ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
FACULTY OF COMPUTER AND MATHEMATICAL SCIENCES
DEPARTMENT OF COMPUTER SCIENCE

Flow-Based E-mail Spam Detection

By: Zelalem Hailu

Advisor : Mulugeta Libsie (PhD)

A thesis submitted to
the School of Graduate Studies of Addis Ababa University
in partial fulfillment of the requirements for the Degree of
Masters of Science in Computer Science

November, 2011

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

FACULTY OF COMPUTER AND MATHEMATICAL SCIENCES

DEPARTMENT OF COMPUTER SCIENCE

Flow-Based E-mail Spam Detection

By: Zelalem Hailu

Name and Signature of members of the examining board

Chairman	Signature
Advisor	Signature
Examiner	Signature

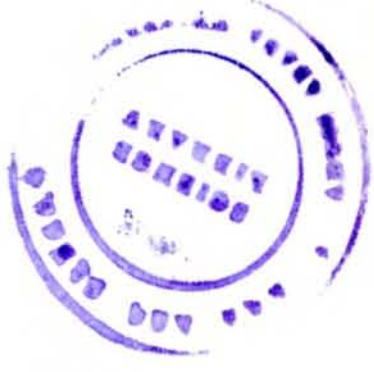
Table of Contents

Acknowledgment

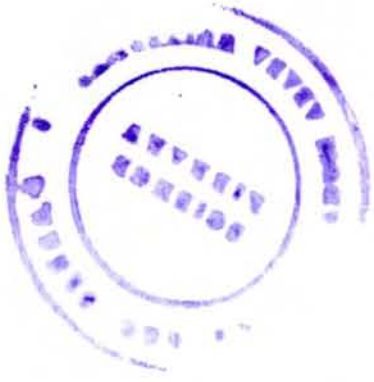
It is a pleasure to thank the people who made this thesis work possible. First and foremost, I would like to express my sincere gratitude to my advisor Dr. Mulugeta Libsie for his continuous and tireless follow up of this work. His thoughtful guidance and detailed inspection of errors, tirelessly, gave me a lifetime lesson of striving for perfection.

I am also appreciative of the computer science department. It is very kind of them to provide us stationery materials and printing service.

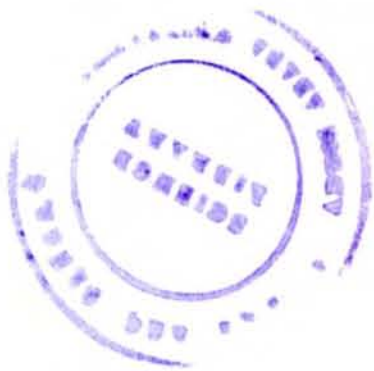
Finally, I am indebted to my fellow classmates who have been very close to me in good and bad times. Friends, it is your cheering that always comes to my mind when I think of you. I really enjoyed it. Thank you for being my friends.



List of Figures



List of Tables



Acronyms and Abbreviations

AAU: Addis Ababa University

CSC: Chi-Square Classifier

DNS: Domain Name System

FNR: False Negative Rate

FPR: False Positive Rate

IM: Instant Messaging

IMAP: Internet Message Access Protocol

MDA: Mail Delivery Agent

MTA: Mail Transport Agent

MUA: Mail User Agent

MX: Mail Exchange

NBC: Naïve Bayesian Classifier

OCR: Optical Character Recognition

PBC: Payload Based Classifier

POP: Post Office Protocol

ROC: Receiver Operating Characteristics

SMOTE: Synthetic Minority Over-sampling Technique

SMTP: Simple Mail Transfer Protocol

SVM: Support Vector Machine

TPR: True Positive Rate

Abstract

The volume of unsolicited commercial e-mails, also known as spam, is in such a rapid increase that almost over 90% of all e-mail messages are spam. We are in a state where an average of 200 billion e-mail spams are sent each day. This problem is exacerbated by the fact that many of these spams contain some sort of malicious code for attack. In addition to wasting of users' time and attack threats, the huge amount of spam also consumes bandwidth and storage spaces illegally. There have been efforts over the years to combat spam messages. The most popular ones are based on e-mail content analysis and IP address reputation. Techniques based on e-mail content analysis are falling behind because of spammers' ability to trick such filters using legitimate e-mail-like words in their contents. The introduction of image and PDF spams is also another headache for content based filters. Filters based on IP address reputation are also not coping well with the spammers because of the dynamic nature of IP addresses and the difficulty of hunting down malicious addresses before significant damages are done. Our approach is to filter out spam messages before they are delivered to the user's inbox based on packet flow characteristics. This is a complimentary approach that can be used with other techniques to reduce the number of spam messages reaching users' inbox. Our approach is based on over 55,000 packet flow records. We have identified nine features that best differentiate spam from legitimate e-mail. Based on these attributes and a classification model with an accuracy of 99.5% and a false-positive of 2.6%, we have developed a ranking algorithm that scores a given flow into one of five categories. Based on these scores, a given packet flow will be accepted, rejected or will be passed for further examination by other techniques. In addition to giving the advantage of not relying on e-mail content or IP address to filter spam, our method also avoids the wastage of resources like bandwidth and storage space by spam messages.

Keywords: Network flow, E-mail spam, Feature selection, Classification, Ranking algorithm

CHAPTER ONE: INTRODUCTION

1.1 Background

Communication is a vital part of human life. People use different communication techniques to convey information among each other; to carry out business correspondence or just to keep in touch with relatives. Electronic mail, also known as e-mail, is one of the most popular communication channels being widely used these days. E-mail is an electronic message transmitted over a network from one user to another [15]. Because of its low cost and fast communication, e-mail has been widely used for years.

Spam (also called junk e-mail) is an unwanted e-mail that has been sent to large number of recipients on the Internet for several purposes; including advertising, spreading malicious programs, annoying, and phishing. Spam is a problem that practically every e-mail user encounters. More than 85% of all e-mail messages are likely to be spam and this level is still rising [6]. The problem of spam begins with the definition of the term itself. It depends on the attitude, position, temper or even mood of the person receiving an e-mail, whether he/she classifies an incoming mail as spam.

The following are often referenced definitions of spam:

> *"Spamming is the abuse of electronic messaging systems to send unsolicited bulk messages, which are generally undesired"* [3].
>
> *"An electronic message is spam if [4]:*
>
> *(1) The recipient's personal identity and context are irrelevant because the message is equally applicable to many other potential recipients AND*
>
> *(2) The recipient has not verifiably granted deliberate, explicit and still-revocable permission for it to be sent".*

Spam is a problem that plagues every organization from the biggest of enterprises to the smallest of businesses and individuals. According to a study by Symantec, one out of every 617 spam messages contains a malicious code or malware [1]. From 2001 to 2008, spam volume grew from 8% of all e-mail volume to over 80% of all e-mails. Even many

enterprise customers are finding that over 95% of their e-mail traffic is spam. Dealing with spam can severely hamper employee productivity as well as time and resources from IT departments [1, 5].

Current approaches to identify and filter spam can be classified into two categories: those based on characterizing the properties of the sending SMTP server, and those based on analyzing the contents of the e-mail message. These approaches are also called pre-acceptance and post acceptance tests, respectively. These techniques have their own advantages and disadvantages. The details regarding these techniques are dealt in Chapter two.

1.2 Motivation

The spam filtering techniques mentioned in the last paragraph of the previous section could not prevent the explosive growth of spam. Usually spammers seem to go one step ahead of the defense mechanisms. Thus, researchers are still looking for novel techniques that are capable of preventing or effectively identifying spam. Currently, most spam detection is accomplished by filters at the receiving end of the e-mail life cycle. These spam filters have become complex and yet ineffective against some spammer evasion techniques like image spam and parasitic spam (embedding spam into legitimate messages). Another negative aspect of filtering at the receiving end is that it does not stop the billions of packets that traverse the Internet and consume resources. Therefore, anti-spam tools need not only filter spam at the users' inbox, but also before delivering the message to the user, at the network core, preventing the illegal consumption of storage space on mail servers.

Whereas the spammers have the flexibility to alter the content of e-mails as users update spam filters, they have far less flexibility when it comes to altering the network-level properties of the spam they send. Thus, effective and accurate pre-acceptance filtering can significantly reduce the load on SMTP servers and enhance their ability to identify and mitigate spam.

Spammers are even able to evade existing pre-acceptance tests like blacklisting using zombie (compromised computers by attackers) computers and botnets (network of zombies). As pointed out in [9], 85% of all spam is estimated to come from a few botnet

clusters. Therefore, it is very critical to detect this botnet machines based on their network-level behavior (rather than simple IP address blocking) to mitigate billions of spam e-mails that originate from the spamming machines.

In this thesis, we propose a spam detection approach that does not rely on e-mail content which is very easy for spammers to change and delude protection mechanisms. Network-level behavior of spammers will be analyzed to develop an algorithm that will help in the design of more robust pre-acceptance techniques.

Studies show that the network layer behavior of suspicious hosts differs substantially from that of legitimate mail servers, both in activity and incoming/outgoing traffic patterns [7, 8]. Recently a few researchers have proposed some approaches and techniques to detect malicious machines based on network-level behaviors [8, 10, 11, 12, 13]. This work differs from the past approaches in that we try to identify novel traffic features that are general enough to detect spammers based on traffic information only and we focus specifically on detecting e-mail spam, not on detecting botnets in general, unlike most others.

In this study, from the network level properties, those properties that are general enough to detect spamming machines and are more difficult to alter to the spammers will be explored and used to develop a spam detection algorithm.

1.3 Statement of the problem

In 2009, there were 1.4 billion e-mail users on the Internet sending 90 trillion e-mail messages. Out of these, 73 trillion were spam messages, averaging 200 billion e-mail spams per day. The spam amount has shown 24% increase from the previous year [2]. According to [17], US companies alone are losing over $70 billion a year in lost user productivity.

Even though there have been plenty of research works done on combating e-mail spam over the years, there still exists a huge gap in finding optimum solution to the problem. To this end, we have identified the following main problems among others.

- The existing techniques are easily evadable

- Most existing techniques have high false positive rates
- The existing techniques impose huge load on the mail servers
- The existing techniques do not prevent wastage of resources illegitimately
- The existing techniques are expensive to maintain

1.4 Objectives

1.4.1 General Objective

The general objective of this study is to develop a spam detection algorithm based on the network-level behavior of spammers.

1.4.2 Specific Objectives

This study strives to meet the following specific objectives:

- To understand how spammers behave on the flow level
- To observe how spamming differs from normal e-mail traffic on flow level
- To identify behaviors which are identifying features of spammers
- To come up with an algorithm that can detect spamming machines with low false positives

1.5 Scope and Limitation

This study is limited to observing the flow behaviors of spammers and devising an algorithm to mitigate future attacks. We focus only on the network traffic flow to observe flow behaviors of legitimate and spam e-mails. This thesis will not deal with analyzing e-mail contents or mere IP addresses to detect spammers and spam.

1.6 Methods

Data Set

The flow data that we will be using for the study will be collected using net flow capturing software tools. From the existing open sources tools, we will select the one that fits best for this purpose and network traffic will be collected for about 7 days.

Data Selection

From the collected data, we will filter out the ones that we are interested in. The whole data captured will be too large and the response time may also be infeasible. Therefore, data selection step will be used to filter out important data and the chosen data will be stored in a separate database.

Feature Extraction and classification

From the filtered data set, features from flows will be observed and those features that are typical of spam messages will be extracted. These are the properties that differentiate spam from a legitimate e-mail.

Algorithm Construction

Based on the properties observed, an initial algorithm will be developed that can be used to detect spam. This algorithm will be fine-tuned using validation techniques.

Validation and Testing

Validation can be done using already known legitimate traffic and spam traffic collected during the data collection period.

1.7 Application of the Results

The findings from this study could be used by small to large mail servers worldwide. Perhaps the results from this research could shift the burden from mail servers down to other network devices in detecting spam. This in turn means reduction in the number of misclassification by mail servers due to too much load in high speed real time networks. The effective classification and filtering of spam messages in the network core means no need of storage for spam in user inboxes, no need of resources and bandwidth wastage to transport spam folders from servers to users' mail agents, no wastage of users' time in reading and deleting spam and of course, no threat of attack via spam messages. These benefits are for the vast number of mail servers and other network resources globally and for the nearly one and half billion e-mail users.

1.8 Organization of the Thesis

The remaining part of this document is organized in six chapters. Literature review is dealt with in Chapter two. Works related with ours are presented in the third Chapter. Chapter four presents the proposed model to come up with the desired solution. The results from this thesis work are presented in Chapter five together with thorough discussion about the findings. The final Chapter concludes the whole work.

CHAPTER TWO: LITERATURE REVIEW

This Chapter reviews literatures on how e-mail systems work and the different spam categories. Better evaluation of anti-spam methods can be made by grouping them into classes. Each category acts on a different level during the flow of ham (legitimate e-mail) and spam e-mails. In order to describe these categories (that will be done in the next Chapter), a general view of how e-mail systems work is presented below. Describing the groups of anti-spam methods can be done much easier by considering the flow of e-mail messages from the sender to the receiver. Literatures regarding spam will be dealt with in the final part of this Chapter.

2.1 E-mail and its Protocols

2.1.1 How e-mail works

An e-mail can be simply a few lines of text sent from one user to another, or includes attachments such as pictures or documents. It can be a newsletter sent to subscribers daily detailing what is happening and when, or an encrypted message sent between government organizations containing classified information.

E-mail has come a long way since the advent of the Internet, with billions of e-mails being sent and received each day. It is perhaps the Internet's 'killer' application as more than 81% of people who use the Internet also use e-mail [2] and there are more number of e-mail accounts than Internet users [16]. E-mail has become the way to communicate efficiently, quickly and cheaply. E-mail is thought to have a major impact in advancing the spread of the Internet more than any other application or use of the Internet. When the Internet (or ARPANET as it was called) was first introduced it did not take long before 75% of all network traffic was e-mail related [15].

As the Internet has no distance issues, it is far simpler, cheaper and quicker to send an e-mail halfway round the world than it is to send a traditional letter. It is just as easy to send an e-mail to the person sitting next to you, as it is to send an e-mail to a hundred different people around the world.

From the user standpoint, e-mail seems so simple. You set the e-mail address of the person to whom you want to send the e-mail, compose your message and click 'Send'. In reality, it is a bit more complicated than that. Figure 2.1 illustrates the components that take part in delivering a given message from the source to the destination [15]. Each step is described in the subsequent paragraphs.

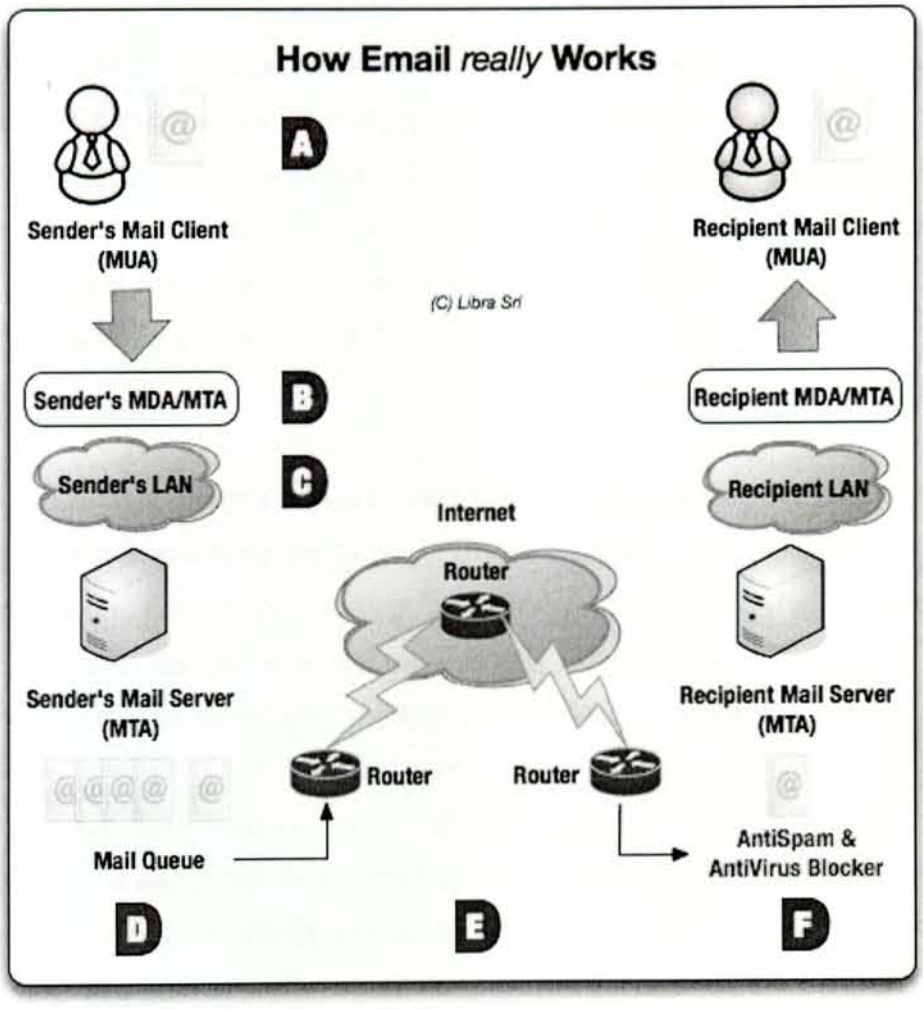


Figure 2.1: Components in e-mail messaging

Step A: Sender creates and sends an e-mail

The originating sender creates an e-mail in his/her Mail User Agent (MUA) and clicks 'send'. MUA is the application the originating sender uses to compose and read e-mail, such as Outlook, Entourage, etc.

Step B: Sender's MDA/MTA routes the e-mail

The sender's MUA transfers the e-mail to a Mail Delivery Agent (MDA). Frequently, the sender's MTA also handles the responsibilities of an MDA. Several of the most common MTAs do this, including sendmail and gmail. The MDA/MTA accepts the e-mail, then routes it to local mailboxes or forwards it if it isn't locally addressed. In Figure 2.1, an MDA forwards the e-mail to an MTA and it enters the first of a series of "network clouds," labeled as a "Sender's LAN" cloud.

