SNAPS: EMBEDDED NETWORK SECURITY APPLIANCE

By

DILEEP KUMAR LABANA

(07MCE024)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

MAY 2009

SNAPS: EMBEDDED NETWORK SECURITY APPLIANCE

Major Project

Submitted in partial fulfillment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

By

DILEEP KUMAR LABANA

(07MCE024)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AHMEDABAD-382481

May 2009

Certificate

This is to certify that the Major Project entitled "SNAPS: EMBEDDED NETWORK SECURITY APPLIANCE" submitted by Dileep Kumar Labana (07MCE024), towards the partial fulfillment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University of Science and Technology, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Mr. Dhaval Gajjar
Guide, Senior Software Engineer,
Solusoft Technologis Pvt. Ltd.,
Ahmedabad

Prof. Priyanka Sharma
Co-Guide, Assistant Professor,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Prof. D. J. Patel
Professor and Head,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Dr K Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad

Abstract

The bandwidth management is implemented for the purpose of availability of resource. So that fare sharing of available bandwidth between employees. The Hierarchical Token Bucket(HTB) concept is used to control the traffic between different classes. The secure socket layer VPN is implemented to provide secure access of organizations network from outside the organization. The traffic between the Web browser and the SSL VPN device is encrypted with the SSL protocol.Despite the popularity of SSL VPNs, here not intended to replace Internet Protocol Security (IPsec) VPNs. The two VPN technologies are complementary and address separate network architectures and business needs.

The growing problem of spam mails has generated a need for reliable Antispam filters. Antispam(Spam Filtering) can be applied at the client level or the server level. Several options are available at the client level for spam filtering. However, such lists are used by service providers and network administrators to block an email before it is sent. The unintended consequence of maintaining these blacklists is that sometimes, innocent senders are inadvertently blocked from sending legitimate emails.Identifying and removal of spam from the email delivery systems allows end users to regain a useful means of communication.

Acknowledgements

With immense pleasure, I would like to present this report on the dissertation work related to "SNAPS: Embedded Network Security Appliance". I am very thankful to all those who helped me for the successful completion of the dissertation and for providing valuable guidance throughout the project work. I would like to thank Solusoft India Pvt. Ltd, Ahmedabad for giving me an opportunity to successfully complete of the dissertation work in the specified duration of time. I am very thankful to our CEO Mr. Kiran Thakrar, for his support. I would like to thank our Project Leader, Mr. Dhaval Gajjar for having guided me at every step and for giving me enough freedom to use my creativity in the conception and implementation of the project idea. I am also thankful to Mr. Roysan Rajan(VP) and Mrs. Rama Ruparelia(MD) for their cooperation and help.

I like to give my special thanks to Prof. Priyanka Sharma, Assistant Professor, Department of Computer Engineering, Institute of Technology, Nirma University, Ahmedabad for her continual kind words of encouragement and motivation throughout the Major Project. I am thankful to Dr. S. N. Pradhan, Prof In Charge M Tech CSE, Institute of Technology, Nirma University, Ahmedabad for his suggestions to improve quality of work.

My kind and sincere thanks and gratitude to Dr K. Kotecha, Director Institute of Technology, Nirma University, Ahmedabad for his persistent kind words of encouragement and motivation throughout the Dissertation work. My sincere thanks and gratitude to Prof D J Patel, Professor and Head Computer Engineering Department, Institute of Technology, Nirma University, Ahmedabad for his words of encouragement and motivation throughout the Dissertation work.

The blessings of God, department faculty members and my family members makes the way for completion of major project. I am very much grateful to them.

- Dileep Kumar Labana

(07MCE024)

Contents

Certificate iii

Abstract iv

Acknowledgements v

List of Tables viii

List of Figures ix

Abbreviation x

1 Introduction 1
- 1.1 General Overview . . . 1
- 1.2 Objective of Study . . . 3
- 1.3 Scope of Work . . . 3
- 1.4 Organization of Thesis . . . 4

2 Literature Survey 6
- 2.1 Elements of Traffic Control . . . 6
  - 2.1.1 Shapping . . . 7
  - 2.1.2 Scheduling . . . 7
  - 2.1.3 policing . . . 8
  - 2.1.4 Tokens and Buckets . . . 8
- 2.2 Linux Traffic Control . . . 8
  - 2.2.1 Introduction . . . 8
  - 2.2.2 Queuing Discipline(qdisc) . . . 9
  - 2.2.3 Class . . . 10
  - 2.2.4 Filter . . . 10
  - 2.2.5 Handle . . . 11
- 2.3 Virtual Private Network . . . 11
  - 2.3.1 Types of VPN . . . 12
- 2.4 Antispam(spam filter) . . . 14
  - 2.4.1 Anti Spam Technologies . . . 15

3 Project Archicture **18**
3.1 Introduction . . . 18
3.2 Versioning System . . . 18
3.3 Project Architecture . . . 19
3.4 Installation and Configuration . . . 20
3.4.1 Installation and configuration of iproute2tool package . . . 20
3.4.2 Kernel Configuration and Compilation . . . 21

4 Bandwidth Management **23**
4.1 Introduction . . . 23
4.2 Advantages . . . 24
4.3 Disadvantages . . . 25
4.4 Implementation . . . 25
4.4.1 iproute2 tools(tc) . . . 25
4.4.2 Classful Queuing Disciplines . . . 26
4.4.3 Hierarchical Token Bucket(HTB) . . . 27
4.5 Summary . . . 33

5 SSL VPN **35**
5.1 Introduction . . . 35
5.2 Implementation of SSL VPN using Openvpn . . . 37
5.2.1 Openvpn . . . 37
5.2.2 Installation and configuration of Openvpn . . . 37
5.2.3 Creating configuration file for clients and server . . . 41
5.3 Summary . . . 48

6 Anti spam email relay server **49**
6.1 Introduction . . . 49
6.2 Proposed Spam Filter . . . 50
6.3 Implementation . . . 51
6.3.1 Postfix configuration . . . 52
6.3.2 Configuring Amavisd-new . . . 56
6.3.3 Configuring Spamassassin . . . 57
6.3.4 Postfix Anti-Spam settings . . . 57
6.3.5 Postfix content filtering control files . . . 61
6.3.6 Autolearning and sidelining emails . . . 62
6.4 Result . . . 64

7 Conclusion and Future Scope **66**
7.1 Conclusion . . . 66
7.2 Future Scope . . . 66

References **68**

List of Tables

I Private IP Address space 38
II Key Files used by clients and server 41

List of Figures

1.1 Components of SNAPS. 2
1.2 Layers of SNAPS. 3
1.3 Working of SNAPS. 4

2.1 Token bucket filter. 9
2.2 PPTP VPN. 13

3.1 Project Architecture. 20

4.1 HTB class structure and borrowing 29
4.2 HTB class states and potential actions taken 30
4.3 Linux traffic control using HTB. 32

6.1 Annual Spam Evolutions[5]. 50
6.2 Modification in master.cf file. 56
6.3 update in main.cf and transport file. 57
6.4 update in main.cf and transport file. 58
6.5 code for Spamassassin. 59
6.6 Comparison between different MTAs. 64
6.7 Performance of Spamassassin and proposed method 65

Abbreviation

SNAPS	Solusoft Network Analysis and Protection System
SSL	Secure socket Layer
VPN	Virtual Private Network
IPS	Intrusion Prevention System
TBF	Token Bucket Filter
HTB	Hierarchical Token Bucket
qdisc	Queuing Discipline
CBQ	Class Based Queuing
DNSBL	Domain Name System Blacklist
DNS	Domain Name System
CVS	Concurrent Versioning System
FIFO	First IN First Out
QoS	Quality of Service
IPsec	Internet Protocol Security
PKI	public key infrastructure
ISP	Internet service Provider
SMTP	Simple Mail Transport Protocol
MTA	Mail Transfer Agent

Chapter 1

Introduction

This chapter covers general overview of the Thesis. It include the motivation and the scope of the Thesis work. It also guides about the organization of the Thesis report.

In today's world we see that there are many standalone systems available and we use them for our day to day use. These systems include computers as one of their parts. The computers interact with the hardware and then do the operation. That's called embedded systems. This thesis, focus on developing some modules of SNAPS(Solusoft Network Analysis and Protection System). SNAPS is unique security product which is developed by Solusoft Pvt. Ltd. It is used for surf protection, bandwidth management, Antispam, SSL VPN and firewall.

Organizations become more and more averse to risk as they mature. An organization under the gun to demonstrate increased CMM(Capability Maturity Model) level is not going to go looking for real challenge.

1.1 General Overview

SNAPS,unique network security solution in an integrated and easy-to-use and manage package. With SNAPS, complete security comes from integrating a variety of security countermeasures into the security appliance to counteract multiple types of security threats.

SNAPS provides with six critical security applications - Firewall, VPN Gateway, Intrusion Protection, Anti-Virus, Surf Protection, Traffic Controller as shown in Figure 1.1.

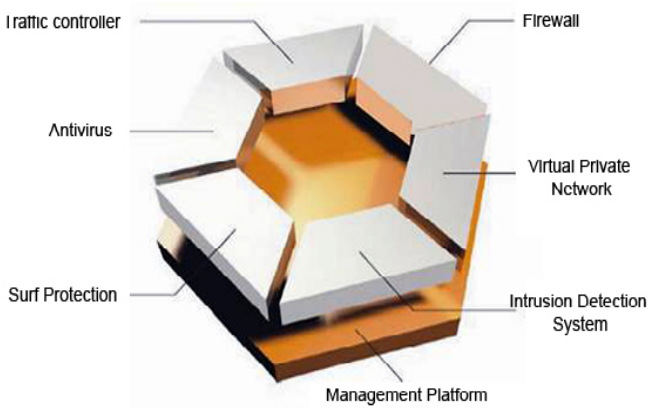

Figure 1.1: Components of SNAPS.

In SNAPS, There are three basic layers as shown in figure 1.2. First layer (Lower Layer) contains hardware and OS. We are using specific hardware specified in requirement section. We are using Linux as OS platform.

Second Layer (Middle Layer) contains some Software which is used to implement the features like Firewall, IPS, VPN, Serf Protection, Traffic Controller and Antispam. For various features we are using iptables for Firewall, snort for IPS, openvpn for VPN, squid for Serf Protection, Postfix for Antispam and havp for Antivirus proxy. Third Layer (Upper Layer) contains the user interface through which a user can configure SNAPS. We have used PHP as a platform for making GUI.

Figure 1.3 shows us how SNAPS works on the network. SNAPS has four LAN port. First is called Green Port, which is connected to internal LAN computer. Second Port is called Red port, which is connected to the internet. Third port is called Orange port, which is connected to public servers. So administrator can specify some rules for LAN computer which are there in the configuration page.

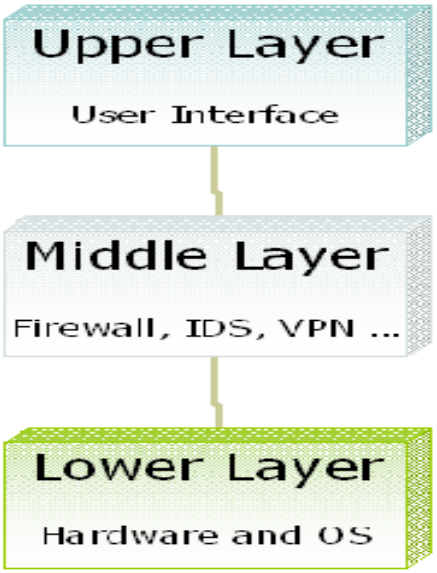

Figure 1.2: Layers of SNAPS.

1.2 Objective of Study

The goal of this project is to design and implement network security appliance that provides complete cost effective protection against a wide range of network security threats and bandwidth management. The system will be built in hard drive provides for extensive reporting and allows transaction logs to be stored locally. The compact size, high reliability and economical price makes this appliance a perfect fit for small to medium size organization.

1.3 Scope of Work

The bandwidth management is implemented for the purpose of availability of resource. So that fare sharing of available bandwidth between employees. And secure socket layer VPN is developed to provide secure access of organization's network from outside the organization. Also the Antispam is developed to block the spam mail to reach the end user.

