# DESIGN AND IMPLEMENTATION OF CLASS BASED QUEUING IN PROXY SERVER

**A DISSERTATION**

***Submitted in partial fulfilment of the requirements for the award of the degree***

***of***

MASTER OF TECHNOLOGY

***in***

INFORMATION TECHNOLOGY

***By***

**GAURAV JAIN**

*ER & DCI NOIDA*

**IIT Roorkee-ER&DCI, Noida**
**C-56/1, "Anusandhan Bhawan"**
**Sector 62, Noida-201 307**
FEBRUARY, 2003

Enrolment No. 019017

621.380285
JAI

## CANDIDATE'S DECLARATION

I hereby declare that the work presented in this dissertation titled **"Design & Implementation of Class Based Queuing in Proxy Server"**, in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Information Technology,** submitted in **IIT, Roorkee – ER&DCI Campus, Noida,** is an authentic record of my own work carried out during the period from August 2002 to February, 2003 under the guidance of **Mr. B. K. Das,** Scientist-C, Internet Division, N.I.C. New Delhi and **Mr. Munish kumar,** Project Engineer, ER&DCI, Noida.

The matter embodied in this dissertation has not been submitted by me for award of any other degree or diploma

Date : 21/02/03

Place: Noida

[illegible]

**(Gaurav Jain)**

## CERTIFICATE

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

[illegible]

**(Mr. Munish Kumar)**

Project Co-Guide

Project Engineer,

ER&DCI, Noida,

[illegible]

Date: 21-02-03

Place: Noida

[illegible]

**(Mr. B. K. Das)**

Project Guide

Scientist-C,

Internet Division,

N.I.C., New Delhi.

Date: 21/02/03

Place: New Delhi

## ACKNOWLEDGEMENT

First of all I would like to pay my humble regards to **Prof. Premvrath**, Director IIT-Roorkee, **Mr. R. K. Verma,** Executive Director ER&DCI-Noida, **Mr. H. M. Reddy,** Technical Director, Internet Division, N.I.C. New Delhi, for providing me with this valuable opportunity to carry out this work. I am also very grateful to **Dr. A. K. Awasthi**, Programme Director, M.Tech (I.T) & Dean PG S&R, IIT-Roorkee, **Prof. R.P. Agarwal,** Course-Coordinator M.Tech (I.T.), IIT-Roorkee, **Mr. V. N. Shukla,** Course-Coordinator M.Tech. (I.T.), ER&DCI-Noida for their support and help generously extended to me without which this project could not have been completed in time.

I owe a special sense of gratitude to my honorable guide **Mr. B. K. Das**, Scientist C, Internet Division N.I.C., New Delhi and my respected co-guide **Mr. Munish Kumar,** Project Engineer, ER&DCI-Noida for their prodigious guidance, painstaking attitude and giving reformative suggestions throughout my project

My sincere thanks are due to **Dr. (Ms) P.R. Gupta**, Reader, ER&DCI, Noida for the encouragement and valuable suggestions she provided me during the course of my project work.

Above all I would like to thank my family. My parents provided me a perfect environment for my studies and supported me throughout. Finally, I would like to extend my gratitude to all those persons who directly or indirectly helped me in the process and contributed towards this work.

Gaurav Jain

**(Gaurav Jain)**
Enrollment no: 019017

# CONTENTS

# ABSTRACT

Growth of Internet as a fundamental network has resulted in very slow Internet access. This situation is further exacerbated by a small number of users who use a disproportionate share of the available bandwidth for file downloads, etc., starving other users of bandwidth. Therefore some means of controlling and managing the bandwidth available to users is necessary. We can manage bandwidth by classifying traffic in to various categories and allocating each class a portion of total available bandwidth, thus we can restrict high bandwidth consuming applications like movie, real audio, MP3 download to a limit, as specified by network administrator. Most organization provides web access to end users through cache/proxy servers. As all web accesses are channeled through the cache server, using the same server for bandwidth management is expedient. Squid is a popular proxy and web cache server, which contain a bandwidth management feature called "delay pools". However the existing implementation of delay pools did not provide satisfactory performance, as bandwidth allocations are static and does not adept to varying usage patterns. Our solution adjusts the bandwidth allocated to each class to provide an optimum level of access to all users. During off-peak hours, each class of application can use essentially the full available bandwidth, while during peak hours it's bandwidth is limited to min value as specified by network administrator. Thus light interactive users get better performance at the expense of heavy and non-interactive users.

Chapter1

# INTRODUCTION

Internet access in National Informatics Center, like many other organizations, is constrained by the cost of available bandwidth. Although the speeds of LANs are in the order of a Gb/s, the high cost of Internet bandwidth often means that dozens of users share few a Mb/s of bandwidth.

Providing enough bandwidth to satisfy all users is financially infeasible, so I approached the problem from the angle that bandwidth is a fixed resource, which should be managed to yield the maximum benefits. Some consider that the cost of bandwidth is dropping rapidly, therefore this problem will only be temporary. However, observations are, that although the cost of bandwidth is reducing. At the same time, the demand for bandwidth is increasing. The net result has remained an insufficiency of bandwidth, and can be expected to be so in the future.

## 1.1 Objective of Thesis

Mostly web access is provided through cache/proxy servers. As all web accesses are channeled through the cache server, using the same server for bandwidth management is expedient. For achieving the goal of providing fair access to all users, we have to:

- Shape traffic by the use of Class Based Queuing at proxy server, which in turn requires classification of Internet traffic in to different classes based on some "magic word" present in URL such as .mp3, .avi, .html, .jpeg etc. and allocate each class a portion of total available bandwidth.

- Secondly we have to derive a mechanism so that bandwidth allocated to each class could vary according to changing usage pattern and provide a optimum level of access to all the users.

As a result of which, during off-peak hours, each class of applications could use essentially the full available bandwidth, while during peak hours they are limited to the minimum level of bandwidth set by the network administrator. Thus light interactive users get better performance at the expense of heavy and non-interactive users.

## 1.2 Scope

Design and Implementation of Class Based Queuing is a live project at National Informatics Center – New Delhi, they are developing hardware for it by implementing it in router, but I have implemented it in a proxy/cache server.

Tools for bandwidth management are required because Internet is decentralized in nature, composed of multiple administrative domains with wide range of resource limitations, the control of Internet resources involve the local decisions, thus measures for providing a fair quantity of bandwidth for each application requires link shearing at users place only.

By implementing Class Based Queuing in proxy server we could prevent few users from acquiring a unequal fraction of available bandwidth, and thus provide fair access to all users.

## 1.3 Overview

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

When bandwidth is precious (as it is always) and it is not possible to meet exact bandwidth needs, traffic shaping procedure could be used by implementing a class based queuing of outgoing Internet request, thus ensuring proper utilization of available resources.

In most of the organizations, a proxy server is used to give access to network services; this means that if you are sitting at networked computer that is connected to Internet via proxy server, then when you start a web browser and point it to www.iitr.ac.in, then your browser sends request to proxy server, proxy server retrieves the web page on your behave and send it to your computer [1].

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

We have decided to use squid proxy server because Squid is the only proxy server that implements a bandwidth management mechanism called "Delay Pools", written by David Luyer [3]. Using this delay pools, it is possible to limit Internet traffic in a reasonable way, depending on so-called 'magic words', existing in any given URL for example, a magic word could be '.mp3', '.exe' or '.avi', etc. Any distinct part of a URL (such as .avi ) can be defined as magic word.

With that, we can classify the in coming requests in to different classes and assign each class a different delay pool thus a different bandwidth.

Thus we can tell squid to download certain kinds of files (like sound files) at a specified speed (in our example, it will be about 5 Kbytes/s). If our LAN users download files at the same time, they will be downloaded at about 5 Kbytes/s altogether, leaving remaining bandwidth for web pages, e-mail, news, etc. Thus provides fair access to other users and avoids conges ion in the line.

But major drawback with delay pool is that delay pools parameters are statically defined in the Squid configuration file and cannot be changed dynamically to suit the changing conditions in the network. i.e. if a portion of available bandwidth to a class of applications is not fully utilized then in that case it leads to the wastage of the precious bandwidth allocated to that class.

Therefore I modified the freely available source code so that bandwidth allocated to each application could be adjusted according to the varying usage pattern and an optimum level of access could be provided to each user.

## 1.4 Organization of the Thesis

Chapter 2 Literature Survey gives a brief introduction about TCP/IP architecture, IP addressing and address Subnetting. Along with this, I also discuss basic concepts of proxy server along with their usages.

Chapter 3. Provides in-depth analysis of the problem posed before me along with design of the solution proposed by me.

Chapter4 Discuss implementation aspects to transfer proposed solution in to machine-readable code in Chapter 5 I have provided some of the results obtained with a brief discussion about the results obtain.

Finally I conclude my thesis in chapter 6.

Chapter 2

# LITERATURE SURVEY

Internet a worldwide network that is widely used to connect universities, government offices, companies, and of late, private individuals, is successor of the ARPANET.

APRANET was a research network sponsored by DoD (U.S. Department of Defense). It eventually connected hundreds of universities and government installations using leased telephone lines. When satellite and radio networks are added later, the existing protocols had trouble internetworking with them, so new reference architecture was needed. Thus the ability to connect multiple networks together in a seamless way was one of the major design goals from the very beginning. The architecture later becomes known as TCP/IP reference model, after it's two primary protocols, TCP (Transmission Control Protocol) and IP (Internet Protocol).

Given the DoD's worry that some of it's precious hosts, routers, and internet work gateways might get below to pieces at a moment's notice, another major goal was that the network be able to service loss of subnet hardware, with existing conversation not being broken off. In other words, DoD wanted connection to remain intact as long as the source and destination machine were functioning, even if some of the machines or transmission lines in between were suddenly put out of operation. Furthermore, a flexible architecture was needed, since applications with divergent requirements were envisioned, ranging from transferring files to real time speech transmission.

## 2.1 TCP/IP Overview

Before we examine each layer of TCP/IP, we should cover a few basic concepts. At its most basic level, a computer network is simply a series of connections between computers that allow them to communicate. The content, scope, size, speed, and reliability of the network vary depending on its protocols and implementation. Protocols are pre-established means of communication. The term TCP/IP (Transmission Control Protocol/Internet Protocol) actually refers to a whole family of protocols, of which TCP and IP are just two. Figure 2.1 contains the standard "stack" diagram of TCP/IP. Rather than make protocols monolithic (which would mean ftp, telnet, and gopher would each have a full network protocol implementation, including separate copies of kernel code for the devices each protocol uses), the designers of TCP/IP broke the job of a full network protocol suite into a number of tasks. Each layer corresponds to a different facet of communication. Conceptually, it is useful to envision TCP/IP as a stack. In implementations, programmers often blur the layers for increased performance.

| | |
|---|---|
| Application | Telnet, FTP, RPC, etc. |
| Transport | TCP, UDP |
| Network | IP, ICMP, IGMP |
| Link | Network interface and device driver |

**Figure 2.1: The Layers of the TCP/IP Protocol Suite**

The first, the link layer, is responsible for communicating with the actual network hardware (e.g., the Ethernet card). Data it receives off the network wire it hands to the network layer; data it receives from the network layer it puts on the network wire. This is where device drivers for different interfaces reside.