Step C: Network Cloud

An e-mail can encounter a network cloud within a large company or ISP, or the largest network cloud in existence: the Internet. The network cloud may encompass a multitude of mail servers, DNS servers, routers, and other devices and services too numerous to mention. These are prone to be slow when processing an unusually heavy load, temporarily unable to receive an e-mail when taken down for maintenance, and sometimes may not have identified themselves properly to the Internet through the Domain Name System (DNS) so that other MTAs in the network cloud are unable to deliver mails as addressed. These devices may be protected by firewalls, spam filters and malware detection software that may bounce or even delete an e-mail. When an e-mail is deleted by this kind of software, it tends to fail silently, so the sender is given no information about where or when the delivery failure occurred.

E-mail service providers and other companies that process a large volume of e-mail often have their own private network clouds. These organizations commonly have multiple mail servers, and route all e-mail through a central gateway server (i.e., mail hub) that redistributes mail to whichever MTA is available. E-mail on these secondary MTAs must usually wait for the primary MTA (i.e., the designated host for that domain) to become available, at which time the secondary mail server will transfer its messages to the primary MTA.

Step D: E-mail Queue

The e-mail in Figure 2.1 is addressed to someone at another company, so it enters an e-mail queue with other outgoing e-mail messages. If there is a high volume of mail in the queue—either because there are many messages or the messages are unusually large, or both—the message will be delayed in the queue until the MTA processes the messages ahead of it.

Step E: MTA to MTA Transfer

When transferring an e-mail, the sending MTA handles all aspects of mail delivery until the message has been either accepted or rejected by the receiving MTA. As the e-mail clears the queue, it enters the Internet network cloud, where it is routed along a host-to-host chain of servers. Each MTA in the Internet network cloud needs to "stop and ask directions" from the DNS in order to identify the next MTA in the delivery chain. The exact route depends partly on server availability and mostly on which MTA can be found to accept e-mail for the domain specified in the address. Most e-mail takes a path that is dependent on server availability, so a pair of messages originating from the same host and addressed to the same receiving host could take different paths. These days, it is mostly spammers that specify any part of the path, deliberately routing their message through a series of relay servers in an attempt to obscure the true origin of the message.

To find the recipient's IP address and mailbox, the MTA must drill down through the DNS, which consists of a set of servers distributed across the Internet: beginning with the root name servers at the top-level domain, then domain name servers that handle requests for domains within that top-level domain, and eventually to name servers that know about the local domain.

DNS resolution and transfer process

- There are 13 root servers serving the top-level domains (e.g., .org, .com, .edu, .gov, .net, etc.). These root servers refer requests for a given domain to the root name servers that handle requests for that top-level domain. In practice, this step is seldom necessary.

- The MTA can bypass this step because it already knows which domain name servers handle requests for these top-level domains. It asks the appropriate DNS server which Mail Exchange (MX) servers have knowledge of the subdomain or local host in the e-mail address. The DNS server responds with an MX record: a prioritized list of MX servers for this domain.

  An MX server is really an MTA wearing a different hat, just like a person who holds two jobs with different job titles (or three, if the MTA also handles the responsibilities of an MDA). To the DNS server, the server that accepts messages is an MX server. When it is transferring messages, it is called an MTA.

- The MTA contacts the MX servers on the MX record in order of priority until it finds the designated host for that address domain.

- The sending MTA asks if the host accepts messages for the recipient's username at that domain (i.e., username@domain.top-level domain) and transfers the message.

Step F: Firewalls, Spam and Virus Filters

The transfer process described in the last step is somewhat simplified. An e-mail may be transferred to more than one MTA within a network cloud and is likely to be passed to at least one firewall before it reaches its destination. An e-mail encountering a firewall may be tested by spam and virus filters before it is allowed to pass inside the firewall.

These filters test to see if the message qualifies as spam or malware. If the message contains malware, the file is usually quarantined and the sender is notified. If the message is identified as spam, it will probably be deleted without notifying the sender.

Spam is difficult to detect because it can assume so many different forms, so spam filters test on a broad set of criteria and tend to misclassify a significant number of messages as spam, particularly messages from mailing lists. When an e-mail from a list or other automated source seems to have vanished somewhere in the network cloud, the cause is usually a spam filter at the receiver's ISP or company.

Finally, the e-mail makes it past the hazards of the spam trap or filter, and is accepted for delivery by the receiver's MTA as shown in Figure 2.1. The MTA calls a local MDA to deliver the mail to the correct mailbox, where it will be stored until it is retrieved by the recipient's MUA.

2.1.2 Common E-mail Protocols

SMTP (Simple Mail Transfer Protocol) [34] was developed during the 1980s as a standard for the transmission of messages. It is the language that most mail servers use to send messages between each other.

When the SMTP mail system sends a message it uses DNS to convert the domain part of an e-mail address (such as @aau.edu.et) to the IP network address of the machine that maintains the domain (such as 10.4.15.109). It then connects to that IP address on port 25 and uses very simple commands to communicate the sender's and recipient's e-mail addresses and the body of the message.

POP3 (Post Office Protocol 3) [35] is used by clients to collect e-mail from e-mail servers. Clients must supply a username and password to the server in order to log into their account or POP3 mailbox. The e-mail server will respond with the number of messages waiting and the client can initiate a 'deque' command to download the queued e-mails. The messages will either be deleted from the e-mail server or marked as read so they are not downloaded again.

POP3 is very good at the simple task of collecting e-mail and can be used as and when required. The POP3 transactions between client and the e-mail server are similar to SMTP.

IMAP (Internet Message Access Protocol) [36] is used for accessing e-mail stored on a server. It is a protocol that allows users to perform certain electronic mail functions on a remote server rather than on their local computer. The fundamental difference between accessing e-mail via IMAP as opposed to via POP3 is that IMAP does not download the messages and stores them locally as POP3 does. All message manipulations, such as opening, closing and deleting, are carried out on the server. This makes backup simpler

and security tighter since no e-mails are actually stored locally on the user's own computer.

2.2 E-mail Threats

There are plenty of attacks on e-mail systems by individuals and organizations that seek to cause some form of technical damage or hope to make money in an illegal fashion. The following are some of the major threats posed on e-mail systems [49].

- **Viruses, Worms and Trojan Horses:** Delivered as e-mail attachments, destructive code can devastate a host system's data, turn computers into remote control slaves known as botnets and cause recipients to lose serious money. Trojan horse keyloggers, for example, can secretly record system activities, giving unauthorized external parties access to corporate bank accounts, internal business web sites and other private resources.

- **Phishing:** According to the Anti-Phishing Working Group-a trade organization that consists of financial organizations, software publishers and other concerned parties - phishing attacks utilize social engineering to steal consumers' personal and financial data. The attacks rely on "spoofed" e-mails that direct recipients to bogus web sites that are designed to trick them into revealing confidential financial data such as credit card numbers, account usernames, passwords and social security numbers. Phishing committers typically operate by hiding under fake identities that they have stolen from banks, online merchants and credit-card companies.
- **Spam:** Junk e-mail can quickly overwhelm an inbox, making it difficult or even impossible for its owner to view legitimate messages. The spam problem has gotten so bad that it is commonplace for users to drop e-mail accounts that are overrun with spam rather than try to fight the problem. Spam is also the delivery medium of choice for both phishers and virus attackers. The problem has reached to a level that hundreds of billions of spam messages are sent every day.

2.3 Spam

Spam, as defined in Chapter one, is the use of electronic messaging systems to send unsolicited bulk messages indiscriminately. While the most widely recognized form of spam is e-mail spam, the term is applied to similar abuses in other media: instant messaging spam, Usenet newsgroup spam, Web search engine spam, spam in blogs, wiki spam, online classified ads spam, mobile phone messaging spam, Internet forum spam, junk fax transmissions, social networking spam, television advertising and file sharing network spam.

Spamming remains economically viable because advertisers have no operating costs beyond the management of their mailing lists, and it is difficult to hold senders accountable for their mass mailings. Because the barrier to entry is so low, spammers are numerous, and the volume of unsolicited mail has become very high.

2.3.1 Types of Spam

- **E-mail Spam**

  Spam in e-mail started to become a problem when the Internet was opened up to the general public in the mid-1990s. It grew exponentially over the following years, and today composes more than 85% of all the e-mail in the world, by a "conservative estimate" [6]. Pressure to make e-mail spam illegal has been successful in some jurisdictions, but less so in others. Spammers take advantage of this fact, and frequently outsource parts of their operations to countries where spamming will not get them into legal trouble.

  Increasingly, e-mail spam today is sent via zombie networks, networks of compromised personal computers in homes and offices around the globe. According to Daily Mail [32], Rustock botnet, a collection of infected machines independently sending messages, had sent 200 billion junk e-mails a day, half the global total, in 2010 before being shut down by U.S. authorities in March 2011.

  Many modern worms install a backdoor (a means of access to a computer program that bypasses security mechanisms) which allows the spammer access to the computer and use it for malicious purposes. This complicates attempts to

control the spread of spam, because in many cases the spam doesn't even originate from the spammer.

- **Instant Messaging Spam**

  Instant Messaging spam makes use of instant messaging systems. Although less ubiquitous than its e-mail counterpart, according to a report from Ferris Research, 500 million spam instant messages were sent in 2003, twice the level of 2002 [29]. As instant messaging tends not to be blocked by firewalls, it is an especially useful channel for spammers.

- **Newsgroup and Forum Spam**

  Newsgroup spam is a type of spam where the targets are Usenet newsgroups. Spamming of Usenet newsgroups actually pre-dates e-mail spam. Usenet convention defines spamming as excessive multiple posting, that is, the repeated posting of a message. Similar to newsgroup spam is forum spam. Forum spam is the creating of messages that are advertisements or otherwise unwanted on Internet forums. It is generally done by automated spambots. Most forum spam consists of links to external sites, with the dual goals of increasing search engine visibility in highly competitive areas such as weight loss, pharmaceuticals, gambling, pornography, real estate or loans, and generating more traffic for these commercial websites. Some of these links contain code to track the spambot's identity if a sale goes through, when the spammer behind the spambot works on commission.

- **Mobile Phone Spam**

  Mobile phone spam is directed at the text messaging service of a mobile phone. This can be especially irritating to customers not only for the inconvenience but also because of the fee they may be charged per text message received in some markets.

- **Online Game Messaging Spam**

  Many online games allow players to contact each other via player-to-player messaging, chat rooms, or public discussion areas. What qualifies as spam varies

from game to game, but usually this term applies to all forms of message flooding, violating the terms of service contract for the website.

- **Spam targeting search engines (spamdexing)**

Spamdexing (from *spamming* and *indexing*) refers to a practice on the World Wide Web of modifying HTML pages to increase the chances of them being placed high on search engine relevancy lists. These sites use black hat search engine optimization techniques to unfairly increase their rank in search engines. Recently The New York Times has reported an incident of a company named J.C.Penny that had managed to turn up in the number one spot of Google search results for months using some sort of spamdexing [33].

Many modern search engines modified their search algorithms to try to exclude web pages utilizing spamdexing tactics. For example, the search bots (robotic agents that continuously search the Internet to find all the best websites on the searcher's behalf) will detect repeated keywords as spamming by using a grammar analysis. If a website owner is found to have spammed the webpage to falsely increase its page rank, the website may be penalized by search engines.

- **Blog and Wiki Spam**

Blog spam, or "blam" for short, is spamming on weblogs (a web site that consists of a series of entries arranged in reverse chronological order). In 2003, this type of spam took advantage of the open nature of comments in the blogging software Movable Type by repeatedly placing comments to various blog posts that provided nothing more than a link to the spammer's commercial web site [18]. Similar attacks are often performed against wikis (a website that allows the creation and editing of any number of interlinked web pages via a web browser) and guestbooks (an electronic means for a visitor to acknowledge their visitation to a site, and leave their name, address, and a comment or note, if desired), both of which accept user contributions

- **Spam targeting video sharing sites**

Video sharing sites, such as YouTube, are now being frequently targeted by spammers. The most common technique involves people or spambots posting

links to sites, most likely pornographic or dealing with online dating, on the comments section of random videos or people's profiles. Another frequently used technique is using bots to post messages on random users' profiles to a spam account's channel page, along with tempting text and images, usually of a sexually suggestive nature. These pages may include their own or other users' videos, again often suggestive. The main purpose of these accounts is to draw people to their link in the home page section of their profile. YouTube has blocked the posting of such links. In addition, YouTube has implemented a CAPTCHA (a type of challenge-response test as an attempt to ensure that the response is not generated by a computer) system that makes rapid posting of repeated comments much more difficult than before, because of abuse in the past by mass-spammers who would flood people's profiles with thousands of repetitive comments.

Yet another kind is actual video spam, giving the uploaded movie a name and description with a popular figure or event which is likely to draw attention or within the video has a certain image timed to come up as the video's thumbnail image to mislead the viewer. The actual content of the video ends up being totally unrelated, sometimes offensive, or just features on-screen text of a link to the site being promoted.

CHAPTER THREE: RELATED WORK

This Chapter deals with research works that are published by different researchers to combat the problem of e-mail spam. Even though there are a number of approaches to fight the problem of e-mail spam, in this Chapter we point out the most commonly used broader categories for the purpose of understandability.

The related works we have reviewed are discussed in the following two broad categories: IP reputation (envelope analysis) and content analysis (data analysis) techniques [14]. The IP reputation methods deal with canceling the SMTP connection before receiving the actual e-mail's data. The content analysis methods deal with filtering out malicious messages based on analysis of e-mail header and content.

3.1 IP Reputation

Esquivel *et al.* [19] proposed an IP reputation based pre-acceptance spam filtering mechanism. The authors classified SMTP senders in to three main categories: legitimate servers, end hosts, and spam gangs, and empirically studied the limits of effectiveness regarding an IP reputation filtering for each category. Then the authors developed techniques that build custom IP reputation lists which improve the performance of existing IP reputation lists.

They have also developed a technique to correlate domain names and IP addresses with sender policy framework records. The authors were able to keep track of IP addresses of spammers by examining the DNS sender policy framework records for their domain.

Even though IP-based blacklist is an effective way to filter spam e-mails, building and maintaining individual IP addresses in the blacklist is difficult as new malicious hosts continuously appear and their IP addresses may also change over time [20]. Thus, Qian *et al.* proposed to replace individual IP addresses in the blacklist with IP clusters [20]. They proposed a new clustering approach that considers both network origin and DNS information.

The authors studied the three common clustering approaches: BGP (Border Gateway Protocol)-based, DNS based, and combined clusters, and proposed a combined cluster approach that can be easily incorporated in to SpamAssassin [30], a popular spam filtering system. In addition to previously used BGP information, the authors also examined reverse DNS records as a way to construct IP clusters.

SpamTracker is a system that classifies e-mail senders using behavioral blacklisting proposed by Ramachandran *et al.* [21]. This system classifies e-mail senders based on their sending patterns. The authors believe that many spammers exhibit stable sending patterns that can act as fingerprints for spamming behavior.

SpamTracker clusters e-mail senders based on the set of domains that they target. It uses these sending patterns of confirmed spammers to build blacklist clusters, each of which has an average vector that represents a spamming fingerprint for that cluster. SpamTracker tracks sending patterns of other senders and computes the similarity of their sending patterns to that of a known spam cluster as the basis for a spam core.

Erickson *et al.* [22] built and deployed a whitelisting e-mail service named DoE-mail. The authors have shown that with whitelists it is easy to identify the much smaller set of users who can legitimately send us e-mail. The system (DoE-mail) had been used for two years by over 120 users. To use the system, a user signs up for DoE-mail account and redirects his/her existing e-mail through it. Then the user owns and manages whitelists and blacklists. Along with the DoE-mail service, the authors implemented Thunderbird add on, a free e-mail client that makes management of the whitelists and blacklists easy.

Another whitelisting approach is the one proposed by Chirita *et al.* in [23]. MailRank, a system that builds a global whitelist from the e-mail contacts of all MailRank users and updates it automatically based on their e-mail activities, is the proposed approach. To prevent attackers from putting their e-mail addresses in the whitelist, the e-mail contacts of all MailRank users are aggregated in to a single e-mail network which is then used to compute a rank for each e-mail address. This approach assumes that e-mails from within a person's social network are typically not spam with a high probability, such that the rank can be used to identify the e-mail address of spammers.

Approaches based on IP reputation have the problem of quick reputation change. Condition of blocked IP might change more quickly than its reputation and once a whitelisted server begins sending spam, it might take time to discover the misbehaving server as connections from whitelisted servers are not checked at all. In addition to this, an e-mail server that uses whitelisting does not escape from the need to filter out connections from unknown servers. Therefore, whitelisting by itself is not a full solution rather part of the whole solution.

Risk assessment of blacklists is a very complex process and needs support from experienced groups. Blocking SMTP connections without looking into the e-mails might be dangerous, since no quarantining is used and hence no recovery of false positives is feasible. This might cause complete lose of valuable e-mail sent from an IP address that is blacklisted by mistake, for example, or has changed its reputation lately.

3.2 Content Analysis

Nowadays the most common counter measures against spam are spam filters such as SpamAssassin which reject or accept e-mail messages based on their content. One of the most common spam filtering techniques used these days, Bayesian filter, is proposed in [24]. The authors sought to employ a Bayesian classification technique to the problem of junk e-mail filtering in one of early works for spam filtering. In addition to message texts, the authors incorporated domain knowledge about the particular task at hand through the introduction of additional features in the Bayesian classifier.

Among features used in classification is examining message texts for the appearance of specific phrases known to be indicative of junk e-mail. Domain types of the sender (.edu, .com,...), examining whether the recipient of a message is an individual user or the message was sent via a mailing list, whether a message has attached documents and the percentage of non-alphanumeric characters in the subject of the mail message are among the indicative features the authors identified.