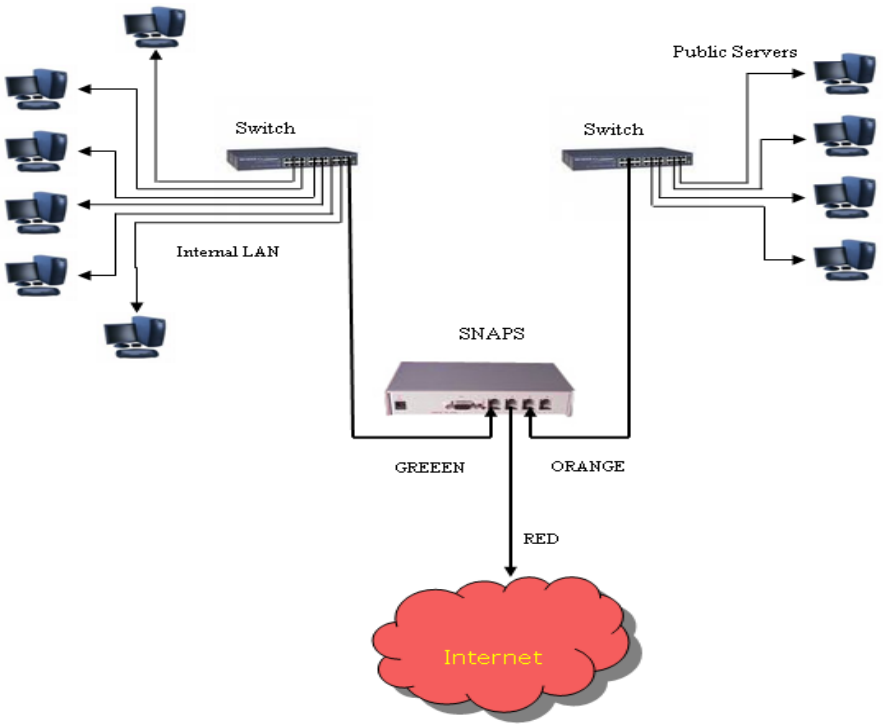

Figure 1.3: Working of SNAPS.

1.4 Organization of Thesis

The Thesis covers the different modules of SNAPS. It covers mainly three modules of SNAPS which are bandwidth management, SSL VPN and Antispam(spam filter).

Chapter 2, *Literature Survey*, covers a literature survey for different technique of bandwidth management,different methods of VPN and survey reports on various security threats.

Chapter 3, *Project Archicture*, covers the project architecture,installation and configuration of different software packages. It also includes kernel configuration and compilation.

Chapter 4, *Bandwidth Management*, covers the different techniques of bandwidth management and implementation of HTB technique. It also covers the advan-

tages and disadvantages of different techniques.

Chapter 5, *SSL VPN*, covers the introduction of VPN, different types of VPN and implementation of SSL VPN.

Chapter 6, *Antispam*, covers the study of different spam filter techniques and implementation of spam filter.It includes the experiment steps and proof towards the goal.

Chapter 7, *Conclusion and Future Scope*, This chapter includes conclusion and future scope of the Thesis work.

Chapter 2

Literature Survey

This chapter covers the related work and the latest studies in the literature on each of these building blocks.There have been a number of surveys about network security, banwidth management, secure socket layer virtual private network(SSL VPN) and spam filter. The survey we present here covers only those work that are in the same context as our study. However, for comprehensive completeness, we also give brief information on some techniques which are used for similar tasks that are not covered in our study.

Most IT people when having network congestion problems consider, without doing an in-depth analysis of the situation that the obvious and only solution consists in buying more bandwidth to have the job done. They forget that what seems to be a logical solution, is not more than a source of unnecessary expenses because the problem is alleviated temporarily, to become again, sooner than later, in the same, or perhaps worst problem.

2.1 Elements of Traffic Control

Traffic control is the name given to the sets of queuing systems and mechanisms by which packets are received and transmitted on a router. This includes deciding which (and whether) packets to accept at what rate on the input of an interface and

determining which packets to transmit in what order at what rate on the output of an interface. In the overwhelming majority of situations, traffic control consists of a single queue which collects entering packets and dequeues them as quickly as the hardware (or underlying device) can accept them.

2.1.1 Shapping

Shaping is the mechanism by which packets are delayed before transmission in an output queue to meet a desired output rate. This is one of the most common desires of users seeking bandwidth control solutions. The act of delaying a packet as part of a traffic control solution makes every shaping mechanism into a non work conserving mechanism, meaning roughly: "Work is required in order to delay packets."

Shapers attempt to limit traffic to meet but not exceed a configured rate (frequently measured in packets per second or bits/bytes per second). As a side effect, shapers can smooth out bursty traffic. One of the advantages of shaping bandwidth is the ability to control latency of packets.

2.1.2 Scheduling

Schedulers arrange and/or rearrange packets for output. Scheduling is the mechanism by which packets are arranged (or rearranged) between input and output of a particular queue. The overwhelmingly most common scheduler is the FIFO (first_in first_out) scheduler. From a larger perspective, any set of traffic control mechanisms on an output queue can be regarded as a scheduler, because packets are arranged for output.

Other generic scheduling mechanisms attempt to compensate for various networking conditions. A fair queuing algorithm attempts to prevent any single client or flow from dominating the network usage. A round_robin algorithm gives each flow or client a turn to dequeue packets. Other sophisticated scheduling algorithms attempt to prevent backbone overload.

2.1.3 policing

Policers measure and limit traffic in a particular queue. Policing, as an element of traffic control, is simply a mechanism by which traffic can be limited. Policing is most frequently used on the network border to ensure that a peer is not consuming more than its allocated bandwidth. A policer will accept traffic to a certain rate, and then perform an action on traffic exceeding this rate.

2.1.4 Tokens and Buckets

Two of the key underpinnings of a shaping mechanisms are the interrelated concepts of tokens and buckets. In order to control the rate of dequeuing, an implementation can count the number of packets or bytes dequeued as each item is dequeued, although this requires complex usage of timers and measurements to limit accurately. Instead of calculating the current usage and time, one method, used widely in traffic control, is to generate tokens at a desired rate, and only dequeue packets or bytes if a token is available.

In summary, tokens are generated at rate, and a maximum of a bucket's worth of tokens may be collected. This allows bursty traffic to be handled, while smoothing and shaping the transmitted traffic. As shown in Figure 2.1. The concepts of tokens and buckets are closely interrelated and are used in both TBF (one of the classless qdiscs) and HTB (one of the classful qdiscs).

2.2 Linux Traffic Control

2.2.1 Introduction

Traffic control encompasses the sets of mechanisms and operations by which packets are queued for transmission/reception on a network interface. The operations include enquiring, policing, classifying, scheduling, shaping and dropping.

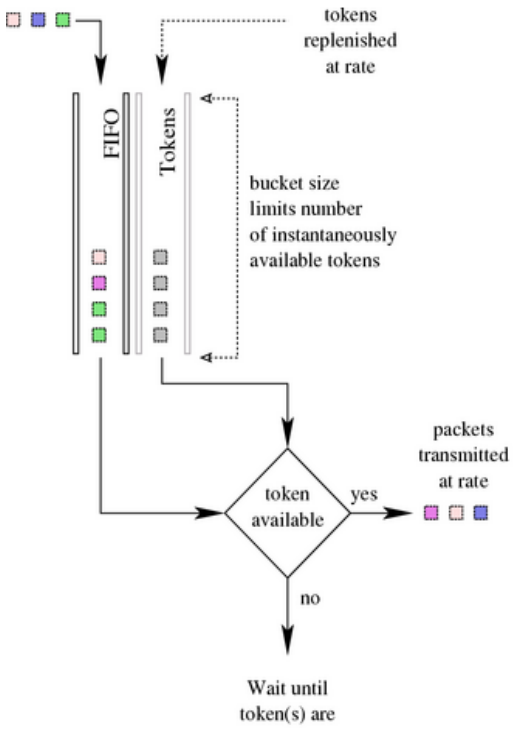

Figure 2.1: Token bucket filter.

Linux offers a very rich set of tools for managing and manipulating the transmission of packets. The larger Linux community is very familiar with the tools available under Linux for packet mangling and firewalling(netfilter, and before that, ipchains) as well as hundreds of network services which can run on the operating system.

2.2.2 Queuing Discipline(qdisc)

The qdisc is the major building block on which all of Linux traffic control is built, and is also called a queuing discipline.Simply put, a qdisc is a scheduler. Every output interface needs a scheduler of some kind, and the default scheduler is a FIFO. Other qdiscs available under Linux will rearrange the packets entering the scheduler's queue in accordance with that scheduler's rules.

The classful qdiscs can contain classes, and provide a handle to which to attach filters. There is no prohibition on using a classful qdisc without child classes, although

this will usually consume cycles and other system resources for no benefit. The classless qdiscs can contain no classes, nor is it possible to attach filter to a classless qdisc. Because a classless qdisc contains no children of any kind, there is no utility to classifying. This means that no filter can be attached to a classless qdisc.

2.2.3 Class

Classes only exist inside a classful qdisc (e.g., HTB and CBQ). Classes are immensely flexible and can always contain either multiple children classes or a single child qdisc. There is no prohibition against a class containing a classful qdisc itself, which facilitates tremendously complex traffic control scenarios.

Any class can also have an arbitrary number of filters attached to it, which allows the selection of a child class or the use of a filter to reclassify or drop traffic entering a particular class. A leaf class is a terminal class in a qdisc. It contains a qdisc (default FIFO) and will never contain a child class. Any class which contains a child class is an inner class (or root class) and not a leaf class.

2.2.4 Filter

The filter is the most complex component in the Linux traffic control system. The filter provides a convenient mechanism for gluing together several of the key elements of traffic control. The simplest and most obvious role of the filter is to classify packets. Linux filters allow the user to classify packets into an output queue with either several different filters or a single filter.

Filters can be attached either to classful qdiscs or to classes, however the enqueued packet always enters the root qdisc first. After the filter attached to the root qdisc has been traversed, the packet may be directed to any subclasses (which can have their own filters) where the packet may undergo further classification.

2.2.5 Handle

Every class and classful qdisc requires a unique identifier within the traffic control structure. This unique identifier is known as a handle and has two constituent numbers, a major number and a minor number. These numbers can be assigned arbitrarily by the user in accordance with the following rules. The numbering of handles for classes and qdiscs are as follows:

Major : This parameter is not used by the kernel. The user may use an arbitrary numbering scheme, however all objects in the traffic control structure with the same parent must share a major handle number. Conventional numbering schemes start at 1 for objects attached directly to the root qdisc.

Minor : This parameter unambiguously identifies the object as a qdisc if minor is 0. Any other value identifies the object as a class. All classes sharing a parent must have unique minor numbers.

The special handle ffff: 0 is reserved for the ingress qdisc. The handle is used as the target in classid and flowid phrases of tc filter statements. These handles are external identifiers for the objects, usable by user and applications. The kernel maintains internal identifiers for each object.

2.3 Virtual Private Network

Virtual private networks (VPNs) are a fairly complex subject; there is no single defining product, nor even much of a consensus among VPN vendors as to what comprises a VPN. Consequently, everyone knows what a VPN is, but establishing a single definition can be remarkably difficult. Some definitions are sufficiently broad as to enable one to claim that Frame Relay qualifies as a VPN when, in fact, it is an overlay network. Although an overlay network secures transmissions through a public network, it does so passively via logical separation of the data streams.

VPNs provide a more active form of security by either encrypting or encapsulating data for transmission through an unsecured network. These two types of security-encryption and encapsulation-form the foundation of virtual private networking. However, both encryption and encapsulation are generic terms that describe a function that can be performed by a myriad of specific technologies. To add to the confusion, these two sets of technologies can be combined in different implementation topologies. Thus, VPNs can vary widely from vendor to vendor.

2.3.1 Types of VPN

A VPN (Virtual Private Network) is a way of creating a secure connection to and from a network or computer. VPNs have been used for years, but they have become more robust in recent years. They are more affordable and also much faster. There are many different types of VPNs available. Let's take a look at most common types.

PPTP VPN(Dial-up VPN)

A simple method for VPN is PPTP. It is a software based VPN system that uses your existing Internet connection. By using your existing Internet connection, a secure "tunnel" is created between two points allowing a remote user to connect to a remote network. You can setup this type of connection with various types of software or hardware. Windows Server has a PPTP build-it and you can connect to it via a native VPN client within Windows. Cisco also have this ability, but require a 3rd party software to be loaded on remote workstations. There is some overhead associated with this as all data transmitted and received in encrypted. This can be referred to as the poor man's VPN. There is little to no cost to setup this type of VPN, and you can often use your existing equipment and software. It is sometimes referred to as "dial-up VPN" because when the client software connects it looks like it's dialing up. See the diagram below(Figure 2.2):

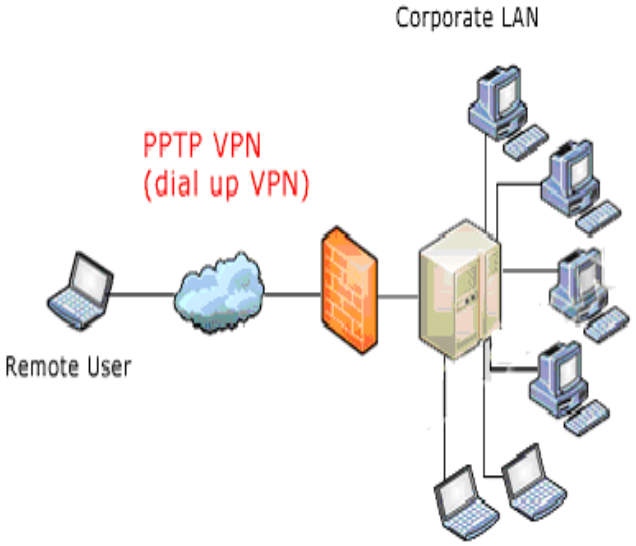

Figure 2.2: PPTP VPN.