The second, the network layer, is responsible for figuring out how to get data to its destination. Making no guarantee about whether data will reach its destination, it just decides where the data should be sent.

The third, the transport layer, provides data flows for the application layer. It is at the transport layer where guarantees of reliability may be made.

The fourth, the application layer, is where users typically interact with the network. This is where telnet, ftp, email, IRC, etc. reside.

Packets are the basic unit of transmission on the Internet. They contain both data and header information. Simply put, headers generally consist of some combination of checksums, protocol identifiers, destination and source addresses, and state information. Each layer may add its own header information, so it can interpret the data the lower layer is handing it. In Figure 2.2, we see a sample Ethernet frame. This is the product of a packet that has gone from that application layer all the way to the link layer. Each layer takes the previous layer's packet, viewing almost all of it as data, and puts its own header on it.

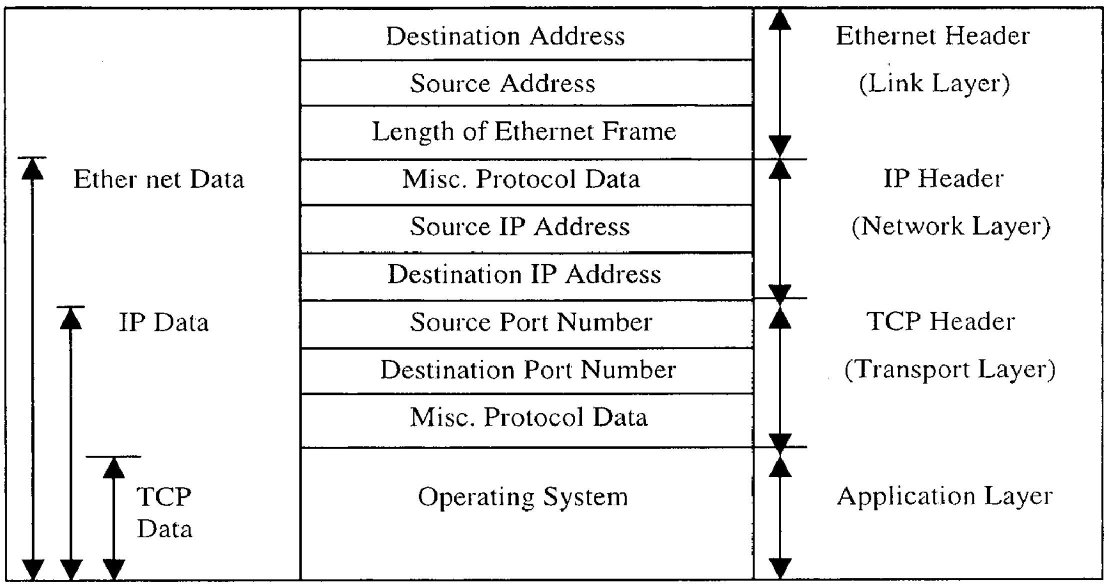


**Figure 2.2: A Sample Ethernet Frame**

### 2.1.1 The Link Layer

The link layer is the simplest layer to understand. Composed of the network hardware and the device drivers, the link layer is the lowest level of the protocol stack. When receiving data from the network, it takes packets from the network wire, strips away any link layer header information, and hands it off to the network layer. When transmitting data onto the network, it takes packets from the network layer, sticks a link layer header on them, and sends them out over the wire.

The benefit of separating out the hardware layer is that protocol implementers only have to write the network layer once. Then they provide a common interface to the network layer by writing different device drivers for each kind of network interface.

### 2.1.2 The Network Layer

This is where the Internet Protocol (IP) and the Internet Control Message Protocol (ICMP), among others, reside. ICMP is used to provide network reliability information by utilities like ping and trace-route. IP is used for almost all other Internet communication. When sending packets, it is figures out how to get them to their destination; when receiving packets, it figures out where they belong. Because it does not worry about whether packets get to where they are going or whether they arrive in the order sent, its job is greatly simplified. If a packet arrives with any problems (e.g., corruption), IP silently discards it. Upper layers are responsible for insuring reliable reception of packets. We refer to IP's behavior as ''stateless" or ''connectionless" because the existence of previous or future packets is irrelevant when processing the current packet. We could unplug the network wire, wait a minute, plug it back in, and IP would never know the difference.

IP is able to get packets to their destinations because every network interface on the Internet has a unique, numeric address. Oddly enough, these numbers are called IP addresses. Notice, every interface has its own address. If a machine has multiple interfaces (as is the case with a router), each one has its own IP address. The Internic is responsible for assigning sets of addresses to organizations, thereby insuring uniqueness.

Because it's a pain to refer to machines with strings of numbers, the designers of TCP/IP allowed network administrators to associate names with IP addresses. And so was born the domain name system (DNS). It is a distributed database of IP addresses and their natural language names, called host names. In fact one IP address can have multiple names associated with it.

The steps IP takes to send a packet are simple: based on its IP address, figure out how to get it there and send it on its way.

Deciding out how to get the packet there, routing, is the critical task for IP. Fortunately, sender doesn't have to know how to get a packet all the way to reciever; it just needs to figure out which local router is responsible for getting packets to receiver. A router differs from a typical machine on the net because it has at least two network interfaces -- this allows it to connect to two or more networks. For a small organization, there will typically be a local network (e.g., Ethernet) and then a leased-line link to the Internet. The organization's router is connected to both the local network and the Internet link. All packets bound for the Internet are sent to the router, which then puts it on the leased line, bound for the next router.

Each router only needs to know about the routers to which it is connected. To determine where a given packet will go next, machines on the Internet maintain routing tables. They consist of three major items: addresses of routers, addresses they can handle, and the interface to which they are connected. In the case of a machine on a local net , it probably has three entries: one for the loop back interface (which allows a host to connect to itself), one for the local network and a default entry.

The local network entry lets IP know that the machine is directly connected to a certain set of IP addresses. Rather than try and route those packets, IP figures out the hardware address of the Ethernet interface to which the IP address corresponds and sends the packet there. With this entry, that machine is essentially the router, the addresses it can handle are all the IP addresses on the local net, and the destination interface is an Ethernet card on the local net.

The default entry says ''for all other addresses, send it to this router." Instead of trying to deliver a packet for a machine on foreign network to a machine on the local net, host computer sends it to the router's interface, saying, ''Here, I don't know where this goes, you figure it out." The router then looks at its table, sees it doesn't have a direct connection to concerned machine, and sends it to its default destination. And so the process continues.

When receiving data, IP takes the packet from the link layer, checks for any blatant corruption, and hands the packet to the proper process at the transport layer. If there is any problem with the packet, IP silently discards it because it doesn't have to worry about whether a packet reaches its destination.

### 2.1.3 The Transport Layer

There are two protocols at the transport layer: the Transmission Control Protocol (TCP) and the User Data-gram Protocol (UDP). TCP provides end-to-end reliable communication and UDP doesn't. UDP is as unreliable as IP, but allows people to write user level software that creates its own packet formats, which is particularly helpful if you want to write new protocols, don't have the kernel sources, and don't want the overhead of TCP.

TCP creates a ''virtual circuit" between two processes. It insures that packets are received in the order they are sent and that lost packets are retransmitted. We won't go into the details of how it works, but interactive programs like ftp and telnet use it.

So far we have discussed addressing on the host level -- how to identify a particular machine. But once at a machine, we need a way to identify a particular service (e.g., mail). This is the function of ports -- identification numbers included with every UDP or TCP packet. TCP/IP ports are not hardware-based. They are a just a way of labeling packets. A process on a machine ''listens" on a particular port. When the transport layer receives a packet, it checks the port number and sends the data to the corresponding process. When a process starts up, it registers a port number with the TCP/IP stack. Only one process per protocol can listen on a given port. So while a process using UDP and one using TCP can both listen on port 111, two processes that both used TCP could not. There are a number of ports that are reserved for standard services. For example, SMTP, the mail protocol, is always on port 25, and telnet-data is always on port 23. We've examined how ports work on the server end -- specific ports are reserved for set tasks. On the initiator end, port assignment is dynamic. When a telnet client starts up, it gets a new port number (e.g., 1066). This is the source port which client's TCP layer puts on every packet. This allows the telnet daemon on server to responds to the correct telnet process on client side. The combination of source/destination IP addresses and ports provides a unique conversation identifier. Each conversation is called a flow.

## 2.2 IP Addressing

In order to deliver IP packet to a machine each machine on network has an IP address. An IP address is a 32 bit binary number usually represented as 4 decimal values,

each representing 8 bits, in the range 0 to 255 (known as octets) separated by decimal points. This is known as "dotted decimal" notation.
Example: 140.179.220.200

It is sometimes useful to view the values in their binary form.
140 .179 .220 .200
10001100.10110011.11011100.11001000

Every IP address consists of two parts, one identifying the network and one identifying the node. The Class of the address and the subnet mask determine which part belongs to the network address and which part belongs to the node address.

### 2.2.1 Address Classes

There are 5 different address classes. You can determine which class any IP address is in by examining the first 4 bits of the IP address.

Class A addresses begin with 0xxx, or 1 to 126 decimal.
Class B addresses begin with 10xx, or 128 to 191 decimal.
Class C addresses begin with 110x, or 192 to 223 decimal.
Class D addresses begin with 1110, or 224 to 239 decimal.
Class E addresses begin with 1111, or 240 to 254 decimal.

Addresses beginning with 01111111, or 127 decimal, are reserved for loop back and for internal testing on a local machine. [You can test this: you should always be able to ping 127.0.0.1, which points to yourself] Class D addresses are reserved for multicasting. Class E addresses are reserved for future use. They should not be used for host addresses.
Now we can see how the Class determines, by default, which part of the IP address belongs to the network (N) and which part belongs to the node (n).

Class A -- NNNNNNNN.nnnnnnnn.nnnnnnn.nnnnnnn
Class B -- NNNNNNNN.NNNNNNNN.nnnnnnnn.nnnnnnnn
Class C -- NNNNNNNN.NNNNNNNN.NNNNNNNN.nnnnnnnn

becomes acceptable. This approach might involve upgrading the backbone to a high-speed technology such as Gigabit Ethernet. If you have fairly light traffic in general, more bandwidth may be all you need to ensure that applications receive the high priority and low latency they require.

However, this simplistic strategy collapses if a network is even moderately busy. In a complex environment - one that has a lot of data packets moving in many paths throughout the network, or that has a mixture of data and real-time applications - you could run into bottlenecks and congestion.

Also, simply adding bandwidth doesn't address the need to distinguish high-priority traffic flows from lower-priority ones. In other words, all traffic is treated the same. In the network realm, such egalitarianism is not good, because network traffic is, by its nature, unpredictable. Further bandwidth is always costly, as demands for bandwidth are increasing day by day.

A fairly new idea in the QoS realm is, to make the network itself more intelligent by incorporating something called policy-based management. With this method, network managers could define policies that spelled out what levels of service certain types of traffic requires that is the reason why I have implemented class based queuing in proxy server to have a proper bandwidth management.

## 2.5 Proxy Servers

A Proxy is someone who does something on behalf of another. For example, if you cannot be present at an electoral vote, you can often nominate another person to register your vote for you. This is a "real world" example of using a proxy.