The authors used a corpus of 1789 actual e-mail messages of which 1578 were junk and 211 were legitimate. Among these, 1538 messages were used for training and the remaining 251 messages were used for testing.

Wang *et al.* [25] demonstrated the use of Support Vector Machines (SVM) as a base classifier to address the problem of separating legitimate e-mails from unsolicited ones. In their work, the authors addressed the feasibility of adaptive learning strategy to spam filtering task. They have also used active learning for selecting the most useful example for labeling and adding the labeled example to training set to retrain a model, and an online learning for solving the problem of dynamic nature of data and drifting concepts.

This work is inspired by the works of [26, 27, 28]. Even though the works of [26] and [27] do not involve filtering spam traffic, they point out the possibilities of using network traffic data for the purpose of identifying different activities in flow level and hence give us an insight that we can filter spam traffic in flow level.

In [26] Bonfiglio *et al.* proposed a technique to successfully reveal traffic of a specific application, Skype traffic, in real time. The authors proposed a framework based on two complementary techniques to reveal Skype traffic in real time. The first approach, based on Pearson's Chi-Square test and agnostic to VOIP-related traffic characteristics, is used to detect Skype's finger print from the packet framing structure, exploiting the randomness introduced at the bit level by an encryption process. The authors call this approach Chi-Square Classifier (CSC). The second approach is based on a stochastic characterization of Skype traffic in terms of packet arrival rate and packet length, which are used as features of a decision process based on Naïve Bayesian Classifiers (NBC).

To proof-check the correctness of the statistical techniques, the authors develop a Payload Based Classifier (PBC), that relies on traditional technique of deep-packet inspection, combined with a per host analysis that allows to identify Skype clients and their generated traffic. The PBC is used to cross-check the results obtained from the statistical approaches. In particular, from the controlled test bed experiments and from real traffic traces as well, the PBC is used to create a benchmark dataset in which the authors classify Skype flows with a very high confidence level. Moreover, the benchmark dataset is used to tune parameters of the above classifiers, as well as to quantify the number of NBC/CSC false-positives and false-negatives. The authors assessed the effectiveness of the two classifiers, when they are either separately or jointly used, by running the NBC and CSC onto the benchmark dataset. They anticipated that the combination of NBC and CSC yields astonishingly good results. Finally, the authors

found out that the joint NBC+CSC classification method effectively limits the number of false positives, yielding conservative results, in the sense that the number of non-Skype flows erroneously classified as such is negligible.

Moore and Zuev [27] successfully utilized traffic flow characteristics to classify network traffic into categories. They have used supervised machine learning to classify network traffic. The authors used data that has been hand-classified (based upon flow content) to one of a number of categories (like Bulk, Database, Interactive, Mail, Services, www, P2P, Attack, Games, and Multimedia). Sets of data consisting of the (hand-assigned) category combined with descriptions of the classified flows (e.g., flow length, port numbers, time between consecutive flows) are used to train the classifier. The authors tested their algorithm using data-sets consisting of only the object descriptions and then compared the predicted category with the actual category for each object. In contrast with the full-payload data used to enable precise classification, this method, once trained with data of a known classification, is able to be used to classify data for which only the TCP-headers are available.

The authors illustrated the performance both in terms of accuracy and trust in the resulting classification of traffic. They showed that in its most basic form a Naive Bayes classifier is able to provide 65% accuracy for data from the same period and can achieve over 95% accuracy when combined with a number of simple refinements. They also illustrated the temporal stability of their technique using test and training sets separated by over 12 months. Significantly, while a relatively low performance is achieved for the simplest Naive Bayes configuration, the authors then showed the full benefits of their refined techniques which led to an accuracy of up to 95%.

The authors have demonstrated that a classification model constructed using known data is able to be applied when far less information is available about the traffic. Critically, they have illustrated a classification technique that may be used retrospectively on data-traces that have previously not been examined in any detail due to the lack of complete information.

The feasibility of detecting spam connections using flow statistics has been shown by Zadnik and Michlorsky in [28]. To this end, the authors analyzed several days of SMTP communication collected at middle-sized email server. They collected data from the

SMTP server hosting mailboxes of the Liberouter [31] project group. In parallel, all SMTP and SMTPS traffic was dumped in a file. The file was processed later on by a script that measured 30 unidirectional flow characteristics in each direction per each connection. In offline mode, the delivered mails were classified by SpamAssassin version 3.2.3 into two groups: relevant e-mails and spam.

In their first experiment, the training set consists of 66% of all annotated traffic statistics and the remainder serves for evaluation of the classifier accuracy. The training of the classification model took approximately 4 minutes while the evaluation was very fast (less than 1 second). Results confirm that the classification of SMTP traffic into specified categories (Spam, NotSpam, Rejected, Outgoing and Others) is possible.

The goal of their second experiment was to find out whether the classifier would still work if it is trained and evaluated only on the incoming flow of the connection. The experiment was motivated by situations where it is possible to monitor only one direction of the connection, for example due to asymmetric routing. The performance of the classifier that observes the incoming stream only is nearly the same as that of the bidirectional one.

The aging of the classifiers was a point of the authors' interest during their last experiment. The classifier trained on whole measurement period was evaluated on data collected two months later. The performance of the classifiers, both bidirectional and incoming flow, degraded significantly.

Sperotto *et al.* [7] investigated if it is possible to detect spammers at the flow level, without relying on email content. Their findings show that the network behavior of suspicious hosts differs substantially from that of a legitimate mail server, both in activity level and incoming/outgoing traffic patterns. Based on these observations, they propose a detection algorithm that makes use of network characteristics. The algorithm has been validated using trusted blacklisting services. The results show that spamming machines can be detected with 92% accuracy for the traces on which the authors validated their approach, meaning that the algorithm has a low probability to report false positives.

The algorithm consists of two main phases and a post-processing step. In the first phase, hosts that do not satisfy three basic selection criteria are filtered out. The criteria are:

number of outgoing connections, connection ratio, and number of distinct destinations. This phase aims to reduce the amount of data to be analyzed and to improve the overall performance of the algorithm. The hosts selected in the first phase are then ranked in the second phase by means of five ordering criteria according to their likelihood of being spammers. These criteria are: number of distinct destinations, percentage of idle time, number of peaks, number of incoming connections, and irregularity in activity.

Finally, ranked hosts are once again filtered according to a post-processing criterion. The criterion specifies the number of hosts to be filtered. The algorithm analyzes the SMTP traffic sent and received from the network that is monitored. Of course, this means spam traffic generated by a spammer outside the monitored network and targeting a different network cannot be considered for the analysis. However, the results show that it is not necessary to have a complete overview of all the traffic generated by a spammer to achieve a good detection level.

The authors evaluated their algorithm over three data sets collected at the University of Twente: a reference data set used to develop the algorithm and two newly collected data sets. Each data set spans over a period of seven days, with an average of ~15M flows. The time windows over which the data sets span are not overlapping. The implementation of their approach uses SQL scripts and can process a data set in a period of 5 hours.

The last two works are the ones that have direct relation with ours. Both works try to inspect malicious flows using network behaviors. In [28], the authors tried to point out the feasibility of detecting spam connections using flow statistics. As clearly stated in their report, the intention of the authors was to use a suitable classification method to validate that spam can be detected in statistics collected upon SMTP connection and to prove its feasibility for spam detection and not to tune the classification to come up with a new algorithm suitable for the purpose.

And in [7], the authors have developed a spam detection algorithm which is able to discriminate between benign and malicious hosts. But their algorithm needs the access to all hosts so that they can watch properties like the host's idle time, the number of outgoing connections from the host at a time, the number of distinct destinations for a given message from a host, the number of incoming connections and idle time of the host

so that they can classify hosts as malicious and benign. But having access to spammers' resources is far from possible. Their technique can be applied to see whether a given corporation's mail servers are suspicious of sending out spam or not but not to successfully filter out spam coming into a corporation's servers from outside.

Content based filters are getting easily evaded by spammers as spammers use a method called 'Bayesian Poisoning' to trick spam filters by inserting ham-like (legitimate e-mail) word salad in the e-mail. The increasing number of image spam is another headache for content based filters as they need to do OCR (Optical Character Recognition) operations to decode the image into text before they can filter. In addition to this, managing the policy list of filters is very time consuming and tedious.

Therefore, the need for new and better spam mitigation techniques is undebatable. The current rate of increment of spam attacks also suggests that the existing techniques seem to lose the war against the spammers. This has given a room for researchers to come up with new spam fighting techniques. To this end, we are proposing a technique to stop spam before it reaches the users inbox, like blacklisting, but that does not depend on the reputation of the sending IP address, unlike blacklisting. We propose to identify spam flow in the network level. This has a big advantage in saving the Internet resource in terms of bandwidth and storage space and user's time from a corporation's point of view.

This is where our approach fits. We propose to develop an algorithm to filter out suspicious connections based on just flow data. We neither need the access to spammers' machines nor want to just see the feasibility of this approach rather we develop and fine tune our algorithm to classify flow data into categories based on observable properties. Due to the high risk of breaking all suspicious connections, our algorithm will score the connections so that the ones with highest scores of suspicion can be dropped and the less suspicious ones (legitimate ones) with below the threshold score can be sent to the inbox. The other connections that are in between the above two categories can be passed for further analysis together with their scores. Therefore, our algorithm is a kind of scoring algorithm which categorizes connections into three groups: highly likely spam, highly likely ham and suspicious. As mentioned above, the highly likely category connections can be dropped and the highly likely ham connections can be delivered to the user's

inbox while the suspicious category connections can be passed for further check-up by filters like SpamAssassin which is an open source and can be integrated with our system.

CHAPTER FOUR: SOLUTION DESIGN

This chapter deals with describing the details of the proposed solution. The first section of this Chapter gives an overview of how our proposed system works while the other section discusses the detail architecture of our system and the individual components in the system.

4.1 Flow based spam detector

Our solution is based on TCP/IP packet flow information. Packets traversing the network will be monitored on the gateway routers to formulate statistics that will be used in identifying spam and ham flows. Based on the formulated statistics and classification model, our ranking algorithm will assign class categories for every traversing flow. These categories will be used to accept, reject or simply pass the specific flow for further examination. To formulate the statistics we will use the TCP/IP header information that will be discussed in the next section. The data we will be using and the ranking algorithm with the classifier are briefly discussed in the consecutive sections.

In general, our system works in the following way. We collect a training data that is a flow record of packets traversing a network and manually classify the records into spam and ham classes. This will be a training data. Based on this training data we train a classification algorithm producing a model for future use. This model will be the best performing among the existing classification algorithms. We have even experimented improving and customizing an existing algorithm so that it fits well for our purpose. Based on this model and a number of best rules gained using the model on the training data, we extract confidence values for each rule. Then, we will go to the real time data collection of a single flow at a time to see weather that flow is of a spam or ham connection. This detection is done using the readymade model with the confidence values for each rule and our ranking algorithm. In real time, the ranking algorithm scores a given flow into categories so that that flow will be rejected or accepted.

The detailed format of our input data, the general architecture of our system and the ranking algorithm are discussed in brief in the next section.

4.2 Design Architecture

In this section, first a pictorial depiction of the design architecture will be presented followed by descriptions of each component in subsequent subsections. As shown in Figure 4.1, the proposed solution has four major components: input data, classifier, text rule, and a ranking algorithm.

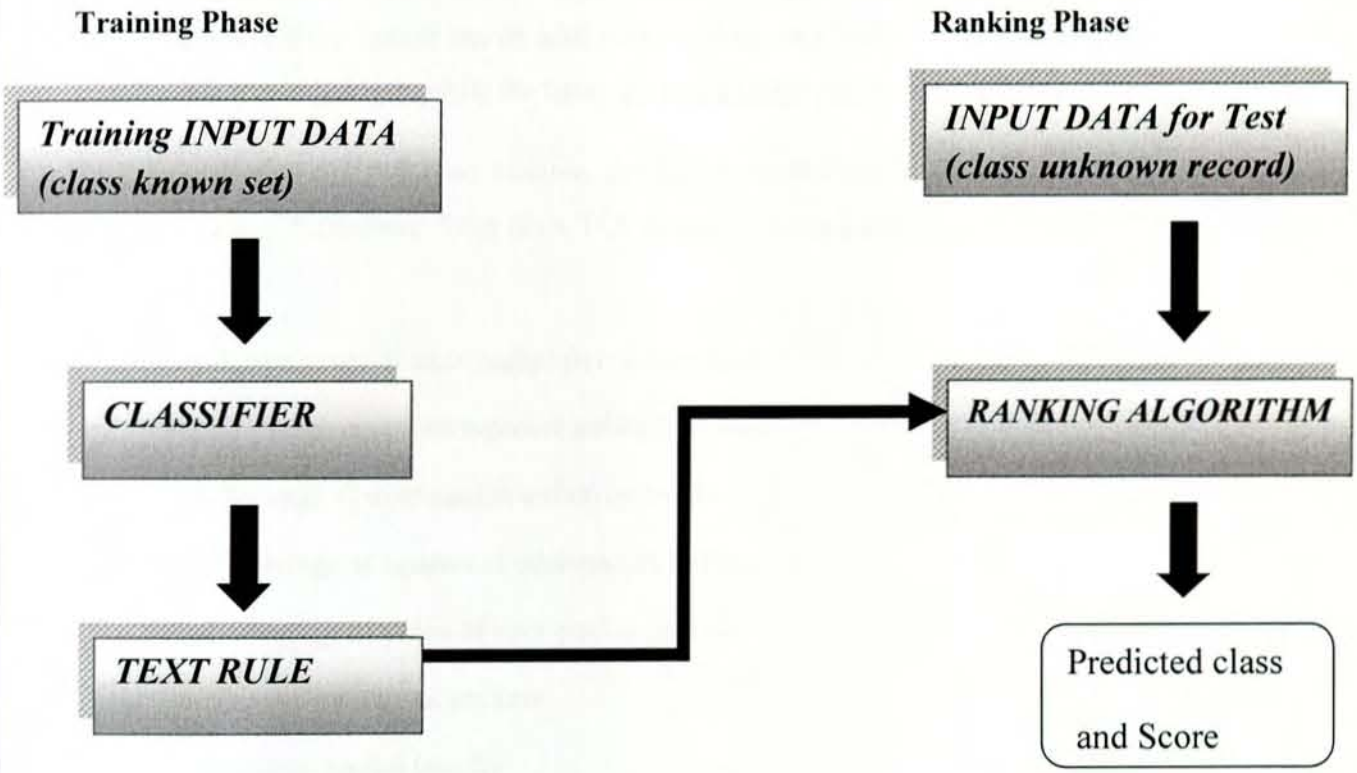


Figure 4.1: Design Architecture

During the training phase, the input data is manually classified and class assigned (spam or ham) record. This set of records is used to generate rules for future prediction based on the best classifier available. The input data in the ranking phase is a single record with unknown class of which we want to know its class.

4.2.1 Input data

Input data is captured from network traffic. It is a real-time collection of data as it traverses over a network. Attributes will be formulated from the packet headers by calculating averages, minimum values, squares, and others and important attributes will be fed to the classifier for training. All the attributes formulated are described in the next paragraphs. In the case of ranking, a single record will be fed to the ranking algorithm to come up with a class prediction. The difference between input data for training and input data for test is that the former has an additional attribute that indicates the class type and contains much more records while the latter has just a single record with unknown class.

For the purpose of our proposed solution, the following statistical data is required. The statistical data is formulated from IPv4 TCP header, Figure 4.2 [54], and IPv4 header, Figure 4.3.

Minii- minimum of inter-packet arrival intervals

Maxii- maximum of inter-packet arrival intervals

Sipi-Average of inter-packet arrival intervals

Sipi2-Average of squares of inter-packet arrivals

Sipi3-Average of cubes of inter-packet arrivals

Sp-Average number of packets

Spl-Average packet lengths

Spl2-Average of squares of packet lengths

Spl3-Average of cubes of packet lengths

Minpl-minimum of packets' lengths

Maxpl-maximum of packets' lengths

Spss-Average of packets with syn set

Slss- Average of packets' lengths with syn set

Spas- Average of packets with ack set

Slas- Average of packets' lengths with ack set

Spfs- Average of packets with fin set

Slfs- Average of packets' lengths with fin set

Sprs- Average of packets with reset set

Slrs- Average of packets' lengths with reset set

Spps- Average of packets with push set

Slps- Average of packets' lengths with push set

Sttl-Average of all TTL fields

Sttl2-Average of squares of TTL fields

Minttl-minimum of seen TTL fields

Maxttl-maximum of seen TTL fields

Stw-Average of TCP window sizes

Mintw-minimum of TCP window size

Maxtw-maximum of TCP window size

Sztwp-Average number of packets with TCP window size zero

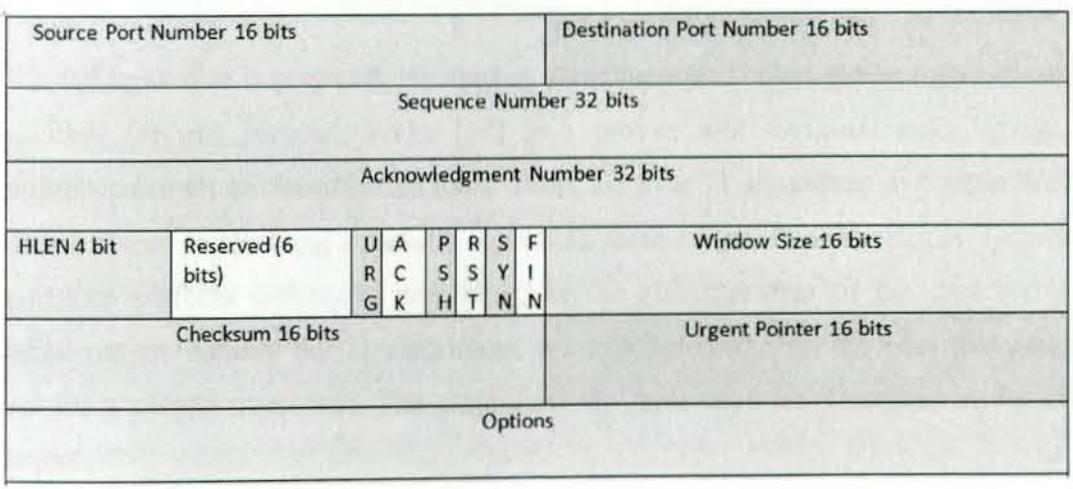


Figure 4.2: IPv4 TCP header

4 bits	4 bits	8 bits	16 bits
Version	**Header Length**	**Service Type**	**Total Length**
Identification		**Flags**	**Fragment Offset**
Time to live (TTL)		**Protocol**	**Header Checksum**
Source IP Address			
Destination IP Address			
IP Options			

Figure 4.3: IPv4 IP header

The packet capture can be done using any of the available capturing tools like Tcpdump [37], Wireshark [40] or Ethereal [39]. We have got a readymade data already captured by Tcpdump and we have not used these tools in our thesis work.