Site-to-Site VPN

Site-to-site is as same the point-to-point except there is no "dedicated" line in use. Each site has it's own internet connection which may not be from the same ISP or even the same type. One may have a T1 while the other only has DSL. Unlike point-to-point, the routers at both ends do all the work.

They do all the routing and encryption. This is an easy way to connect two offices without having each user "dial-up" using a PPTP connection. Site-to-site VPNs can work with hardware or firewall devices. On the software side, you can use something like Clark connects. On the hardware side, you can have many different devices to choose from.

Point-to-Point VPN

Another site to site VPN is a point-to-point Simply put, two or more networks are connected using a dedicated line from an ISP. The main strength of using a leased line is that is a circuit-based point-to-point connection. It does not go out over the

public Internet, so there performance is not degraded by routing problems, latency, and external congestion.

Most client to site VPNs are based around IPSec (short for IP Security), which is a suite of protocols developed by the IETF to support secure exchange of packets at the IP layer. Typically, an IPSec tunnel connection will be created from a Client software component to a VPN gateway (or firewall with VPN functionality). Following the initialization of this tunnel, all packets destined for the remote corporate network will be routed down this tunnel. The tunnel provides the necessary security, by encrypting each packet (using one of a selection of algorithms) before forwarding it to the remote gateway. When packets reach the remote gateway, they are decrypted and then forwarded 'in the clear' to the final destination.

IPSec was initially devised for site to site VPN connections, so to add the necessary functionality to IPSec to allow effective client to site connections and management, each vendor has added vendor specific features to it's IPSec implementation.

2.4 Antispam(spam filter)

When the average number of spam messages received is continually increasing exponentially, both the Internet Service Provider and the end user suffer. The lack of an efficient solution may threaten the usability of the email as a communication means.

Email filtering is the process of monitoring incoming (or outgoing) email, and then taking certain actions when an email is considered to be SPAM [4]. Spam constitutes a major problem for both e-mail users and Internet Service Providers (ISP) [5]. In general the word "spam" is used to refer to unwanted, "junk" email messages. Spam can often be referred to as unsolicited commercial e-mail or unsolicited bulk email; however, not all unsolicited e-mails are necessarily spam.

2.4.1 Anti Spam Technologies

Over the past few years, a lot of anti-spam tools and solutions based on different technological approaches have been developed [7]. However, as you will see below, there are significant differences in terms of the effectiveness of each approach.

Centralized filtering server

In this architecture, a single anti-spam filter runs on a centralized organization-wide mail server. This approach eliminates the need to deploy software to email clients or to train users. Centralized filters have the disadvantage that they do not typically use the specific preferences and opinions of the user.

Gateway Filtering

In this approach, all inbound email is routed through a filtering gateway before being delivered to the mail server. Gateway services work well with webbased and mobile access to email, and may increase robustness since they queue emails if the client network or server is off-line. On the other hand, the gateway itself is a single point of failure and may be difficult to manage in the presence of multiple mail servers within an organization [3].

Rule-based filtering

Rule-based filters assign a spam score to each email based on whether the email contains features typical of spam messages, such as keywords and HTML formatting like fancy fonts and background colors. A major problem with rule-based scores is that since their semantics are not well-defined, it is difficult to aggregate them and to establish a threshold that can actually limit the number of false positives.

Heuristic Filtering

In essence, heuristic filtering is a method of spam detection that uses baseline artificial intelligence to deliver an automated spam deletion process [5]. These automated mechanisms categorize incoming email messages as spam or legitimate based on known spam patterns. In theory, the advantage of this process lies in its automated nature and the fact that it should require no human intervention in the process of message classification. In reality, however, the greatest advantage of heuristics emerges as its greatest weakness.

Collaborative spam filtering

In collaborative approaches, server-side automatic monitoring systems consider whether incoming messages are to be known spam after these messages are classified by an automatic mechanism or by final recipients. These solutions have achieved considerable success as they overcome the single point of failure typical of centralized architecture. All the solutions presented above have strengths and weaknesses.

It is clear that no single technology is powerful enough to block all the spam that might flood an average mail server [7]. In fact, most anti-spam solutions combine two or more technologies in an attempt to improve their overall effectiveness, while decreasing their false positives ratio.

Blacklists

The efficiency of a blacklist can be measured by the rate in which the list is updated [11]. As soon as a machine is detected as a spam source, its IP address should be included on the blacklist. The period of time in which addresses that are no longer sending spam are removed is also important, to reduce false positives. Blacklists can be either managed by users or by organizations responsible for the list management. User-managed blacklists are rare, as they require constant user interaction to remove and add addresses. The blacklists commonly used are managed by organizations.

The lists are queried using the DNS protocol and are called DNSBL (Domain Name System Blacklist) [12]. The DNS protocol is used to query these lists because it is a well established protocol, its implementation is already mature and it also provides cache of queries, reducing the bandwidth usage.

Bayesian Filters

Bayesian filters are an evolution of rule-based systems because the rules to classify the messages are automatically created when training the filter. The filter uses a Bayesian classifier, which is trained with some messages previously classified as either legitimate or spam. Using these training messages, the classifier can automatically discover characteristics that are present on both legitimate and spam messages. To classify the messages it is verified if the characteristics already learned by the filter are present on the message. The probability of a message be classified as spam given its characteristics is calculated by the product of the probability that the characteristics are present on spam messages multiplied by the probability of a message be spam divided by the probability that the characteristics are present on all the messages used to train the filter.

It is clear that no single technology is powerful enough to block all the spam that might flood an average mail server [7]. In fact, most anti-spam solutions combine two or more technologies in an attempt to improve their overall effectiveness, while decreasing their false positives ratio.

Chapter 3

Project Archicture

3.1 Introduction

SNAPS contain six modules and each module requires exhaustive analysis and research oriented work. Three modules implemented in this system which are Bandwidth management, SSL VPN and spam filter.

3.2 Versioning System

In this CVS (Concurrent Versioning System)is used for coordination between developers and for taking backup of code. CVS uses client-server architecture: a server stores the current version(s) of the project and its history, and clients connect to the server in order to check out a complete copy of the project, work on this copy and then later check in their changes. Typically, client and server connect over a LAN or over the Internet, but client and server may both run on the same machine if CVS has the task of keeping track of the version history of a project with only local developers. Our CVS server runs on Debian Linux machine while clients are running on Linux as well as Windows machines.

Each of the developer is allowed to edit files within their own working copy of the project, and sending (or checking in) their modifications to the server. To avoid the

possibility of people stepping on each other's toes, the server will only accept changes made to the most recent version of a file. Developers are therefore expected to keep their working copy up-to-date by incorporating other people's changes on a regular basis. This task is mostly handled automatically by the CVS client, requiring manual intervention only when a conflict arises between a checked-in modification and the yet-unchecked local version of a file.

3.3 Project Architecture

This done using SCRUM development model for developing modules of SNAPS. Scrum is an iterative incremental process of software development commonly used with software development. Despite the fact that "Scrum" is not an acronym, some companies implementing the process have been known to adhere to an all capital letter expression of the word, i.e. SCRUM. The SCRUM methodology is an approach for systems development which is based on both defined and black box process management. SCRUM is an enhancement of the iterative and incremental approach to deliver product. The SCRUM approach assumes that the analysis, design, and development processes in the Sprint phase are unpredictable. A control mechanism is used to manage the unpredictability and control the risk. Flexibility, responsiveness, and reliability are the results.

- The first and last phases (Planning and Closure) consist of defined processes, where all processes, inputs and outputs are well defined. The knowledge of how to do these processes is explicit. The flow is linear, with some iteration in the planning phase.

- The Sprint phase is an empirical process. Many of the processes in the sprint phase are unidentified or uncontrolled. It is treated as a black box that requires external controls. Accordingly, controls, including risk management, are put on each iteration of the Sprint phase to avoid chaos while maximizing flexibility.

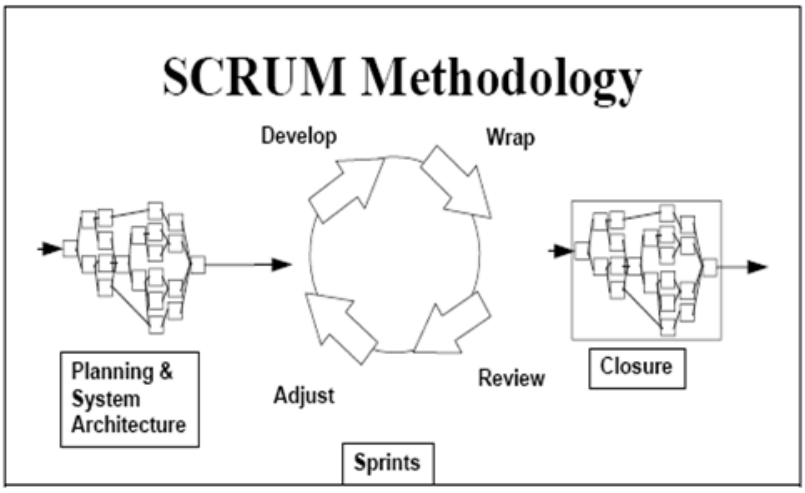

Figure 3.1: Project Architecture.

- Sprints are nonlinear and flexible. Where available, explicit process knowledge is used, otherwise tacit knowledge and trial and error is used to build process knowledge. Sprints are used to evolve the final product.

- The project is open to the environment until the Closure phase. The deliverable can be changed at any time during the Planning and Sprint phases of the project. The project remains open to environmental complexity, including competitive, time, quality, and financial pressures, throughout these phases.

3.4 Installation and Configuration

3.4.1 Installation and configuration of iproute2tool package

Most linux distributions are starting to provide the iproute2 package, because of the new redesigned network subsystem implemented in kernels 2.2 and up. The old comands 'ifconfig' and 'route' are now been deprecated because of their faulty and unexpected behaviour under these kernels. This new routing and filtering code provides many advantages and features that weren't available before, and ip/tc are

the tools to handle it. This package requires db. So first we require to download tar file then install:

http://www.sleepycat.com/update/snapshot/db-4.1.25.tar.gz

tar-zxvf db-4.1.25.tar.gz

cd db-4.1.25/dist

./configure –prefix=/usr –enable-compat185

make

make install

In linux Debian flavor we can install using following command:

[**root@debian]sudo aptitude install 'package name'**

iproute2-2.4.7-now-ss020116-try.tar.gz and kernel-2.4.20,

tar -zxvf htb3.6-020525.tgz

tar -zxvf iproute2-2.4.7-now-ss020116-try.tar.gz

cd iproute2

3.4.2 Kernel Configuration and Compilation

Many distributions provide kernels with modular or monolithic support for traffic control (Quality of Service). Custom kernels may not already provide support (modular or not) for the required features. If not, this is a very brief listing of the required kernel options.

Kernel compilation options: #

#QoS and/or fair queueing

#

CONFIG_NET_SCHED=y

CONFIG_NET_SCH_CBQ=m

CONFIG_NET_SCH_HTB=m

CONFIG_NET_SCH_CSZ=m

CONFIG_NET_SCH_PRIO=m

CONFIG_NET_SCH_RED=m

CONFIG_NET_SCH_SFQ=m

CONFIG_NET_SCH_TEQL=m

CONFIG_NET_SCH_TBF=m

CONFIG_NET_SCH_GRED=m

CONFIG_NET_SCH_DSMARK=m

CONFIG_NET_SCH_INGRESS=m

CONFIG_NET_QOS=y

CONFIG_NET_ESTIMATOR=y

CONFIG_NET_CLS=y

CONFIG_NET_CLS_TCINDEX=m

CONFIG_NET_CLS_ROUTE4=m

CONFIG_NET_CLS_ROUTE=y

CONFIG_NET_CLS_FW=m

CONFIG_NET_CLS_U32=m

CONFIG_NET_CLS_RSVP=m

CONFIG_NET_CLS_RSVP6=m

CONFIG_NET_CLS_POLICE=y

A kernel compiled with the above set of options will provide modular support for almost everything discussed in this documentation. The user may need to modprobe module before using a given feature.