Simply put, then, a proxy server is a computer that sits between your network and the Internet, and deals with any requests for data from the Internet that users on your network may make. This means that if you are sitting at a networked computer that is connected to the Internet via a proxy server, then when you start your web browser and point it at www.yahoo.com then your browser sends the request to the proxy server, the proxy server retrieves the web page on your behalf, and then sends it to your computer

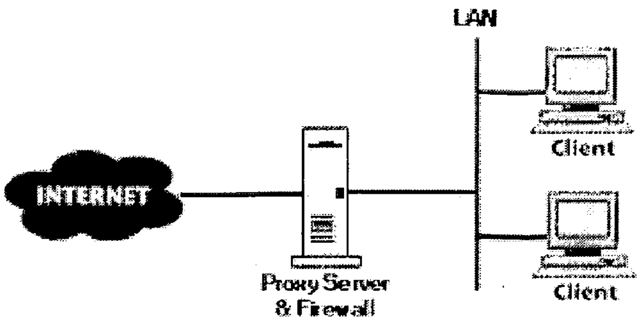


**Figure 2.3: Proxy server in network**

This is a good thing for a couple of reasons. Firstly, it enables an organization to have just one Internet connection, and secondly it makes it easier to manage how this connection is used.

### 2.5.1 Benefits of a Proxy Server

If you connect to the Internet, there are a number of reasons you may want to use a proxy server.

Security: If you configure a proxy server correctly you reduce the exposure of your internal network to hackers without reducing your users access to the Internet.

Cost: The cost of having one big connection to the Internet via a proxy server is smaller than having a modem on each desk and letting each user dial out as they please.

Administration: Are you worried about how your company uses the Internet? Need to block certain sites? A proxy server allows you to monitor what sites people visit and how much time they spend doing it, and while this is a thorny issue for some, it give you an option to block sites you consider unsuitable.

- Caching: Some proxy servers offer the ability to "store" web pages as they are accessed for later re-use. This means that your users see a boost in their web browser performance.

- NAT: Ability to use a small amount of public IP addresses to make your network visible to the outside world, regardless of the amount of workstations using private IP addresses on your LAN via Network Address Translation (NAT)

### 2.5.2 Problems with Proxy Servers

- Training: Requires some knowledge to set up and maintain in peak condition. While it is true that just about anyone can set up a proxy server, some knowledge is required to get the best out of the server.
- Single point of failure: Running a proxy server gives your Internet usage a single point of failure, if the proxy server fails all clients attached to it cannot use the Internet until it is fixed.
- Planning: It can be difficult for someone inexperienced with proxy servers to estimate specifications for a proxy server. Depending on what proxy server you use, and what roles you have it perform, a proxy can be network, memory, disk and processor intensive, in any combination of the above.
- Security: You need to know how to configure the operating system the proxy runs on securely; in effect, by placing all your clients behind a proxy you are inviting the outside world to concentrate any attacks on your network on your proxy server.
- Remote users: Those who use laptops and work from home or hotel rooms, may have trouble using the Internet via an dial up ISP because all their software is configured to use your proxy server. Allowing these users to work with your proxy server and dial into an ISP while on the road requires a bit of planning

### 2.5.3 Functions of Proxy Server

#### Caching

Caching attempts to increase performance for your users, and reduce the amount of traffic flooding your Internet connection. A Cache stores web page components locally on the proxy server, and when a user requests a page that is stored in the cache it can return the page to them from the cached copy rather than requesting it from the Internet server.

each representing 8 bits, in the range 0 to 255 (known as octets) separated by decimal points. This is known as "dotted decimal" notation.
Example: 140.179.220.200

It is sometimes useful to view the values in their binary form.
140 .179 .220 .200
10001100.10110011.11011100.11001000

Every IP address consists of two parts, one identifying the network and one identifying the node. The Class of the address and the subnet mask determine which part belongs to the network address and which part belongs to the node address.

### 2.2.1 Address Classes

There are 5 different address classes. You can determine which class any IP address is in by examining the first 4 bits of the IP address.

Class A addresses begin with 0xxx, or 1 to 126 decimal.
Class B addresses begin with 10xx, or 128 to 191 decimal.
Class C addresses begin with 110x, or 192 to 223 decimal.
Class D addresses begin with 1110, or 224 to 239 decimal.
Class E addresses begin with 1111, or 240 to 254 decimal.

Addresses beginning with 01111111, or 127 decimal, are reserved for loop back and for internal testing on a local machine. [You can test this: you should always be able to ping 127.0.0.1, which points to yourself] Class D addresses are reserved for multicasting. Class E addresses are reserved for future use. They should not be used for host addresses.
Now we can see how the Class determines, by default, which part of the IP address belongs to the network (N) and which part belongs to the node (n).

Class A -- NNNNNNNN.nnnnnnnn.nnnnnnn.nnnnnnn
Class B -- NNNNNNNN.NNNNNNNN.nnnnnnnn.nnnnnnnn
Class C -- NNNNNNNN.NNNNNNNN.NNNNNNNN.nnnnnnnn

In the example, 140.179.220.200 is a Class B address so by default the Network part of the address (also known as the Network Address) is defined by the first two octets (140.179.x.x) and the node part is defined by the last 2 octets (x.x.220.200).
In order to specify the network address for a given IP address, the node section is set to all "0"s. In our example, 140.179.0.0 specifies the network address for 140.179.220.200. When the node section is set to all "1"s, it specifies a broadcast that is sent to all hosts on the network. 140.179.255.255 specifies the example broadcast address. Note that this is true regardless of the length of the node section.
There are some restrictions on IP address. Node addresses of all "0"s and all "1"s are reserved for specifying the local network (when a host does not know it's network address) and all hosts on the network (broadcast address), respectively. This also applies to subnets. A subnet address cannot be all "0"s or all "1"s. This also implies that a 1-bit subnet mask is not allowed.

## 2.3 Subnetting

Subnetting an IP Network can be done for a variety of reasons, including organization, use of different physical media (such as Ethernet, FDDI, WAN, etc.), preservation of address space, and security. The most common reason is to control network traffic. In an Ethernet network, all nodes on a segment see all the packets transmitted by all the other nodes on that segment. Performance can be adversely affected under heavy traffic loads, due to collisions and the resulting retransmissions. A router is used to connect IP networks to minimize the amount of traffic each segment must receive.

### 2.3.1 Subnet Masking

Applying a subnet mask to an IP address allows you to identify the network and node parts of the address. The network bits are represented by the 1s in the mask, and the node bits are represented by the 0s. Performing a bit-wise logical AND operation between the IP address and the subnet mask results in the Network Address or Number.

Default subnet masks:

- Class A - 255.0.0.0 - 11111111.00000000.00000000.00000000
- Class B - 255.255.0.0 - 11111111.11111111.00000000.00000000
- Class C - 255.255.255.0 - 11111111.11111111.11111111.00000000

### 2.3.2 Private Subnets

There are three IP network addresses reserved for private networks. The addresses are 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16. They can be used by anyone setting up internal IP networks, such as a lab or home LAN behind a NAT or proxy server or a router. It is always safe to use these because routers on the Internet will never forward packets coming from these addresses.

## 2.4 QoS in Internet

The growth of Internet as a universal network raises several issues for both enterprises IT departments and ISPs, not the least of which is how to guarantee that applications will receive the service levels they require to perform adequately across the network. For example, network managers might need a way to define a low level of latency and packet loss to ensure that a large file or a business-critical traffic flow gets to its destination on time and without delay. Or, they may need to ensure that a real-time session such as voice or video over IP doesn't look choppy or out of sequence.

The problem with Internet is, IP that is used for transferring packet across network, is a connectionless technology and does not guarantee bandwidth. Specifically, the protocol will not, in itself, differentiate network traffic based on the type of flow to ensure that the proper amount of bandwidth and prioritization level are defined for a particular type of application.

Because current version IP i.e. IP-v4 does not inherently support the preferential treatment of data traffic, it's up to network managers and service providers to make their network components aware of applications and their various performance requirements.

Quality of Service (QoS) generally encompasses bandwidth allocation, prioritization, and control over network latency for network applications. There are several ways to ensure QoS, the easiest one is simply to throw bandwidth at the problem until service quality becomes acceptable. This approach might involve upgrading the backbone to a high-speed technology such as Gigabit Ethernet. If you have fairly light traffic in general, more bandwidth may be all you need to ensure that applications receive the high priority and low latency they require.

However, this simplistic strategy collapses if a network is even moderately busy. In a complex environment - one that has a lot of data packets moving in many paths throughout the network, or that has a mixture of data and real-time applications - you could run into bottlenecks and congestion.

Also, simply adding bandwidth doesn't address the need to distinguish high-priority traffic flows from lower-priority ones. In other words, all traffic is treated the same. In the network realm, such egalitarianism is not good, because network traffic is, by its nature, unpredictable. Further bandwidth is always costly, as demands for bandwidth are increasing day by day.

A fairly new idea in the QoS realm is, to make the network itself more intelligent by incorporating something called policy-based management. With this method, network managers could define policies that spelled out what levels of service certain types of traffic requires that is the reason why I have implemented class based queuing in proxy server to have a proper bandwidth management.

## 2.5 Proxy Servers

A Proxy is someone who does something on behalf of another. For example, if you cannot be present at an electoral vote, you can often nominate another person to register your vote for you. This is a "real world" example of using a proxy.

Simply put, then, a proxy server is a computer that sits between your network and the Internet, and deals with any requests for data from the Internet that users on your network may make. This means that if you are sitting at a networked computer that is connected to the Internet via a proxy server, then when you start your web browser and point it at www.yahoo.com then your browser sends the request to the proxy server, the proxy server retrieves the web page on your behalf, and then sends it to your computer

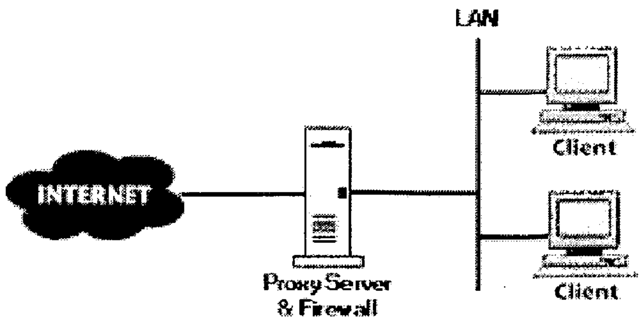


**Figure 2.3: Proxy server in network**

This is a good thing for a couple of reasons. Firstly, it enables an organization to have just one Internet connection, and secondly it makes it easier to manage how this connection is used.

### 2.5.1 Benefits of a Proxy Server

If you connect to the Internet, there are a number of reasons you may want to use a proxy server.

Security: If you configure a proxy server correctly you reduce the exposure of your internal network to hackers without reducing your users access to the Internet.

Cost: The cost of having one big connection to the Internet via a proxy server is smaller than having a modem on each desk and letting each user dial out as they please.

Administration: Are you worried about how your company uses the Internet? Need to block certain sites? A proxy server allows you to monitor what sites people visit and how much time they spend doing it, and while this is a thorny issue for some, it give you an option to block sites you consider unsuitable.

- Caching: Some proxy servers offer the ability to "store" web pages as they are accessed for later re-use. This means that your users see a boost in their web browser performance.

