th second	100 (from config file)	100
1st second	round(100 x 1.02)	102
2nd second	round(102 x 1.02)	104
3rd second	round(104 x 1.02)	106
4th second	round(106 x 1.02)	108
5th second	round(108 x 1.02)	110
6th second	round(110 x 1.02)	112
…		
60th second	round(311 x 1.02)	317
61st second	round(317 x 1.02)	323
62nd second	round(323 x 1.02)	329
63rd second	round(329 x 1.02)	336
64th second	round(336 x 1.02)	343
65th second	round(343 x 1.02)	350
…		
117th second	round(959 x 1.02)	978
118th second	round(978 x 1.02)	998
119th second	round(998 x 1.02)	1018
120th second	round(1018 x 1.02)	1038
121st second	round(1038 x 1.02)	1059
122nd second	round(1059 x 1.02)	1080

Tbl 3-01: Restore value calculation with time

Therefore the *dynamic delay pool system* was improved by simply removing the minimum limit of the restore value. Then restore value could reach even "zero" if the group tokens are so scarce. With this improvement dynamic delay pools can manipulate any number of users, without any prior configuration.

There is another additional advantage of this modification. In the previous dynamic delay pool system, *individual restore value* starts from the minimum value, no matter what the present bandwidth available. Then it should go through many recursions to reach a reasonable restore value. For example assume that *minimum restore value* is 100 (bytes/s). Further assume that the reasonable restore value is 1000 at that time. General restore increment rate is 2%. Then the system needs 119 seconds (2 minutes) to reach this restore rate [Tbl 3-01]. Up to that time, the system is kept under utilized.

Another problem was arisen as I removed the minimum limit of the restore value. General restore evaluating formulae we used are:

Incrementing: Rn+1 = round (Rn x 1.02)
Decrementing: Rn+1 = round (Rn x 0.95)

Let us assume that due to heavy usage of web proxy, restore value drops somehow below 24. This is something very common in our university. In the morning or at the beginning of a break suddenly users start to use the web suddenly. They have their individual buckets full at that time. Then they start requesting beyond their pool limit of the group. As a result restore value starts to degrade very quickly and hits even zero value in some instance.

Now according to the incrementing formula:

Rn = 24 (or even less)
Rn+1 = round (24 x 1.02) = 24

This leads to fix restore value at 24 (or whatever the minimum value; in the worst case “zero”). No matter how small the number of users use the system, restore value can not increase beyond this. Therefore an additional checking is set:

If (Rn == Rn+1) then Rn+1 = Rn + 1

This simple modification completely solved the problem.

3.3.3 Poor Bandwidth Utilization at Off-peak Time

All the above settings work well. However when we do not have many users of the system, what we expect is to share whatever the available bandwidth among those users. If the total available bandwidth of the link is 96kbps and only one user use the system, he or she should get approximately 96kbps.

In the normal operations of our University system, so less a number of users is a rare occurrence. However when we got a separate additional link of 128kbps for testing purposes, we did not have that many users. Then we installed a separate proxy server for this link and initially we allowed only a handful of users.

Test 01: Identification of the Problem

There were situations where I was the only user in the system, and I observed the following points when I downloaded a fairly large file:

1. It started with an acceptable speed of (around) 14KBytes/sec (112 kbps).
2. And suddenly dropped down to 7-8 KB/s (56-64 kbps) within a few seconds.
3. It dropped down to the range 3-4 KB/s (24-32 kbps) after several minutes.

24 kbps is not an acceptable speed at all for an individual, who is using a 128 kbps link alone.

Test 01: Diagnosing the Problem

I pointed out the following line of code in *delay_pools.c*

```
if (common_individual_restore_bps[class_num - 1] >
    rates->aggregate.restore_bps)
        common_individual_restore_bps[class_num - 1] =
           rates->aggregate.restore_bps;
```

squid.conf delay_parameters were as follows:

```
Delay_parameters 1 2500/15000 -1/-1 2500/150000
delay_parameters 2 2500/15000 -1/-1 2500/150000
delay_parameters 3 6000/15000 -1/-1 6000/150000
delay_parameters 4 1000/15000 -1/-1 1000/150000
```

I was a user in the delay pool 2. The problem was very clear. I was limited to aggregate restore value, no matter how much free bandwidth available in the entire link.

Test 01: Solution for the Problem

Simply altered *delay_pools.c* as follows:

```
if (common_individual_restore_bps[class_num - 1] >
    rates->aggregate.max_bytes)
        common_individual_restore_bps[class_num - 1] =
            rates->aggregate.max_bytes;
```

This enabled each individual to reach the maximum restore value equal to maximum bandwidth of its own pool. The maximum bytes that can be retrieved by any delay pool is not more than the rate of link. (15000 above is a corresponding value for 128 kbps (= 16000 Bytes/sec) link). We keep this value slightly below the maximum limit of the bandwidth link to ensure that the link will not be congested under any circumstance.

Test 02: Identification of the Problem

1. The same testing was done with the above settings. The system was started at an acceptable speed at the beginning, but again dropped down.
2. This time it was not that bad as earlier. Final downloading rate was around 7-8 KBytes/sec. Still, this is almost half of the available bandwidth.
3. At one point I thought this is good, since it allowed newly entering users to experience better service without waiting for other users to decrease their restore values to release enough bandwidth.
4. But I had a much stronger point in contrast to this. Is it fair to wait for an uncertain user (who will never come on a Sunday) without offering that benefit to the researchers who will do some really important web browsing?

Test 02: Solution for the Problem

This was an unexpected situation according to the dynamic delay pool algorithm. If bandwidth among delay pools transfers properly, this situation can not occur. Therefore I tested the *excess_bytes* transfer mechanism.

The following lines of code should be executed after delay updates of all delay pools.

```
if (excess_bytes > (max_bandwidth * 5)) //UoM modification
    excess_bytes = (max_bandwidth * 5); //UoM modification
```

I realized that this code is executed at the end of delay updates of each pool. That problem was fixed.

Test 03: Identification of the Problem

1. Even with the above settings proxy could not give expected download speed.
2. I observed that still there was not enough *excess_bytes* to serve the active pool at the desired rate.

Test 03: Solution for the Problem

So *delay pool parameters* were set in *squid.conf* as follows:

```
Delay_parameters 1 2500/25000 -1/-1 2500/100000
delay_parameters 2 2500/25000 -1/-1 2500/100000
delay_parameters 3 6000/25000 -1/-1 6000/100000
delay_parameters 4 1000/25000 -1/-1 1000/100000
```

We cannot allow individual users to reach 25000 in a 128 kb/s link. So limitation of individual restore value was altered to "*aggregate.max_bytes/2*"

```
if (common_individual_restore_bps[class_num - 1] >
      rates->aggregate.max_bytes/2)
        common_individual_restore_bps[class_num - 1] =
            rates->aggregate.max_bytes/2;
```

This system could offer the entire bandwidth of the link to even a single user. When there is more than one user, it manages well to share bandwidth among them.

Final comments:

However the last (third) solution did not show enough stability when there were rapid demand alterations. If a user requested a download while the link is idle, it offers maximum possible initially. Then it starts to run out of bandwidth and the rate drops down. Again it starts to raise the restore value and gradually go up to the maximum possible value. Finally it utilizes the entire bandwidth after 5-10 minutes. [***However I expect much quicker settlement than this. I will compare Solution 02 and 03 with suitable parameter changes to have an optimized solution. Then I can set it in university 512 kb/s link for further realistic analysis.***]

3.4 Support for the User Initiated Bandwidth Management System

Researches were done on efficient bandwidth management as a group, at the University of Moratuwa. Another research gone in parallel was *Mr. Chamara Disanayake*'s *User Initiated Bandwidth Management* [6]. In this research, Chamara wanted to alter bandwidth of each user group that depended on the user demand. The delay pools in squid were altered to support this requirement by simply adding functionalities of alteration of restore value for any delay pool [Seg 3-01].

```
static int delaySetPoolAggregate(int poolNo, int poolVal) {
   int retVal = 0;
   if ( poolNo >= 0 && poolNo < Config.Delay.pools) {
      if (Config.Delay.class[poolNo] == 3) {
         Config.Delay.rates[poolNo]->aggregate.restore_bps = poolVal;
         retVal = 1;
```

```
        }
    }
    return retVal;
}
```

Seg 3-01: Function to Set Restore Value of Delay Pools

4. Bandwidth Monitoring

The lack of visibility into what traffic is traversing the Network forces IT to work in a vacuum. What traffic is consuming the bandwidth? What applications are crossing the Network? Who are the top talkers?