Chapter 4

Bandwidth Management

4.1 Introduction

Traffic control is the name given to the sets of queuing systems and mechanisms by which packets are received and transmitted on a router. This includes deciding which (and whether) packets to accept at what rate on the input of an interface and determining which packets to transmit in what order at what rate on the output of an interface.

In the overwhelming majority of situations, traffic control consists of a single queue which collects entering packets and dequeues them as quickly as the hardware (or underlying device) can accept them. This sort of queue is a FIFO. The default qdisc under Linux is the pfifo_fast which is slightly more complex than the FIFO.

There are examples of queues in all sorts of software. The queue is a way of organizing the pending tasks or data. Because network links typically carry data in a serialized fashion, a queue is required to manage the outbound data packets.

In the case of a desktop machine and an efficient webserver sharing the same uplink to the Internet, the following contention for bandwidth may occur. The web server may be able to fill up the output queue on the router faster than the data can be transmitted across the link, at which point the router starts to drop packets

(its buffer is full!). Now, the desktop machine (with an interactive application user) may be faced with packet loss and high latency. Note that high latency sometimes leads to screaming users! By separating the internal queues used to service these two different classes of application, there can be better sharing of the network resource between the two applications.

Traffic control is the set of tools which allows the user to have granular control over these queues and the queuing mechanisms of a networked device. The power to rearrange traffic flows and packets with these tools is tremendous and can be complicated, but is no substitute for adequate bandwidth. The term Quality of Service (QoS) is often used as a synonym for traffic control.

Packet switched networks differ from circuit based networks in one very important regard. A packet switched network itself is stateless. A circuit based network (such as a telephone network) must hold state within the network. IP networks are stateless and packet switched networks by design; in fact, this statelessness is one of the fundamental strengths of IP.

The weakness of this statelessness is the lack of differentiation between types of flows. In simplest terms, traffic control allows an administrator to queue packets differently based on attributes of the packet. It can even be used to simulate the behavior of a circuit based network. This introduces statefulness into the stateless network.

4.2 Advantages

When properly employed, traffic control should lead to more predictable usage of network resources and less volatile contention for these resources. The network then meets the goals of the traffic control configuration. Bulk download traffic can be allocated a reasonable amount of bandwidth even as higher priority interactive traffic is simultaneously serviced. Even low priority data transfer such as mail can be allocated bandwidth without tremendously affecting the other classes of traffic.

4.3 Disadvantages

Complexity is easily one of the most significant disadvantages of using traffic control. There are ways to become familiar with traffic control tools which ease the learning curve about traffic control and its mechanisms, but identifying a traffic control miss configuration can be quite a challenge.

Traffic control when used appropriately can lead to more equitable distribution of network resources. It can just as easily be installed in an inappropriate manner leading to further and more divisive contention for resources.

The computing resources required on a router to support a traffic control scenario need to be capable of handling the increased cost of maintaining the traffic control structures. Fortunately, this is a small incremental cost, but can become more significant as the configuration grows in size and complexity.

4.4 Implementation

4.4.1 iproute2 tools(tc)

iproute2 is a suite of command line utilities which manipulate kernel structures for IP networking configuration on a machine. Because it interacts with the kernel to direct the creation, deletion and modification of traffic control structures, the tc binary needs to be compiled with support for all of the qdisc you wish to use. In particular, the HTB qdisc is not supported yet in the upstream iproute2 package.

The tc tool performs all of the configuration of the kernel structures required to support traffic control. As a result of its many uses, the command syntax can be described (at best) as arcane. The utility takes as its first non option argument one of three Linux traffic control components, qdisc, class or filter. Here below some of the examples are shown with tc command.

Example1: tc command usage

[root@debian]tc

Usage: tc [OPTIONS]OBJECT { COMMAND — help }
where OBJECT := { qdisc — class — filter}
OPTIONS := { s[tatistics] — d[etails] — r[aw] }

Example2: qdisc

[rootdebian]# tc qdisc add dev eth0 root handle 1:0 htb

Example3: class

[rootdebian]# tc class add dev eth0 parent 1:1 classid 1:6 htb rate 256kbit ceil 512kbit

Example3: filter

[root@debian]# tc filter add dev eth0 parent 1:0 protocol ip prio5 u32 match ip port 22 0xffff match ip tos 0x10 0xff flowid 1:6 police rate 32000bps burst 10240 mpu 0 action drop/continue

Traffic control is implemented using the linux iproute2 package. For this we required to first configure the kernel. In iproute2 package tc command is provide the granular control over traffic.

4.4.2 Classful Queuing Disciplines

The flexibility and control of Linux traffic control can be unleashed through the agency of the classful qdiscs. Remember that the classful queuing disciplines can have filters attached to them, allowing packets to be directed to particular classes and subqueues.

There are several common terms to describe classes directly attached to the root qdisc and terminal classes. Classes attached to the root qdisc are known as root classes, and more generically inner classes. Any terminal class in a particular queuing discipline is known as a leaf class by analogy to the tree structure of the classes.

Besides the use of figurative language depicting the structure as a tree, the language of family relationships is also quite common.

4.4.3 Hierarchical Token Bucket(HTB)

HTB uses the concepts of tokens and buckets along with the class based system and filters to allow for complex and granular control over traffic. With a complex borrowing model, HTB can perform a variety of sophisticated traffic control techniques. One of the easiest ways to use HTB immediately is that of shaping. By understanding tokens and buckets or by grasping the function of TBF, HTB should be merely a logical step. This queuing discipline allows the user to define the characteristics of the tokens and bucket used and allows the user to nest these buckets in an arbitrary fashion. When coupled with a classifying scheme, traffic can be controlled in a very granular fashion.

Below is example output of the syntax for HTB on the command line with the tc tool. Although the syntax for tcng is a language of its own, the rules for HTB are the same.

Example: tc usage for HTB Usage: ... qdisc add ... htb [default N][r2q N]

default	minor id of class to which unclassified packets are sent{0}
r2q	DRR quantums are computed as rate in Bps/r2q10
debug	string of 16 numbers each 0?3 {0}
class add	htb rate R1 burst B1 [prio P] [slot S] [pslot PS] [ceil R2] [cburst B2] [mtu MTU] [quantum Q]
rate	rate allocated to this class (class can still borrow)
burst	max bytes burst which can be accumulated during idle period (computed)
ceil	definite upper class rate (no borrows) (rate)
cburst	burst but for ceil (computed)
mtu	max packet size we create rate map for (1600)
prio	priority of leaf; lower are served first(0)
quantum	how much bytes to serve from leaf at once(use r2q)

HTB is a newer queuing discipline and your distribution may not have all of the tools and capability you need to use HTB.

Shaping

One of the most common applications of HTB involves shaping transmitted traffic to a specific rate. All shaping occurs in leaf classes. No shaping occurs in inner or root classes as they only exist to suggest how the borrowing model should distribute available tokens.

Borrowing

A fundamental part of the HTB qdisc is the borrowing mechanism. Children classes borrow tokens from their parents once they have exceeded rate. A child class will continue to attempt to borrow until it reaches ceil, at which point it will begin to queue packets for transmission until more tokens/ctokens are available. As there are only two primary types of classes which can be created with HTB the figure 4.2 and figure 4.1identify the various possible states and the behavior of the borrowing mechanisms. As shown in figure 4.2 and figure 4.1.

This figure identifies the flow of borrowed tokens and the manner in which tokens are charged to parent classes. In order for the borrowing model to work, each class must have an accurate count of the number of tokens used by itself and all of its children. For this reason, any token used in a child or leaf class is charged to each parent class until the root class is reached.

Any child class which wishes to borrow a token will request a token from its parent class, which if it is also over its rate will request to borrow from its parent class until either a token is located or the root class is reached. So the borrowing of tokens flows toward the leaf classes and the charging of the usage of tokens flows toward the root class.

Note in this diagram that there are several HTB root classes. Each of these root classes can simulate a virtual circuit.

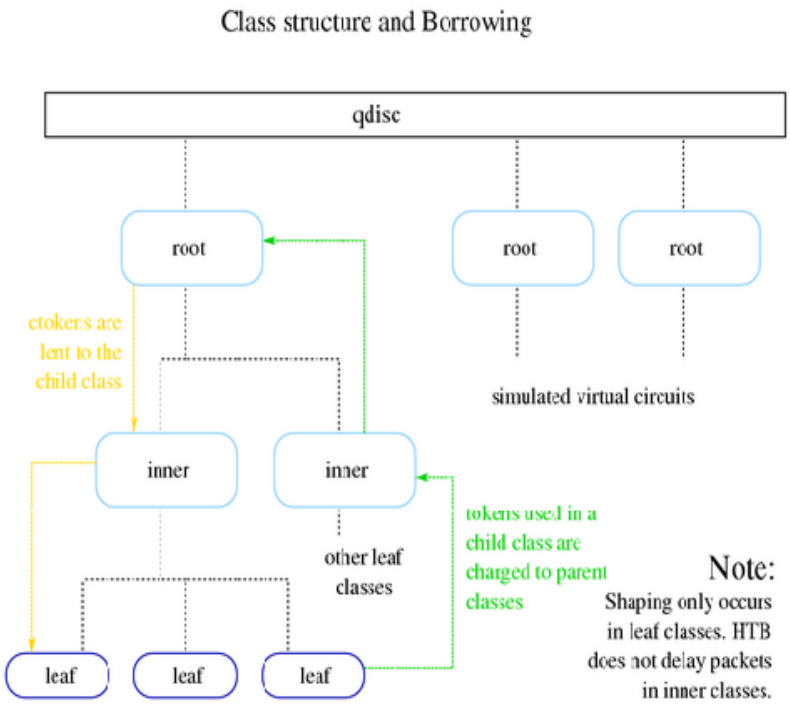

Figure 4.1: HTB class structure and borrowing

HTB class parameters

HTB class uses the following parameter in queuing discipline:

default :

An optional parameter with every HTB qdisc object, the default default is 0, which cause any unclassified traffic to be dequeued at hardware speed, completely bypassing any of the classes attached to the root qdisc.

rate :

Used to set the minimum desired speed to which to limit transmitted traffic. This can be considered the equivalent of a committed information rate (CIR), or the guaranteed bandwidth for a given leaf class.

class	Class state	HTB internal state	Action taken
Leaf	lessthan rate	HTB_CAN_SEND	Leaf class will deque queued byteupto available tokens
	greaterthan rate lessthan ceil	HTB_MAY_BORROW	Leaf class will attempt to borrow tokens from parent class.If Tokens are available, they will lent in quantum increments and the leaf classes deque up to cburst bytes
	greater than ceil	HTB_CANT_SEND	No packets will be dequeued. This will cause packet delay and will increase latency to meet desired rate
Inner root	less than rate	HTB_CAN_SEND	Inner class will lend token to children
	greater than rate lessthan ceil	HTB_MAY_BORROW	Inner class will attempt to borrow tokens from parent class, lending them to competing children in quatum increment per requirement
	greater than ceil	HTB_CANT_SEND	Inner class will attempt to borrow from its parent and will not lend tokens to children class

Figure 4.2: HTB class states and potential actions taken

ceil :

Used to set the maximum desired speed to which to limit the transmitted traffic. The borrowing model should illustrate how this parameter is used. This can be considered the equivalent of "burstable bandwidth".

burst :

This is the size of the rate bucket (see Tokens and buckets). HTB will dequeue burst bytes before awaiting the arrival of more tokens.

cburst :

This is the size of the ceil bucket (see Tokens and buckets). HTB will dequeue cburst bytes before awaiting the arrival of more ctokens.

quantum :

This is a key parameter used by HTB to control borrowing. Normally, the correct quantum is calculated by HTB, not specified by the user. Tweaking this parameter can have tremendous effects on borrowing and shaping under contention, because it is used both to split traffic between children classes over rate (but below ceil) and to transmit packets from these same classes.

r2q :

Also, usually calculated for the user, r2q is a hint to HTB to help determine the optimal quantum for a particular class.

Rules for using HTB

Below are some general guidelines to using HTB for traffic control. These rules are simply a recommendation for beginners to maximize the benefit of HTB until gaining a better understanding of the practical application of HTB.

- Shaping with HTB occurs only in leaf classes.
- Because HTB does not shape in any class except the leaf class, the sum of the rates of leaf classes should not exceed the ceil of a parent class. Ideally, the sum of the rates of the children classes would match the rate of the parent class, allowing the parent class to distribute leftover bandwidth (ceil ? rate) among the children classes. This key concept in employing HTB bears repeating. Only leaf classes actually shape packets; packets are only delayed in these leaf classes. The inner classes (all the way up to the root class) exist to define how borrowing/lending occurs.
- The quantum is only only used when a class is over rate but below ceil.
- The quantum should be set at MTU or higher. HTB will dequeue a single packet at least per service opportunity even if quantum is too small. In such a case, it will not be able to calculate accurately the real bandwidth consumed.