- NAT: Ability to use a small amount of public IP addresses to make your network visible to the outside world, regardless of the amount of workstations using private IP addresses on your LAN via Network Address Translation (NAT)

### 2.5.2 Problems with Proxy Servers

- Training: Requires some knowledge to set up and maintain in peak condition. While it is true that just about anyone can set up a proxy server, some knowledge is required to get the best out of the server.
- Single point of failure: Running a proxy server gives your Internet usage a single point of failure, if the proxy server fails all clients attached to it cannot use the Internet until it is fixed.
- Planning: It can be difficult for someone inexperienced with proxy servers to estimate specifications for a proxy server. Depending on what proxy server you use, and what roles you have it perform, a proxy can be network, memory, disk and processor intensive, in any combination of the above.
- Security: You need to know how to configure the operating system the proxy runs on securely; in effect, by placing all your clients behind a proxy you are inviting the outside world to concentrate any attacks on your network on your proxy server.
- Remote users: Those who use laptops and work from home or hotel rooms, may have trouble using the Internet via an dial up ISP because all their software is configured to use your proxy server. Allowing these users to work with your proxy server and dial into an ISP while on the road requires a bit of planning

### 2.5.3 Functions of Proxy Server

#### Caching

Caching attempts to increase performance for your users, and reduce the amount of traffic flooding your Internet connection. A Cache stores web page components locally on the proxy server, and when a user requests a page that is stored in the cache it can return the page to them from the cached copy rather than requesting it from the Internet server.

There are two types of caching you might see employed. If both are available, you should use a combination of each of them in order to provide the best performance.

Firstly, we have passive caching, which all proxy servers that support caching will support. This means that whenever your proxy server gets a user request a web page for the first time the page components will be copied into cache ready in case someone asks for that page again. This is most efficient when several people from your network visit the same page and works quite well in education where a lecturer might ask everyone in a class to visit a list of web sites as research.

Secondly, you may see some proxy servers that support active (or pre-emptive) caching. Active caching allows the administrator to tell the proxy server to copy down certain popular web pages into the cache at regular intervals, so that when the first user goes to access the page it is already in the cache. You would probably use this to pre-load popular web sites and search engines that your users commonly use as 'portals' to the web.

**IP Packet Filtering**

In order to provide security a proxy server can allow or disable very specific types of network traffic. This is an important component of any system that connects an organization's network to the Internet. Packet filtering consists of defining a set of filters detailing exactly what types of traffic an interface allows to pass through it. You can set different filters for incoming and outgoing traffic of the same type. It is also possible to filter packets based on the Internet address they are coming from, for example, blocking web sites you consider unsuitable for your organization.

**Logging**

All proxy servers should allow at least a rudimentary form of logging. All use of a proxy server can be written to a log for later review, allowing to you see how your computer users are using your Internet connection, time spent using the connection, what sites are visited etc. Some proxy servers also collect log information about external connections, allowing you to log any attacks on your proxy server from the Internet. Some of the better proxy servers allow you to record this data in several ways. For example, some use a database to enable you to analyze trends and produce reports on usage quite easily. Sounds good but it can be a bear to wade through the data logging produces. Still it's nice to know the information is there if you want it.

**Network Address Translator (NAT)**

Most proxy servers support some kind of NAT function, supporting at least in part the RFC 1631 definition of a NAT router. NAT allows you to use any addressing system you like internally and have your clients communication with external hosts pass through a NAT enabled router and appear to of come from the legitimate IP address you have assigned to the router or proxy server performing NAT. When the external host replies, it sends it's reply to the one legitimate IP address it knows, the address of your NAT enabled proxy server, which then forwards the reply back to your internal client.

NAT has two primary functions from our point of view here. Firstly, it reduces the amount of public Internet address blocks an organization needs to connect everyone to the Internet. Instead of one address for each client on your network, you only need one for each proxy server that stands between your network and the Internet. As public addresses are in short supply, this reduces the amount of address blocks you need to purchase.

Secondly, NAT can increase security. If you are hiding behind a proxy server using NAT then no one can see your computer's IP address on the Internet. If no one can see you (and if you use private IP address ranges, reach you) via the proxy server then they can't "hack" your network. Before allowing an external source to send information back to a client on your internal network, NAT assesses whether an address/port mapping, either static or dynamic, exists for the packet. If a mapping does not exist for the packet, the NAT silently discards it.

This behavior protects the internal network from malicious users on the Internet. The only way that Internet traffic is forwarded to the private network is either in response to traffic initiated by a private network user that created a dynamic mapping or because a static mapping exists so that Internet users can access specific resources on the private network. (For example, with a web or email server.)

Of course, this does mean that you have to take a lot of care securing your proxy servers and other external servers such as web and email, but it is easier to secure 10 controlled servers than 10,000 uncontrolled desktop computers.

**Socks Proxy**

Many good proxy servers support a form of socks proxy. This allows applications that run on clients behind the proxy server to talk to hosts on the Internet as if they were directly connected to the Internet, without a proxy server in-between them.

Chapter 3

# ANALYSIS AND DESIGN

In many organizations, the unavailability of sufficient bandwidth results in very slow access to Internet This situation is further exacerbated by a small number of users who use a disproportionate share of the available bandwidth for file download etc., starving other users for bandwidth. Therefore, to ensure reasonable and fair levels of access to all users, I decided to implement Class Based Queuing of out going Internet traffic, preventing high bandwidth consuming applications from utilizing a disproportionate proportion of the available resources.

## 3.1 Class Based Queuing

Class Based Queuing (CBQ) is a traffic management algorithm developed by the Network Research Group at Lawrence Berkeley National Laboratory as an alternative to traditional router-based technology [3]. Now in the public domain as an open technology, CBQ is deployed by companies at the boundary of their WANs.

Network manager can use CBQ to easily classify traffic to meet business priorities and to ensure each traffic class has the appropriate Quality of Service (QoS). CBQ integrate easily with companies existing network policies to protect its network and provide IT manager with more control over the network, thus reducing bandwidth cost.

CBQ divides user traffic into a hierarchy of classes based on any combination of IP addresses, protocols and application types [5]. A company's accounting department, for example, may not need the same Internet access privileges as the engineering department. Because every company is organized differently and has different policies and business requirements, it is vital for traffic management technology to provide flexibility and granularity in classifying traffic flows.

Priority level and borrowing privilege are two key attributes of Class Based Queuing.

Priority levels allow higher priority queues to be serviced first with in the limits of their bandwidth allocation so that the delay for more sensitive real-time traffic classes is reduced.

Borrowing is an explicit class-based queuing mechanism for distribution of excess bandwidth. A class with borrowing privilege may initiate a borrowing request when it needs more bandwidth.
If another class is not using it's full bandwidth, synchronization features indicate that bandwidth is available. Borrowing bandwidth is granted to higher-priority classes before lower-priority classes [4].

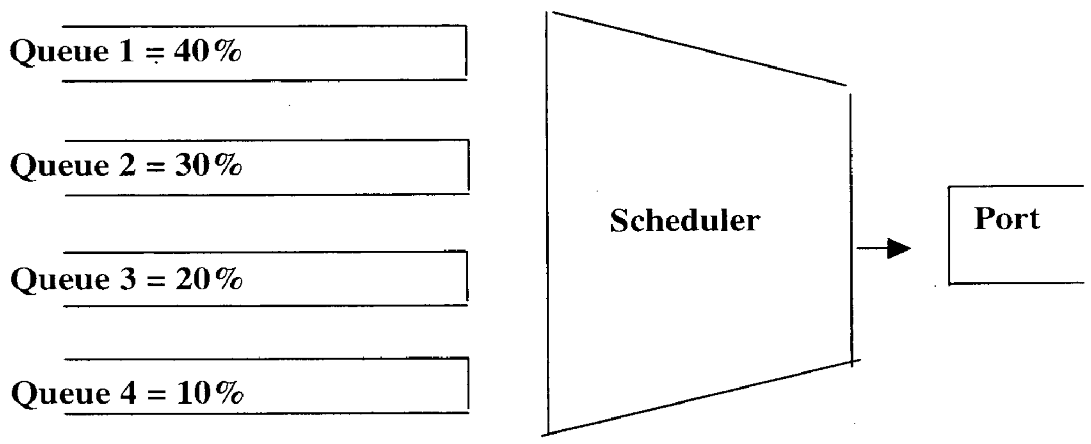


**Figure 3.1: Internet traffic classified into different classes with each allocated a portion of available bandwidth**

By providing network manager with better control over user traffic, CBQ lets companies meet the need of response-time-sensitivity applications, supports service-level agreement and keeps inappropriate traffic off the network.
Because it operates at the IP network layer, CBQ provides the same benefits across any Layer 2 technology and is equally effective with any IP protocol, such as TCP and User Data-gram Protocol (UDP). It also operates with any client or server TCP/IP stack variation, since it takes advantage of standard TCP/IP flows control mechanisms to control end-to-end traffic [5].
In our case, according to the CBQ algorithm, I classified Internet traffic in to six classes, and each class is allocated a portion of bandwidth, thus restricting the request rate of high

bandwidth consuming traffic such as multimedia downloads and providing a fair access to applications like mail, news which consumes smaller amount of bandwidth. Thus utilizing Internet bandwidth in an intelligent way.

## 3.2 Proxy Servers

In most of the organizations, a proxy server is used to give access to network services indirectly. Proxy server acts as an intermediary between a workstation users and the Internet thus all accesses to Internet are handled through the proxy server. There are three main reasons for using proxy servers; to enhance security, to conserver IP address and to reduce network bandwidth.

- The rationale for enhancing security through use of a proxy server is that only the computer running the proxy server will access the Internet directly. Therefore it is only that machine that is vulnerable to attacks from the Internet. And since securing one machine is much simpler than securing all machines, security is enhanced.
- Since IP-v6 is not yet implemented and still today it is a theoretical concept thus IP are limited. Thus by the use of a proxy server we can save much precious IP addressees as it is only proxy server that has routable IP address.
- Network bandwidth can be reduced by the caching requests. If several users request the same page it will only need to be fetched once over the network. Unfortunately not many pages are cacheable, since there are so much dynamic pages out there. But it is usually possible to cache images

A major side-effect of this is that all web accesses have to go through the proxy server and this provides a focal point for monitoring and controlling how users utilize the available resources [1].

### 3.2.1 Squid Cache Server

Squid Internet Object Cache is one of the more popular proxy servers on the Internet for a simple reason -- it's a free, open source code and easy-to-configure tool that cuts down on

Web traffic for large Internet sites. Thus I decided to use squid proxy server to implement class based queuing and provide a fair Internet access to all of the users.

## 3.3 Delay Pool in Squid

The bandwidth management component in Squid is called “Delay Pools”, written by David Luyer [2]. Using this delay pools, it is possible to limit Internet traffic in a reasonable way, depending on so-called 'magic words', existing in any given URL for example, a magic word could be '.mp3', '.exe' or '.avi', etc. Any distinct part of a URL (such as .avi ) can be defined as magic word.