Successfully optimizing network performance requires unified insight into the applications and conditions on the network. Many of the commercial and non-commercial parties have made monitoring and reporting tools a cornerstone of its network optimization technology.

4.1 The Importance of Bandwidth Monitoring in My Research

The main objective of my research was not to do something with bandwidth monitoring. However in order to check the performance variations with the parameter changes, different configurations and application alterations, I wanted to have a better evaluation mechanism. Then I thought of using an existing bandwidth monitoring tool which can satisfy my requirements. The basic requirements of the monitoring system were:

1. The overall monitoring system should be able to produce results which can be used to determine the utilization of the existing bandwidth.
2. The system should support the identification of different types of users in the system. i.e. whether they are interactive users, downloaders or multibrowsers.
3. Feasibility of analysing data only for a particular group for a given period.
4. The system should support both user based and time based analysis, since the dynamic behaviour of the different types of users is required.
5. Independency or minimum interaction with the Squid processes and monitoring system is preferable to avoid performance degrading of the proxy server.

4.2 Existing Bandwidth Monitoring Tools

All the existing network monitoring systems are designed keeping certain objectives in mind. Those objectives do not directly coinciding with our requirements. Therefore we have to use a combination of multiple tools to check all the combinations of our requirements.

4.2.1 MRTG

Multi-Router Traffic Graph (MRTG) is one of the world's popular link monitoring tools [26]. It gives a minutely, hourly and daily basis simple graphical view of the bandwidth usage of the link. This is used to analyse the time based bandwidth usage of the link [Fig 4-01]. However MRTG does not support user based bandwidth analysing.

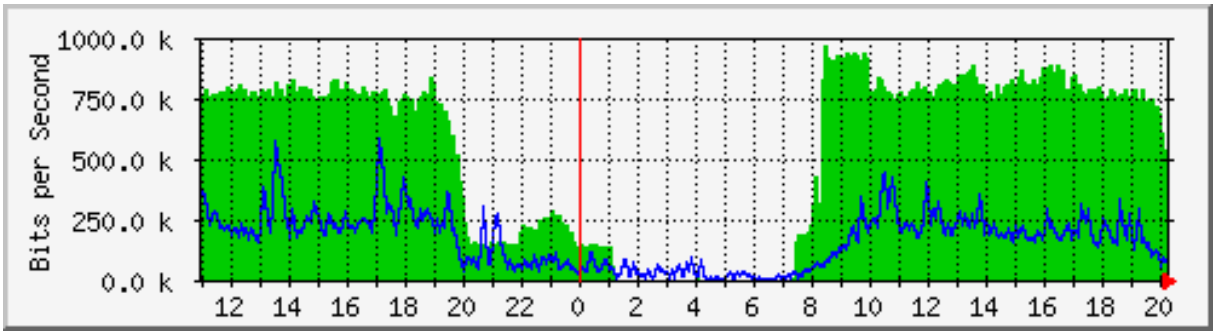


Fig 4-01: MRTG Link Analysis – University of Moratuwa (1st February 2005 at 20:16)

4.2.2 Webalizer

The *Webalizer* is a web server log file analysis program which produces usage statistics in HTML format for viewing with a browser [27]. It supports squid log files too. The results are presented in both columnar and graphical format, which facilitates interpretation. Yearly, monthly, daily and hourly usage statistics are presented, along with the ability to display usage by site, URL, referrer, user agent (browser), search string, entry/exit page, username and country [Fig 4-02]. Processed data may also be exported into most database and spreadsheet programs that support tab delimited data formats.

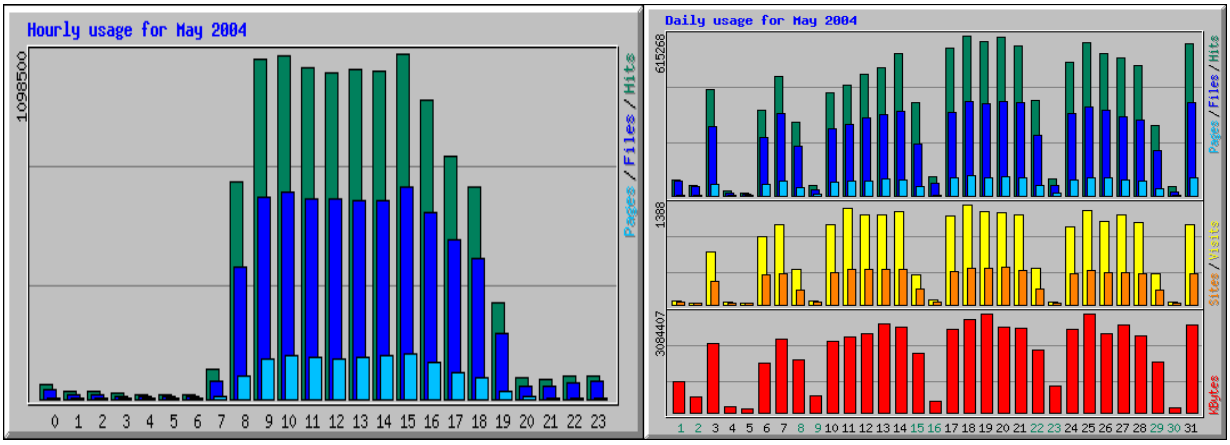


Fig 4-02: Sample Result from Webalizer – University of Moratuwa, May 2004

There are many other utilities like squidview [***and many others***] [28] which do the squid performance analysis. However all these were not able to exactly meet my requirements mentioned above. Therefore I thought developing my own *bandwidth monitoring system*.

4.3 Own Approaches

The most straight forward way of bandwidth usage monitoring is by simply *analysing squid log files*. The other advantage is that, we can do the analysis *off-line*. Squid logs each request with corresponding response information including time, delay, IP, response time, type and size of the response.

4.3.1 Squid Log Analyser

Introduction

I developed an analysis system to which squid log file can be fed. Then we can specify time range, number of records needed per hour and the number of records going to a single file. The system does the analysis and reports of a lot important information with respect to both users and time. Our own development of squid log analyser has the capability of producing the following results from analysed results.

1. **Total number of requests:** This gives the total number of requests done within each time frame specified. User can specify the span of the time frame and the entire period to analyse. Analysis can be done for individual users within specific user groups. Otherwise analysis can be done by comparing user groups.

2. **Total Bytes:** This results works very similar to above except analysis done based on the amount of bytes received rather than the number of requests. This is what actually corresponds to the amount of useful data retrieved.

3. **Average Gross Rate:** is simply obtained by calculating the straight time average of the amount of bytes received within the considering time period. When we take values for small time spans Average Gross Rate gives bit of meaningful results. Otherwise the effect of idle time of the user influences negatively. Therefore AGR is not so important in our decision making even though it shows a better overall view for the administrators.

4. **Average Burst Rate:** This is a representative value for instantaneous rates for the users. This is obtained by dividing the amount of data transferred by the time taken for it. Average Burst Rate is the average value of above calculated values for each request within the period considered. The expectation of dynamic delay pools is to achieve a higher value for Average Burst Rate for interactive users over multibrowsers and downloaders.

5. **Average Delay per Byte:** Even this value is simply the inverse of Average Burst Rate in practice, but not in theory. Average Delay is dealt with link latency while Burst Rate dealing with the throughput of the link under consideration. However the amount of information provided in the squid log file is not sufficient to evaluate these two quantities in a distinguished manner.

6. **Average Delay per Request:** This can be considered as a simplified version of Average Delay per Byte. As name implies, it is an average value of the delays of each request, while Average Delay per Byte represents delay per byte, which is more logical.

[***Another very important advantage we got in our own analyzer is the customarisability to match with our own requirements. Squid logs all the web traffic including local site references. However the local traffic by passes the bandwidth management in delay pools, since it does not

consume scarce international bandwidth. Exceptions for these unwanted logs were setup in Squid Log Analyzer [Seg 4-01].

```
if (strstr(url, "ac.lk") != NULL) continue;
if (strstr(dest, "192.248.8.") != NULL) continue;
if (strstr(dest, "192.248.9.") != NULL) continue;
if (strstr(dest, "192.248.11.") != NULL) continue;
if (strstr(dest, "192.248.10.") != NULL) continue;
if (strstr(dest, "192.248.24.") != NULL) continue;
if (strstr(dest, "192.248.1.") != NULL) continue;
if (strstr(dest, "localhost") != NULL) continue;
```

Seg 4-01: Squid Log Analyzer Exception for Local Traffic*]**

Interesting Investigations

I was led to a couple of interesting findings in my analysis. I had noticed at the beginning that download speed is unbelievably high as soon as we start a download. Then it rapidly drops down to an acceptable rate **[***Fig Need***]**.