- Parent classes lend tokens to children in increments of quantum, so for maximum granularity and most instantaneously evenly distributed bandwidth, quantum should be as low as possible while still no less as MTU.

- A distinction between tokens and ctokens is only meaningful in a leaf class, because non?leaf classes only lend tokens to child classes.

- HTB borrowing could more accurately be described as "using".

Finally the simplified linux traffic control snerio is shown in the following figure 4.3.

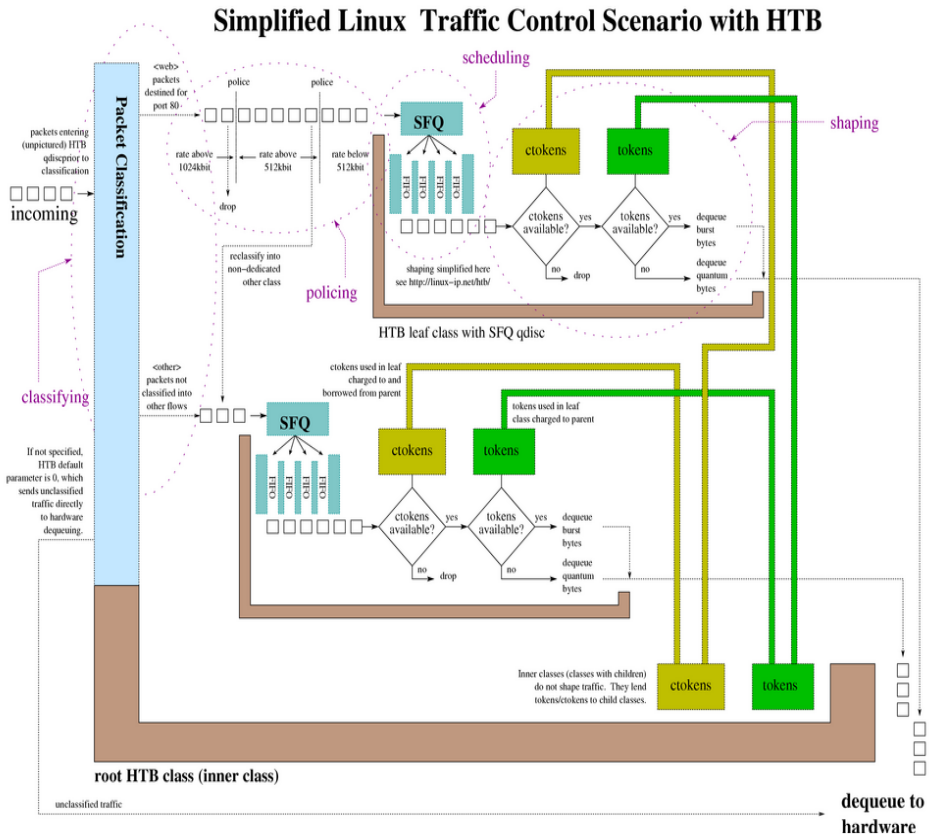

Figure 4.3: Linux traffic control using HTB.

4.5 Summary

In bandwidth management takes lot of time to analysis the current situation. So first thing is require to identifying organization requirement. And SNAPS has four port for interfaces with external and internal connection.

First thing is require to do is define general scenario for traffic control so that first divide bandwidth management in two modules fixed and dynamic bandwidth allocation on four interfaces like eth0, eth1, eth2 and eth3.

The classful qdisc Hierarchical token bucket concept is used. As explained earlier htb uses classes with subclasses so we can define different class for different users with different rate. In this we can assign different access rate and limitation with user level. As for example if any organization consist three type of user one is CEO level, second is Manager level and third is Employee level . In this each level of user has different requirement and priority. In this type of situation we can use htb qdisc for better control over the available bandwidth.

One script was written to implement traffic control for above situation.For this one algorithm is defined and steps of algorithm are shown blow.

Step 1: Initially define access rate in Kilo bits per second (kbps), Kilo bytes per second (KB), mega bits per second (mbps), or Mega byte (MB)

Step 2: Choose interface on which we want to implement traffic control. Example eth0, eth1 etc.

Step 3: Define fixed or dynamic bandwidth allocation

Step 4: Define available bandwidth and total no. of users.

Step 5: If we want to implement fixed bandwidth allocation then bandwidth is equally divided in to all users than goto step7. If dynamic then network administrator can assign access rate according to requirement and goto step 6.

Step 6: Assign upload and download limit.

Step 7: Assign ceil rate and enter the IP addresses of user.

step 8: Next if you want to implement bandwidth management on another inter-

face than repeat step 2 or goto step9.

Step 9: Exit.

This script shows the stopping criteria, So anyone can stop access rate limitation on particular interface like eth0 than it behaves normally. And also mention some command for result which shows that what access rate is available in particular class. Here it uses the HTB qdisc so that shapping and borrowing can done as explained earier. This is done using assigning 2 kbps download limit and 1kbps upload limit on interface eth0 and try to download some files from internet it take lot time. It was checked using command "tc -s -d class show dev eth0", it show rate less than 2kbps means it work.

Chapter 5

SSL VPN

5.1 Introduction

Secure Sockets Layer (SSL) virtual private networks (VPN) provide secure remote access to an organization's resources. A VPN is a virtual network, built on top of existing physical networks, that can provide a secure communications mechanism for data and other information transmitted between two endpoints. Because a VPN can be used over existing networks such as the Internet, it can facilitate the secure transfer of sensitive data across public networks. An SSL VPN consists of one or more VPN devices to which users connect using their Web browsers.

The traffic between the Web browser and the SSL VPN device is encrypted with the SSL protocol or its successor, the Transport Layer Security (TLS)protocol. This type of VPN may be referred to as either an SSL VPN or a TLS VPN. SSL VPNs provide remote users with access to Web applications and client/server applications, and connectivity to internal networks. Despite the popularity of SSL VPNs, they are not intended to replace Internet Protocol Security (IPsec) VPNs. The two VPN technologies are complementary and address separate network architectures and business needs. SSL VPNs offer versatility and ease of use because they use the SSL protocol, which is included with all standard Web browsers, so the client usually does not

require configuration by the user. SSL VPNs offer granular control for a range of users on a variety of computers, accessing resources from many locations.There are two primary types of SSL VPNs:

SSL Portal VPN : This type of SSL VPN allows a user to use a single standard SSL connection to a Web site to securely access multiple network services. The site accessed is typically called a portal because it is a single page that leads to many other resources. The remote user accesses the SSL VPN gateway using any modern Web browser, identifies himself or herself to the gateway using an authentication method supported by the gateway, and is then presented with a Web page that acts as the portal to the other services.

SSL Tunnel VPN : This type of SSL VPN allows a user to use a typical Web browser to securely access multiple network services, including applications and protocols that are not web-based, through a tunnel that is running under SSL. SSL tunnel VPNs require that the Web browser be able to handle active content, which allows them to provide functionality that is not accessible to SSL portal VPNs.

SSL VPN products vary in functionality, including protocol and application support. They also vary in breadth, depth, and completeness of features and security services. Some recommendations and considerations include the following:

- SSL VPN manageability features such as status reporting, logging, and auditing should provide adequate capabilities for the organization to effectively operate and manage the SSL VPN and to extract detailed usage information.

- The SSL VPN high availability and scalability features should support the organization's requirements for failover, load balancing and throughput. State and information sharing is recommended to keep the failover process transparent to the user.

- SSL VPN portal customization should allow the organization to control the look and feel of the portal and to customize the portal to support various devices such as personal digital assistants (PDA) and smart phones.

- SSL VPN authentication should provide the necessary support for the organization's current and future authentication methods and leverage existing authentication databases. SSL VPN authentication should also be tested to ensure interoperability with existing authentication methods.

- The strongest possible cryptographic algorithms and key lengths that are considered secure for current practice should be used for encryption and integrity protection unless they are incompatible with interoperability, performance and export constraints.

5.2 Implementation of SSL VPN using Openvpn

5.2.1 Openvpn

OpenVPN is a full-featured SSL VPN which implements OSI layer 2 or 3 secure network extension using the industry standard SSL/TLS protocol, supports flexible client authentication methods based on certificates, smart cards, and/or username/password credentials, and allows user or group-specific access control policies using firewall rules applied to the VPN virtual interface. OpenVPN is not a web application proxy and does not operate through a web browser.

5.2.2 Installation and configuration of Openvpn

Steps for Installation of openvpn in Linux Debian

1. It is also possible to install OpenVPN on Linux using the universal ./configure method. First expand the .tar.gz file using following command:

tar xfz openvpn-[version].tar.gz

2. Than cd to the top-level directory type:

./configure

make

make install

3. Determine whether to routed or brigded VPN. Here routed VPN is implemented.

./configure

make

make install

4. Setting up a VPN often entails linking together private subnets from different locations.

The Internet Assigned Numbers Authority (IANA) has reserved the following three blocks of the IP address space for private internets (codified in RFC 1918):

While addresses from these netblocks should normally be used in VPN configura-

10.0.0.0	10.255.255.255	$(10/8 prefix)$
172.16.0.0	172.31.255.255	$(172.16/12 prefix)$
192.168.0.0	192.168.255.255	$(192.168/16 prefix)$

Table I: Private IP Address space

tions, it's important to select addresses that minimize the probability of IP address or subnet conflicts. The types of conflicts that need to be avoided are:

- conflicts from different sites on the VPN using the same LAN subnet numbering, or
- remote access connections from sites which are using private subnets which conflict with your VPN subnets.

For example, if the popular 192.168.0.0/24 subnet used as private LAN subnet. Now trying to connect to the VPN from an internet cafe which is using the same subnet

for its WiFi LAN. Then there will be a routing conflict because machine won't know if 192.168.0.1 refers to the local WiFi gateway or to the same address on the VPN.

As another example, If multiple sites are linked each other by VPN, but each site is using 192.168.0.0/24 as its LAN subnet. This won't work without adding a complexifying layer of NAT translation, because the VPN won't know how to route packets between multiple sites if those sites don't use a subnet which uniquely identifies them.

The best solution is to avoid using 10.0.0.0/24 or 192.168.0.0/24 as private LAN network addresses. Instead, use something that has a lower probability of being used in a WiFi cafe, airport, or hotel where anybody might expect to connect from remotely. The best candidates are subnets in the middle of the vast 10.0.0.0/8 netblock (for example 10.66.77.0/24).

5. Setting up your own Certificate Authority (CA) and generating certificates and keys for an Openvpn server and multiple clients. The first step in building an OpenVPN 2.0 configuration is to establish a PKI (public key infrastructure). The PKI consists of:

- a separate certificate (also known as a public key) and private key for the server and each client, and

- a master Certificate Authority (CA) certificate and key which is used to sign each of the server and client certificates.

OpenVPN supports bidirectional authentication based on certificates, meaning that the client must authenticate the server certificate and the server must authenticate the client certificate before mutual trust is established.Both server and client will authenticate the other by first verifying that the presented certificate was signed by the master certificate authority (CA), and then by testing information in the now-authenticated certificate header, such as the certificate common name or certificate type (client or server).This security model has a number of desirable features from the VPN perspective:

- The server only needs its own certificate/key – it doesn't need to know the individual certificates of every client which might possibly connect to it.

- The server will only accept clients whose certificates were signed by the master CA certificate (which we will generate below). And because the server can perform this signature verification without needing access to the CA private key itself, it is possible for the CA key (the most sensitive key in the entire PKI) to reside on a completely different machine, even one without a network connection.

- If a private key is compromised, it can be disabled by adding its certificate to a CRL (certificate revocation list). The CRL allows compromised certificates to be selectively rejected without requiring that the entire PKI be rebuilt.

6. Generate the master Certificate Authority (CA) certificate and key

In this section a master CA certificate/key, a server certificate/key, and certificates/keys for 3 separate clients are generated. Following commands are used to generate master certificate and keys:

[root @debian] cd /usr/share/doc/packages/openvpn Next, initialize the PKI. On Linux/BSD/Unix: **. ./vars ./clean-all ./build-ca**

7. Generate certificate & key for server Server key can be generated using the following command ./build-key-server server As in the previous step, most parameters can be defaulted. When the Common Name is queried, enter "server". Two other queries require positive responses, "Sign the certificate? [y/n]" and "1 out of 1 certificate requests certified, commit? [y/n]".

8. Generate certificate & key for server Generating client certificates is very similar to the previous step.

./build-key client1

./build-key client2

./build-key client3

Remember that for each client, make sure to type the appropriate Common Name when prompted, i.e. "client1", "client2", or "client3". Always use a unique common name for each client.