We can tell squid to download certain kinds of files (like sound files) at a specified speed (in our example, it will be about 5 Kbytes/s). If our LAN users download files at the same time, they will be downloaded at about 5 Kbytes/s altogether, leaving remaining bandwidth for web pages, e-mail, news, etc. Thus provides a fair access to all the users by preventing few users from using a disproportionate portion of available resources

### 3.3.1 Working of delay pool

Delay pool uses Token Bucket algorithm [2] for bandwidth management. Using this program, we can manipulate the bandwidth available to each class of application by changing two parameters:

- The max value and
- The restore value

The max value identifies the maximum size of the token bucket and the restore value is the rate at which the tokens are issued into the bucket. When this feature is used, the time-averaged bandwidth available to each class is limited by the rate at which tokens are issued to the bucket (the restore rate). However, if the class has not requested any new data for some time and has accumulated tokens in the bucket, when a new request is submitted the data is delivered in a burst, up to the number of tokens in the bucket. In this manner, we can limit bandwidth-consuming applications from filling all the available bandwidth and provide fair access to all the users [6].

### 3.3.2 Drawbacks of available delay pool

When we deployed delay pools in the NIC network, we found that though it functioned properly and enabled fair access to all class of applications, it had a major weakness. The delay pools parameters are statically defined in the Squid configuration file and cannot be changed dynamically to suit the changing conditions in the network. Bandwidth usage patterns change considerably at different times of the day, and restore value for a class applications that gives fair access to all classes during peak hours is unduly restrictive during slack times, confining an class to a small bandwidth and generally leaving a large proportion of the available bandwidth unused. Though it is possible to vary the delay pool configuration by using time based controls this does not respond to the actual demand at a given time and can be very cumbersome to implement in even a moderately complex environment.

Since the Squid source code was available for modification, the most attractive solution to this problem was to modify the delay-pools code to dynamically reconfigure the parameters to share the available bandwidth fairly among the currently being used applications, based on the principles of link-sharing described by Floyd and Jacobson [7].

## 3.4 Design

As we have decided to modify the freely available squid source code, we tried the various implementations, and here we present them:

### 3.4.1 Earlier experiments

Before arriving at the present implementation, we tested several different modifications and changes to the Squid code. Initially, we keep track each application's usage independently and dynamically allocate each application a separate bandwidth value up to a certain fixed ceiling. However, in this case the fairness criteria were invalidated as heavy used classes were quickly assigned a large amount of bandwidth, to determinate lightly used class.

Later, we tried to keep track of the number of users and allocate bandwidth to each application based on the number of active users. However, this also proved to be inefficient, as it leads to a large portion of available bandwidth unused further not all users connected to a particular class would be browsing the Internet at the same time. Some would be reading data already rendered and others would be waiting for requests delayed due to congestion in other parts of the Internet.

### 3.4.2 Current modification

The current algorithm that we have implemented [Figure – 3.2] trades sophistication for simplicity and speed of execution. Speed of execution is a very important factor as we dynamically evaluate and adjust the values once per second and an inefficient algorithm would pose a heavy workload for the system running the program. This algorithm comprises of two stages:

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

**i. Allocation of bandwidth to a class**

The algorithm uses the aggregate number of tokens in the delay pool as the criteria for whether the class's link bandwidth is fully utilized or not. If the number of tokens exceeds the cutoff value, indicating that there is some capacity being underutilized, each individual class's bandwidth allocation is increased by a fraction [presently 10% (max) of the aggregate delay pool size]. If the number of tokens is less than the cutoff value, indicating that the requests are backing up and link bandwidth is fully utilized, individual class's bandwidth allocation is reduced by a fraction. This is shown in the flow chart below:
Usually the link bandwidth is used both for the restore rate (at which tokens are issued to the pool) and the maximum size of the pool. The aim of the algorithm is to keep the number of tokens in the pool of a class as close to zero as possible. Practically, we have set a cutoff value that is 16% of the maximum bucket size and when the available tokens drop below this value the user's bandwidth is reduced. A value that is larger than zero is used, as it is difficult to precisely control the number of tokens in the pool. Depending upon the number of tokens in the aggregate delay pool, the bandwidth an individual user receives can range between a small minimum value e.g. 100 Bytes per second, and

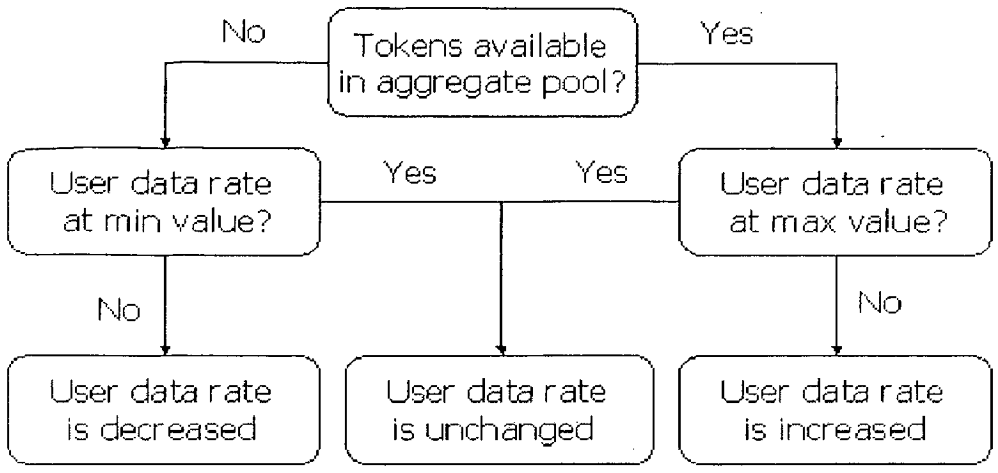

No
Tokens available in aggregate pool?
Yes
User data rate at min value?
Yes
Yes
User data rate at max value?
No
No
User data rate is decreased
User data rate is unchanged
User data rate is increased

**Figure 3.2: Dynamic delay pools algorithm**

usually the link bandwidth is used both for the restore rate (at which tokens are issued to the pool) and the maximum size of the pool. The aim of the algorithm is to keep the number of tokens in the pool of a class as close to zero as possible. Practically, we have set a cutoff value that is 16% of the maximum bucket size and when the available tokens drop below this value the user's bandwidth is reduced. A value that is larger than zero is used, as it is difficult to precisely control the number of tokens in the pool. Depending upon the number of tokens in the aggregate delay pool, the bandwidth an individual user receives can range between a small minimum value e.g. 100 Bytes per second, and the total available bandwidth of the link. The small minimum value is set to ensure that even at peak load times users get some access and practically this limit is not usually reached. Setting the maximum value to the rate at which tokens are issued to the aggregate pool ensures that even if there is only one application at that instant, users of that application can use all available resources.

### ii. Transferring bandwidth among delay pools

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

The Squid code was therefore further modified to allow bandwidth assigned to a particular delay pool, which is not being utilized, to be transferred to another delay pool [Fig. 3.3]. This modification checks the level of tokens in each delay pool after adding the token allocation available to that particular pool. If the level of tokens is greater than the maximum allowable value, then the excess tokens are transferred to a buffer and if a delay pool whose level of tokens is less than maximum permissible value is encountered, the tokens in the buffer are used to "top up" the pool up to it's max value [or until the tokens in the buffer are exhausted]. Accordingly, excess tokens that would be unutilized by lightly loaded delay pools are distributed among the pools that are using their token allocation fully. The maximum size of the buffer is set to a few seconds worth of tokens.

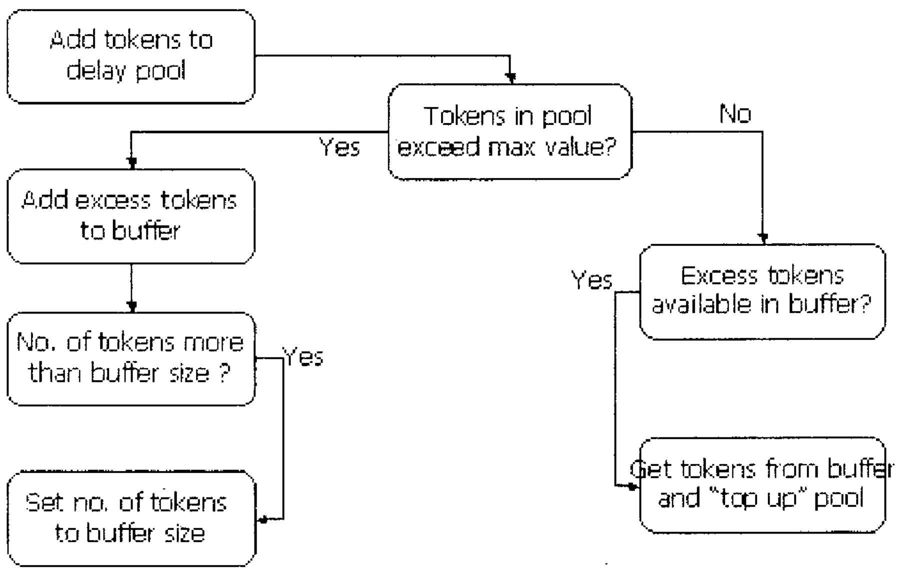

Add tokens to delay pool
Tokens in pool exceed max value?
Yes
No
Add excess tokens to buffer
No. of tokens more than buffer size ?
Yes
Set no. of tokens to buffer size
Yes
Excess tokens available in buffer?
Get tokens from buffer and "top up" pool

**Figure 3.3: Token distribution [among pool] algorithm**

Chapter 4

# IMPLEMENTATION ASPECTS

For implementing Class Based Queuing in proxy server first of all we have to classify the Internet traffic in to various classes and then assign a portion of bandwidth to each class so as to provide a fair access to all the users. We also have to take in to account the varying usage pattern and should vary bandwidth allocated accordingly so that there is a optimum bandwidth utilization, and each user is provided a fair access.

## 4.1 Classification of traffic in to classes

For implementing Class Based Queuing in squid proxy server I have classified the total traffic in to six classes based on the format of the file being transferred they are:

- Plain text files: they contains files with extension .html, .htm and .txt.
- Formatted document file: they contain files with extension .doc, .pdf and .ps.
- Compressed and encoded files: they contain files with extension .arc, .arj, .bin, .exe, .gz/.gzip, .hqz, .sit, .sea, .tar/.tar.gz/.tar.Z, .uu, .Z, .zip.
- Graphics files: they contain files with extension .gif, .tiff, .jpg/.jpeg/.jfif.
- Sound files: they contain files with extension .au .aiff .mp3 .ra .wav.
- Video files: they contain files with extension .avi, .mov/.movie, .mpg/.mpeg, .qt, .ram.