I initially thought that this is simply due to the fact that interactive users get higher speed. Generally a particular user's bandwidth bucket has a reasonable amount of bandwidth tokens before he/she starts downloading. So there is not any restriction at dynamic delay pools for them and they get higher speed.

The above screen shots **[***Fig Need***]** were taken at the beginning and after a couple of minutes of download done at 128 kb/s link. As you see initial download speed of 115Kb/p (= 920 kb/s) of the above example can not be explained in this way. There is no way of raising the downloading rate beyond the link speed.

Later I realized that download start as soon as user click the URL. Then it continues downloading to the client buffer. If the client has to specify the location to save, downloading will stop when it fills up the client buffer. Then the client gets all the data in the buffer at the instance of starting the download. This will be visible as an incredible download speed.

But as time goes on, the client starts to receive the data at his/her allowed speed. Therefore it drops down gradually.

Problems in Squid Log Analyser

If a file download of a particular request has happened for several seconds, the speed distribution might have the following sort of pattern **[***Fig Need***]**:

But squid will log it as a 10 KB file downloaded at 10:10:40 and the delay is 13 seconds. In current *squid_log_analysis* we consider this as a file downloaded at 10:10:23 at a speed of 1 KB/s (10KB/10s). Here we have ignored the period of 10:10:10 to 10:10:23. One improvement

suggested is to consider the average download speed (1KB/s) for the entire period. But this is also not fair, since the speed of downloading might not be evenly distributed. (According to dynamic delay pools concept speed can vary from second to second. Therefore the best place to track the bandwidth usage is within the delay pools. We could reach the much better accurate instantaneous usage of bandwidth by this method.

[***However from the user's point of view they do not care about the variation of download speed at the midway, except for the pages they can start reading even at a partial download. Therefore what matters is the average download speed. Therefore above readings with proper average distribution over the time is important.***]

4.3.2 Bandwidth Monitoring using delayBytesIn() Function

delayBytesIn() is the function that releases the controlled bandwidth (available amount of tokens) to the users. This function gives the user (*delay_id*) and the amount of released bytes (*qty*) at each bandwidth release to the user. I built another squid monitoring system based on these values which will be able to solve the problems mentioned in the above section.

A simple modification of *delayBytesIn()* function of the *delay_pools.c* allows to check the downloading user and downloaded amount at each data transfer to the user. Following segment of codes simply display some of important values.

```
unsigned short position = d & 0xFFFF; // d is the delayed
debug(77, 2) ("\t%2d\t%4d\t%4d\t%7d\n",
      pool, delay_data[pool].class3->network_map[position >> 8],
      delay_data[pool].class3->individual_map[position >> 8]
            [position & 0xFF], qty);

// delay pool: pool
// network   : delay_data[pool].class3->network_map[position >> 8]
// host      : delay_data[pool].class3->individual_map[position >> 8]
                        [position & 0xFF]
// amount    : qty
```

Seg 4-02: Bandwidth Monitoring through Delay Pools

Squid is a processor intensive critical program. Therefore calculation related to bandwidth monitoring should not be done within the squid process. Therefore only the essential processing kept inside squid. Then we used another program to process these data and arranged them according to time and user/user groups in an effective way.

We had two major ways of sharing information between Squid and the analysing process. One way is to use shared memory. The other is to logging the usage down on the fly and analyse them offline.

First method is an online mechanism. All the analysis must be performed on the fly. There is a possibility of degrading performance of Squid when we run both processes on the same computer. On the other hand there is a danger of missing some of the data transfer information at highly

loaded times. Further, we do not need to do the analysis of bandwidth usage online. We should be able to collect whatever the simplest raw data necessary for bandwidth usage analysis, with minimal load on the Squid process. Then we need to take those data anywhere and analyse by the other program; may be on another computer. Ultimately we need just comparable information about the bandwidth usage of our interested users and user groups.

Therefore we finally decided to use squid logging process for sharing information because of the following advantages.

1. Squid already has a very good logging process. We used the same process for our purpose.
2. Squid logging process is customizable. We can disable all other type of logging messages and enable only our information on log files. This makes our shared information clearer on the logs.
3. Squid log file is a very simple text file. It can be used anywhere with suitable regular expressions to extract whatever the important data needed to produce our required information. Therefore we have better performance, portability and flexibility.

4.4 Analysing Results

5. Inter-Proxy Bandwidth Negotiation

Our effort in the University of Moratuwa is not to stop at the level of a single organization. Simply the reason is the existing high bandwidth network among all the Universities in the country.

5.1 Necessity of Inter-Proxy Bandwidth Communication

According to the model we described at the beginning [Fig 1-01] we need one higher level of communication for better utilization of all collaborative bandwidth. However the independency of each entity is very high at this level. Co-relation among Universities is entirely different and very loosely coupled to the controllability of staff & students within a single university. We can not assume a common unique policy for bandwidth sharing for all Universities. However most of the Universities are willing to share the bandwidth whenever it is free. At the same time they need full authority of their own bandwidth. Of course they will expect getting some additional bandwidth from other universities when they run out of bandwidth. With these issues the centralized control of bandwidth is not feasible at this level.

With expansion of universities there are internet links obtained by each department with their own budget. With this trend we see the requirement of Inter-Proxy Bandwidth Negotiation modal even within a single university.

Further any of the existing technologies does not support enough for multiple uplinks for the optimal user distribution. This is also one another issue in moving to our new Inter-Proxy Bandwidth Communication System.

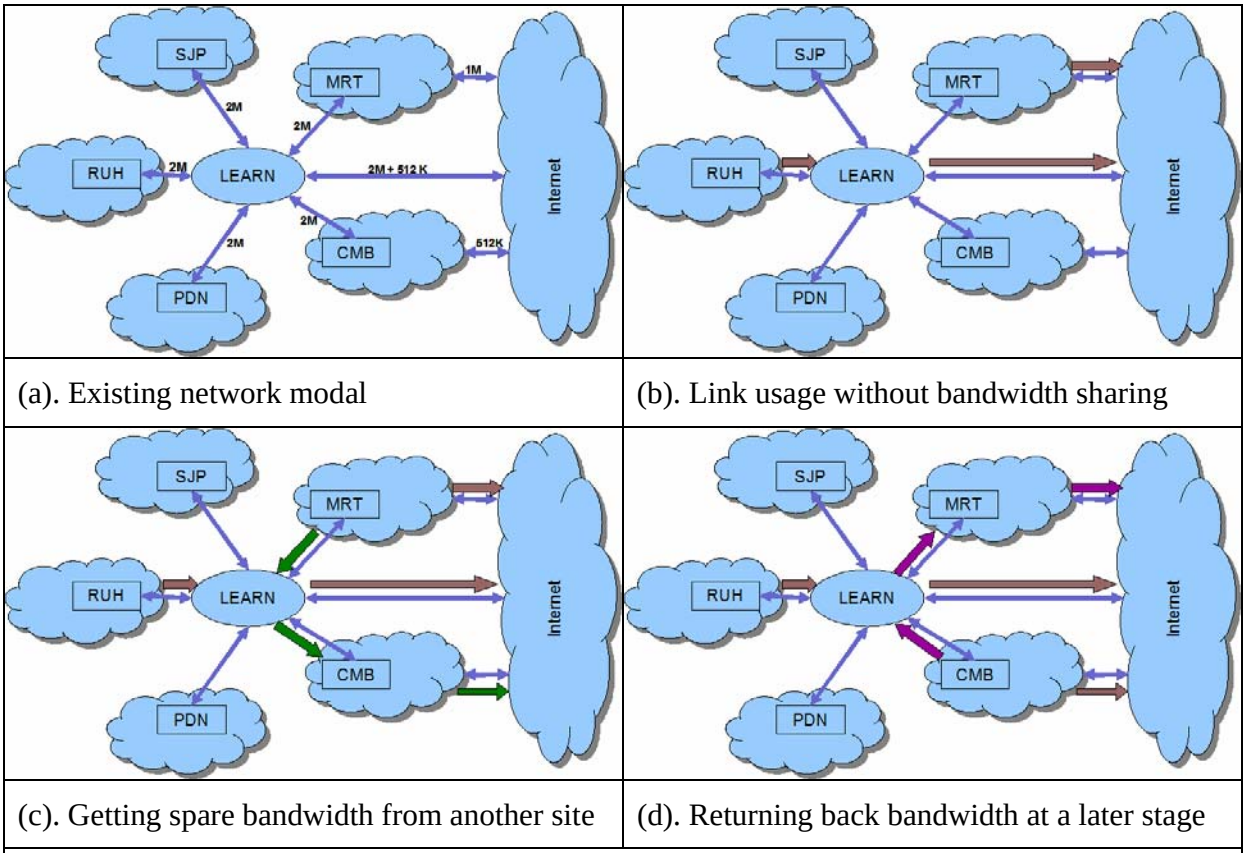
5.2 Overall System

We have multiple links to the Internet from different sites, as illustrated in Fig 5-01 (a). At the same time we have 2M capacity links among all the universities. According to current traffic among Universities these links are at under utilization. Therefore we can assume enough bandwidth for communication among Universities. Further these links are not so expensive to increase when compare to international links. According this status our assumption is that we have enough internal bandwidth, while having scarceness of international bandwidth.