4.2.2 Classifier

After the input data is prepared, we need to experiment and select the best classification algorithm for our purpose. Weka [41] is a proven tool for such tasks. It has a comprehensive set of classification tools. There are over 71 algorithms in total in Weka. When it comes to choosing a specific algorithm, there are no obvious choices. Different algorithms perform differently depending on the characteristics of the data and the desired output. Among the 71 algorithms, we experimented with the ones that can be used for a classification task. The algorithms we have used are described in the next Chapter.

There are also various methods that assist us in getting the best out of our data. Details regarding tools and techniques used will be presented in the next Chapter. After experimenting with the classification algorithms, we have selected the best algorithm for our purpose. We have also made improvements to the best performing algorithm so that it can best fit to our purpose. There are different measures to prefer one algorithm from

another. We will deal with these, as well, in detail in the next Chapter. Having done training and testing the algorithm, we will use the rules from the trained algorithm for future class prediction of unknown-class data.

4.2.3 Text Rule

This is a rule generated from the trained algorithm in a text format. Weka outputs are not in a format we want for our ranking algorithm as input. In order to incorporate the results from the trained algorithm with our ranking algorithm, we converted the output rules into a notepad file. This text rule is organized in a way that our ranking algorithm will use it for future prediction. This rule will be generated during the training phase and will be used thereafter. The rule might be periodically updated to see changes in behaviors of classes. For this purpose, a new data for training has to be captured with already known class type. If there happens to be a major change in the induction rule, our previous rule has to be replaced with the new one without the need of changing the ranking algorithm.

4.2.4 The Ranking Algorithm

The ranking algorithm takes the induction rule (text rule) and data to be predicted as input. Based on the induction rule, this algorithm scores the class probability of an input data. There are scores from 1 to 5. A score of 1 indicates that the data belongs to ham class and a score of 5 indicates a spam class. Data with a score of 2 is most probably ham while data with score of 4 is most probably spam. The middle category with a score of three is a group of data that needs further assessment to classify.

We thought that it is better to accept ham likely e-mails (score 2) instead of rejecting them in expense of losing ham messages. Losing a single ham message could sometimes be catastrophic. In doing so, we might accept some spam messages that resemble ham but this should be acceptable than losing a ham message. We proposed two options for the score 4 category: **tight filtering** and **loose filtering**. When tight filtering is turned on, score 4 category will be rejected like that of score 5 category. But if loose filtering is turned on, the score 4 category will be passed for further filtering like score 3 category. This implementation depends on the user's interest. Some users might prefer to tolerate some amount of spam to their inbox while others might need complete blockage of

anything suspicious of spam behavior. Spammers will, most probably, not resend or communicate with the destination due to huge amount of messages they send and because of their use of automated machines that send messages autonomously. In order to avoid legitimate messages being rejected again and again by suspicion, we proposed assigning a counter for every message rejected. If that message reappears for a given number of times, it can be accepted. Currently, it is believed that spammers do not resend a failed message so that we can accept all messages that are resent. But there is no guarantee that the spammers will not resend in the future. Therefore, this number should be adjusted according to the spammers' behavior and more research has to be done to see the spammers' resending behavior. The pseudo code for the ranking algorithm is shown in Figure 4.4.

```
Accept Input Data for Ranking
Read Induction Rule
Until all rules are checked AND Not Solved
        Check Rule
        If  Not Solved
          Rule++
If Solved
   Calculate Percentage
   If Class Spam
        If Percent equals 100
           Score equals 5
        ElseIf Percent less than hundred AND greater than or equals 95
           Score equals 4
        Else
          Score equals 3
   If Class Ham
        If Percent equals 100
           Score equals 1
        ElseIf Percent less than hundred AND greater than or equals 90
          Score equals 2
        Else
          Score equals 3
Else
   Score equals 3
```

Figure 4.4: Pseudo code for the ranking algorithm

The algorithm starts with taking the record to be scored. In real time, this will be a single flow traversing a network that is captured with the packet capturing tools described. The ranking algorithm, then reads the already prepared induction rules from the text data. There are a number of rules together with their confidence values. Confidence is calculated based on how accurately they performed on the training data. Some rules classify the training data correctly without misclassifications, having 100 percent confidence, while others have some misclassifications reducing their confidence value. These confidence values are referred to as percent in the algorithm.

Having the input data for classification, the algorithm checks whether the data satisfies the first rule or not. If the rule is satisfied, a flag called Solved will be turned on. If not, every rule will be checked until a satisfying rule is found. When the satisfying rule is found, its confidence value will be checked to assign a rank to the record. If there is no rule that the data satisfies, the record will be assigned a score of 3.

We have categorized the confidence values into three categories. The first category contains rules that perfectly classified the training dataset into the respective class. Such rules have 100 confidence values. If the input data satisfies such rules, then the record will be given a score of 5, if it is Spam Class, or a score of 1, if it is Ham Class. That means, a record of spam class with a confidence value of 100 will be automatically rejected and a record of ham class with a confidence value of 100 will be accepted.

The second category for spam class records is those with confidence value between 95(inclusive) and 100. We have done some experimentation to come up with the boundary of 95 confidence value. This is a score 4 class and records in this category are very suspicious to be spam. Therefore, there is a probability that these records will be rejected, for example, when tight filtering is turned on. Based on our training data, we have seen that it is risky to include other records with less confidence values than 95. Hence, we have decided to include the remaining records (with less than 95 confidence value) in the third category with score 3. But, records with confidence value of 95 and more are very close the definitely spam category and including them in score 4 is necessary and less risky.

When the records are of ham class, we have the lower bounder of 90 confidence value (inclusive) and an upper bound of 100. The records having these confidence values and ham class are assigned score of 2. These records are very likely to be ham and it is even possible to accept such records. But, there is a low probability of accepting spam messages and it should be acceptable. Reducing the boundary to a confidence value of below 90 will let in many spam messages into ham territory and is against the very first idea of filtering out spam.

CHAPTER FIVE: EXPERIMENTATION AND IMPLEMENTATION

Starting with describing the tools and techniques we have used, this Chapter presents, in detail, our experimentation. The results and findings from the experiments are also discussed together with how our final solution is implemented.

5.1 Tools and Techniques

The following are the tools that are essential for the realization of the proposed solution. The techniques are also explained along with the tools.

TCPDUMP

Tcpdump is a powerful command line packet analyzer. It can print out a description of the contents of packets on a network interface or it can save the packet data to a file for later analysis. Furthermore, it can read packets from saved files for analysis. This tool is used for capturing packets either for training the algorithm or doing the scoring task in real-time.

When Tcpdump finishes capturing packets, it reports counts of packets captured, packets received by filter if specified, and packets dropped by the kernel due to lack of buffer space.

There are options to be used with Tcpdump in command line to specify the type and format of an output. For example:

-e: print the link-level header on each dump line

-F: use file as input for filter expression

-K: do not attempt to verify IP, TCP or UDP checksums

-n: do not convert addresses to names

-S: print absolute, rather than relative, TCP sequence number

-tt: print an unformatted timestamp on each dump line

-XX: when parsing and printing, in addition to printing the headers of each packet, print data of each packet, including its link-level headers, in hexadecimal and ASCII.

Based on the type of the protocol, the output formats differ. For the '-e' option, on Ethernets, the source and destination addresses, protocol, and packet length are printed. On FDDI (Fiber Distributed Data Interface) networks, frame control field, source and destination addresses and packet lengths are printed.

The following is the general format of a TCP protocol line:

Src>dst: flags data-seqno ack window urgent options

It is also possible to capture packets with particular flag combinations, meaning just capturing packets with some flags set and some others not, for example.

We were not able to collect raw packets and hence used already collected data. We have not used Tcpdump in our thesis work. We described it here because of the fact that it is necessary for future use of our algorithm.

MS Office Excel

After first captured by Tcpdump, our input data is converted into an Excel file for further processing. Using Excel sheets, we pre-processed our data so that irrelevant and incomplete data can be removed. We have removed four attributes found to be not important for the classification task and some records with missing values have also been removed. Details regarding these are explained in Section 5.2.2. Excel's filtering feature has also been of great help to visualize and analyze data in subsets.

WEKA

WEKA (Waikato Environment for Knowledge Analysis) [41] is an open-source software which is a collection of machine learning algorithms for data mining tasks. It contains tools for data pre-processing, classification, regression, clustering, association, and visualization. For our experimental purpose, we have used different classification algorithms that exist in Weka and selected the best performing algorithm for our ranking algorithm implementation. The algorithms we have used and their performances in our data set are described in section 5.2.5 in detail.

Notepad

We have used notepad to formulate the induction rules that have been generated by the selected algorithm during classification. Our ranking algorithm reads the induction rules from this Notepad formulated file.

Python

Python [42] is a free-to-use programming language that runs on Windows, Linux/Unix, and Mac OS. It is a simple but effective object-oriented programming language. Python is an excellent tool for scanning and manipulating textual data. Because of this fact, we have used Python for writing our ranking algorithm which reads input from a text file.

5.2 Data Collection and Experimentation

5.2.1 Data Collection

Initially, the data for this thesis work was planned to be collected from Addis Ababa University (AAU) mail server. But, due to changes in management personnel and over-suspicious privacy concerns, we were not able to use data from AAU. We have also tried to collect our data from other local universities, but the same concerns persisted in wherever we went. As a matter of this fact, we are obliged to use data from sources outside the country. The data we have used is described in detail in [28]. The data was collected from an SMTP server that hosts over one hundred mailboxes in the Czech Republic. The data was collected for nine days. The raw packets were dumped with Tcpdump and in parallel, received e-mails were saved to separate folders. The delivered e-mails were classified by SpamAssassin in offline mode. Later, the results from the classification were manually verified and the misclassified ones were corrected. Figure 5.1 shows the data collection setup [28].

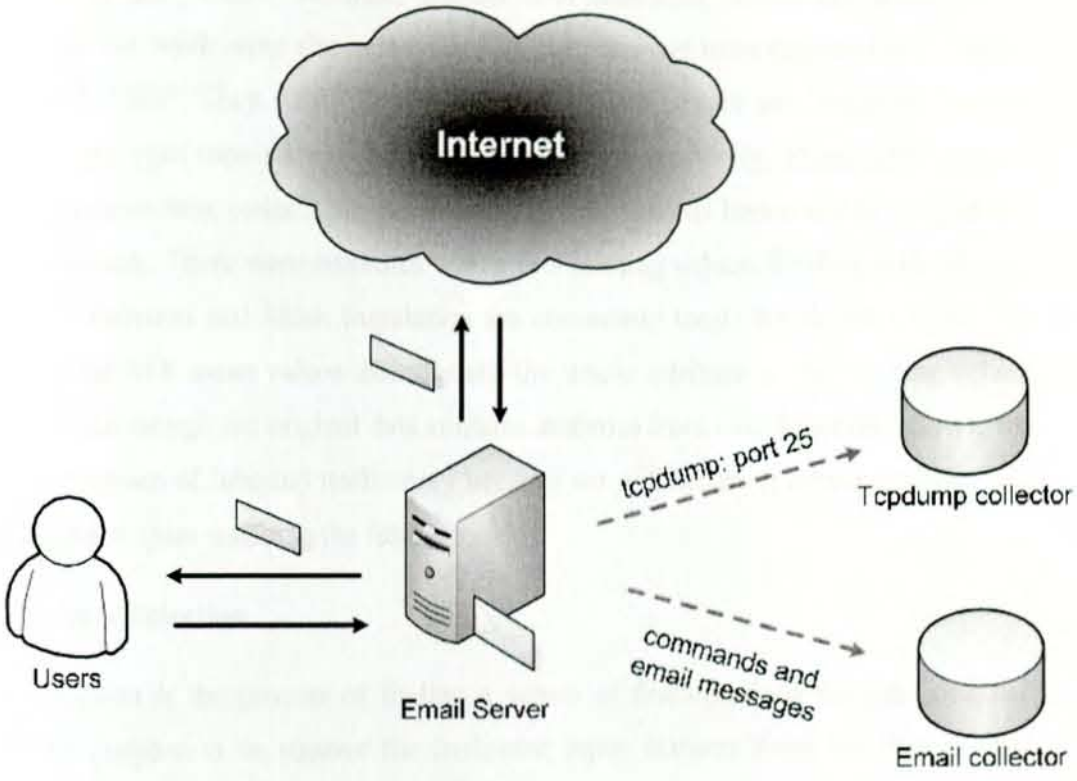


Figure 5.1: Data collection setup

5.2.2 Data Preprocessing

Data preprocessing describes any type of processing performed on raw data to prepare it for another processing procedure. It transforms the data into a format that will be more easily and effectively processed for the purpose of the user. As pointed out in Section 5.1, we have used MS Excel to preprocess our data. Using Excel and Weka, we have removed 629 invalid and irrelevant records from the original data. Invalid records are records containing many attributes with missing values. The original data we have received had an attribute called "valid" and was assigned a value of 1 for valid records and 0 for the non-valid ones. We removed the invalid records and then also removed the whole attribute ("valid") as there is no need to keep the attribute when all the remaining records contain the same value of 1 for that attribute. An attribute containing the same value for all the records is irrelevant for our classification.

In addition to the "Valid" attribute, another four attributes which are found to be irrelevant for our work were also removed. The attributes we have removed are "Spus", "Slus", "Sts", "Ets". They stand for sum of packets with urgent set, length of packets with urgent set, start time stamp, and end time stamp, respectively. These attributes are irrelevant because they contain the same values for all records hence not helping in the classification task. There were attributes with a few missing values. To deal with missing values Case Deletion and Mean Imputation are commonly used. We decided to fill the missing values with mean values than discard the whole attribute as the missing values were few. Even though the original data contains statistics from two directional flows, we have used statistics of inbound traffic only because we will be using inbound flows only to determine the spam traffic in the future.

5.2.2.1 Feature Selection

Feature selection is the process of finding a subset of features from the total original features. Its purpose is to remove the irrelevant input features from the data set for improving the classification accuracy. Feature selection is essential for model construction. Irrelevant and redundant features may lead to a complex model as well as poor detection accuracy [52].

There are many data mining methods for selecting important features from training data sets such as information gain, gain ratio, document frequency, mutual information, CHI, and term strength [43]. For a text data set like ours, information gain, gain ratio and CHI methods can be used. In our thesis work, to select the important input attributes from the original training data set, the three methods were tested and we found that information gain approach performed better. Information gain is the amount of information associated with an attribute value that is related to the probability of occurrence. Our selection of this approach is based on its performance on our dataset and other related experiments like those in [43].

Information gain measures the amount of information about the class prediction, if the only information available is the presence of a feature and the corresponding class distribution. It measures the expected reduction in entropy (uncertainty associated with a random feature). Hence, attributes with random distributions have higher entropy and lower information gain; therefore, such attributes are less interesting.

Information gain ratio is also used instead of information gain. It is a similar technique with information gain but biases the decision tree against considering attributes with a large number of distinct values. However, attributes with very low information values then appeared to receive an unfair advantage.

In probability theory and statistics, the chi-square distribution with k degrees of freedom is the distribution of a sum of the squares of k independent standard normal random variables. It is one of the most widely used probability distributions in inferential statistics. The chi-square distribution is used in the common chi-square tests for goodness of fit of an observed distribution to a theoretical one. Top ten features that are selected based on the three methods are shown in Table 5.1. These features were described in section 4.1.

Table 5.1: Top ten Features selected by the three methods

Rank Number	**Information Gain**	**Gain Ratio**	**CHI-square**
1	Spl3	Minpl	Spl3
2	Spl2	Slfs	Spl2
3	Sipi2	Slas	Sipi2
4	Sipi	Spl	Sipi
5	Maxii	Spps	Maxii
6	Sipi3	Spl2	Stw
7	Stw	Maxpl	Sipi3
8	Spl	Spl3	Slas
9	Maxpl	Sp	Spl
10	Slas	Sipi3	Maxpl

It can be seen from Table 5.1 that information gain and chi square select almost similar features. Experimenting with these selected features, we have seen that information gain features are much better. Experimental results obtained using this method are explained in Section 5.2.3. Features selected by information gain are also in accordance with results we obtained by manually selecting best features which are also explained in Section 5.2.3.

5.2.2.2 Unbalanced Dataset Handling

A dataset is said to have a class imbalance if it contains many more examples of one class than the other. Such situations pose challenges for classifiers since they tend to ignore small classes while concentrating on classifying larger once accurately. Our dataset contains 48,933 spam instances and 1592 ham instances. This is one example of unbalanced datasets that needs some balancing work before classification to avoid classifier bias.

However, there is one critical difference between our dataset imbalance and other datasets that often have class imbalance problems such as fraud detection, fault detection, and anomaly detection which typically contain much fewer instances of the event of interest (abnormal activity in the mentioned examples) than of irrelevant events (normal activities) [44]. Our dataset contains much more instances of spam class (the feature of the Interest) than ham class and this class imbalance problem of ours is not as severe as the other way round because we have more sets of interesting instances. Even though our dataset is not cursed with the problem of few instances of interest class, it still needs balancing tasks to achieve high classification accuracy.