9. Generate Diffie-hallman parameter Diffie Hellman parameters must be generated for the OpenVPN server. On Linux/BSD/Unix:

./build-dh

10. Key Files Following key files are used at sever side and client side(as shown in Table II):

Filename	**Needed By**	**Purpose**	**Secret**
ca.crt	server + all clients	Root CA certificate	NO
ca.key	key signing machine only	Root CA key	YES
dhn.pem	server only	Diffie Hellman parameters	NO
server.crt	server only	Server Certificate	NO
server.key	server only	Server Key	YES
client1.crt	client1 only	Client1 Certificate	NO
client1.key	client1 only	Client1 Key	YES
client2.crt	client2 only	Client2 Certificate	NO
client2.key	client2 only	Client2 Key	YES
client3.crt	client3 only	Client3 Certificate	NO
client3.key	client3 only	Client3 Key	YES

Table II: Key Files used by clients and server

11. The final step in the key generation process is to copy all files to the machines which need them, taking care to copy secret files over a secure channel.

5.2.3 Creating configuration file for clients and server

After installing openvpn and generating key files for server and client, then create server config file and client config file.

Editing the server configuration file

The sample server configuration file is an ideal starting point for an OpenVPN server configuration. It will create a VPN using a virtual TUN network interface (for routing), will listen for client connections on UDP port 1194 (OpenVPN's official port number), and distribute virtual addresses to connecting clients from the 10.8.0.0/24 subnet.At this point, the server configuration file is usable; however it might require customizing it further:

- Enabling TUN/TAP interface.

  In openvpn two options are available for establishing tunnel between client and server. Here if bridge is established than it required to enable TAP driver in Linux or TUN driver for routing. This is enabled using following command: To check whether or not the TUN/TAP drivers are properly loaded:

  [root @debian]$ lsmod — grep tun Tun 12672 1

  If not than first enabling using

  [root@debian]$ modprobe tun

- Server is identified by writing mode as server in server config file

- **Tunnel port**

  Default source and destination tunneling port is UDP 1194. We should keep the default setting unless we need to change it for Firewall reasons otherwise we can keep it. Prefer UDP ports. The use of TCP can lead to degraded performances. So for this udp port was used.

  Port to udp

  Port 1194

- Next server certificate and key files are used as shown in Table II.

Editing the client configuration files

Like the server configuration file, first edit the ca, cert, and key parameters to point to the files you generated in the PKI section above. Note that each client should have its own cert/key pair. Only the ca file is universal across the OpenVPN server and all clients.

Next, edit the remote directive to point to the hostname/IP address and port number of the OpenVPN server (if your OpenVPN server will be running on a single-NIC machine behind a firewall/NAT-gateway, use the public IP address of the gateway, and a port number which you have configured the gateway to forward to the OpenVPN server).

Finally, ensure that the client configuration file is consistent with the directives used in the server configuration. The major thing to check for is that the dev (tun or tap) and proto (udp or tcp) directives are consistent. Also make sure that comp-lzo and fragment, if used, are present in both client and server config files. Starting up the VPN and testing for initial connectivity Starting the server. First, make sure the OpenVPN server will be accessible from the internet. That means:

- opening up UDP port 1194 on the firewall (or whatever TCP/UDP port you've configured), or

- setting up a port forward rule to forward UDP port 1194 from the firewall/gateway to the machine running the OpenVPN server.

Next, make sure that the TUN/TAP interface is not firewalled. To simplify troubleshooting, it's best to initially start the OpenVPN server from the command line (or right-click on the .ovpn file on Windows), rather than start it as a daemon or service: Starting the client.As in the server configuration, it's best to initially start the OpenVPN server from the command line (or on Windows, by right-clicking on the client.ovpn file), rather than start it as a daemon or service: openvpn [client config file]

A normal client startup on Windows will look similar to the server output above, and should end with the Initialization Sequence Completed message. Now, try a ping across the VPN from the client. If you are using routing (i.e. dev tun in the server config file), try:

ping 10.8.0.1

If bridging is implemented than(i.e. dev tap in the server config file), try to ping the IP address of a machine on the server's ethernet subnet. If the ping succeeds, congratulations! You now have a functioning VPN. Configuring OpenVPN to run automatically on system startup

The lack of standards in this area means that most OSes have a different way of configuring daemons/services for autostart on boot. The best way to have this functionality configured by default is to install OpenVPN as a package, such as via RPM on Linux or using the Windows installer. Linux

If OpenVPN is installed via an RPM package on Linux, the installer will set up an initscript. When executed, the initscript will scan for .conf configuration files in /etc/openvpn, and if found, will start up a separate OpenVPN daemon for each file.Including multiple machines on the server side when using a routed VPN (dev tun).

Once the VPN is operational in a point-to-point capacity between client and server, it may be desirable to expand the scope of the VPN so that clients can reach multiple machines on the server network, rather than only the server machine itself.

For the purpose of this example, we will assume that the server-side LAN uses a subnet of 10.66.0.0/24 and the VPN IP address pool uses 10.8.0.0/24 as cited in the server directive in the OpenVPN server configuration file. First, you must advertise the 10.66.0.0/24 subnet to VPN clients as being accessible through the VPN. This can easily be done with the following server-side config file directive:

push "route 10.66.0.0 255.255.255.0"

In a typical remote access scenario, the client machine connects to the VPN as a single machine. But suppose the client machine is a gateway for a local LAN (such

as a home office), and each machine on the client LAN to be able to route through the VPN.

For this example, first assume that the client LAN is using the 192.168.4.0/24 subnet, and that the VPN client is using a certificate with a common name of client2. The goal is to set up the VPN so that any machine on the client LAN can communicate with any machine on the server LAN through the VPN.Before setup, there are some basic prerequisites which must be followed:

- The client LAN subnet (192.168.4.0/24 in our example) must not be exported to the VPN by the server or any other client sites which are using the same subnet. Every subnet which is joined to the VPN via routing must be unique.

- The client must have a unique Common Name in its certificate ("client2" in our example), and the duplicate-cn flag must not be used in the OpenVPN server configuration file.

First, make sure that IP and TUN/TAP forwarding is enabled on the client machine. Next, would be deal with the necessary configuration changes on the server side. If the server configuration file does not currently reference a client configuration directory, add one now:

client-config-dir ccd

In the above directive, ccd should be the name of a directory which has been precreated in the default directory where the OpenVPN server daemon runs. On Linux this tends to be /etc/openvpn and on Windows it is usually in Program Files. When a new client connects to the OpenVPN server, the daemon will check this directory for a file which matches the common name of the connecting client. If a matching file is found, it will be read and processed for additional configuration file directives to be applied to the named client. The next step is to create a file called client2 in the ccd directory. This file should contain the line:

iroute 192.168.4.0 255.255.255.0

This will tell the OpenVPN server that the 192.168.4.0/24 subnet should be routed to client2. Next, add the following line to the main server config file (not the ccd/client2 file):

route 192.168.4.0 255.255.255.0

Route and iroute statements required because that route controls the routing from the kernel to the OpenVPN server (via the TUN interface) while iroute controls the routing from the OpenVPN server to the remote clients. Both are necessary. Next, if anybody would like to allow network traffic between client2's subnet (192.168.4.0/24) and other clients of the OpenVPN server. If so, add the following to the server config file.

push "route 192.168.4.0 255.255.255.0"

This will cause the OpenVPN server to advertise client2's subnet to other connecting clients.The last step, and one that is often forgotten, is to add a route to the server's LAN gateway which directs 192.168.4.0/24 to the OpenVPN server box (this won't be need if the OpenVPN server box is the gateway for the server LAN). Suppose If this step was missed and tried to ping a machine (not the OpenVPN server itself) on the server LAN from 192.168.4.8. The outgoing ping would probably reach the machine, but then it wouldn't know how to route the ping reply, because it would have no idea how to reach 192.168.4.0/24. The rule of thumb to use is that when routing entire LANs through the VPN (when the VPN server is not the same machine as the LAN gateway), make sure that the gateway for the LAN routes all VPN subnets to the VPN server machine.

Similarly, if the client machine running OpenVPN is not also the gateway for the client LAN, then the gateway for the client LAN must have a route which directs all subnets which should be reachable through the VPN to the OpenVPN client machine. Routing all client traffic (including web-traffic) through the VPN Overview

By default, when an OpenVPN client is active, only network traffic to and from the OpenVPN server site will pass over the VPN. General web browsing, for example, will be accomplished with direct connections that bypass the VPN.

In certain cases this behavior might not be desirable – If a VPN client to tunnel all network traffic through the VPN, including general internet web browsing. While this type of VPN configuration will exact a performance penalty on the client, it gives the VPN administrator more control over security policies when a client is simultaneously connected to both the public internet and the VPN at the same time.

Implementation

Add the following directive to the server configuration file: If VPN setup is over a wireless network, where all clients and the server are on the same wireless subnet, add the local flag:

push "redirect-gateway def1" push "redirect-gateway local def1"

Pushing the redirect-gateway option to clients will cause all IP network traffic originating on client machines to pass through the OpenVPN server. The server will need to be configured to deal with this traffic somehow, such as by NATing it to the internet, or routing it through the server site's HTTP proxy. In Linux, a command such as this to NAT the VPN client traffic to the internet:

iptables -t nat -A POSTROUTING -s 10.8.0.0/24 -o eth0 -j MASQUERADE

This command assumes that the VPN subnet is 10.8.0.0/24 (taken from the server directive in the OpenVPN server configuration) and that the local ethernet interface is eth0. When redirect-gateway is used, OpenVPN clients will route DNS queries through the VPN, and the VPN server will need handle them. This can be accomplished by pushing a DNS server address to connecting clients which will replace their normal DNS server settings during the time that the VPN is active. For example:

push "dhcp-option DNS 10.8.0.1"

Will configure Windows clients (or non-Windows clients with some extra server-side scripting) to use 10.8.0.1 as their DNS server. Any address which is reachable from clients may be used as the DNS server address.

5.3 Summary

The installation of Openvpn package requires lot of attention and analysis work. The main thing in establishing SSL VPN between client and server is to RSA key management and common certificate generation. Also generating server's private key and each client's private key. The server and client configuration files are created with the help of sample config file. Modify sample config file according to described in section 5.2. The client/server tunnel is tested using the ping command which shows that both are reply each other with standard format of ping reply.

Chapter 6

Anti spam email relay server

6.1 Introduction

A lot of users see spam as annoying e-mails they can simply delete. They do not realize their real monetary impact. Actually spam is costly for both users and the ISP [6]. The spam cost to the ISP is more dramatic and can be seen at two levels: an increase on the load of e-mail servers and the waste of bandwidth. In addition, the average number of spam messages received is increasing exponentially. Figure 6.1 shows recent statistics on the number of spam messages received by one e-mail user, and taken from [5]. Fighting spam is necessary. The lack of an efficient solution may threaten the usability of email as a communication means.

Spam filtering can be applied at the client level or the server level. Several options are available at the client level for spam filtering. However, such lists are used by service providers and network administrators to block an email before it is sent; the unintended consequence of maintaining these blacklists is that sometimes, innocent senders are inadvertently blocked from sending legitimate emails. Spam filters are also effective against mass mailings of spam mail.

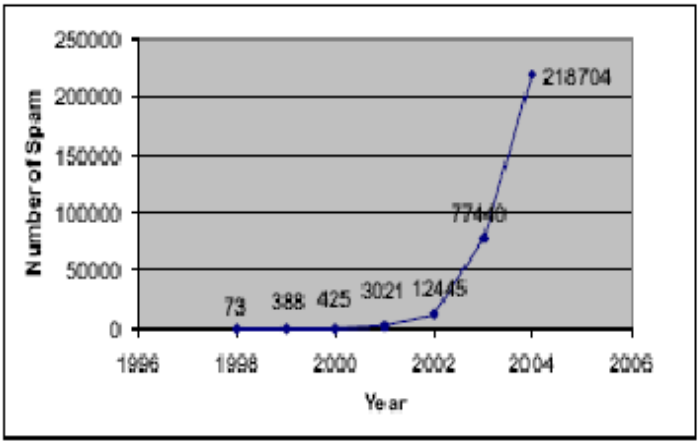

Figure 6.1: Annual Spam Evolutions[5].

6.2 Proposed Spam Filter

The proposed anti spam filtration scheme, is divided into three parts. Like every other filtration mechanism, it also has white list dictionary which is maintained by administrator. Black list is not necessary for system but user can edit both lists. Secondly, system has capability to calculate spam score using spamassassin. Last filtration chamber is consisted of Bayesian filter, based on the probabilistic approach. The Pseudo code of Filtration is given in algorithm 1.

When an email arrives at SMTP proxy server, it checks the senders mailing address. If it is in white list, without performing the other filtration procedure, mail will be sent to mail box. Other wise, apply the filtering procedure, to find out detected email is HAM or SPAM.