5.2.1 Understanding the Functionality

If we assume the absence of our system, University system works as illustrated in Fig 5-01 (b). Here the universities use their own bandwidth links, if any. The universities do not have their own bandwidth links, share the central bandwidth provided by the LEARN system.

Most of the universities have their own bandwidth have a shared portion LEARN bandwidth as well. Current policy for distribution of the local traffic among these multiple links is to send different types of dedicated traffic through each link.



(a). Existing network modal

(b). Link usage without bandwidth sharing

(c). Getting spare bandwidth from another site

(d). Returning back bandwidth at a later stage

Fig 5-01: How Inter-Proxy Bandwidth Negotiation Works

E.g. Moratuwa University:

- Emails: LEARN shared link (64kbps)
- Web: Own link (1Mpps)

This made both links under utilization.

5.2.2 Overall Implementation

With the above background our approach is to use the under utilized bandwidth of one University by another University. To achieve this objective, our approach is to develop a bandwidth sharing protocol for proxies. Then all the proxies at each university can simply communicate with other peer proxies to request some additional bandwidth when they run out of bandwidth.

First thing we wanted in this system was to have a way of checking whether a proxy itself has enough bandwidth, spare bandwidth or run out of bandwidth. Therefore we wanted continues bandwidth monitoring system for each proxy. This sub component was identified as the bandwidth monitoring system of the system.

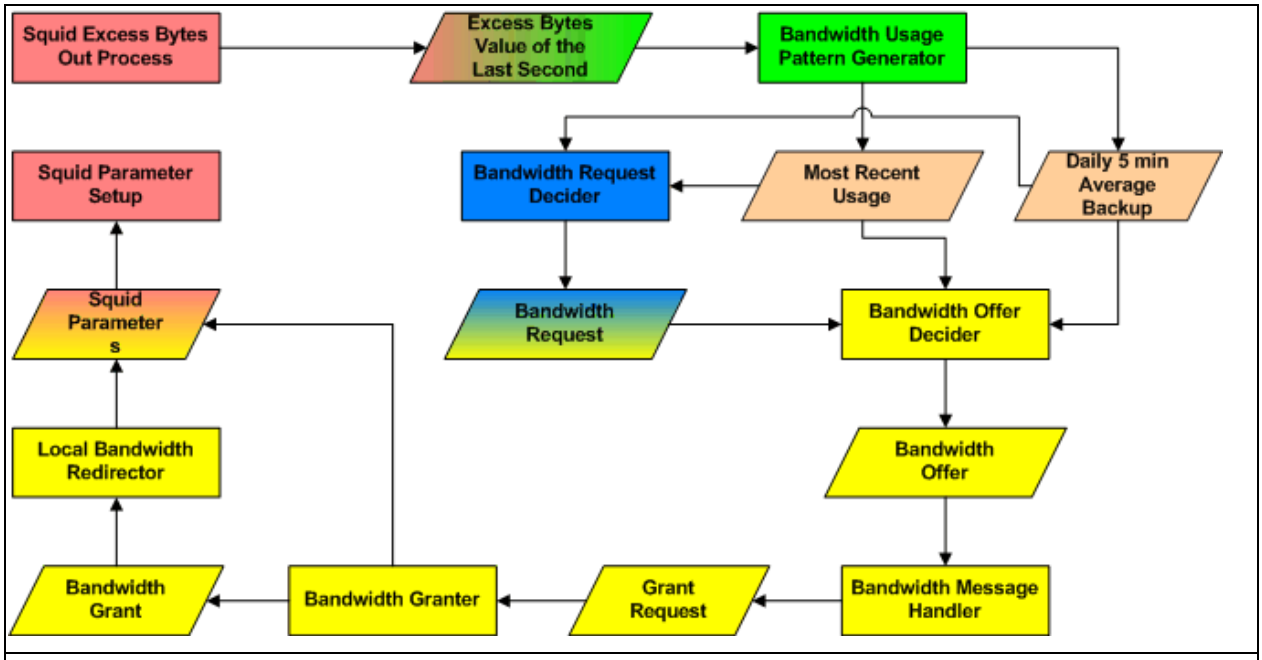


Fig 5-02: Implementation of Inter-Proxy Bandwidth Negotiation

It is not possible to request or offer bandwidth directly based on immediate bandwidth status. Specially to decide on bandwidth offers we should forecast bandwidth usage. This can be done based on the past bandwidth usage patterns. We developed a Bandwidth Usage Pattern Generator for this purpose.

Outcome of the above pattern identified is mainly fed in to two databases. One keeps the Most Recent average bandwidth Usages for:

1. Every minute of last hour
2. Every hour of last the day
3. Every day of last the month
4. Every month of the last year

Even the structure can go up-to above four levels, current implementation uses only first two levels.

The second database averages usage for each 5 minutes. Then it stores for last seven days. According to outcome of the Webalizer **[Need Fig]** we realized that there is a repeated bandwidth usage pattern at each week, of course with couple of exceptions like holidays at the middle of the week. Therefore we simply use the second database in predicting future bandwidth requirements based on the data of the same day of the last week.

Bandwidth request decider is a program triggered by low bandwidth at the proxy. Then it predicts the pattern of the bandwidth usage using Bandwidth Usage Pattern Databases. And send requests out to the neighbour proxies if needed.

All the bandwidth requests received at neighbour proxies are evaluated by the Bandwidth Offer Decider. They built in their own policies to the Bandwidth Offer Decider. Base on local policies and request parameters they send their offers.

The communication among proxies is basically handled by Inter-Proxy Bandwidth Negotiation Protocol. This ensures proper communication among proxies to do the best deal.

With the completion of successful communication of a bandwidth negotiation Bandwidth Granter allocates required bandwidth at the neighbour proxy. Bandwidth requesting proxy is managed to redirect required amount of its local traffic via the newly stabilised link.

Fig 5-02 is a simplified block diagram of the system. It shows the interconnection among the different components of the system. I have dedicated next couple of sections to describe each of above components in details.

5.3 Local Bandwidth Detection of the Proxy

Before anything for bandwidth negotiation, we wanted to have a system to check whether the proxy run out of bandwidth or running with excess bandwidth. We wanted to have a continues monitoring system, that report current bandwidth status of the local proxy server. Originally we tried out couple of methods.

1. **Use the output of the MRTG system –** MRTG (Multi-Router Traffic Graph) is one of the open source product commonly used around the world for link status/utilization monitoring. Currently we use MRTG to monitor traffic in all LEARN and university links. However due to time restrictions we did not try this method.

2. **Use roundtrip time of the ping command –** With increment of congestion of links round trip time increases. We tried this characteristic with specific destinations. This method is affected not only by our local uplink status, but also by the remote site characteristics.

3. **Using connection establishment time of the Squid proxy –** In general, connection establish time directly related to congestion level of the uplink. Therefore we tracked the connection establish time of the Squid proxy server it self. Therefore we have well self-contained the system, rather than collecting small small component from different system. Unfortunately this method was not successful. The implementation of Squid does the function in such a way, that system does not wait for the connection establishment to proceed. Therefore we can not track the connection establishment time from Squid.

4. **Use excess bytes parameter of the dynamic delay pools –** Ultimately we thought to depend on one of the internal parameter of the dynamic delay pools of the Squid. i.e. excess_bytes. This variable holds the collection of excess bytes of all the delay pools of squid. If the users of each delay pool use the bandwidth very highly, there is not contribution to the excess_bytes. Therefore we consider high excess_bytes as high

availability of bandwidth and vice versa. Even though this is not a perfect assumption, it is reliable enough to depend on it.

In our implementation we adopted above 4th method. However in our practical implementation we kept rooms to use any other bandwidth detection mechanisms too. Here we simply transfer the excess_byte value from the squid delay pools to a shared memory location at each second. Bandwidth usage pattern recording system reads values from this shared memory. If we want change the detection system, it's just a simple matter of changing the program that writes to the shared memory location [Fig 5-03].