To this end, we have used a SMOTE (Synthetic Minority Over-sampling Technique), which oversamples the smaller class without making any changes to the larger class. As pointed out in [44], there is no single point where every dataset can be said to be balanced. Some datasets produce good results in the ratio one to ten while others give same results as previous in the ratio one to one hundred. Therefore, we have done little experimenting to see in what ratio we can get the best out of our dataset. The best result we got was obtained by under-sampling the smaller class without changing the larger class until they reach the ratio of about one to eight. Our final balanced set contains 6112 instances of ham class and 48,933 instances of spam class.

5.2.5 Classification

Classification is a data mining technique that assigns items in a collection to target categories or classes. The goal of classification is to accurately predict the target class for each case in the data. A classification task begins with a dataset in which the class assignments are known. In the training process, a classification algorithm finds relationship between the values of the attributes and the values of the target class. Different classification algorithms use different techniques for finding the relationships. These relationships are summarized in a model, which can then be applied to a different dataset in which the class assignments are unknown.

Classification models are tested by comparing the predicted values to known target values in a set of test data. Test metrics are used to assess how accurately the model predicts the known values. The following are the most common metrics [38]:

Confusion Matrix

A confusion matrix displays the number of correct and incorrect predictions made by the model compared with the actual classification in the test data. A confusion matrix has the following general format:

Table 5.2: The general format of a confusion matrix

Classes	**Classified as**		
(actual)	**a**	**b**	**Total**
a	*TP*	*FN*	P
b	*FP*	*TN*	N

Where:

a stands for spam class and b stands for ham class.

TP (True Positive):

Is the number of instances that are Positive and classified as Positive by the model

FN (False Negative):

Is the number of instances that are Positive but classified as Negative by the model

FP (False Positive):

Is the number of instances that are Negative but classified as Positive by the model

TN (True Negative):

Is the number of instances that are Negative and classified as Negative by the model.

P (Positive): Is the total number of positive classifications.

N (Negative): Is the total number of negative classifications.

Two types of errors can be seen from the confusion matrix, Type I and Type II errors. Type I Errors are errors that are False Positive and Type II errors are errors that are False Negative. In our case, Type I Error (FP) is much severe than Type II Error (FN). That means it is better to classify a spam message as ham than classifying ham as spam. Dropping of a single ham message, by mistake, might be catastrophic. This fact has been considered while evaluating a classifier. There are other measures that arise from the confusion matrix, which are discussed below.

TPR (True Positive Rate): the proportion of actual Positives which are predicted Positive

$$\text{TPR} = \frac{\text{TP}}{P} = \frac{TP}{TP+FN}$$ where P stands for actual Positives

TPR is also called sensitivity or recall.

TNR (True Negative Rate): the proportion of actual Negatives which are predicted Negative

$$\text{TNR} = \frac{TN}{N} = \frac{TN}{TN+FP}$$ where N stands for actual Negatives

TNR is also called specificity.

FPR (False Positive Rate): the proportion of actual Positives which are predicted Negative

$$\text{FPR} = \frac{\text{FP}}{\text{N}} = \frac{FP}{TN+FP}$$

Based on the confusion matrix results, accuracy and precision can be computed as follows:

Accuracy refers to the percentage of correct predictions made by the model when compared with the actual classification.

$$\text{Accuracy} = \frac{TP}{TN} \Big/ (\text{P+N})$$

Precision: is the proportion of predicted Positives which are actually Positive.

$$\text{Precision} = \frac{TP}{(TP+FP)}$$

F-measure: is a measure of a test's accuracy. It considers both the precision p and the recall r of the test to compute the score: p is the number of correct results divided by the number of all returned results and r is the number of correct results divided by the number of results that should have been returned. The F -measure can be interpreted as a weighted average of the precision and recall, where an F -measure reaches its best value at 1 and worst score at 0.

ROC Area

Receiver Operating Characteristics (ROC) curve is used to characterize the tradeoff between hit rate and false alarm rate. ROC curves depict the performance of a classifier without regard to class distribution or error costs. They plot the number of positives included in the sample on the vertical axis, expressed as a percentage of the total number of positives, against the number of negatives included in the samples, expressed as a percentage of the total number of negatives, on the horizontal axis.

ROC area is the area under the curve, that is, the area between the ROC curve and the lower right hand corner of the diagram. The ROC area is directly related to the separation of the two conditional distributions, and therefore measures the discrimination ability of

the forecast. The closer the curve is to the upper left hand corner of the graph, the greater the area and the greater the discrimination. The range of the ROC area is 0 to 1. If the ROC lies along the diagonal, corresponding to an area of 0.5, this means the two conditional distributions lie exactly on top of each other and there is no discrimination. Values below 0.5 denote negative or "perverse" discrimination. Although there is some discrimination whenever the ROC area is > 0.5, in most situations the discrimination ability of the forecast is not really considered useful in practice unless the ROC area is greater than 0.7.

Based on the above metrics, we have selected the best performing classification algorithm for our purpose. Below are the algorithms that are found to have better classification results.

Naïve Bayes

The Naïve Bayes algorithm is based on conditional probabilities [38]. It uses Bayes' theorem, a formula that calculates a probability by counting the frequency of values and combinations of values in the historical data. Bayes' theorem finds the probability of an event occurring given the probability of another event that has already occurred.

Naïve Bayes makes the assumption that each predictor is conditionally independent of the other. For a given target value, the distribution of each predictor is independent of other predictors.

The classification results using Naïve Bayes algorithm are shown below.

The results are obtained using stratified cross-validation that divides the input data into ten subsets and uses the nine subsets for training and the remaining one for testing.

Table 5.3: Confusion Matrix of Naïve Bayes Algorithm

Classes (actual)	**Classified as**	
	spam	ham
spam	*46,669*	*2,264*
ham	*35*	*6077*

Table 5.4: Detailed Accuracy by Class for Naïve Bayes algorithm

Class	**TP Rate**	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**ROC Area**
Spam	0.954	0.006	0.999	0.954	0.976	0.998
Ham	0.994	0.046	0.729	0.994	0.841	0.998
Weighted Avg.	0.958	0.01	0.969	0.958	0.961	0.998

Correctly Classified Instances *52746* ***95.8234 %***

Incorrectly Classified Instances *2299* ***4.1766 %***

As can be seen from Tables 5.2 and 5.3, Naïve Bayes algorithm has a good TP value, a good TPR, Precision, and Recall for Spam class but it has a high misclassification (4.17%) making it difficult to adopt for our solution. It has a FPR of 4.6% for ham class which means it classifies 4.6% of ham messages as spam making it a bit difficult for adopting.

JRip

JRip is a rule induction algorithm which generates rules in if-then form. JRip implements a propositional rule learner, RIPPER, which was proposed by Cohon [45]. Ripper builds a rule set by repeatedly adding rules to an empty rule set until all positive examples are covered. Rules are formed by greedily adding conditions to the antecedent of a rule until no negative examples are covered. The classification results using JRip algorithm are shown below.

The results are obtained using stratified cross-validation that divides the input data into ten subsets and uses the nine subsets for training and the remaining one for testing.

Table 5.5: Confusion Matrix of JRip Algorithm

Classes (actual)	**Classified as**	
	spam	ham
spam	*48677*	*256*
ham	*253*	*5859*

Table 5.6: Detailed Accuracy by Class for JRip algorithm

Class	**TP Rate**	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**ROC Area**
Spam	0.995	0.041	0.995	0.995	0.995	0.978
Ham	0.959	0.005	0.958	0.959	0.958	0.978
Weighted Avg.	0.991	0.037	0.991	0.991	0.991	0.978

Correctly Classified Instances *54536* ***99.0753*** *%*

Incorrectly Classified Instances *509* ***0.9247*** *%*

JRip has performed better than Naïve Bayes. It has much reduced the FPR for ham class down to 0.5 and also misclassified only 0.92% of test data. Its Precision and Recall are also way better than Naïve Bayes. However, it still has unpractical FPR. A 3.7% FPR is risky to tolerate.

J48 Decision Tree Algorithm

Decision tree involves decision based classification and adaptive learning over a training dataset [46]. Decision trees are produced by algorithms that identify various ways of splitting a dataset into branch-like segments. These segments form an inverted decision tree that originates with a root node at the top of the tree.

The J48 algorithm is the Weka implementation of the C4.5 decision tree learner. The algorithm uses a greedy technique and determines, at each step, the most predictive attribute and splits a node based on this attribute. It deals with numeric attributes by determining where thresholds for decision should be placed. Results from J48 classification are shown below.

The results are obtained using stratified cross-validation that divides the input data into ten subsets and uses the nine subsets for training and the remaining one for testing.

Table 5.7: Confusion Matrix of J48 Algorithm

Classes (actual)	**Classified as**	
	spam	ham
spam	*48854*	*79*
ham	*175*	*5937*

Table 5.8: Detailed Accuracy by Class for J48 Algorithm

Class	**TP Rate**	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**ROC Area**
Spam	0.998	0.029	0.996	0.998	0.997	0.996
Ham	0.971	0.002	0.987	0.971	0.979	0.996
Weighted Avg.	0.995	0.026	0.995	0.995	0.995	0.996

Correctly Classified Instances *54791* ***99.5386 %***

Incorrectly Classified Instances *254* ***0.4614%***

J48 has given us an improved result in every aspect: high TP, low FP, high Precision and Recall, and high ROC Area as well. It has reduced the FPR down to 2.6% with improved Precision of Ham class to 98.7%. Overall it has a misclassification of 0.46% which is practically tolerable and can be dealt with. This model lets only 0.16% of Spam to be wrongly delivered to the user while correctly detecting 99.84% of spam. This is so lucrative that we used J48 algorithm as a model to construct our ranking algorithm. In the next section, we discuss some of the best predictive attributes we have identified using the J48 model.

The above classification results are based on full dataset usage. We have used all attributes in the classification process. Our next experiment was to see the performance of each attribute alone using the best performing classifier, J48. Below we have shown the results for some of the attributes when used alone. The results of other attributes that are not shown here are shown in Annex I.

Maxpl (Maximum of packets' lengths)

Table 5.9: Confusion Matrix using Maxpl only

	Classified as	
Actual	Spam	Ham
Spam	46410	2523
Ham	1663	4449

Table 5.10: Detailed Accuracy By Class using Maxpl only

Class	**TP Rate**	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**ROC Area**
Spam	0.948	0.272	0.965	0.948	0.957	0.956
Ham	0.728	0.052	0.638	0.728	0.68	0.956
Weighted Avg.	0.924	0.248	0.929	0.924	0.926	0.956

Slas (Average of Packets' Lengths with ack Set)

Table 5.11: Confusion Matrix using Slas only

	Classified as	
Actual	Spam	Ham
Spam	47838	1095
Ham	1892	4220

Table 5.12: Detailed Accuracy By Class using Slas only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.978	0.31	0.962	0.978	0.97	0.909
Ham	0.69	0.022	0.794	0.69	0.739	0.909
Weighted Avg.	0.946	0.278	0.943	0.946	0.944	0.909

Spl2 (Average of Squares of Packet Lengths)

Table 5.13: Confusion Matrix using Spl2 only

	Classified as	
Actual	Spam	Ham
Spam	48022	911
Ham	1243	4869

Table 5.14: Detailed Accuracy By Class using Spl2 only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.981	0.203	0.975	0.981	0.978	0.985
Ham	0.797	0.019	0.842	0.797	0.819	0.985
Weighted Avg.	0.961	0.183	0.96	0.961	0.96	0.985

Spl3 (Average of Cubes of Packet Lengths)

Table 5.15: Confusion Matrix using Spl3 only

	Classified as	
Actual	Spam	Ham
Spam	48465	468
Ham	615	5497

Table 5.16: Detailed Accuracy By Class using Spl3 only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.99	0.101	0.987	0.99	0.989	0.995
Ham	0.899	0.01	0.922	0.899	0.91	0.995
Weighted Avg.	0.98	0.091	0.98	0.98	0.98	0.995

Stw (Average of TCP Window Sizes)

Table 5.17: Confusion Matrix using Stw only

	Classified as	
Actual	Spam	Ham
Spam	48444	489
Ham	1867	4245

Table 5.18: Detailed Accuracy By Class using Stw only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.99	0.305	0.963	0.99	0.976	0.945
Ham	0.695	0.01	0.897	0.695	0.783	0.945
Weighted Avg.	0.957	0.273	0.956	0.957	0.955	0.945

As it can be seen from the above Tables, Spl3 has the highest classification accuracy with an average TPR of 98% and FPR of 9.1%. Spl2 is another attribute with a good classification accuracy having 96% and 18% average TPR and FNR, respectively. Maxpl, Slas and Stw have also better classification accuracies than other attributes.

Next, we experimented to see how the combinations of two or more attributes perform. To this end we have used the combination of the best performing Spl3 attribute with others. Using Spl3 and Spl2 have given us the best improvement over Spl3 alone. We got 98.9% and 5.7% TPR and FPR respectively when Spl3 and Spl2 are used together. This is an expected result as both attributes are the highest performing attributes alone. Believing that using only top performing few attributes may give us the best result, we experimented with three, four, five and more best attributes. But the result was not as we thought. Using the best performing few attributes alone did not give us much improvement over using just Spl3 and spl2.

We have other attributes like "Minii" which perform very weakly when used alone but make significant impact when used with spl3. Minii has 90% TPR and 72% FPR when used alone. Even though Minii has a high FPR, its combination with Spl3 gives us an improved performance of 98.9% TPR and 5.8% FNR. This prompted us to think that we need to use a good combination of both top performing attributes and weaker attributes.

To do this, we have categorized attributes into groups based on their closeness.

Group 1: Minii, Maxii, Sipi, Sipi2, Sipi3

This group contains attributes that deal with packet inter-arrival times

Group 2: Sp, Spl, Spl2, Spl3, Minpl, Maxpl

This group contains attributes that deal with packet sizes.

Group 3: Spss, Slss, Spas, Slas, Spfs, Slfs, Sprs, Slrs, Spps, Slps

This group contains attributes that deal with flags.

Group 4: Sttl, Sttl2, Minttl, Maxttl

This group has attributes dealing with time-to-live values.

Group 5: Stw, Mintw, Maxtw, Sztwp

In this group we have attributes that deal with TCP window size.

We have grouped attributes due to the fact that attributes in a similar category have a very related information and taking one or two from each category can represent information of that category.

The first thing we have done is to take one attribute from each category based on their individual performances. The best performing attributes one from each category are: Sipi2, Spl3, Slas, Sttl, and Stw. The result obtained using these five attributes is shown in Tables 5.19 and 5.20.

Table 5.19: Confusion Matrix using five selected attributes

	Classified as	
Actual	Spam	Ham
Spam	48825	108
Ham	244	5868

Table 5.20: Detailed Accuracy By Class using five selected attributes

Class	**TP Rate**	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**ROC Area**
Spam	0.998	0.04	0.995	0.998	0.996	0.997
Ham	0.96	0.002	0.982	0.96	0.971	0.997
Weighted Avg.	0.994	0.036	0.994	0.994	0.994	0.997

Correctly Classified Instances	54693	99.3605 %
Incorrectly Classified Instances	352	0.6395 %

Furthermore, we have experimented by adding some attributes to the best five we have already used. After some experimentation, adding of another four attributes, maxii, Maxpl, Spps, and Maxtw is found to improve the classification result. The final result using the selected nine attributes is shown in Tables 5.21 and 5.22.

Table 5.21: Confusion Matrix using the selected nine attributes

	Classified as	
Actual	Spam	Ham
Spam	48837	96
Ham	180	5932

Table 5.22: Detailed Accuracy By Class using the selected nine attributes

Class	**TP Rate**	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**ROC Area**
Spam	0.998	0.029	0.996	0.998	0.997	0.993
Ham	0.971	0.002	0.984	0.971	0.977	0.993
Weighted Avg.	0.995	0.026	0.995	0.995	0.995	0.993

Correctly Classified Instances	54769	99.4986 %
Incorrectly Classified Instances	276	0.5014 %

The results using the selected nine attributes are as good as the results we got using the total 29 attributes. But using the reduced number of attributes is advantageous in terms of speed and complexity. The rules derived from using the 29 attributes are very complex and it takes more time to generate the rules.

5.2.6 Results and Discussion

In this section, we discuss the results we found from the experiments in the previous section. Basing our findings on the J48 model we have trained and tested in the above section, we will discuss the attributes which have a profound effect in identifying spam flows. We will also point out any other experimental results that comply or contradict with ours.

As pointed out by Gomes *et.al.* [47] and Beverly *et.al.* [48], there are differences in the ways spam messages and ham messages are sent. These differences are possibly due to the inherent distinct nature of e-mail senders and their connections with e-mail recipients in each group. Whereas a ham transmission is the result of a bilateral relationship, typically initiated by a human being, driven by some social relationship, a spam transmission is basically unilateral action, typically performed by automatic tools and driven by the spammer's will to reach as many targets as possible without being detected.

Because spammers must send large volumes of e-mail to be viable economically, they transmit e-mails continuously, asynchronously, and in parallel [48]. The sources of spam are frequently large compromised botnets; 85% of spam is sent from botnets [9], which are resource constrained and typically connected to the Internet by links with asymmetric bandwidth. Therefore, the flows that comprise spam traffic exhibit behaviors consistent with traffic competing for link access. Furthermore, today's botnet masters and attackers are seeking money, driven by profits, and motivated more by a desire to gain financial benefit than to create havoc [53]. Hence, spammers usually pay for these botnet masters based on the duration of time they actively use the rented botnets, forcing them to send large amount of spams in short period of time in maximum possible way. These factors play a great role in differentiating spam from ham.

The following are the major findings from our research experiment:

There is a notable difference between spam and ham e-mails in terms of size. Our attributes that are related to e-mail size show significant difference between the two categories. In terms of number of packets in a single flow, average size of packets and maximum packet length in a given flow, we found out interesting differences that can be finger prints for detection. These facts are shown in Figures 5.2 to 5.6.