The next step in classifying the traffic in to different classes is to identify the class to which a given request, from the client belongs to then putting that particular request in it's appropriate class. For solution of the above problem I have taken help of access control scheme present in the squid proxy server that, when ever a request comes from client compares its URL with set of rules as defined by the network administrator and then take appropriate action according to the rule set. For classifying the traffic in to six classes I have defined the following rules:

```
acl plain url_regex –i .html . htm .txt
acl formatted url_regex –i .doc .pdf .ps
acl compressed url_regex –i .arc .arj .bin .gz .gzip .hqz .hqx .sit .tar .tar.gz .tar.Z .tarz .uu
.Z .zip
acl graphics url_regex –i .gif .tiff .jpg .jpeg .jfif
acl sound url_regex –i .au .aiff .mp3 .ra .wav
acl video url_regex –i .avi .mov .movie .mpeg .qt .ram.
```

Thus I have put all the traffic that represents html and text file type in to one class and represent it by a tag name plain. Here access element "url_regex" instruct squid proxy to have URL regular pattern matching while key word "–i" makes the pattern checking case insensitive.
By the use this tag name we can control access of the entire traffic that belongs to this class for example if the next rule after putting desired traffic in a class is:

```
http_access deny plain
```

Then all the access to this particular file type will be denied by the proxy server and client will receive access deny message. Similarly we can put access to the Internet traffic of a particular class in delay pool, by specifying:

```
delay_access 1 allow plain
```

Here we instruct squid to have delayed access to all the traffic that belong to the class plain and follow the rules as defined for the class number 1. Thus by using this method we can classify Internet traffic in to six desired classes based on some "magic word" present in their URL and then have access to such type of traffic through their respective delay pools.

However after allowing delayed access to a particular class of internet traffic we should also provide http access to that class because it is the primary access controlling mechanism that prevent malicious and unwanted users from accessing the proxy server. For specifying the delay classes we have to follow certain rules that are:

Firstly we have to specify how many delay pools we are using as in our case we are using six delay pool so we have specified

```
delay_pools 6
```

Secondly we have to specify to which delay class a particular pools belongs as in my case I am concerned with managing over all traffic and not the traffic of an individual connected to the proxy server so I am using the class one delay pool for all the six traffic categories. Thus I specified:

```
delay_class 1 1
delay_class 2 1
delay_class 3 1
delay_class 4 1
delay_class 5 1
delay_class 6 1
```

These rules specify that delay pool number one is of class 1 and so on.

Third step in defining rules for delay classes is to specify the restore rate of the tokens in the bucket and the size of the bucket that is used to collect the tokens. Thus I have specified:

```
dalay_parameters 1 restore1/(total bandwidth)
delay_parameters 2 restore2/(total bandwidth)
delay_parameters 3 restore3/(total bandwidth)
delay_parameters 4 restore4/(total bandwidth)
dalay_parameters 5 restore5/(total bandwidth)
delay_parameters 6 restore6/(total bandwidth)
```

Here key word 1 after the rule tag delay_parameter specifies that the particular rule sets are for delay pool number: 1. restore1, the restore rate of the tokens in the bucket is supplied by the network administrator at the time when he opt for traffic shaping in the interface provided by me to the software. As time average bandwidth for a class is the rate at which tokens are added to the bucket, thus bandwidth of a class is limited to the restore value, restore1. I have chosen the total bandwidth available, as size of the bucket for each class. Because when there is any request traffic in a given class after a long ideal period, data could be delivered in a bust up to the number of tokens present in the bucket, as tokens are accumulating in the bucket.

Since the existing implementation of the delay pool did not provide the satisfactory performance as bandwidth allocation are static and do not adopt to the varying usage pattern. Therefore I decided to modify the available squid source code for improving its performance.

## 4.2 Modification of source code

In the implementation of proposed algorithms for making delay pool dynamic involves the modification of three files present in the, squid proxy/cache server source code, they are:

- structs.h
- cache_cf.c
- delay_pools.c

While the first two files undergo minor modification, but the last one that is major file for the delay pool present in the squid proxy undergoes a major modifications. However modification should be such that it should not affect the normal working of the delay pool and should be bug free because squid source files are interrelated and we could not compile a file separately to fix out the bugs.

### 4.2.1 Modification of the file structs.h

This file contains data structure, which contains the values supplied by the user for allocated bandwidth (restore rate of the tokens) and maximum size of the bucket. As we are varying the available bandwidth for a class according to the usage pattern therefore we should have a variable which stores the minimum bandwidth allocated to a class. As bandwidth is changing according to varying usage pattern, and if later, we decide to reduce allocated bandwidth after increasing it we should know what is the minimum value to which we can reduce bandwidth. Therefore I have introduced a new variable named:

```
int restore_bps_orig;
```

It is of type integer and added in the structure "_delaySpec" which contains specification about the delay pools.

### 4.2.2 Modification of the file cache_cf.c

This file reads the values supplied by the network administrator from squid's configuration file "squid.conf" and assign it to appropriate variable. Here in this file I have assigned the newly introduced variable "restore_bps_orig" with the value supplied by the user for restore rate of tokens in the bucket.

I have assigned the value of the restore rate of the tokens in the bucket to two variables, namely "restore_bps" and "restore_bps_orig". As value of first variable can vary according to varying usage pattern, and we could encounter a situation in which after increasing bandwidth, we have to reduce it then in that situation we should know what is the minimum value to which reduction is possible.

This file also acts as a parser, that is it checks whether the value supplied by the network administrator are correct and according to the format as described in the configuration file. If there is any discrepancy error is generated at the time of starting the proxy server. To have a error free environment we have provide an interface to the software.

### 4.2.3 Modification of the file delay_pools.c

This file undergoes a large modification, as this is the main file from where the operation of the delay pool is controlled. The main operations that are carried out in this file are:

When squid proxy is started it first of all identifies how many and of what class delay pools are being used so allocate appropriate amount of memory for the related variables needed to control the operation of delay pool. In next step it initializes the values of the variables. In our case it allocate memory for six -class one, structures and initializes the variable that contain size of the bucket, restore rate, original restore rate for each delay pool number.

Whenever there is a request from the client it first of identify the client from it's IP address and then tries to identify to which delay class a particular request belongs. It than requests delay bytes form that particular delay pool, however all the six delay pools are updated every second to increment the restore bytes to the delay bucket, if number of bytes increases the total size of the bucket the excess bytes are distracted.

Since the current implementation of delay pools does not take in to account the current usage pattern therefore we have to modify function "delayPoolsUpdate", which updated current aggregate of bucket holding the tokens. And introduced a new function "delayPoolsRestoreUpdate" for updating the restore values of the tokens in the bucket, for each pool.

In the function "**delayPoolsUpdate()**"

I have introduced a new variable "excess_bytes" which takes in to account excess bytes, transfers unused bytes from one pool to another. If for a particular pool the number of bytes after adding the current aggregate bytes to increment from restore bytes increases the maximum limit that a bucket could hold, then excess bytes are added to variable "excess_bytes". On the other hand if number of bytes in a bucket are less than the maximum limit then they are supplied from excess_bytes this helps in transferring bandwidth among delay pools.

However I have put a limit to the tokens that excess_bytes could hold to avoid excess_bytes from running out of memory.

I have put a limit excess_bytes to 11 times to that of total available bandwidth.

In function "**delayPoolRestoreUpdate**()"

I have added this new function to vary restore bytes according to the usage pattern. This function is executed whenever times limit from last restore value update increases one second.

If for a particular pool, the number of tokens in the aggregate pool reaches below toggle value that is set to one sixth of the total aggregate pool capacity, indicating that requests are building up and link bandwidth is fully utilized, bandwidth allocated to a class is reduced by 5%. This is achieved by multiplying restore_bps by 0.95.

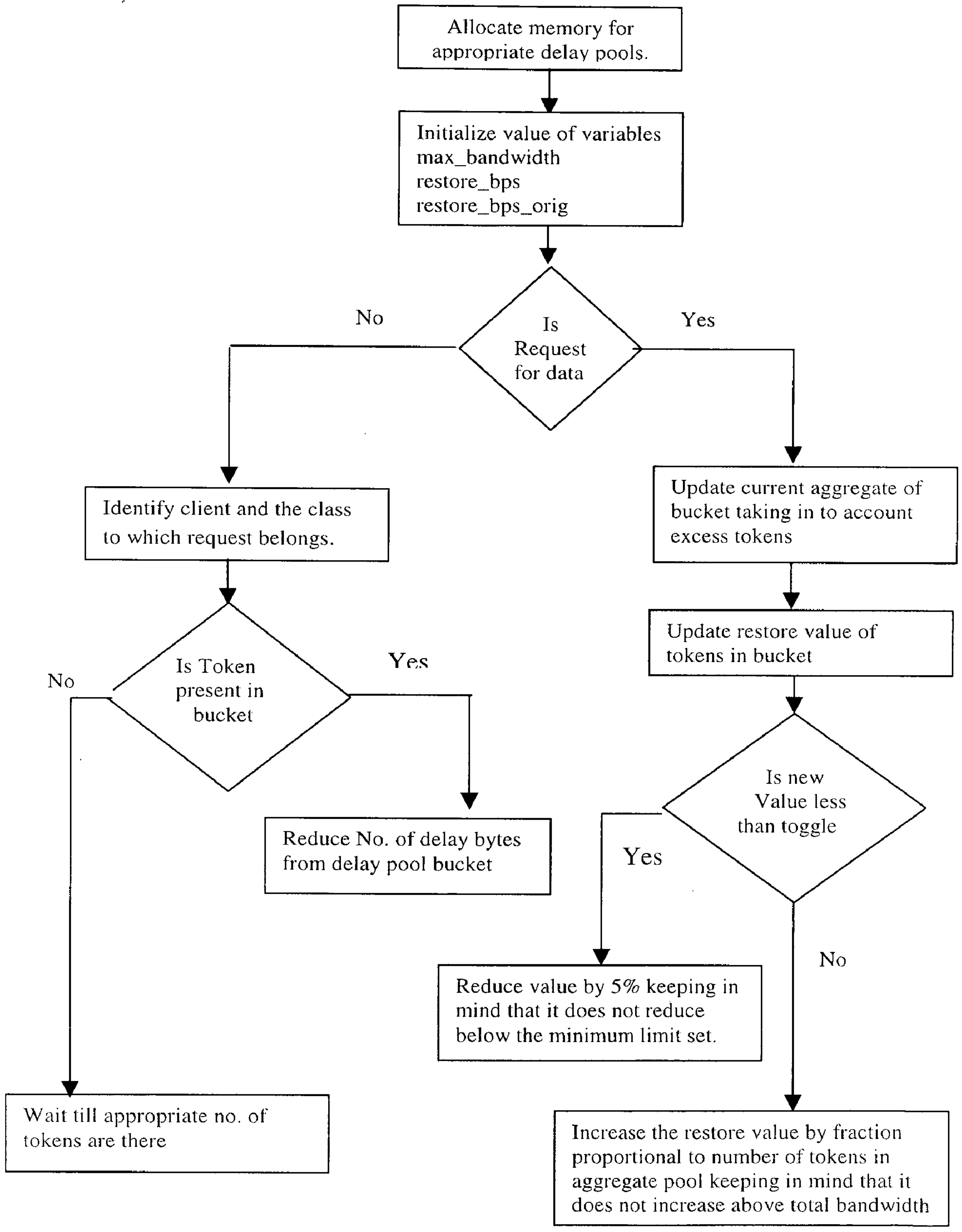
However a limit is put on restore_bps so that allocated bandwidth does not reduce below minimum allocated value as supplied by the network administrator.

On the other hand if current aggregate increases above toggle value bandwidth allocated to a class is increased in proportion to the current aggregate on the bucket of the particular delay pool, so that at a given time restore rate does not increase by more than 10%. The formula used for incrementing value is:

```
temp_restore_val = restore_bps * ( 1.0 + (current aggregate/max bucket capacity));
restore_bps = 0.9 * restore_bps + 0.1 * temp_restore_val;
```

However I have also put a limit in incrementing restore value so that it may not increase above the total bandwidth available for the link.