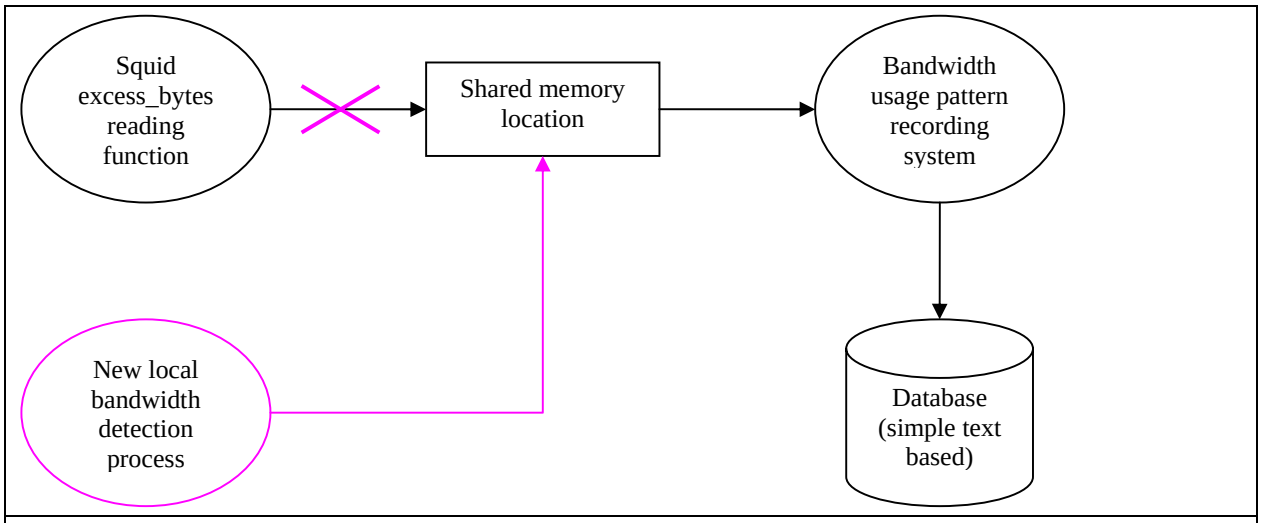


Fig 5-03: Flexibility of Local Bandwidth Detection System

5.4 Bandwidth Usage Pattern Recording

Dynamic delay pools enabled Squid is a huge process with higher memory consumption. Therefore it is important to keep the Squid process alone. We achieved this in our design by having most of data processing components as separate processes. Instance value of excess_bytes of delay pools does not have a meaning. We must keep track of historic values for a future bandwidth usage pattern prediction. Bandwidth usage pattern recording system stores historical values of instance bandwidth usage (availability) in such a way for easy, quick extraction of them for a bandwidth prediction.

We maintain two major set of bandwidth records.

1. **Recent usage pattern –** contains most recent usage behaviour of the local users.
2. **5 min summary for 7days of the week –** contains a summarized usage pattern for last 7 days.

5.4.1 Recent Usage Pattern

```
typedef struct str_bw_usage_pattern {
        float sec[60];
        float min[60];
        float hour[24];
        float day[31];
        float month[12];
        short int next_sec;
        short int next_min;
        short int next_hour;
        short int next_day;
        short int next_month;
} bw_usage_pattern;

bw_usage_pattern *bw_usage;
```

Seg 5-01: Recent Usage Pattern Structure

As you see in above code segment [Seg 5-01] we have allocated 5 floating point number arrays. There are for average bandwidth usage of last 60 seconds, 60 minutes, 24 hours and so on.

In most of our design of bandwidth detection systems (described in Section 5.3) usage of each second represents by an integer. However in pattern recording system we use a floating point array for this purpose also to keep consistency and to support future enhancement.

The above structure [Seg 5-01] supports bandwidth recording up-to 12 months. We do not use this much of levels at present. However memory and performance vice the effect of these higher levels of usage is insignificant. Therefore we keep them with the implementation to support future extendibility.

The short integer variables like next_sec, next_min are simply pointers to the current processing element of corresponding record array. These pointers are upgraded with the system clock. When ever the pointer changes at any level we update the average value for the next level [Seg 5-02].

```
// update average values for last min

bw_usage->min[bw_usage->next_min] =
        (bw_usage->min[bw_usage->next_min] * bw_usage->next_sec +
                bw_usage->sec[bw_usage->next_sec]) / (bw_usage->next_sec + 1)
```

Seg 5-02: Upgrading Upper Level of Usage Patter Records

5.4.2 Five Minute Summary for 7 Days of the Week

According to the result we got from the analysis of web traffic using Webalizer [Fig 4-02], we observed a similar access pattern repeated at each week. Even there are some exceptions in

special cases like holidays, this is a reasonable assumption. Therefore if we want to forecast web usage pattern for a certain period, we can look at the usage pattern of the same time of last week.

We keep bandwidth usage information for last 7 days for this purpose. Here we consider averages of 5 minutes. We selected this level, since we get anonymous amount of data which we cannot handle or otherwise there is not enough data for appropriate decision making.

```
typedef struct type_day_backup {
        struct hour_back {
                float fivemin[12];
        } hour[24];
} str_day_backup;

str_day_backup day_backup;
```

Seg 5-03: Five Minutes Backup Structure per Day

Code Segment [Seg 5-03] shows the structure keeps summarized data per day. We do not use prediction beyond one day in current implementation. Therefore we backup bandwidth usage status of each day in separate disk files and retrieve whenever required.

5.5 Decision Making System

Above two sections mainly discussed the procedures for tracking bandwidth usage of own local proxy and how to store them in an appropriate way for quick data retrieval. The purpose of decision making system is to determine whether local proxy is running out of bandwidth or having excess bandwidth. Based on situation it decides to request bandwidth from neighbours when it runs out of bandwidth. Otherwise it allows offer bandwidth for neighbour requests when system having excess bandwidth.

Dynamic delay pool sets its parameters every second. Inter-Proxy Bandwidth Negotiation System (IPBNS) handles the situations can not manage at that level. Therefore IPBNS checks status every minute. Whenever, it detects drop down of available excess bandwidth by a certain percentage (10% of total), system checks the bandwidth availability for last 5 minutes. If it was low for last 5 minutes IPBNS sends out a bandwidth request. Otherwise it predicts the behaviour for next 5 minutes using same time data of the last week. If the system expects any trend of running out of bandwidth, it immediately request extra bandwidth from the neighbours [Fig 5-04].

Even we say system values as 1 minute, 5 minutes and so on in this explanation; those are configurable to fit requirements of any organizations. We found that these are the reasonable values for our university system.