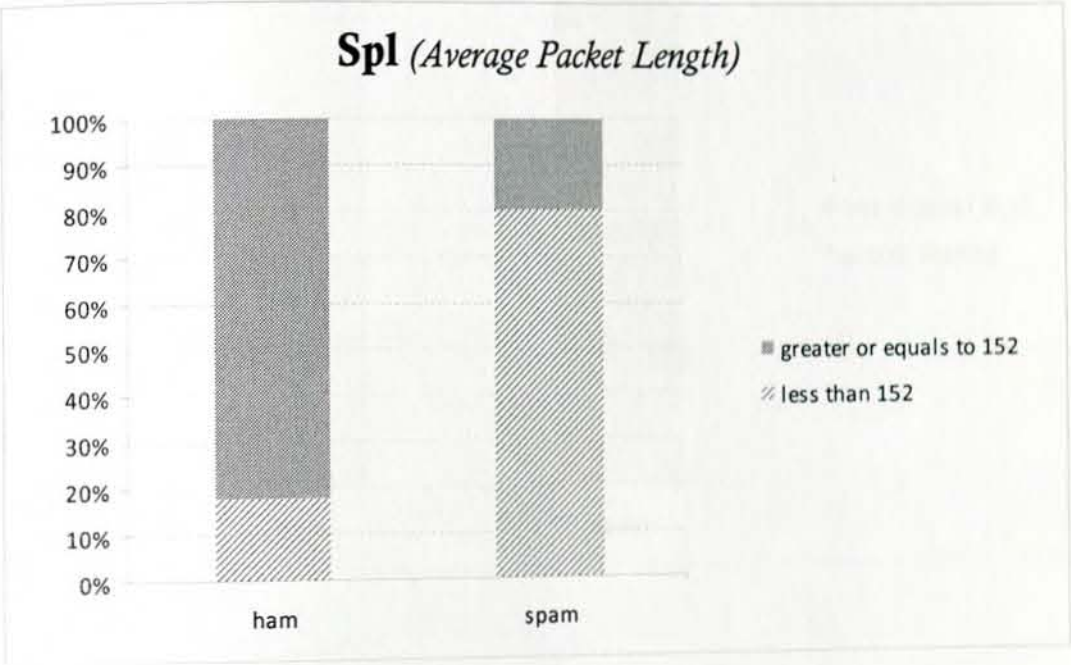


Figure 5.2: Average packet lengths for ham and spam connections

In Figure 5.2, we can see that 82% of spam flows have packet sizes less than 152 bytes, while only 18% of ham flows have less than that figure. This is in parallel with the number of packets for each flow, which is depicted in Figure 5.3. From the figure we can see that when 89% of ham flows have more than ten packets, only 20% of spam flows have more than ten packets per flow. Still continuing with size, we see in Figure 5.4 that 85% of spam flows have maximum packet size (in a flow) less than 658 bytes, while only 14% of ham flows have maximum packet length less than 658 bytes.

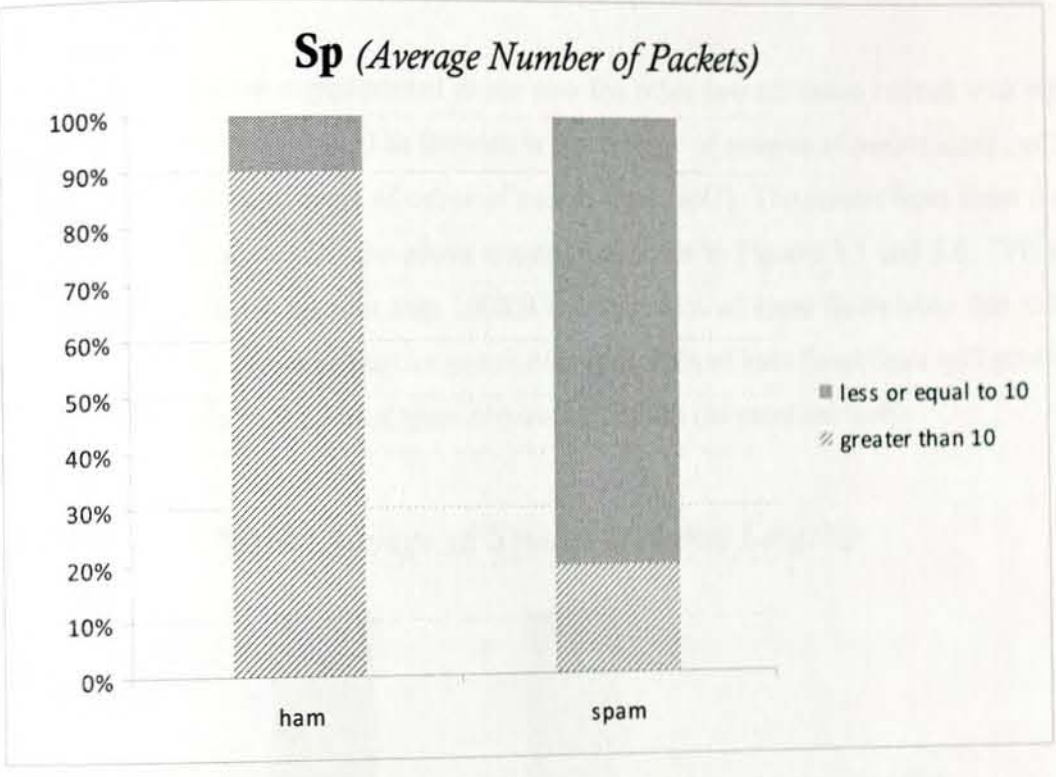


Figure 5.3: Average number of packets for ham and spam flows

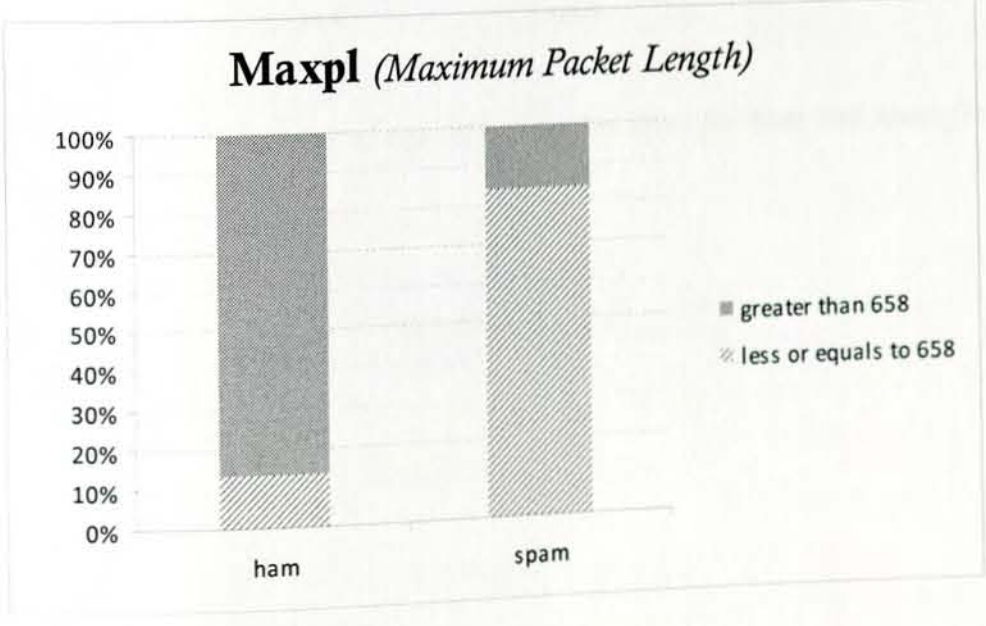


Figure 5.4: Maximum packet lengths in a single flow for ham and spam connections

Furthermore, we have experimented to see how the other two attributes related with size behave in the two categories. The first one is the average of squares of packet sizes (spl2) and the second is the average of cubes of packet sizes (spl3). The results from these two attributes still coincide with the above results. As shown in Figures 5.5 and 5.6, 73% of ham flows have Spl2 greater than 100KB but only 13% of spam flows obey this rule. Spl3 has a slightly better distinctive power than spl2. 76% of ham flows have spl3 greater than 1.5 GB, while only 13% of spam connections fall in the same category.

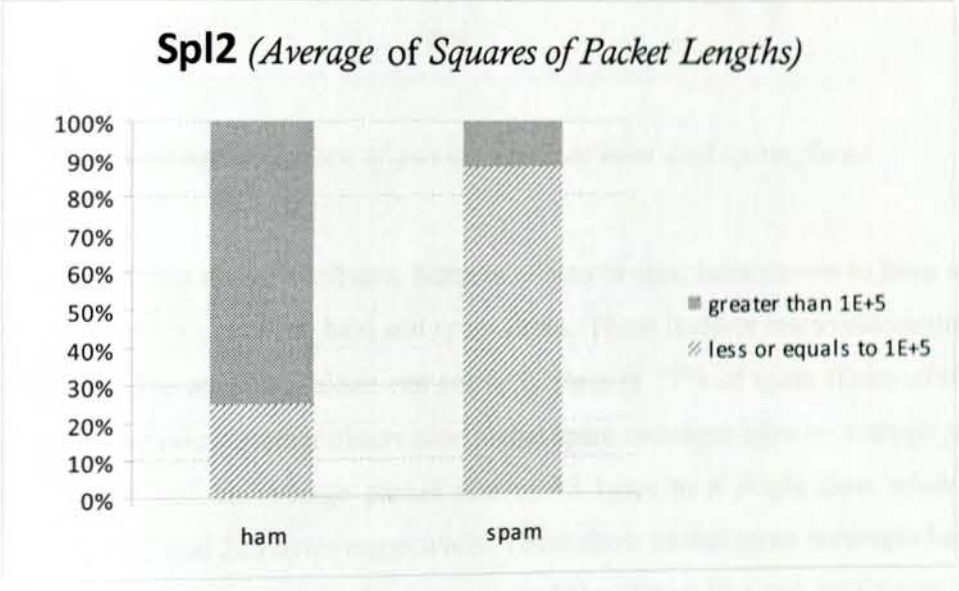


Figure 5.5: Average of squares of packet sizes for ham and spam flows

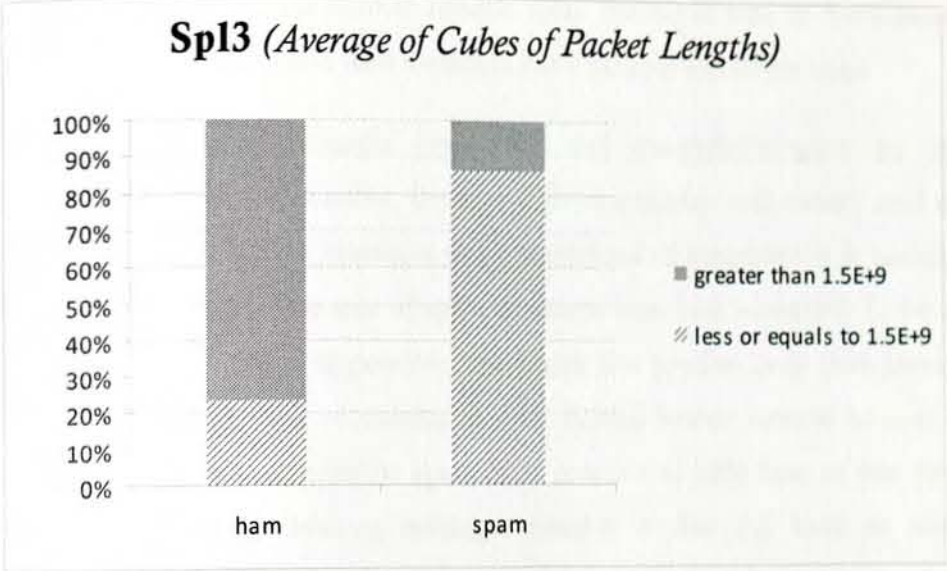


Figure 5.6: Average of cubes of packet sizes of ham and spam flows

The results from the above attributes, features related to size, have shown to have a high discriminatory power between ham and spam flows. These features are so discriminatory that the above five attributes alone can correctly classify 77% of spam flows with only 4% false positive rate. Another observation is that spam messages have an average packet number of nine and an average packet size of 93 bytes in a single flow while ham messages have 22 and 228 bytes respectively. These show us that spam messages have by far less size than ham messages. Ham messages have almost two and half times larger size than spam messages on average.

Variability in size of packets is also another interesting point to note. While ham messages have a very high size differences among themselves (a standard deviation of 302 bytes), spam messages are not too variable in size (a standard deviation of 91 bytes).

Our findings related to packet sizes have been studied by other researchers as well. In [47], the researchers found out that size of packets and coefficient of variability of packet sizes can be used to detect spam flows. Regarding the size and variability, their results coincide with ours. They found out that the average size of a ham message is from six to eight times larger than the average size of a spam and the coefficient of variation of the sizes of ham messages is three times higher than that of spam messages. Hao *et.al.* [51]

have also came up with similar results: spam messages tend to have small size and relatively equal sizes while ham messages have variable and larger sizes.

The fact that spam messages have small and non-variable sizes are attributed to spammers' behavior of sending. Because a given spammer will mostly send the same or similar content in all the messages using some kind of templates, it is normal to expect lower size variance in the size of spam messages than ham messages. To be effective in sending as many spams as possible, spammers also need to keep their messages small. This complies with their economics as well. Rented botnets have to be used effectively by sending as many as possible spam in as possible as little time so that they use their resources effectively. Making messages smaller in size will have an advantage for spammers, in this regard.

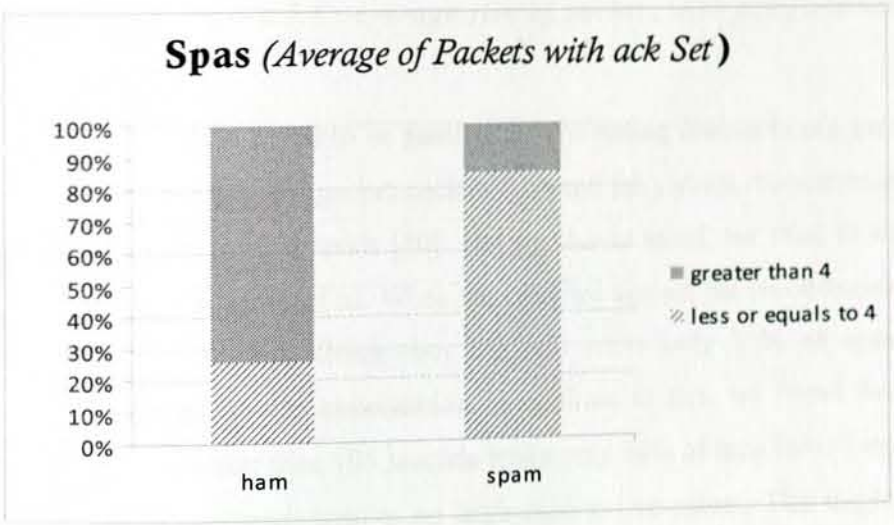


Figure 5.7: Average number of packets with ack field set

Packets with the ack (acknowledgment) flag set are found to be other good indicators. As shown in Figures 5.7 and 5.8, the majority of spam flows have no ack field set, meaning spammers do not waste their time sending acknowledgments. Only 17% spam flows have more than four packets with ack field set, while 71% of ham flows have more than four packets with ack field set. Moreover, 85% of spam flows which have ack filed set have packet sizes less than 41 bytes while only 15% ham flows which have ack field set have packet sizes less than 41.

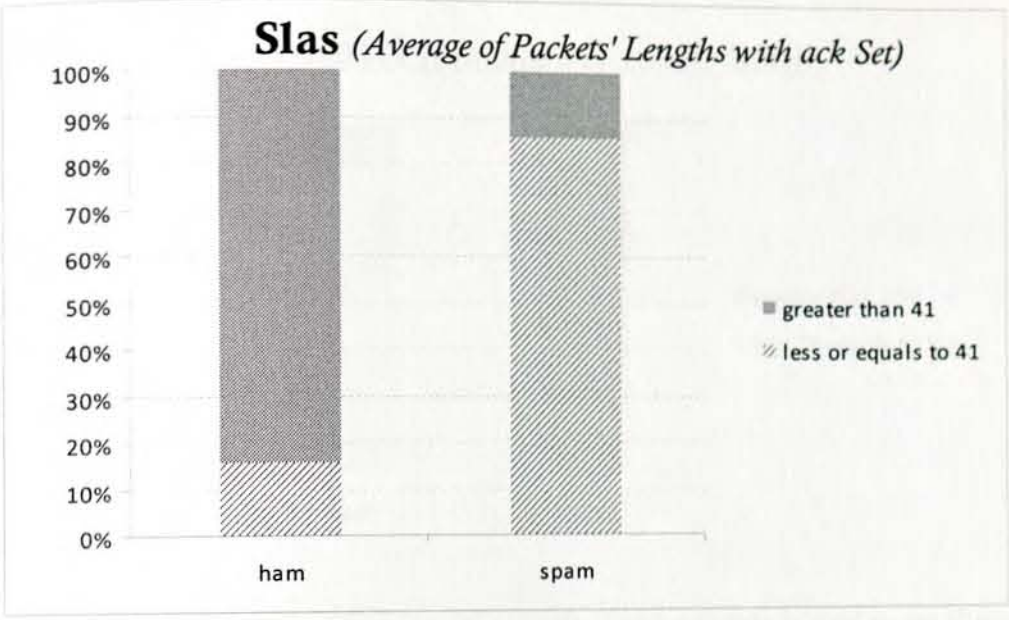


Figure 5.8: Average size of packets with ack field set

Time-to-live (ttl) is found to be another discriminating feature in our experiment. Time-to-live is used to prevent packet endless loop and the current recommended default time-to-live for TCP is 60 seconds [50]. Having this in mind, we tried to see how the two classes behave in terms of ttl. When we compare against the recommended standard (60 seconds), 50% of ham flows obey the rule while only 11% of spam flows are in accordance with the recommendation. In addition to this, we found that 83% of spam flows have ttl greater than 105 seconds while only 36% of ham flows have that rate. This shows us that spammers tend to set large time-to-live values. This might be to keep the packets alive for as long as possible so that they will not be expected to resend removed packets. This makes sense as spammers have too many spams to send and doing the resending task for many packets will be tedious and infeasible for them due to limited resources and time. Figure 5.9 shows the ttls for ham and spam flows.