Thus new control flow diagram will look like:



Allocate memory for appropriate delay pools.
Initialize value of variables max_bandwidth restore_bps restore_bps_orig
Is Request for data
No
Yes
Identify client and the class to which request belongs.
Update current aggregate of bucket taking in to account excess tokens
Update restore value of tokens in bucket
Is Token present in bucket
No
Yes
Reduce No. of delay bytes from delay pool bucket
Is new Value less than toggle
Yes
No
Reduce value by 5% keeping in mind that it does not reduce below the minimum limit set.
Wait till appropriate no. of tokens are there
Increase the restore value by fraction proportional to number of tokens in aggregate pool keeping in mind that it does not increase above total bandwidth

Chapter 4

# RESULTS AND DISCUSSION

Here I present the output from the log file generated by the delay pool.

## 1. When delay pools are created

```
delayInitDelayPool : Pool No. 1, Aggregate = 1000
delayInitDelayPool : Pool No. 2, Aggregate = 1000
delayInitDelayPool : Pool No. 3, Aggregate = 1000
delayInitDelayPool : Pool No. 4, Aggregate = 1000
delayInitDelayPool : Pool No. 5, Aggregate = 1000
delayInitDelayPool : Pool No. 6, Aggregate = 1000
```

Here six delay pools are initialized with there corresponding initial bucket aggregate.

## 2. When delay pools are updated

```
Delay Pool 1 Update: Excess_bytes=0, Aggregate=2000,  Max_bytes=10000, incr=1
Delay Pool 2 Update: Excess_bytes=0, Aggregate=2000,  Max_bytes=10000, incr=1
Delay Pool 3 Update: Excess_bytes=0, Aggregate=2000,  Max_bytes=10000, incr=1
Delay Pool 4 Update: Excess_bytes=0, Aggregate=2000,  Max_bytes=10000, incr=1
Delay Pool 5 Update: Excess_bytes=0, Aggregate=2000,  Max_bytes=10000, incr=1
Delay Pool 6 Update: Excess_bytes=0, Aggregate=2000,  Max_bytes=10000, incr=1

Restore Calc: Class No.=1  Max_Bytes=10000, Current Val=2000, Old Restore=1000,
              New Restore=1100, Orig Restore=1000
Restore Calc: Class No.=2  Max_Bytes=10000, Current Val=2000, Old Restore=1000,
              New Restore=1100, Orig Restore=1000
Restore Calc: Class No.=3  Max_Bytes=10000, Current Val=2000, Old Restore=1000,
              New Restore=1100, Orig Restore=1000
Restore Calc: Class No.=4  Max_Bytes=10000, Current Val=2000, Old Restore=1000,
              New Restore=1100, Orig Restore=1000
Restore Calc: Class No.=5  Max_Bytes=10000, Current Val=2000, Old Restore=1000,
              New Restore=1100, Orig Restore=1000
Restore Calc: Class No.=6  Max_Bytes=10000, Current Val=2000, Old Restore=1000,
              New Restore=1100, Orig Restore=1000
```

Here Delay Pool Update represents the updating of current aggregate of a pool:

Max_Bytes are the maximum number of bytes a pool could hold and incr is the time duration in seconds after which delay pool is updated. Excess_Bytes are used for transferring bandwidth among pools.

Restore Calc represents updating of restore rate at which tokens are added to delay pool. Current Val is current aggregate of delay pool bucket, Old Restore is restore rate before delay pool update and New Restore is after update (which has increased by 10%). Orig Restore is minimum value of bandwidth supplied by network administrator for a given class.

## 3. When there is data request

```
DelayBytesIn: Class Num 1, Old Aggregate=2000, Qty In=879, New Aggregate=1121
DelayBytesIn: Class Num 1, Old Aggregate=1121, Qty In=879, New Aggregate=242

Delay Pool 1 Update: Excess_bytes=0, Aggregate=1342, Max_bytes=10000, incr=1
Delay Pool 2 Update: Excess_bytes=0, Aggregate=3100, Max_bytes=10000, incr=1
Delay Pool 3 Update: Excess_bytes=0, Aggregate=3100, Max_bytes=10000, incr=1
Delay Pool 4 Update: Excess_bytes=0, Aggregate=3100, Max_bytes=10000, incr=1
Delay Pool 5 Update: Excess_bytes=0, Aggregate=3100, Max_bytes=10000, incr=1
Delay Pool 6 Update: Excess_bytes=0, Aggregate=3100, Max_bytes=10000, incr=1

Restore Calc: Class No.=1 Max_Bytes=10000, Current Val=1342, Old Restore=1100,
              New Restore=1100, Orig Restore=1000
Restore Calc: Class No.=2 Max_Bytes=10000, Current Val=3100, Old Restore=1100,
              New Restore=1100, Orig Restore=1000
Restore Calc: Class No.=3 Max_Bytes=10000, Current Val=3100, Old Restore=1100,
              New Restore=1100, Orig Restore=1000
Restore Calc: Class No.=4 Max_Bytes=10000, Current Val=3100, Old Restore=1100,
              New Restore=1100, Orig Restore=1000
Restore Calc: Class No.=5 Max_Bytes=10000, Current Val=3100, Old Restore=1100,
              New Restore=1100, Orig Restore=1000
Restore Calc: Class No.=6 Max_Bytes=10000, Current Val=3100, Old Restore=1100,
              New Restore=1100, Orig Restore=1000
```

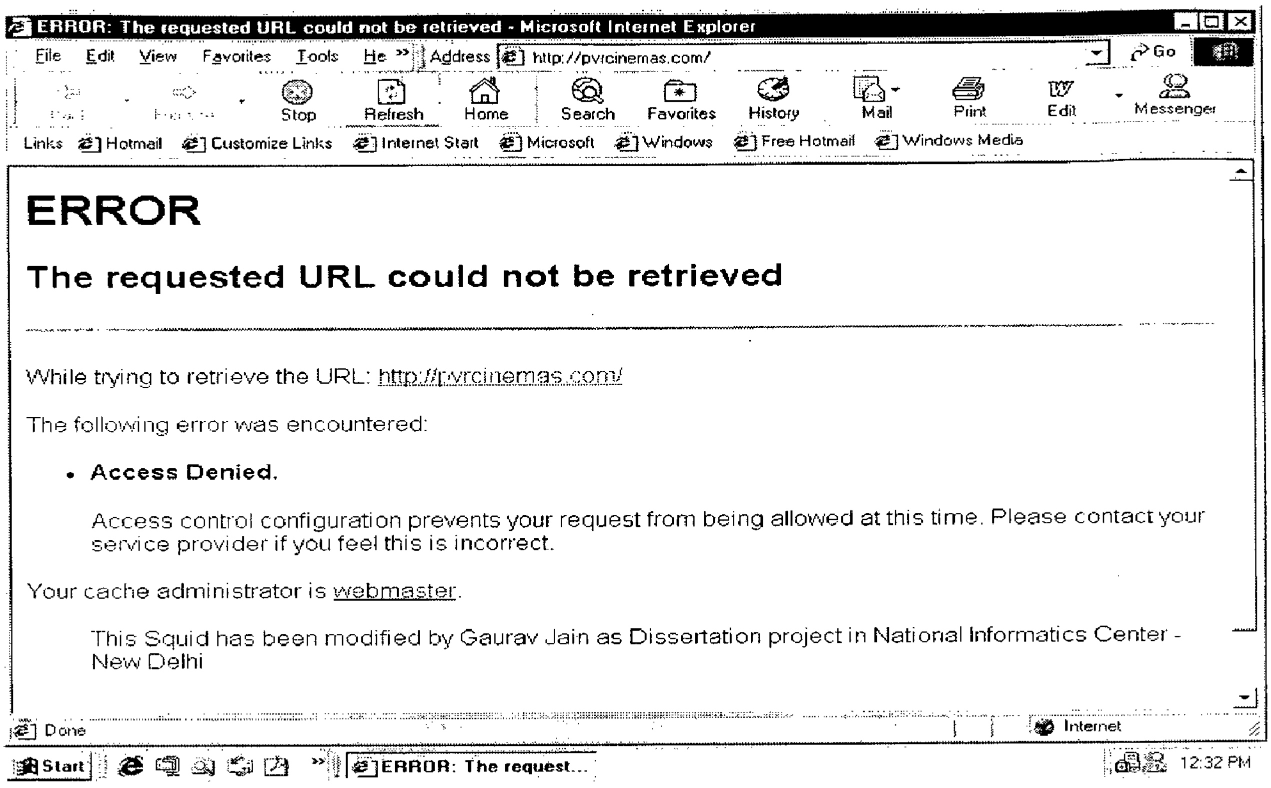
Here we observe that there is no increase in restore rate of class number one because bandwidth allocated to class is fully utilized. If however there is no utilization of bandwidth allocated to other classes' bandwidth allocated to class one will increase by the use of excess bytes but it will take a little bit of time.

```
DelayBytesIn: Class Num 1, Old Aggregate=100000, Qty In=879, New Aggregate=9121
DelayBytesIn: Class Num 1, Old Aggregate=9121, Qty In=879, New Aggregate=8242

Delay Pool 1 Update: Excess_bytes=10000, Aggregate=10000, Max_bytes=10000,
incr=1
Delay Pool 2 Update: Excess_bytes=18242, Aggregate=10000, Max_bytes=10000,
incr=1
Delay Pool 3 Update: Excess_bytes=28242, Aggregate=10000, Max_bytes=10000,
incr=1
Delay Pool 4 Update: Excess_bytes=38242, Aggregate=10000, Max_bytes=10000,
incr=1
Delay Pool 5 Update: Excess_bytes=48242, Aggregate=10000, Max_bytes=10000,
incr=1
Delay Pool 6 Update: Excess_bytes=58242, Aggregate=10000, Max_bytes=10000,
incr=1

Restore Calc: Class No.=1  Max_Bytes=10000, Current Val=10000, Old Restore=10000,
              New Restore=10000, Orig Restore=1000
Restore Calc: Class No.=2  Max_Bytes=10000, Current Val=10000, Old Restore=10000,
              New Restore=10000, Orig Restore=1000
Restore Calc: Class No.=3  Max_Bytes=10000, Current Val=10000, Old Restore=10000,
              New Restore=10000, Orig Restore=1000
Restore Calc: Class No.=4  Max_Bytes=10000, Current Val=10000, Old Restore=10000,
              New Restore=10000, Orig Restore=1000
Restore Calc: Class No.=5  Max_Bytes=10000, Current Val=10000, Old Restore=10000,
              New Restore=10000, Orig Restore=1000
Restore Calc: Class No.=6  Max_Bytes=10000, Current Val=10000, Old Restore=10000,
              New Restore=10000, Orig Restore=1000
```

Here we observe that bandwidth is transferred by excess bytes and class one utilizes full bandwidth. We also see that restore rate does not increase above maximum bandwidth available.

**4. When access to certain site id denied**

When network administrator denies access to certain web site, client will receive an error message as shown in the snap shot taken from the window.