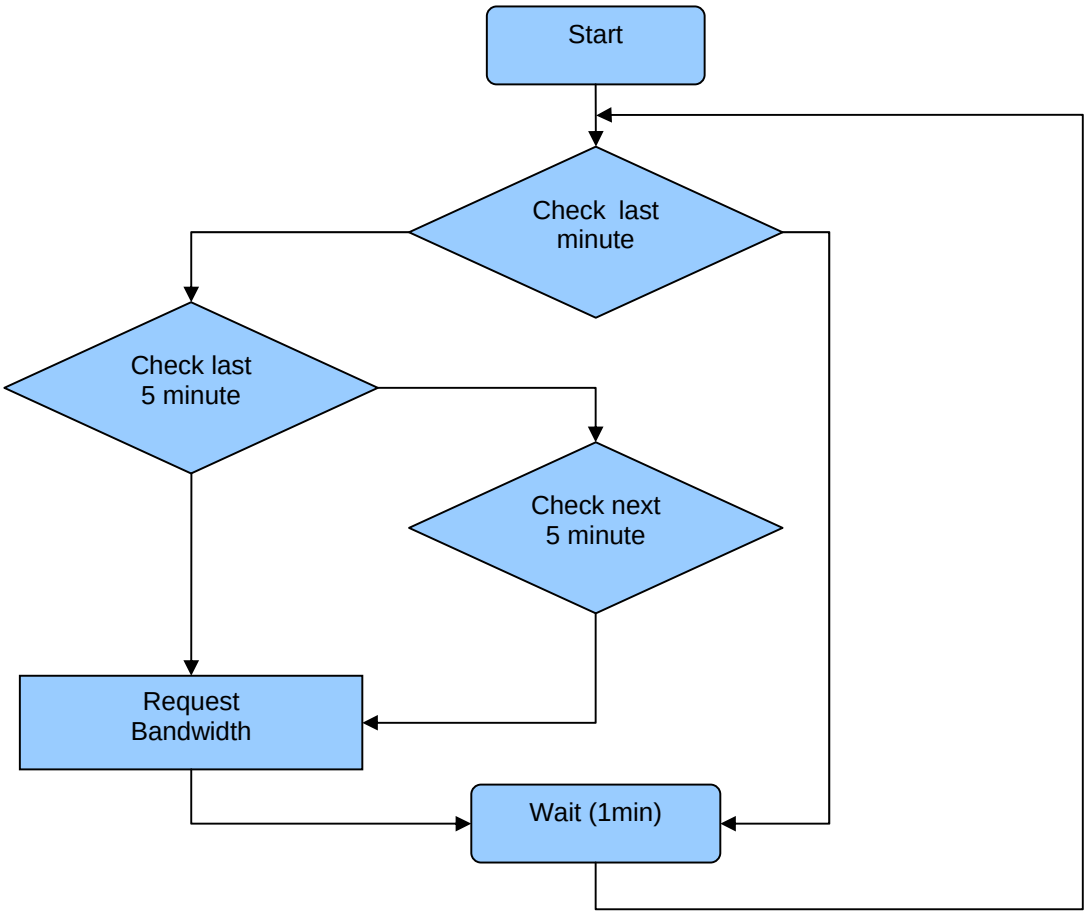


Fig 5-04: Algorithm used in Decision Making System

5.6 Inter-Proxy Communication

The next major component of my development is the Inter-Proxy Bandwidth Negotiation Protocol (IPBNP). This is where the actual communication happens among the proxy servers. IPBNP assures reliable message passing and fair bandwidth negotiation among cooperating proxies. It is basically consists of Message Handling System & Inter-Proxy Bandwidth Negotiation System. These two sub components are described bellow.

5.6.1 Message Handling System

Request Handler in Bandwidth Negotiation Subsystem is responsible for handling incoming bandwidth requests. The system analyses incoming requests and reads the requester's bandwidth requirement. Then those specifications checks against current local bandwidth usage pattern. If the system does not have excess bandwidth to offer it just ignores the message. Otherwise, system responses with its maximum feasible bandwidth offer. [Fig 5-05 (a)]

Bandwidth Offer Management

In the case of receiving a bandwidth offer, the system analyses the offer and check against current local requirements. If there is no need of any extra bandwidth it just ignores the message and proceeds. Otherwise it sends a Bandwidth Grant Request (GRT_REQ) to the bandwidth donor. [Fig 5-05 (a)]

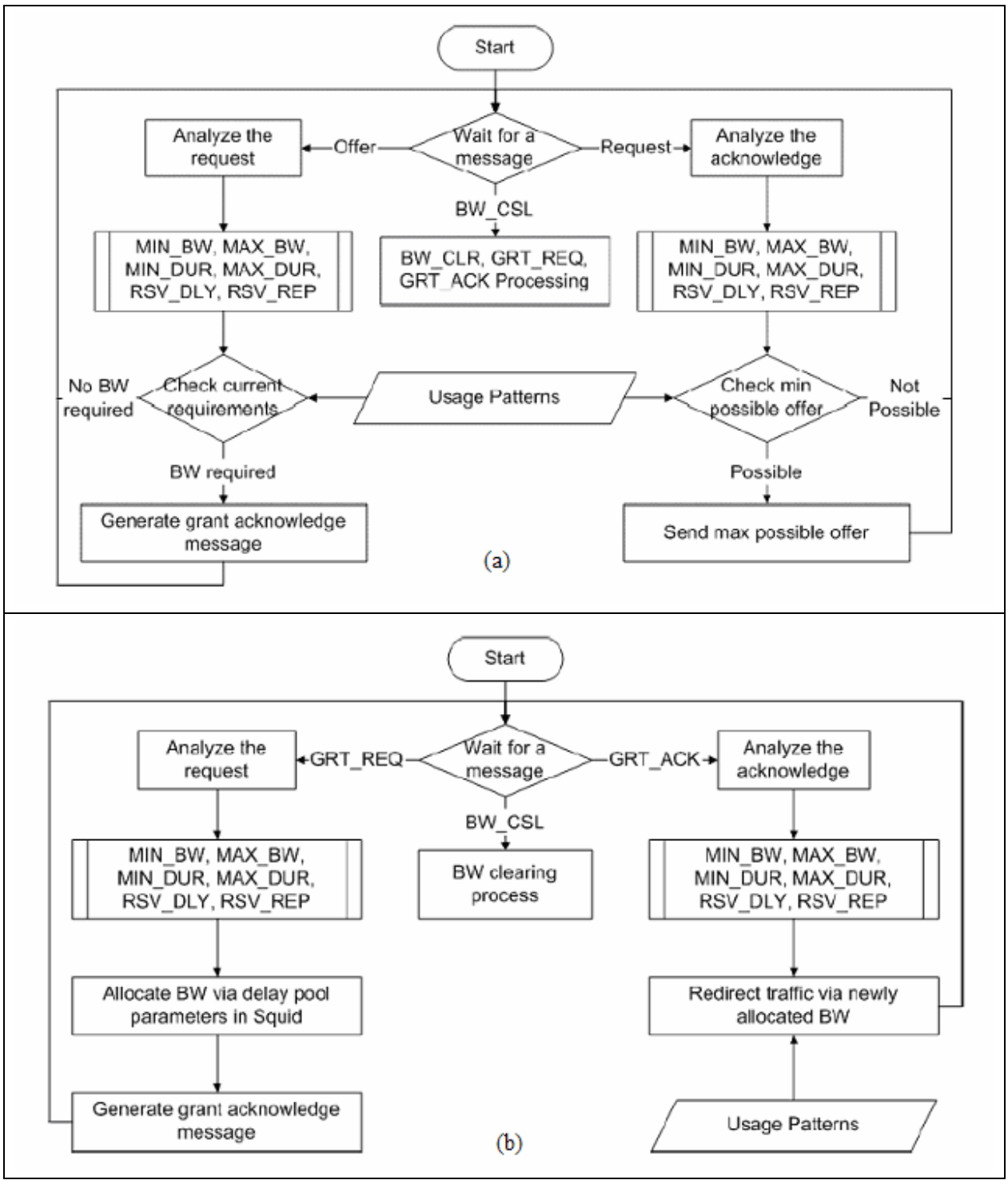


Fig 5-05: Algorithm for Bandwidth Negotiation

Bandwidth Grant Management
As a system receives a bandwidth grant request, it sets corresponding delay pool parameters, to grant corresponding bandwidth to the requested proxy and acknowledges requester back. [Fig 5-05 (b)]

Bandwidth Acknowledge Management
Reception of an acknowledgement is a final message of bandwidth negotiation. Local proxy reallocates its users in such a way as to utilize all bandwidth offers it has.

This is quite a difficult task. But depending on the current restore_bps values for the users, system can decide current usage of each user. Then the squid delay pools could modify to redirect a corresponding number of local users to the newly allocated bandwidth. The sum of average bandwidth usage (restore_bps) all directed users is kept as close as allocated bandwidth.

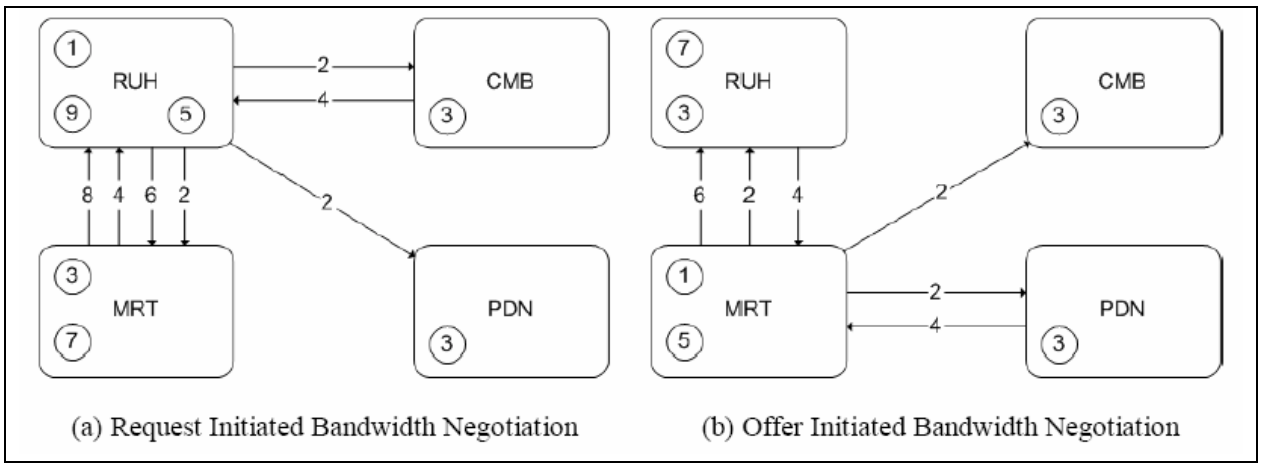
5.6.2 Inter-Proxy Bandwidth Negotiation Protocol

Depending on the decision made at Bandwidth Negotiation Subsystem proxies needs to communicate among them for searching appropriate bandwidth providers or requesters. Inter-Proxy Bandwidth Negotiation Protocol is designed in such a way as to have an efficient bandwidth negotiation among neighbour proxies.