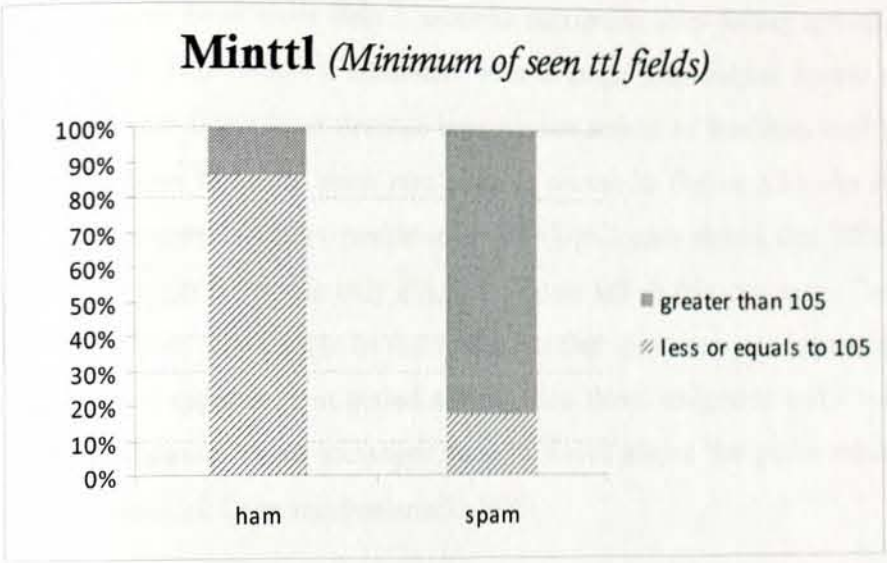


Figure 5.9: Minimum time-to-live values of ham and spam flows

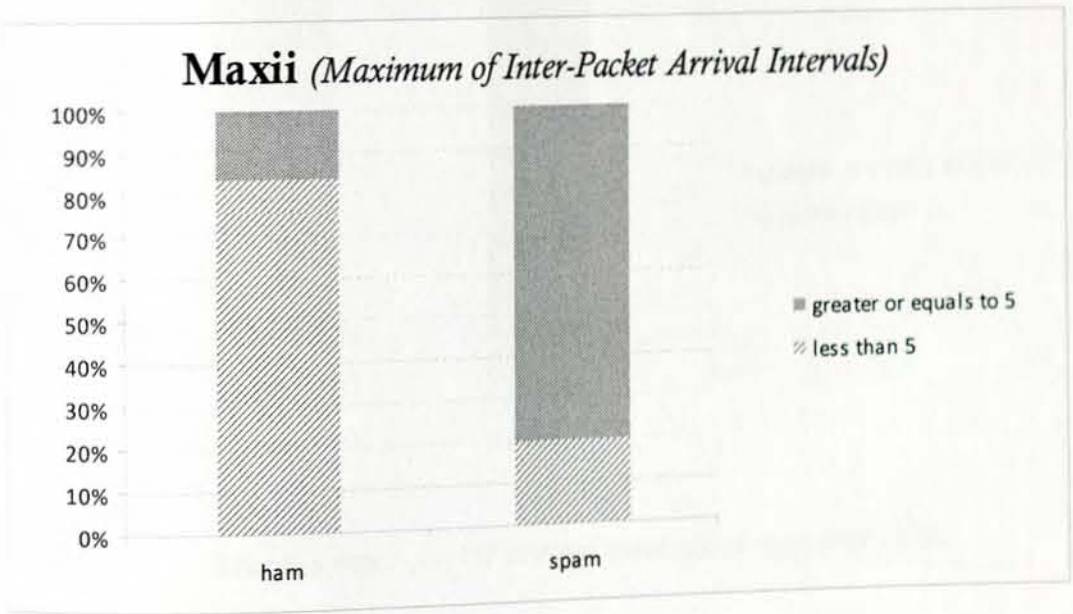


Figure 5.10: Maximum inter-packet arrival intervals of ham and spam connections

Inter-packet arrival times have also some discriminating power. To see this, we have maximum inter-packet arrival time and average inter-packet arrival time attributes. 80% of spam flows are found to have more than 5 seconds maximum inter-packet arrival times while

only 17% of ham flows have more than 5 seconds maximum inter-packet arrival time as shown in Figure 5.10. This behavior continues with average inter-packet arrival times as well. While 67% of ham flows have average inter-packet arrival of less than 0.43 seconds, only 17% of spam flows have the same rate. This is shown in Figure 5.11. As shown in Figure 5.12, sum of squares of inter packet-intervals (Sipi2) also shows that 70% of ham flows have Sipi2 less than 0.5 while only 8% spam flows fall in this category. These large inter-packet times in spam flows might be due to the fact that spammers use constrained links to send large amount of spam in short period making their flows congested and it might also be due to the long distances spam messages have to travel across the globe making their chances of facing congested links and bottlenecks high.

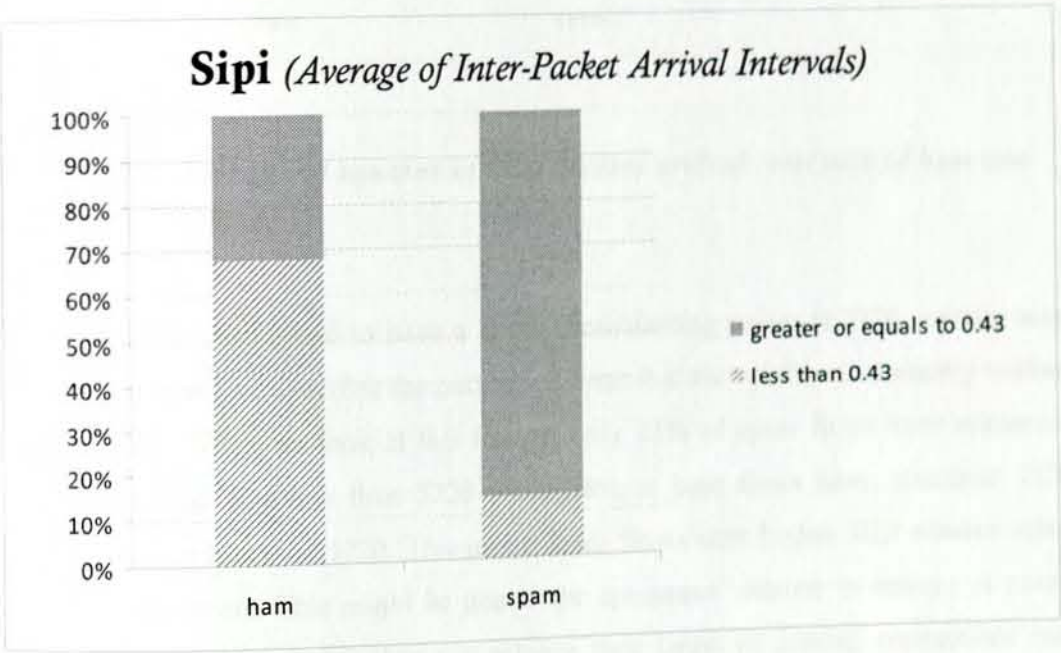


Figure 5.11: Average inter-packet arrival interval of ham and spam connections

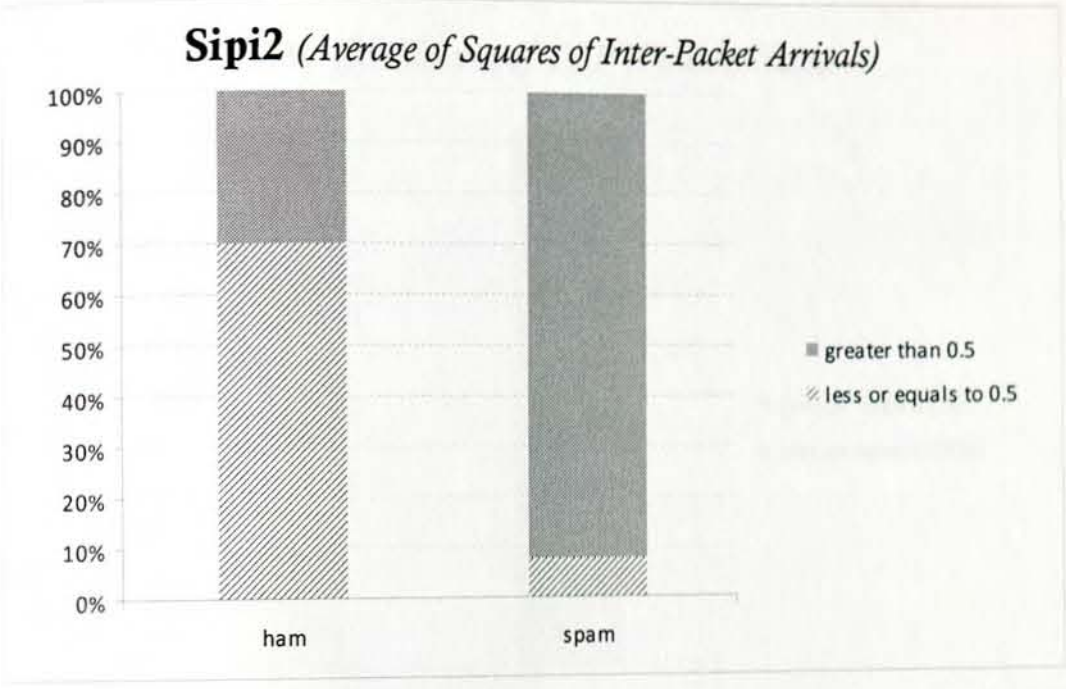
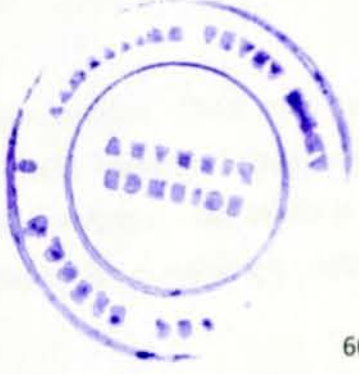


Figure 5.12: Average of squares of inter-packet arrival intervals of ham and spam connections

Another attribute found to have a good discriminating power is TCP window size. TCP window size specifies the number of bytes that the receiver is currently willing to receive. When we look at this feature, only 11% of spam flows have minimum TCP window size less than 5720 while 48% of ham flows have minimum TCP window size less than 5720. This means spam flows have higher TCP window sizes set by spammers. This might be due to the spammers' interest to receive as many bytes as possible so that they can achieve their target of making connections and transferring every data to the destinations. The percentages are shown in Figure 5.13.



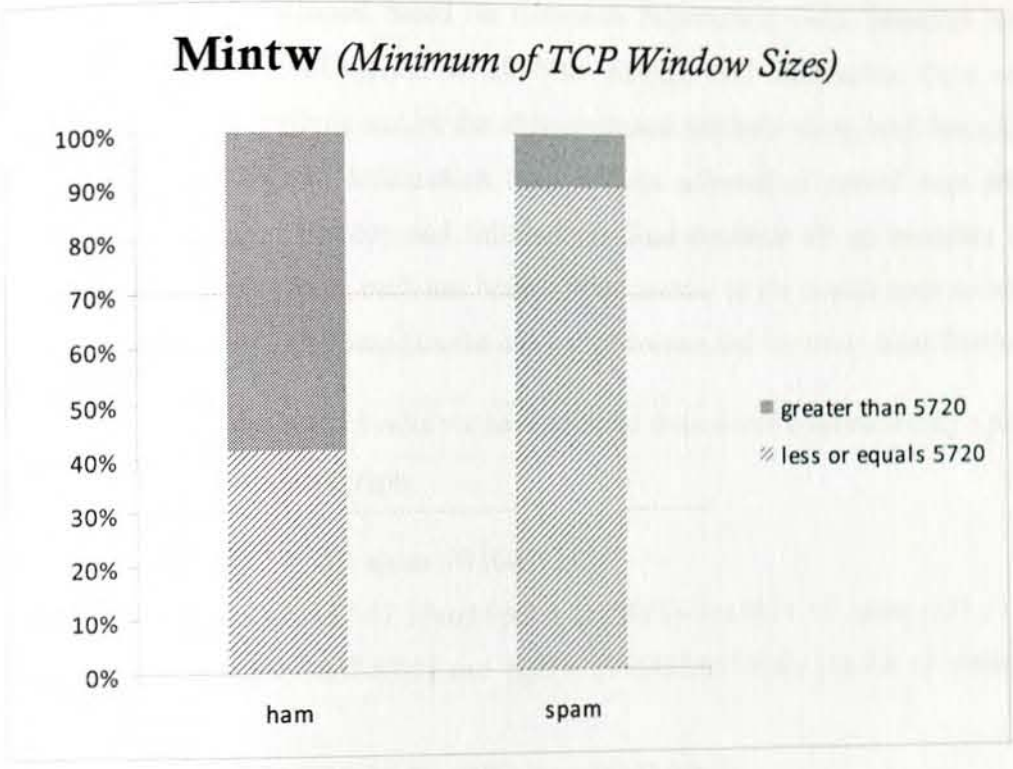


Figure 5.13: Minimum TCP window sizes of ham and spam flows

The features we discussed above are the ones that have higher discriminating capability than the other attributes. However, every attribute we have used in our experimentation has a role in detecting and scoring spam flows.

5.3 Implementation Scenario

Based on the features we have detected in the previous section from our experiments, we developed our ranking algorithm. It accepts a set of rules constructed from the attributes of our training data. Rules that can effectively detect spam flows without any false positive are the most desirable and such rules result in detecting and blocking spam messages in the network core.

We have selected 25 best performing rules for our implementation. These rules are derived from the J48 tree algorithm. The J48 algorithm works based on Entropy and Information Gain principles. The attribute that has the lowest Entropy and hence the highest Information Gain becomes the most important attribute for classification. After

the best attribute is selected based on minimum Information Gain, branches will be formed for each value of the chosen attribute. Entropy and Information Gain will be calculated for each attribute except the already-chosen attribute along each branch. The attribute with the highest Information Gain will be selected as second best and the process of calculating Entropy and Information Gain continue till all branches of all attributes are covered. Then, each tree branch from the root to the branch node constitutes a rule. We select best rules based on the number of correct and incorrect classifications.

The following are the best 25 rules we have selected from a tree constructed by a process described in the above paragraph.

1. Maxpl = '[89.5-451]': spam (31160.0/2.0)
2. Maxpl = '[85.993867-87.5]'and Spl3 = '[1316753-2315634.5]': spam (571.0)
3. Maxpl = '[85.993867-87.5]' and Spl3 = '[878412.861624-1316560.5]': spam (365.0)
4. Maxpl = '[43.706293-84.026422]': spam (4353.0/4.0)
5. Maxpl = '[1498.99018-inf]' and Sttl = '[58.97607-59.083468]' and Maxtw = '[5805.182235-6451.094067]': spam (1285.0)
6. Maxpl = '[1498.99018-inf]'and Sttl = '[56.998096-57.032621]' and Maxtw = '[33432.059258-33741.255595]': ham (59.0)
7. Maxpl = '[1498.99018-inf]'and Sttl = '[47.993357-49.025555]'and Maxii= '[1.030494-1.36728]': ham (64.0)
8. Maxpl = '[1498.99018-inf]'and Sttl = '[46.998195-47.067719]': ham (74.0)
9. Maxpl = '[1498.99018-inf]'and Sttl = '[61.017238-62.726887]': ham (439.0)
10. Maxpl = '[855.5-937.5]': spam (261.0)
11. Maxpl = '[657.5-672.645794]'and Spps = '[3.999973-4.000422]': ham (118.0/1.0)
12. Maxpl = '[1283.5-1299.5]' and Slas = '[-inf-40.009967]': spam (246.0/1.0)
13. Maxpl = '[84.993503-85.993867]'and Spl3 = '(1316753-2315634.5]':spam (220.0)
14. Maxpl = '[1235.5-1271.185413]': spam (757.0/10.0)
15. Maxpl = '[1235.5-1271.185413]' and Spps = '[6.995814-7.003832]': spam (172.0)
16. Maxpl = '[1207-1223.5]': spam (199.0/1.0)
17. Maxpl = '[1194.89529-1205.5]': spam (207.0/1.0)
18. Maxpl= '[87.5-88.5]': spam (646.0)

19. Maxpl = '[1393.508083-1441.184421]' and Spl3 = '[8569461825.5-8629898271.5]': ham (79.0)
20. Maxpl = '[88.5-89.5]' and Spl3 = '[1316753-2315634.5]': spam (523.0)
21. Maxpl = '[1300.5-1321.276703]': spam (199.0/2.0)
22. Maxpl = '[1469.204704-1471.891422]': ham (389.0/4.0)
23. Maxpl = '[1492.005232-1498.99018]': ham (116.0)
24. Maxpl = '[1112.5-1114.962746]' and Slas = '[51.97632-159.5]': ham (119.0)
25. Maxpl = '[1148-1152.598238]': ham (82.0)

These are our best rules and the numbers in each parentheses show the number of correct classifications and incorrect classifications, respectively. It is based on these numbers we calculate the percentage of confidence for each rule. A rule with no misclassification will have 100% confidence value. The numbers in the brackets show the range of values for a given attribute which gives us the classifications shown in parentheses. For instance, rule 18 states that if Maxpl has values between 87.5 bytes and 88.5 bytes, then the given flow will be spam with a confidence value of 100 because there are no misclassifications. All the 646 records satisfying this rule are spam.