Algorithm 1 Mail Message (M) Process
1. if mail address (MA) is in the White list then
2. Mark Message (M) as HAM detected
3. Save M to mail box
4. else
5. calculate spam score(SS)
6. If SS less than Threshold value
7. then mark message as HAM
8. else
9. Send message to Naive Bayesian filter
10. if mark message(M) as SPAM
11. then change the subject as SPAM MAIL
12. else
13. send to mail box
14. endif
15. endif
16. endif

6.3 Implementation

The email MTA is postfix which has a good security record and is fairly easy to setup right. Postfix will listen normally on port 25 for incoming mail. Upon reception it will forward it to Amavisd-new on port 10024. Amavisd-new will then filter the mail through different filters before passing the mail back to Postfix on port 10025 which in turn will forward the mail to the next mail server.

Amavisd-new is a content filtering framework utilizing helper applications for virus filtering and spam filtering. In this setup two helper applications sre used one ClamAV for filtering virus mails and Spamassassin for filtering spam. Spamassassin itself can function as yet another layer of content filtering framework and utilize the helper applications Vipul's Razor2 and DCC.

Unlike many other spam fighting technologies like RBLs and others Spamassassin does not simply accept or reject a given email based on one single test. It uses a lot of internal tests and external helper applications to calculate a spam score for every

mail passed through. This score is based on the following tests:

- Bayesian filtering
- Static rules based on regular expressions
- Distributed and collaborative networks

To implenting spam filter at gateway level required to install and configure no. of packages such as Postfix, Perdition, Amavisd-new and Spamassassin etc. and all this packages are freely available and one can install directly in Linux environment using apt-get install command.

6.3.1 Postfix configuration

By default, Postfix configuration files are in /etc/postfix. The two most important files are main.cf and master.cf; these files must be owned by root. Giving someone else write permission to main.cf or master.cf (or to their parent directories) means giving root privileges to that person. In /etc/postfix/main.cf, it require to set up a minimal number of configuration parameters. Postfix configuration parameters resemble shell variables, with two important differences: the first one is that Postfix does not know about quotes like the UNIX shell does.

The myorigin parameter specifies the domain that appears in mail that is posted on this machine. The default is to use the local machine name, $myhostname, which defaults to the name of the machine. For the sake of consistency between sender and recipient addresses, myorigin also specifies the domain name that is appended to an unqualified recipient address. Examples (specify only one of the following):

/etc/postfix/main.cf:

**myorigin = $myhostname**default: send mail as "user@$myhostname")

myorigin = $mydomain (probably desirable: "user@$mydomain")

The mydestination parameter specifies what domains this machine will deliver locally, instead of forwarding to another machine. The default is to receive mail for

the machine itself.In this default setting is as below:

/etc/postfix/main.cf:

mydestination = $myhostname localhost.$mydomain localhost

mydestination = $myhostname localhost.$mydomain localhost $mydomain

mydestination = $myhostname localhost.$mydomain localhost www.$mydomain ftp.$mydomain

Postfix will forward mail from clients in authorized network blocks to any destination. Authorized networks are defined with the mynetworks configuration parameter. The default is to authorize all clients in the IP subnetworks that the local machine is attached to.If machine is connected to a wide area network then by default mynetworks setting may be easily done. specify only one of the following:

/etc/postfix/main.cf:

mynetworks_style = subnet (default: authorize subnetworks)

mynetworks_style = host (safe: authorize local machine only)

mynetworks = 127.0.0.0/8 (safe: authorize local machine only)

mynetworks = 127.0.0.0/8 168.100.189.2/32 (authorize local machine)

Postfix tries to deliver mail directly to the Internet. According to local conditions. For example, the system may be turned off outside office hours, it may be behind a firewall, or it may be connected via a provider who does not allow direct mail to the Internet. In those cases it needs to configure Postfix to deliver mail indirectly via a relay host. Specify only one of the following:

/etc/postfix/main.cf: **relayhost = (default: direct delivery to Internet)**

relayhost = $mydomain (deliver via local mailhub)

relayhost = [mail.$mydomain] (deliver via local mailhub)

relayhost = [mail.isp.tld] (deliver via provider mailhub)

The Postfix system reports problems to the postmaster alias using different classes of notification.The meaning of the classes is as follows:

bounce Inform the postmaster of undeliverable mail. Either send the postmaster a copy of undeliverable mail that is returned to the sender, or send a tran-

script of the SMTP session when Postfix rejected mail. For privacy reasons, the postmaster copy of undeliverable mail is truncated after the original message headers. This implies "2bounce" (see below). See also the user_relay feature. The notification is sent to the address specified with the bounce_notice_recipient configuration parameter (default: postmaster).

2bounce: When Postfix is unable to return undeliverable mail to the sender, send it to the postmaster instead (without truncating the message after the primary headers). The notification is sent to the address specified with the 2bounce_notice_recipient configuration parameter (default: postmaster).

delay: Inform the postmaster of delayed mail. In this case, the postmaster receives message headers only. The notification is sent to the address specified with the delay_notice_recipient configuration parameter (default: postmaster).

policy: Inform the postmaster of client requests that were rejected because of policy restrictions. The postmaster receives a transcript of the SMTP session. The notification is sent to the address specified with the error_notice_recipient configuration parameter (default: postmaster)

protocol: Inform the postmaster of protocol errors (client or server side) or attempts by a client to execute unimplemented commands. The postmaster receives a transcript of the SMTP session. The notification is sent to the address specified with the error_notice_recipient configuration parameter (default: postmaster).

resource: Inform the postmaster of mail not delivered due to resource problems (for example, queue file write errors). The notification is sent to the address specified with the error_notice_recipient configuration parameter (default: postmaster).

Proxy/NAT external network addresses

Some mail servers are connected to the Internet via a network address translator (NAT) or proxy. This means that systems on the Internet connect to the address of

the NAT or proxy, instead of connecting to the network address of the mail server. The NAT or proxy forwards the connection to the network address of the mail server, but Postfix does not know this.

If a Postfix server is run behind a proxy or NAT, then need to configure the proxy_interfaces parameter and specify all the external proxy or NAT addresses that Postfix receives mail on.

Postfix logging

Postfix daemon processes run in the background, and log problems and normal activity to the syslog daemon. The syslog process sorts events by class and severity, and appends them to logfiles. The logging classes, levels and logfile names are usually specified in /etc/syslog.conf. At the very least you need something like:

/etc/syslog.conf:

mail.err **/dev/console**

mail.debug **/var/log/maillog**

Code

First postfix is configured to listen on port 10025 and remove most of the restrictions as they have already been applied by the postfix instance listening on port 25. Also we ensure that it will only listen for local connections on port 10025. This is done by adding d the following code at the end of /etc/postfix/master.cf As shown in figure 6.2.

The file master.cf tells the postfix master program how to run each individual postfix process. More info with man 8 master. Next it needs the main postfix instance listening on port 25 to filter the mail through amavisd-new listening on port 10024. It also needs to set the next hop destination for mail. Tell Postfix to filter all mail through an external content filter and enable explicit routing to let Postfix know where to forward the mail to.

Code6.1 Changing the master.cf file

```
smtp-amavis unix - - n - 2 smtp
    -o smtp_data_done_timeout=1200
    -o smtp_send_xforward_command=yes
127.0.0.1:10025 inet n - n - - smtpd
     -o content_filter=
     -o local_recipient_maps=
     -o relay_recipient_maps=
     -o smtpd_restriction_classes=
     -o smtpd_client_restrictions=
     -o smtpd_helo_restrictions=
     -o smtpd_sender_restrictions=
     -o smtpd_recipient_restrictions=permit_mynetworks,reject
     -o mynetworks=127.0.0.0/8
     -o strict_rfc821_envelopes=yes
     -o smtpd_error_sleep_time=0
     -o smtpd_soft_error_limit=1001
     -o smtpd_hard_error_limit=1000
```

Figure 6.2: Modification in master.cf file.

Postfix has a lot of options set in main.cf as shown in figure6.3. For further explanation of the file consult manpage postconf or the same online Postfix Configuration Parameters.The format of the transport file is the normal Postfix hash file. Mail to the domain on the left hand side is forwarded to the destination on the right hand side. Code for transport file is shown in figure6.3.

6.3.2 Configuring Amavisd-new

Amavisd-new is used to handle all the filtering and allows you to easily glue together severel different technologies. Upon reception of a mail message it will extract the mail, filter it through some custom filters. It handles white and black listing, filter the mail through various virus scanners and finally it will filter the mail using SpamAssassin.

Amavisd-new itself has a number of extra features.It identifies dangerous file attachments and has policies to handle them per-user, per-domain and system-wide

Code6.2 Changing the main.cf file

```
biff = no
empty_address_recipient = MAILER-DAEMON
queue_minfree = 120000000
transport_maps = hash:/etc/postfix/transport
relay_domains = $transport_maps
```

Code6.3 /etc/postfix/transport

```
mydomain.tld                    smtp:mail.mydomain.tld
```

Figure 6.3: update in main.cf and transport file.

policies. Edit the following lines in /etc/amavisd.conf as in figure6.4.

6.3.3 Configuring Spamassassin

Amavis is using the Spamassassin Perl libraries directly so there is no need to start the service. Also this creates some confusion about the configuration as some Spamassassin settings are configured in /etc/mail/spamassassin/local.cf and overridden by options in /etc/amavisd.conf. Code for local.cf is shown in figure 6.5.

6.3.4 Postfix Anti-Spam settings

When a client (a computer trying to send us mail) connects to Postfix and begins a communication session, Postfix records information about that session. Prior to the point where Postfix accepts mail from that session for delivery, we have the option of evaluating the session and rejecting the mail by setting some restrictions in main.cf.

The configuration below is actually very conservative, allowing most email to come in the front door so amavisd-new and Spamassassin have their shot at it. Adding additional restrictions will increase the likelihood of rejecting valid email from improperly configured computers. Among other things, getting this stuff wrong could reject legitimate mail and/or cause us to become an open relay. Note that restrictions don't always restrict, some also permit. Postfix restriction stages are as follows, and are

Code6.5 Editing /etc/amavisd.conf

```
(Insert the domains to be scanned)
$mydomain = 'example.com';
(Bind only to loopback interface)
$inet_socket_bind = '127.0.0.1';
(Forward to Postfix on port 10025)
$forward_method = 'smtp:127.0.0.1:10025';
$notify_method = $forward_method;
(Define the account to send virus alert emails)
$virus_admin = "virusalert@$mydomain";
(Always add spam headers)
$sa_tag_level_deflt = -100;
(Add spam detected header aka X-Spam-Status: Yes)
$sa_tag2_level_deflt = 5;
(Trigger evasive action at this spam level)
$sa_kill_level_deflt = $sa_tag2_level_deflt;
Do not send delivery status notification to sender)
$sa_dsn_cutoff_level = 10;
Don't bounce messages left and right, quarantine instead
$final_virus_destiny = D_DISCARD; # (defaults to D_DISCARD)
$final_banned_destiny = D_DISCARD; # (defaults to D_BOUNCE)
$final_spam_destiny = D_DISCARD; # (defaults to D_BOUNCE)
```

Figure 6.4: update in main.cf and transport file.

processed in the following order:

- smtpd_client_restrictions
- smtpd_helo_restrictions
- smtpd_sender_restrictions
- smtpd_recipient_restrictions
- smtpd_data_restrictions

We are only going to place entries in the last three restriction stages. Restriction stages are processed in this order regardless of the order listed in main.cf

smtpd_sender_restrictions

This restriction stage restricts what sender addresses this system accepts in MAIL FROM: commands (the envelope sender). We will place three tests (restrictions) in

Code6.10 Create /etc/mail/spamassassin/local.cf

```
# Enable the Bayes system
use_bayes 1
# Enable all networK checks
skip_rbl_checks 0
# Mail using languages used in these country codes will not be marked
# as being possibly spam in a foreign language.
ok_languages da en no sv
# Mail using locales used in these country codes will not be marked
# as being possibly spam in a foreign language.
ok_locales en
# Use a sensible bayes path
bayes_path /var/amavis/.spamassassin/bayes
```

Figure 6.5: code for Spamassassin.

this restriction stage.

A restriction stage holds a list of restrictions (tests). Typically, tests evaluate to either DUNNO, REJECT, or OK. DUNNO means "I don't know what to do, let the next test decide". REJECT simply rejects the mail. OK means no more tests are performed in this restriction stage, tests continue with the next stage (if any). reject_* type tests typically evaluate to REJECT or DUNNO. permit_* type tests typically evaluate to OK or DUNNO, and check_*_access type tests can perform a variety of actions. The illustration shows the basic logic.