There are three major scenarios in IPBNP. These are used for bandwidth request initiation, bandwidth offer initiation and advance reservation.

Request Initiated Bandwidth Negotiation

Here the communication starts with a running out of bandwidth at a node (proxy) (Let's assume RUH [Fig 5 (a)]).



(a) Request Initiated Bandwidth Negotiation (b) Offer Initiated Bandwidth Negotiation

Fig 5-06: Inter-Proxy Bandwidth Negotiation Protocol

1. The proxy monitors the bandwidth usage on its own by Local Bandwidth Usage Monitoring Subsystem.
2. In case of an additional bandwidth is required, RUH a sends bandwidth request to the neighbor proxies (MRT, PDN, CMB and SJP).
3. Those proxies check the availability of bandwidth on offer.
4. And send back offers only by the neighbors who have excess bandwidth which will fit request.
5. Then the requester (RUH) compares the offer it received and decides from whom to obtain bandwidth.
6. And sends requests for a bandwidth grant only to the acceptable donors (Let's say MRT).
7. Then those donors setup squid to grant agreed bandwidth to the requester.
8. And acknowledge requester.
9. Finally requester redirects its traffic via new allocated bandwidth appropriately.

Offer Initiated Bandwidth Negotiation

Here the communication starts when there are some excess bytes at a node (proxy) (Let's assume MRT [Fig 5 (a)]).

1. The proxy monitors the bandwidth usage on its own by Local Bandwidth Usage Monitoring
1. Subsystem.
2. In case of having extra bandwidth MRT sends bandwidth offer to the neighbor proxies.
3. They check whether it is suitable to get a little additional link capacity.
4. And send back request only by the neighbors who are ready to accept the offer.
5. Then the donor (MRT) compares the requests and decides the most suitable neighbor for its offer, and sets Squid parameters accordingly.
6. Then donor sends bandwidth allocation acknowledges to the donated requesters.
7. Finally requesters redirect their traffic appropriately via newly received bandwidth.

In reality offer initiated scenario is just a sub scenario of request initiated scenario. i.e. The first two steps are not required in the second case. Therefore the implementation of the second scenario is simply managed as a subcomponent of the first one.

Bandwidth Reservation in Advance

Advance reservation is simply a derivation of request or offer initiated negotiation except the specific fact that real bandwidth allocation is happened at a later time. Even though the reservation can be done in an offer initiated manner, it does not have practical applications as much as a request initiated one.

However Inter-Proxy Bandwidth Negotiation Protocol supports both type of advance reservation.

We realized the potential in the practical use of advance bandwidth reservation with modern Internet application. As an example in the university system most video conferences are planned in advance. They generally need guaranteed constant bandwidth for the specific period. It is essential to make sure of the availability of sufficient bandwidth for this application even though other universities experience a bit slower Internet access for the specific time period.

Message Format

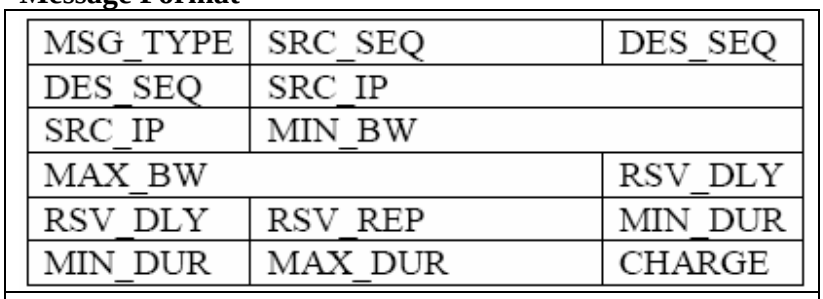


Fig 5-07: Message Format

IPBNP uses simple, small and fixed length message for inter proxy communication since the bandwidth usage in negotiation should be small. Fig 5-07 above shows all the fields of the message which have following meanings:

MSG_TYPE (1 Byte) - Type of the message can be one of the following:

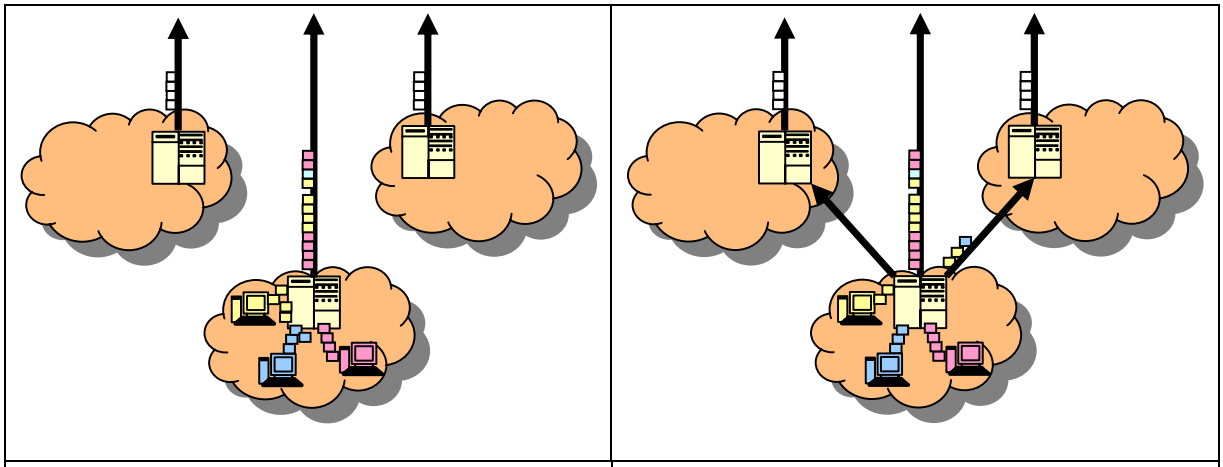
1. BW_REQ - Bandwidth Request
2. BW_OFR - Bandwidth Offer
3. BW_CSL - Bandwidth Cancellation
4. GRT_REQ - Grant Request
5. GRT_ACK - Grant Acknowledge

SRC_SEQ (2 Bytes) - Source Sequence generated by the message source in order to keep track of the issued messages and corresponding responses.
DES_SEQ (2 Bytes) - Set to the original Sequence number received from the source on a reply to a particular message.
SRC_IP (4 Bytes) - Source IP can be used to identify the proxy from which message originated.
MIN_BW (3 Byte) - Minimum Bandwidth requested or offered, specified in bytes.
MAX_BW (3 Byte) - Maximum Bandwidth requested or offered specifies in bytes. The value 0 sets MAX_BW to as much as possible, but not less than MIN_BW.
RSV_DLY (2 Bytes) - Reservation Delay in minutes. This value says how many minutes later the real bandwidth allocation happens. The maximum possible latency is about 45 days.
RSV_REP (2 Bytes) - Reservation Repetition is also specified in minutes.
MIN_DUR (2 Bytes) - Minimum Duration for the bandwidth negotiation which has been specified in minutes.
MAX_DUR (2 Bytes) - Maximum Duration for the bandwidth negotiation is also specified in minutes. Setting this value to 0 means allocation is valid until either side (requester or donor) cancels it. However if the MIN_DUR is set duration should not be less than that value.

CHARGE (1 Bytes) - Anyone is not ready to give any thing free. There should be a possibility for charging (even not always monetary). CHARGE parameter allows proxies to assign particular value for their service. This field is left for the support future developments.

5.7 Bandwidth Receiver End: Redirecting Local Traffic

A proxy gets an additional bandwidth from another organization at the end of proper communication with its neighbours as described in the above section. The next problem wanted to address was, *how Squid can effectively redirect its local users* through these additional and existing own bandwidth links while changing the number of links and their capacities. This section describes the mechanism used achieve this objective.