ERROR: The requested URL could not be retrieved - Microsoft Internet Explorer
File Edit View Favorites Tools He
Address http://pvrcinemas.com/
Go
Stop Refresh Home Search Favorites History Mail Print Edit Messenger
Links Hotmail Customize Links Internet Start Microsoft Windows Free Hotmail Windows Media
Done
Internet
Start
ERROR: The request...
12:32 PM

ERROR

The requested URL could not be retrieved

While trying to retrieve the URL: http://pvrcinemas.com/

The following error was encountered:

- **Access Denied.**

  Access control configuration prevents your request from being allowed at this time. Please contact your service provider if you feel this is incorrect.

Your cache administrator is webmaster.

This Squid has been modified by Gaurav Jain as Dissertation project in National Informatics Center - New Delhi

# CONCLUSION

I have implemented Class Based Queuing i n p roxy s erver f or h andling v arious categories of Internet traffic, so as to over come slow Internet access speed.
Squid is the only cache/proxy server that has a bandwidth management feature called "delay pool". However, the existing implementation of delay pools did not provide satisfactory performance as bandwidth allocation are static and do not adapt to varying usage pattern.
Our solution adjust the bandwidth allocation dynamically, thus bandwidth allocated could vary between the total link bandwidth available and a minimum value supplied by the network administrator. Providing optimal access to all the users.

Although the current system allows bandwidth allocated to a single server to be distributed nearly optimally, we still face the situation of some caches on our network not using their bandwidth allocation while others are overloaded. I propose to extend the system to allow bandwidth to be exchanged among a set of caches. So that unused bandwidth from lightly loaded cache could be transferred to one that is heavily loaded.

# REFERENCES

1. Robert Moir; Proxy Servers for beginners, from www.robertmoir.co.uk/security/fog000000000018.html

2. Squid Documentation from www.squid.nlanr.net

3. Ashley Stephenson; A Class by itself: Class-based queuing can provide simplified quality-of-service guarantees for high-bandwidth networks, from http://www.currentissues.telephonyonline.com/ar/telecome_class_itself_classbased

4. Class-based Queuing variant Leads to Dynamic bandwidth Allocation, from http://www.itworld.com/nl/sup_mgr/05132002/

5. Blumenffld Steven M.; Class-based queuing, from http://www.broadcastengineering.com/ar/broadcasting_classbased_queuing

6. Bandwidth Limiting HOWTO, from http://squid-docs.sourceforge.net/latest/html/

7. Sally Floyd and Vam Jacobson; Link-shearing and Resource Management Model for Packet Network, IEEE/ACM Transaction on Networking, Vol. 3 No. 4, August 1995

8. Tanenbaum Andrew S.; Computer Networks, Third Edition, Prentice-Hall India. 2000, [illegible]

9. Comer Douglas E.; Internetworking with TCP/IP, Volume 1, Third Edition, Prentice-Hall India. 2001 [illegible]

10. Ashley Stephenson; A Class by itself: Class-based queuing can provide simplified quality-of-service guarantees for high-bandwidth networks, from http://www.currentissues.telephonyonline.com/ar/telecome_class_itself_classbased

11. Schultz Briam; Class-based queuing divvies bandwidth, from http://www.nwfussion.com/news/tech/0612tech.html

12. Class-based Queuing variant Leads to Dynamic bandwidth Allocation, from http://www.itworld.com/nl/sup_mgr/05132002/

# APPENDIX - A

# APPENDIX - A

## A.1 Squid Proxy/Cache Server

Squid is the result of efforts by numerous individuals from the Internet community. Many organizations have provided support for squid's development. Squid is copyrighted by Regents of University of California. Squid incorporates software development. This means, program is free software so we can redistribute it and/or modify it under the terms of the GNU General Public License as published by Free Software Foundation.

More than a mere proxy, the Squid Internet Object Cache is one of the more popular proxy servers on the Internet for a simple reason -- it's a free and easy-to-configure tool that cuts down on Web traffic for large Internet sites.

Squid is software that **caches** Internet data. It does this by accepting requests for objects that people want to download and handling their requests in their place. In other words, if a person wants to download a web page, they ask squid to get the page for them. Squid then connects to the remote server (for example http://www.squid-cache.org/) and requests the page. It then transparently streams the data through itself to the client machine, b ut at the same time keeps a copy. The next time someone wants that page; squid simply reads it off disk, transferring the data to the client machine almost immediately.

## A.2 Concept behind Internet caching

Some questions should come to mind as to how useful caching can be, and when objects should and should not be cached. It is totally inappropriate to cache (for example) credit card numbers, the results of scripts executed on remote servers, sites that change often (like www.timesofindia.com) or even sites that simply don't want to be cached.

Squid handles these circumstances elegantly (but needs the remote sites to work according to the standards, of course). Executable cgi-bin scripts are not cached, pages that return the correct headers are cached for limited periods of time, and you can specify extra rules as to what, and what not, to cache, and for how long to cache it.

## A.3 Systems supported by Squid

Squid runs on most versions of Unix and OS/2 It is known to work on:

AIX, Digital Unix, FreeBSD, HP-UX, Irix, Linux, NetBSD, Nextstep, SCO, Solaris

## A.4 Hardware requirements

Caching stresses certain hardware subsystems more than other. Although the key to good cache performance is good overall system performance, the following list is arranged in order of decreasing importance:
Disk random seek time

- Amount of system memory
- Sustained disk throughput
- CPU power

Seek time is one of the most important considerations if your cache is going to be loaded. The smaller these value the better: it is the average time that the disk's heads take to move from a random track to another (in milliseconds). Having the world's fastest drive is not useful, though, if it holds a tiny amount of data. To cache effectively you need disks that can hold a significant amount of downloaded data, but that are fast enough to not slow your cache to a crawl.
Squid keeps an in-memory table of objects in RAM. Because of the way that Squid checks if objects are in the file store, fast access to the table is very important. Squid slows down dramatically when parts of the table are in swap.
Each object stored on disk uses about 75 bytes (approx.) of RAM in the index. The average size of an object on the Internet is about 13kb, so if you have a gigabyte of disk space you will probably store around about 80,000 objects. At 75 bytes of RAM per object, 80,000 objects require about six megabytes of RAM. If you have 8gigs of disk you will need 48Mb of RAM just for the object index. It is important to note that this excludes memory for your operating system, the Squid binary, memory for in-transit objects and spare RAM for disk cache.

Squid is not generally CPU intensive. On startup Squid can use a lot of CPU while it works out what is in the cache, and a slow CPU can slow down access to the cache for the first few minutes upon startup, but later a Pentium 133 machine generally runs pretty idle, while receiving 7 TCP requests a second. Also multiprocessor machine generally doesn't increase speed dramatically as only certain portions of the Squid code are threaded.

## A.5 Supported Protocols

Squid supports the following incoming protocol request types (when the proxy requests are sent in HTTP format)

- Hyper Text Transfer Protocol (HTTP), which is the specification that the WWW is based on.
- File Transfer Protocol (FTP)
- Gopher – it isn't much used any more.
- Secure Socket Layer - which is used for secure online transactions.

As squid can also be part of Internet caching hierarchy i.e. you don't have to cache Internet content, which another computer participating in hierarchy has already cached. So beside these protocols squid also supports:

- Internet Cache Protocol (ICP). ICP is used to find out if a specific object is in another cache's store.
- Cache Digests. This protocol is used to retrieve an index of objects in another cache's store. When a cache receives a request for an object it does not have, it checks this index to determine which cache does have the object.
- Simple Network Management Protocol (SNMP). Common SNMP tools can be used to retrieve information about your cache.

## A.6 Directory structure

Squid normally creates a few directories. They are normally as follows:

```
/usr/local/squid
                    /bin
                    /cache
                    /etc
                    /logs/
                    /src
```

/bin contains the squid program itself, as well as programs like ftpget, which are used by squid to perform various functions.
The /cache/ directory is where the actual cache data is stored. It contains directories along the lines of /00/ /01/ /02/ and /03/ which contain more directories, and eventually the actual data of the cache. Storing the data in multiple directories means that getting to files in large caches is still fast, since your operating system can take a long time to read a directory with 10 000 files in it.
/etc/ contains the squid.conf file, which is the only squid configuration file.
The /logs/ directory can also get large, especially if you have siblings, as they will query you with each connection, which could double the log size. Note that there is a file called /log/ in the cache directory, but you can't delete or remove it. It is an index to the /usr/local/squid/cache/ directories described above.
/src/ generally contains the source to the version of squid you are running

## A.7 Squid's log files

Squid (in it's default configuration) makes 4 log files.

- /usr/local/squid/logs/access.log
- /usr/local/squid/logs/cache.log
- /usr/local/squid/logs/store.log
- /usr/local/squid/cache/log

The first three of these log files can be safely be cycled. Although the file in the cache directory is called a log, it is actually an index of each object on disk (when squid is started it reads this file, rather than reading each and every object in the cache), and cannot thus be cycled, though it does get smaller if you send squid a kill command.

- **access.log** – Contains entries that describe each time the cache has been hit or missed when a client requests HTTP content. Along with that information is the identity of the host making the request (IP address) and the content they are requesting. We can use this information to find out when content is being used from cache and when the remote server must be accessed to obtain the content. Here is what some of the access result codes mean:

  1. TCP_DENIED: Squid denied access from the request.
  2. TCP_HIT: Cache contained a valid copy of object.
  3. TCP_IMS_HIT: A fresh version of the requested object was still in cache when the client asked if the content had changed.
  4. TCP_IMS_MISS: An If-Modified-Since request was issued by the client foe a stale object.
  5. TCP_MEM_HIT: Memory contained a valid copy of the object.
  6. TCP_MISS: Cache did not contain the object.
  7. TCP_NEGATIVE_HIT: The object was negatively cached, meaning that an error was returned (such as the file not being found) when the object requested.
  8. TCP_REF_FAIL_HIT: A stale object was returned from the cache because of a failed request to validate the object.
  9. TCP_REFRESH_HIT: A stale copy of the object was in cache, but a request to the server return information that the object has not been modified.
  10. TCP_REFRESH_MISS: A stale cache object was replaced by new, updated content.
  11. TCP_SWAPFAIL: An object could not be accessed from the cache, despite the belief that the object should have been there.

- **cache.log** – Contains valuable information about squid configuration when the squid demon starts up. We can se how much memory is available (Max Mem), how much swap space (Max Swap), the location of cache directory, the type of connection being accepted (HTTP, ICP, and SNMP), and the port on which connections are being accepted. We can also see a lot of information about the cached objects (such as how many are loaded, expired, or canceled).

- **store.log:** Contains entities that show when the content is being swapped out from memory to the cache (SWAPOUT), swapped back into the memory from the cache (SWAPIN), or released from the cache (RELEASE). You can see where the content come from originally and where it is being place in the cache. Time is logged in this file in raw UNIX time (in milliseconds).

## A.8 Tools in Squid

squid.conf file has number of options that can be adjusted to control the operation of squid proxy server:

- Network options: which specifies various port numbers on which a squid server receives and sends request.
- Options that affect the neighbor selection for caching purposes.
- Options that affect the cache size.
- Options that specifies log file pathnames and cache directories.
- Options that control the access to various users and servers.
- Different delay pool parameters.
- Certain administrative parameters
- Options for external support programs.