1) check_sender_access Postfix compare the envelope sender to entries in an /etc /postfix /sender _access database and act upon those entries if a match is found. It also define what action is taken there (OK, DUNNO, REJECT etc.) on a sender by sender basis. If the sender is not listed in the file, the test evaluates to DUNNO, and the next test is performed.

2) reject_non_fqdn_sender Reject when the envelope sender mail address is not in the proper format. Remember, the "envelope sender" is what the sending mail server gives in the "MAIL FROM:" line during the SMTP session, not the

header "From:" line. "Joe" is not allowed to send us mail (because we can't reply to "Joe") but "Joe@example.com" is at the very least an email address. If the sender does not get rejected at this point, this test evaluates to "DUNNO".

3) reject_unknown_sender_domain Reject when the envelope sender's domain part of the mail address has no DNS "A" or "MX" record at all. This setting kicks about 35% of the mail coming in my mail server. It is common for spammers to use a bogus domain name so they don't have to deal with the backlash of rejected mail. It is also important for us not to fill up our queue with bounce notices that can never be delivered due to the fact that the sender's domain does not even exist. If the sender's domain has an "A" or "MX" record, this test will also evaluate to "DUNNO".

smtpd_recipient_restrictions

The access restrictions that the Postfix SMTP server applies in the context of the RCPT TO: command. This refers to the "envelope recipient" which is what the client gave in the "RCPT TO:" line during the SMTP session, not the header "To:" line.

Restrictions that would normally go in these prior restriction stages can alternately be placed in smtpd _recipient _restrictions. Therefore, some people prefer to place all the smtpd _* _restrictions that would normally go in prior restriction stages into smtpd _recipient _restrictions (in the proper order) and leave the prior stages unconfigured (empty). In our case it is safer to use smtpd _sender _restrictions and smtpd_recipient _restrictions. Those specific restrictions(tests) are placed in smtpd _recipient _restrictions:

1)permit_mynetworks Allows machines listed in "mynetworks" to skip the rest of the tests in this restriction stage (permit = OK). In other words, it exits this stage and is tested in the next stage (smtpd_data _ restrictions). Because permit_mynetworks is placed in front of reject_ unauth_destination, this means machines in $mynetworks are allowed to relay mail to any domain. Without

this, we would only be able to send mail to our own domain(s). If the IP address of the sender is not listed in $mynetworks, the test evaluates to "DUNNO" and continues on to the next test (reject_unauth_destination).

2) reject_unauth_destination This, along with permit_mynetworks is used for relay control. This setting, in essence, means that mail bound for any domain that we have not configured our machine to accept mail for will be rejected. If the domain is listed in relay_domains, this test evaluates to "DUNNO" and the session is allowed to go on to the next test (if any). Just like "mynetworks", this setting is extremely critical.

By placing permit_ mynetworks directly ahead of reject _unauth _ destination, we are assured that we can send mail to domains other than ours, but we will only accept mail addressed to us from computers outside our network, thus permit_mynetworks and reject_unauth_destination work as a team.

smtpd_data_restrictions

Optional access restrictions that the Postfix SMTP server applies in the context of the SMTP DATA: command. Like smtpd_recipient _restrictions, this is a restriction stage.

6.3.5 Postfix content filtering control files

Postfix has /etc/postfix/header_checks and /etc/postfix/body_checks files. These files will list certain "strings" of text, and tell Postfix what to do with mail if it encounters these strings in email headers or the body of the message.

header_checks: Optional (only use if you intend on using header_checks): postconf -e "header _checks = pcre:/etc/ postfix/header _checks"

body_checks: Optional (only use if you intend on using body _checks): postconf -e "body _checks = pcre:/etc /postfix/body _checks"

content_filte: Here's where we tell Postfix to use amavisd-new. postconf -e "content _filter = smtp -amavis: [127.0.0.1]:10024"

6.3.6 Autolearning and sidelining emails

If the spam score is very low than there is the possibility of some false positives. These are filtered into the folder likely-spam. These are manually reviewed and any false positive is moved to the redeliver mailfolder. Now here we required to write redeliver.pl script and copy to the /usr/local/bin/.

Code Listing 6.14: redeliver.pl #!/usr/bin/perl -w

```
# Redelivers mail using a modified version of smtpclient
# By: Jens Hilligsoe ¡gentoo@hilli.dk¿
use strict;
if(!($#ARGV == 0)) {
        die "Usage:\n$0 maildir_mail\n";
}
my $mail = $ARGV[0];
my $to = "";
my $from = "";
sub prunefile ( $ );
# Retrieve To and From envelope adresses open (MAIL, $mail) or die
"Could not open $mail: $?\n"; while(< MAIL >) {
   if(($to eq "") —— ($from eq "")) {
      chop;
      (my $key, my $value) = split (/:/);
       if($key eq "X-Envelope-To") {
          $to = $value;
                              after
       }
```

```
        if($key eq "X-Envelope-From") {
            $from = $value;
                                    if($from eq "") {
                $from = "postmaster";
            }
        }
    }
    if($to eq "") {
        prunefile($ARGV[0]); # Just nuke it if to is empty
    } else {
        my $redelivercmd = "cat $ARGV[0] — smtpclient -F -S 127.0.0.1 -P 10025
    -f $from        $to";
        unless (system($redelivercmd) == 0 ) {
        die "Unable to redeliver: $?";
        }
        prunefile($ARGV[0]); # Clean up
    }
    sub prunefile ( $ ) {
        my ($file) = @_;
        unless (unlink $file) {
            die "Unable to remove mail: $?";
        }
    }
```

6.4 Result

Antispam filter is implemented at gateway level due to this it required to setup to two proxy server for receiving and sending mail. Here postfix is used as SMTP proxy server and perdition as POP/IMAP proxy server. Spamassassin is used to calculate spam score and based on this HAM or SPAM is decided. Amavisd-new is used for content filter such as body checks, header checks etc.

No MTA scores well by all measures. The needs of users vary greatly and some criteria are mutually orthogonal. Commonly cited MTA selection criteria are:Ease of administration, Security, Performance and Long-term viability.

Postfix is better than Sendmail for implementing spam filter. Here following table shows the comparison between different Mail Transfer Agents(MTAs). As shown in below figure6.6, Postfix is better than other MTAs according to the overall performance criteria.

if you are...	qmail	Exim	Sendmail	Postfix	Notes
Inexperienced	0	3	1	3	Exim and Postfix have good docs and clear examples
Worried about security	3	2	0	3	Postfix is secure and modern; qmail is secure but very old and cranky; Exim is secure to different criteria (read above.)
Relying on Sendmail milters	0	1	3	2	Postfix can run milters; can use equivalent Exim routers/filter script
Wanting minimum hassle	0	3	0	3	Sendmail has some easy front-ends, but the deeper you go the worse it gets. Postfix and Exim are more predictable.
Resource-constrained	3	2	1	2	See *Embedded Application* below for other comments
On Windows	0	2	3	0	Sendmail has a native Windows port; Exim is in the Cygwin distro
Needing commercial support	1	3	3	3	There are competent companies for all MTAs; qmail is inherently less supportable being so old

Figure 6.6: Comparison between different MTAs.

In this implement spamassassin is used for calculating spam score. It is important to set particular threshold value. For this a series of experiments are carried out for spam detection If it is not properly define than increase in false positive and false negative. As graph shows comparative result at different threshold values. Here we could got good result at threshold value 5.0 as shown in figure 6.7.

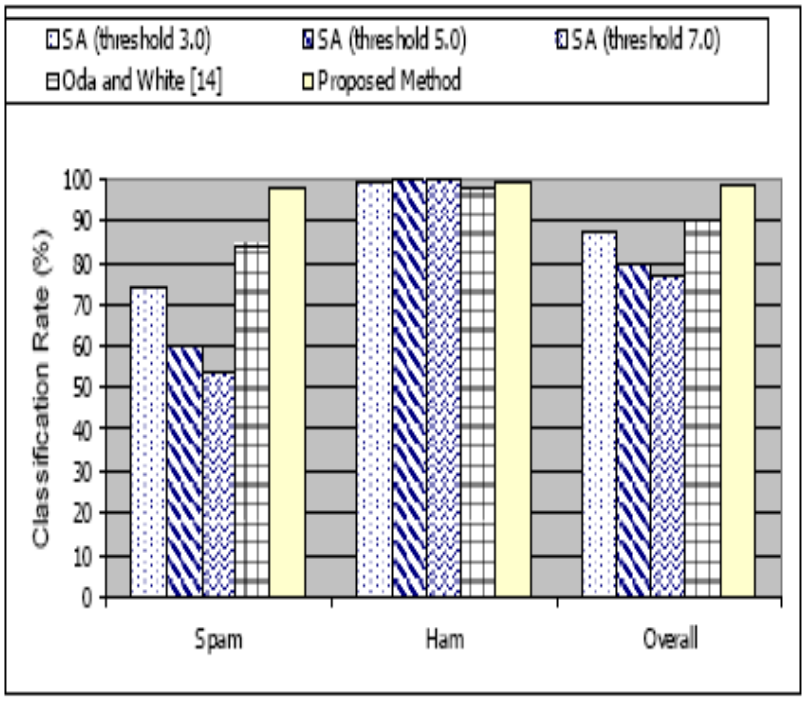

Figure 6.7: Performance of Spamassassin and proposed method

Chapter 7

Conclusion and Future Scope

7.1 Conclusion

In bandwidth management HTB classful qdisc is more preferable from implementation point of view, due to its classful qdisc so we can divide one class into further subclass so better control over traffic. Using openvpn Linux free distribution package we can establish more secure virtual private network, due to it uses the SSL/TLS protocol.

In the Antispam, spamassassin is used for calculating spam score. It is important to set particular threshold value. If it is not properly define than increase in false positive or false negative. Antispam gives good result at threshold value 5.0.

7.2 Future Scope

To improve the performance of the application several approaches can be made. First, a thorough examination of the characteristics of the organization because organizations are vary in size. Second, attempts should be made in concert with Network Administrators to determine how fairy distribute the available bandwidth between users and how to apply different policies to control spam mail. Finally, Some new modifications can make this system more versatile. In the future, we evaluate the system, analyze the system in a formal way and investigate the offline change of indi-

vidual addresses. Future improvement may be the integration of more modules into the system to look more facets of the email. The images, sender and receiver of email and subject part may help us to further classify a mail as a spam or ham. Most of the spam emails use the URL of some website. If we can make such a powerful system which can follow the URL and parse the text inside the website, this can also lead us to identify the spam behavior.

References

[1] W. Diffie, M. E. Hellman *Exhaustive Cryptanalysis of the NBS Data Encryption Standard.* Computer, pp.[74-84], June 1977.

[2] Yang Kuihe, Chu Xin *Implementation of Improved VPN Based on SSL.* Hebei University of Science and Technology, Shijiazhuang 050054, China, 2000.

[3] B. Braden, D. Clark, S. Shenker *Integrated Services in the Internet Architecture: an Overview.* Request for Comments (RFC) 1633, IETF, June 1994.

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

[5] Mingjun Lan, Wanlei Zhou *Spam Filtering based on Preference Ranking.* School of Information Technology, Deakin University, Australia, 2005.

[6] Symantec Internet Security Threat Report Trends for January 06-June-06 Symantec white paper, Volume X, Sept 2006.

[7] Usman Tariq, ManPyo Hong, Wonil Kim Quick Fix, an expeditious approach to diminish SPAM. Technical report, Royal Institute of Technology, 2003.

[8] Banit Agrawal, Nitin Kumar, Mart Molle *Controlling Spam Emails at the Routers.* Department of Computer Science & Engineering University of California, Riverside, California, 92521, 2005.

[9] Minh Tran, Grenville Armitage *End-users' resource consumption of spani and a 3D anti-spani evaluation framework.* Centre for Advanced Internet Architectures Swinburne University of Technology Melbourne, Australia, 2006.

[10] J. Jung, E. Sit *An Empirical Study of Spam Traffic and the Use of DNS Black Lists.* Empirical Study of Spam Traffic and the Use of DNS Black Lists", Proc. of the 4th ACM SIGCOMM Conference on Internet Measurement, Sicily, Itaty, Oct 2004.

[11] G. Lindberg *Anti-Spam Recommendations for SMTP MTAs.* RFC 2505, Internet Engineering Task Force, Feb 1999

[12] Danilo Michalczuk Taveira, Otto Carlos Muniz Bandeira Duarte *A Monitor Tool for Anti-spam Mechanisms and Spammers Behavior.* P.O. Box 68504 - 21945-970, Rio de Janeiro, RJ, Brazil, 2008

[13] S. L. Pfleeger, G. Bloom *Canning spam: Proposed solutions to unwanted email.* IEEE Security & Privacy Magazine, vol. 3, no. 2, pp. 4047, Mar. 2005.