(a) Sending traffic only though own link | (b) Sending traffic via multiple uplinks

Fig 5-08: Redirection of Local Traffic via Multiple Uplinks

Fig 5-08 (a) shows the general functionality of Squid. Each proxy uses its own bandwidth link. The improved system behaviour is demonstrated on Fig 5-08 (b). Uplink selection was done using the *Parent Cache* concept of the Squid [29]. *Parent Cache* allows squid to send its queries through another proxy.

```
1.      // Proposed methodology of optimal uplink sharing
2.      //
3.      // 1. Is this the starting phase of the link usage?
4.      //    - p->stPhase = 1
5.      //    - If so we fillup to 75% of the link
6.      // 2. What is the current usage of the link
7.      //    - linkUsage = ??%
8.      //    - If stPhase then
9.      //              - If linkUsage < 75% then
10.     //                - If !(request->ip in p->userlist) then
11.     //                      - add(request-ip, p->userlist)
12.     //              - else
13.     //                - p->stPhase = 0
14.     //                - If !(reqest->ip in p->userlist) then
```

```
15.     //                      - return 0
16.     //        - else
17.     //              - If (linkUsage < 50%) then
18.     //                - If !(request->ip in p->userlist) then
19.     //                      - add(request->ip, p->userlist)
20.     //              - else if (linkUsage < 90%) then
21.     //                - if !(request->ip in p->userlist) then
22.     //                      - return 0
23.     //              - else
24.     //                - removeLast(p->userlist)
25.     //                - if !(request->ip in p->userlist) then
26.     //                      - return 0
```

Seg 5-04: Five Minutes Backup Structure per Day

Then the question is how to select the local users to redirect through the newly established link. Seg 5-04 shows the algorithm we used for this. A queue named *userlist* is used here to keep track of users redirected through each link. Then that queue is filled until the usage (*linkUsage*) reaches 75% of the link. If the link usage was dropped bellow 50%, new users are added to the queue to reach 75% usage of the link. If it was raise beyond 90%, users are removed from the queue.

5.8 Bandwidth Donor End: Allocation for the Requesters

6. Other Approaches

6.1 tc

7. Results & Further Works

7.1 Multi-disciplinary Decision Making

7.2 Inter-Proxy Communication via Multicast

8. Conclusion

References

[1] Home Page, Squid Web proxy Cache: January 2003, http://www.squid-cache.org

[2] Gihan Dias & Chamara Gunaratne, "Using Dynamic Delay Pools for Bandwidth Allocation", Department of Computer Science, University of Moratuwa, Sri Lanka.

[3] Offline downloading applications??

[4] High guaranteed bandwidth applications, Current trend of shifting bandwidth usage to real time applications??

[5] Limited bandwidth problem of the region. We have to live with it for another four to five years??

[6] Chamara Disanayake, Gihan Dias and C. R. de Silva, "A Market-Based approach to control Web bandwidth Usage", accepted to APAN 2004.

[7] The change of bandwidth usage from peak hours to off-peak hours.??

[8] Shantha sir paper??

[9] Paper discussing the Bandwidth limitation problems in the South-Asian & Latin-American regions??

[10] Paper describing firewall functionality of the proxy server??

[11] A fully descriptive documentation for squid cache??

[12] Whole list of features of the proxy server??

[13] Original delay pool configuration in squid??

[14] Different approaches in bandwidth optimization??

[15] Pradee's paper??

[16] One another view point to utilize the bandwidth??

[17] Managing bandwidth at the proxy level??

[18] Advantages of using a proxy server for bandwidth management??

[19] ACL description of squid??

[20] Delay Pools in squid ??

[21] Andrew S. Tanembaum "Computer Networks" 3rd Edition pp. 381–384.

[22] Squid proxy cache configuration file (squid.conf)??

[23] Improvements of squid from version 2.4 to 2.5??

[24] Details of Squid 3.0 including the statement saying it is a C++ implementation??

[25] Squid web based administrative interface??

[26] MRTG FAQ Website http://faq.mrtg.org/ Referred on 01st February, 2005.??

[27] Webalizer Homepage http://www.mrunix.net/webalizer/ Referred on 01st February, 2005??

[28] Other Squid performance monitoring tools??

[29] Squid Parent Cache concept description???

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

[31] Cho-Yu Chiang, Mikihiro Ueno, Ming T. Liu, Mervin E. Muller "Modeling Web Caching Hierarchy Schemes", Department of Computer and Information Scinece, The Ohio State University, Columbus, Ohio pp. 20–p28.

[32] D. Wessels and K. Claffy, "Internet Cache Protocol" and "Application of ICP", RFC 2186 and RFC 2187.

[33]

Appendices

App 01: Delay Pools Setting in Squid

```
# DELAY POOL PARAMETERS (all require DELAY_POOLS compilation option)
# -----------------------------------------------------------------------------

#  TAG: delay_pools
#     This represents the number of delay pools to be used.  For example,
#     if you have one class 2 delay pool and one class 3 delays pool, you
#     have a total of 2 delay pools.
#
#Default:
# delay_pools 0

#  TAG: delay_class
#     This defines the class of each delay pool.  There must be exactly one
#     delay_class line for each delay pool.  For example, to define two
#     delay pools, one of class 2 and one of class 3, the settings above
#     and here would be:
#
#Example:
# delay_pools 2      # 2 delay pools
# delay_class 1 2    # pool 1 is a class 2 pool
# delay_class 2 3    # pool 2 is a class 3 pool
#
#     The delay pool classes are:
#
#           class 1           Everything is limited by a single aggregate
#                       bucket.
#
#           class 2     Everything is limited by a single aggregate
#                       bucket as well as an "individual" bucket chosen
#                       from bits 25 through 32 of the IP address.
#
#           class 3           Everything is limited by a single aggregate
#                       bucket as well as a "network" bucket chosen
#                       from bits 17 through 24 of the IP address and a
#                       "individual" bucket chosen from bits 17 through
#                       32 of the IP address.
#
#     NOTE: If an IP address is a.b.c.d
#           -> bits 25 through 32 are "d"
#           -> bits 17 through 24 are "c"
#           -> bits 17 through 32 are "c * 256 + d"
#
#Default:
# none

#  TAG: delay_access
#     This is used to determine which delay pool a request falls into.
#     The first matched delay pool is always used, i.e., if a request falls
#     into delay pool number one, no more delay are checked, otherwise the
#     rest are checked in order of their delay pool number until they have
#     all been checked.  For example, if you want some_big_clients in delay
```

```
#     pool 1 and lotsa_little_clients in delay pool 2:
#
#Example:
# delay_access 1 allow some_big_clients
# delay_access 1 deny all
# delay_access 2 allow lotsa_little_clients
# delay_access 2 deny all
#
#Default:
# none

#  TAG: delay_parameters
#     This defines the parameters for a delay pool.  Each delay pool has
#     a number of "buckets" associated with it, as explained in the
#     description of delay_class.  For a class 1 delay pool, the syntax is:
#
#delay_parameters pool aggregate
#
#     For a class 2 delay pool:
#
#delay_parameters pool aggregate individual
#
#     For a class 3 delay pool:
#
#delay_parameters pool aggregate network individual
#
#     The variables here are:
#
#           pool        a pool number - ie, a number between 1 and the
#                       number specified in delay_pools as used in
#                       delay_class lines.
#
#           aggregate   the "delay parameters" for the aggregate bucket
#                       (class 1, 2, 3).
#
#           individual  the "delay parameters" for the individual
#                       buckets (class 2, 3).
#
#           network           the "delay parameters" for the network buckets
#                       (class 3).
#
#     A pair of delay parameters is written restore/maximum, where restore is
#     the number of bytes (not bits - modem and network speeds are usually
#     quoted in bits) per second placed into the bucket, and maximum is the
#     maximum number of bytes which can be in the bucket at any time.
#
#     For example, if delay pool number 1 is a class 2 delay pool as in the
#     above example, and is being used to strictly limit each host to 64kbps
#     (plus overheads), with no overall limit, the line is:
#
#delay_parameters 1 -1/-1 8000/8000
#
#     Note that the figure -1 is used to represent "unlimited".
#
#     And, if delay pool number 2 is a class 3 delay pool as in the above
#     example, and you want to limit it to a total of 256kbps (strict limit)
#     with each 8-bit network permitted 64kbps (strict limit) and each
```

```
#     individual host permitted 4800bps with a bucket maximum size of 64kb
#     to permit a decent web page to be downloaded at a decent speed
#     (if the network is not being limited due to overuse) but slow down
#     large downloads more significantly:
#
#delay_parameters 2 32000/32000 8000/8000 600/8000
#
#     There must be one delay_parameters line for each delay pool.
#
#Default:
# none

#  TAG: delay_initial_bucket_level  (percent, 0-100)
#     The initial bucket percentage is used to determine how much is put
#     in each bucket when squid starts, is reconfigured, or first notices
#     a host accessing it (in class 2 and class 3, individual hosts and
#     networks only have buckets associated with them once they have been
#     "seen" by squid).
#
#Default:
# delay_initial_bucket_level 50
```