Such rules will be formulated with the help of classification algorithms and will be given to the ranking algorithm as an input together with a flow statistics to be ranked. To be consistent with the spammers' behavior, these rules have to be updated frequently. There is no need to change the ranking algorithm, rather classification algorithms which assist in generating induction rules have to be tested in regular intervals and the best classifier has to be selected to generate the induction rules.

To improve the performance of the J48 algorithm, we have used a similar tree induction algorithm called ID3. It is a predecessor of the J48 algorithm with similar performance. We have experimented with the ID3 algorithm to improve its classification results. ID3 works like the J48 algorithm by selecting the best attribute based on Information Gain value. We have found out that attributes with many distinct values get higher Information Gain values and the algorithm gets biased to such attributes. So, attributes having so many distinct values will get an unfair advantage of getting selected while formulating a rule.

To reduce the amount of bias in selecting attributes, we proposed a method of artificially biasing attribute selection to those attributes with a few distinct values and known to have a good classification performance either from experience or observation. We have got a better result with a sample classification problem by introducing artificial attribute bias towards attributes with a few distinct values. Since we do not have experience how each attribute should be biased, we have not used biasing attributes. But, in the future after implementing our solution, if some attributes are found to have a good classification power, then we can assign good values to such attributes so that we can bias the classifier for good. This can be done during maintenance where changes are needed based on spammers' future behaviors.

As our solution is a flow level solution, it can be deployed on the edge router of, for example, a corporation. Such design will have a huge advantage of preventing a company's illegal bandwidth consumption in addition to saving employees' time and preventing them from attacks via spam. This design can be extended to prevent whole citizens of a given country by deploying the solution at the edge routers of the main gateways of the country. Such solutions will have notable benefits to countries like ours where bandwidth is so limited and an expensive asset. In terms of a country, the overall advantage of protection will have such a great impact. A general architecture of a company's network and the proposed position of deploying our solution are shown in Figure 5.14.

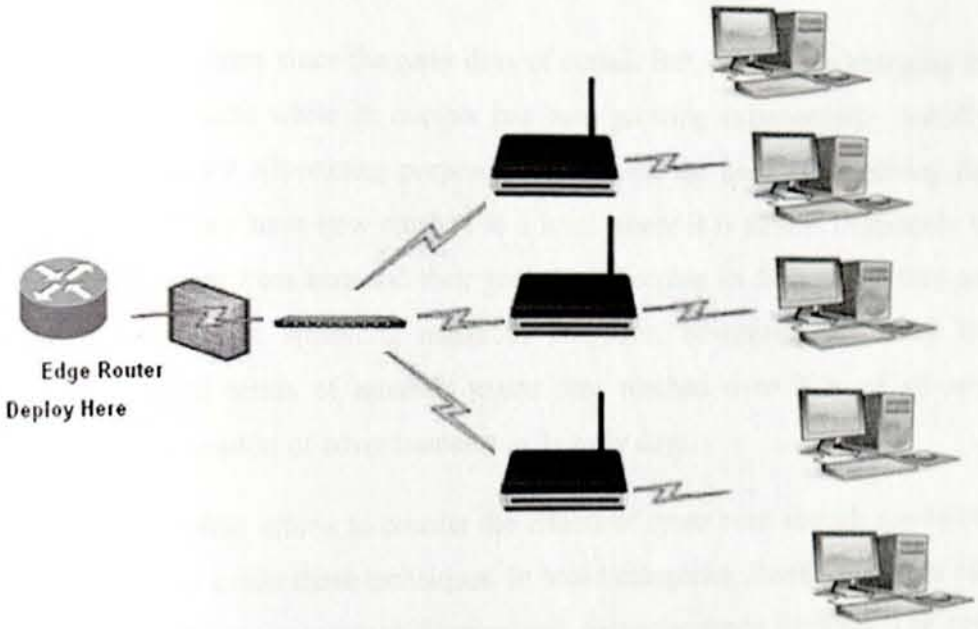


Figure 5.14: A deployment option of our system in a sample company's network

CHAPTER SIX: CONCLUSION AND RECCOMMENDATION

E-mail spam has been there since the early days of e-mail. But, it has been changing its mission, form and content while its number has been growing exponentially. Initially spams were sent just for advertizing purpose even without the need of disguising the sender information. They have now reached to a level where it is almost impossible to easily differentiate spam from ham and their goals have become so diverse that they are used for phishing attacks, spreading malicious programs, advertising and even for annoying purposes. In terms of number, spams sent reached over 90% of all sent messages from just a handful of advertisements in its early days.

There have been different efforts to counter the effects of spam even though spammers also have been able to evade those techniques. In broad categories, these techniques can be classified into two: pre-acceptance filtering and post-acceptance filtering. The pre-acceptance filtering techniques try to block spams even before they reach the recipient's mail server. Such techniques usually rely on the reputation of the sending IP addresses to whether accept or reject a given message. To do this, such techniques build reputation lists, either blacklists or whitelists, to look up when a given mail server is trying to connect. Blacklists are lists of bad IP addresses so that any connection establishment attempt from such addresses is automatically rejected. Whitelists, in contrast, are lists of good IP addresses so that any message sent from such addresses is accepted right away. Such techniques, IP reputation based, are so reliant on IP addresses and IP addresses are so dynamic that it is becoming very difficult to totally rely on such systems. These systems have a notable difficulty in hunting down all possible bad IP addresses and because of this so many IP addresses continue to send spams without being detected. It is even much difficult to change a given IP address' reputation once it is blacklisted by mistake or otherwise. The problem with whitelists is that it is difficult to communicate with new machines and if a whitelisted IP address starts sending spam it will not be easily detected early.

The post-acceptance filtering techniques try to prevent spam messages from reaching the user's inbox by filtering out and storing separately the spam messages. They usually use content filters to detect spam messages. But such techniques are being highly tricked by spammers as they use a variety of disguising methods to make spams inseparable from

legitimate e-mail messages. Such content-based filters are almost losing the war against spams. It has become very difficult to detect spams just by looking the content for some common spam words.

Considering the disadvantages of totally relying on IP address reputation and the fact that content filters are losing to spammers, we have come up with a complementary approach that neither relays on IP addresses nor needs the content to filter out spam messages. It is based on packet flow data. We have found out that by just looking into flow data we can have differentiating fingerprints between ham and spam messages. Packet data was collected and based on this data we have developed a classification model using the J48 decision tree algorithm. Using this model, from a total of 29 features we selected the best performing combination of nine attributes. The nine attributes selected together with our J48 model gave us 99.5% accuracy and a false-positive of 2.6%. This result is so impressive that we developed a ranking algorithm based on our model. The ranking algorithm takes a new record of a single flow and assigns a rank to that flow so that it can be accepted, rejected or passed for further processing. There are scores from 1 to 5. A flow with a score of 1 is a ham flow and that with a score of 2 is most probably ham. Score 1 flows are automatically accepted and score 2 flows can also be accepted as they have a high tendency of being ham. So, we accept flows with scores of 1 and 2. A flow record with a score of 5 is spam and that with a score of 4 is most probably spam. The middle category with a score of 3 will be passed for further filtering. Score 5 flows are automatically rejected but flows with score of 4 can be rejected or treated like score 3 category based on an assigned option. If **tight filtering** is turned on, score 4 flows will be rejected as that of score 5 but if **loose filtering** is turned on, the score 4 flows will be passed for further filtering by other systems.

Any message rejected will be reported to the sender as a failure of delivery so that no message sent by a legitimate party will vanish in the middle of nowhere.

We developed this system based on flow behaviors of spam and ham messages. We have noticed that there are notable differences between ham and spam messages in terms of packet size, size of flag fields, inter-packet arrival intervals, number of packet and TCP window sizes, among others. These features are good enough that we are able to filter out over 99% of spam flows with a very low false positive.

There are differences in the ways spam messages and ham messages are sent. These differences are possible due to the distinct nature of e-mail senders and their connections with e-mail recipients in each group. A ham transmission is the result of a bilateral relationship typically initiated by a human being, driven by social relationship. While spam transmission is basically a unilateral action performed by automatic tools and driven by the spammers' will to reach as many targets as possible without being detected. Flows that comprise spam traffic exhibit behaviors consistent with traffic competing for link access. Such inherent behaviors play a huge role in detecting spam based on flow characteristics.

There always is a difference between the two groups of senders (spammers and legitimate users) that our system will not be easily evaded. Even though changes are made by spammers, there is still a high probability of their flows being different enough to be detected. Otherwise, the spammers' business will not be viable and they cannot continue their business if they cannot send too much spam in a short period of time in a constrained link with messages of same or similar content. Another advantage of our system is that it takes the burden of filtering spam from mail servers down to network devices. There were spams passing by mail servers due to huge load on the servers. Now, mail servers can do their regular job of processing e-mail (whether it is storing, transmitting or receiving). Preventing illegal wastage of storage space on mail servers and preventing bandwidth consumption while transporting spam messages from mail servers to the client is also another advantage of our system.

A point worth mentioning here is that, our system needs to be maintained regularly to see changes in spammer behaviors. This can be done by collecting packet flows and testing our model using the collected data. If our current model fails to perform up to the expectation, it can be updated by giving the collected data as an input. Such maintenance steps are neither difficult nor expensive and should be done on regular intervals.

As a recommendation for future work, we believe that flow level behaviors of spam have to be seen in a wide scale setup and at a country level so that a wider view and more general properties can be selected to detect spam messages.

References

[1] http://www.expresscomputeronline.com/2008/208/market02.shtml Accessed on August 14, 2010.

[2] http://royal.pingdom.com/2010/01/22/internet-2009-in-numbers , Accessed on July 28, 2010.

[3] www.wikipedia.org/wiki/spam Accessed on July 24, 2010.

[4] www.spamhaus.org/definition.html Accessed on July 24, 2010.

[5] J. Goodman and G. Hulten, "Junk E-mail Filtering", Tutorial; KDD 2004.

[6] Symantec Enterprise Security: "State of Spam and Phishing", a monthly report April 2010.

[7] A. Sperotto, G. Vlick, R. Sadre, and A. Prag," Detecting Spam at the Network Level", EUNICE 2009, pp 28-216.

[8] A. Ramachandran and N. Feamster," Understanding the Network Level Behavior of Spammers", In Proceedings of ACM SIGCOMM '06, 2006.

[9] Threat Research and Content Engineering: Sribi now leads the spam pack http://www.marshal.com/trace/traceitem.asp?article=567,feb.2008., Accessed on October 15, 2010.

[10] K. Smith, E. Al-shaer, and K. Elbadawi, "Information Theoretic Approach for Characterizing Spam Botnets Based on Traffic Properties", In the IEEE ICC'09 proceedings, 2009.

[11] R. Clayton, "Stopping Spam by Extrusion Detection", University of Cambridge, 2004.

[12] Y. L. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten and I. Osipkov, "Spamming Botnets: Signature and Characteristics", ACM SIGCOMM'08, August 2008.

[13] H. Lin, C. Meichen, and J. Tzeng, "Flow Based Botnet Detection", In Proc. of IEEE International conference on Innovative Computing, Information and Control , 2009.

[14] C. Rossow," Anti-spam measures of European ISPs/ESPs", Institute for Internet Security, 2007.

[15] Technical paper, "How E-mail Works", Becta, 2004.

[16] The Radicati Group, "E-mail Statistics Report", 2010.

[17] Nucleus Research, "Spam: The Repeat Offender", April 2007.

[18] Wired Magazine, "The (Evil) Genius of Comment Spammers", March 2004.

[19] H. Esquivel, A. Akella, and T. Mori, "On the Effectiveness of IP Reputation for Spam Filtering", Proc. IEEE/ACM COMSNETS'10, Jan. 2010.

[20] Z. Qian, M. Mao, Y. Xie, and F. Yu, "On the Network-level Clusters for Spam Detection", In the 17th Annual Network and Distributed System Security Symposium (NDSS), 2010.

[21] A. Ramachandran, N. Feamster, and S. Vempala, "Filtering Spam with Behavioral Blacklisting", In SIGCOMM'06: Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, ACM Press, 2007.

[22] D. Erickson, M. Casado, and N. McKeown, "The Effectiveness of Whitelisting: a User-Study", In Conference on E-mail and Anti-Spam, Mountain View, California, 2008.

[23] P. Chirita, J. Diederich, and W. Nejdl, "MailRank: Global Attack-Resistant Whitelists for Spam Detection", Proc. of the 14th ACM International Conference on Information and Knowledge Management, 2005.

[24] M. Sahami, S. Dumais, D. Heckrman, and E. Horvitz, "A Bayesian Approach to Filtering Junk E-mail", AAAI Workshop on Learning for Text Categorization, Madison, Wisconsin. AAAI Technical Report, 1998.

[25] Q. Wang, Y. Guan, and X. Wang, "SVM-Based Spam Filter with Active and Online Learning", Proceedings of the Fifteenth Text Retrieval Conference, TREC 2006.

[26] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli, "Revealing Skype Traffic: When Randomness Plays With You", SIGCOMM'07, 2007.

[27] A. Moore, and D. Zuev, "Internet Traffic Classification Using Bayesian Analysis Techniques", SIGMETRICS'05, 2005.

[28] M. Zadnik, Z. Michlovsky, "Is Spam Visible in Flow-level Statistics?", CESNET Technical Report 6, 2008.

[29] http://www.pcworld.com/article/115422/im_spam_set_to_triple.html, Accessed on December 2010.

[30] http://spamassassin.apache.org Accessed on February 20, 2011.

[31] http://www.liberouter.org Accessed on March 15, 2011.

[32] http://www.dailymail.co.uk/sciencetech/article-1368436/Spam-canned-Worlds-biggest-source-junk-emails-shut-Micsoft-seizes-internet-servers-controlling-million-PCs.html Accessed on March 25, 2011.

[33] http://www.nytimes.com/2011/02/13/business/13search.html?_r=1 Accessed on February 24, 2011.

[34] http://www.ietf.org/rfc/rfc2821.txt Accessed on March 26, 2011.

[35] http://www.ietf.org/rfc/rfc1939.txt Accessed on March 26, 2011.

[36] http://www.ietf.org/rfc/rfc3501.txt Accessed on March 26, 2011.

[37] www.tcpdump.org Accessed on April13, 2011.

[38] I. Witten and E. Frank, "Data mining: Practical machine learning tools and techniques", Morgan Kaufmann series in data management systems, Morgan Kaufman Publishers, second ed., 2005.

[39] www.ethereal.com Accessed on April13, 2011.

[40] www.wireshark.org Accessed on April13, 2011.

[41] www.cs.waikato.ac.nz/ml/weka Accessed on January 15, 2011.

[42] www.python.org Accessed on May 05, 2011.

[43] Y. Yang, J.O. Pederson, "A comparative study on feature selection in text categorization", in international conference on machine learning, 1997.

[44] T. Jo, N. Topkowicz, "a multiple resampling method for learning from imbalanced datasets", computational intelligence, 2004.

[45] W. Cohen "fast effective rule induction in machine learning", proceedings of the twelfth international conference, 1995.

[46] J.R. Quilan, "C4.5: programs for machine learning", Kaufmann publishers inc., 1994.

[47] L. Gomes, C. Cazita, J. Almeida, V. Almeida, W. Meira, "Characterising a spam traffic", proceedings of the 4th ACM SIGCOMM conference on Internet measurement, 2004.

[48] R. Beverly, K. Sollins, "exploiting transport level characteristics of spam", in 5th conference on e-mail and anti-spam, 2008.

[49] http://en.wikibooks.org/wiki/Internet_Technologies/Email_security, Accessed on September 20, 2011.

[50] www.ietf.org/rfc/rfc1700.text, Accessed on May 24, 2011.

[51] S. Hao, N. Syed, N. Feamester, A. Gray, S. Krasser, "detecting spammers with SNARE: spatio-temporal network level automatic reputation engine", in 18th USENIX security symposium, 2009.

[52] E.Yu, S.Cho, "constructing response model using ensemble based on feature subset selection", in expert systems with applications, 2006.

[53] Z. Li, Q. Liao, A. Striegel, "botnet economics: uncertainty matters", in M. Johnson, managing information risk and the economics of security, 2008.

[54] http://wwww.indiastudychannel.com/resources/129782, Accessed on July 13, 2011.

Annex I

The performance of individual attributes when used alone using J48 algorithm

Maxii (Maximum of inter-packet arrival intervals)

Confusion Matrix of J48 Algorithm using Maxii only

	Classified as	
Actual	Spam	Ham
Spam	48795	138
Ham	2368	3744

Detailed Accuracy By Class using Maxii only

Class	**TP Rate**	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**ROC Area**
Spam	0.997	0.387	0.954	0.997	0.975	0.965
Ham	0.613	0.003	0.964	0.613	0.749	0.965
Weighted Avg.	0.954	0.345	0.955	0.954	0.95	0.965

Maxttl (Maximum of seen TTL fields)

Confusion Matrix using Maxttl only

	Classified as	
Actual	Spam	Ham
Spam	48516	417
Ham	3825	2287

Detailed Accuracy By Class using Maxttl only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.991	0.626	0.927	0.991	0.958	0.822
Ham	0.374	0.009	0.846	0.374	0.519	0.822
Weighted Avg.	0.923	0.557	0.918	0.923	0.909	0.822

Maxtw (Maximum of TCP Window Sizes)

Confusion Matrix using Maxtw only

	Classified as	
Actual	Spam	Ham
Spam	48669	264
Ham	4182	1930

Detailed Accuracy By Class using Maxtw only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.995	0.684	0.921	0.995	0.956	0.822
Ham	0.316	0.005	0.88	0.316	0.465	0.822
Weighted Avg.	0.919	0.609	0.916	0.919	0.902	0.822

Minii (Minimum of Inter-Packet Arrival Intervals)

Confusion Matrix using Minii only

	Classified as	
Actual	Spam	Ham
Spam	48641	292
Ham	4954	1158

Detailed Accuracy By Class using Minii only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.994	0.811	0.908	0.994	0.949	0.889
Ham	0.189	0.006	0.799	0.189	0.306	0.889
Weighted Avg.	0.905	0.721	0.895	0.905	0.877	0.889

Minpl (Minimum of Packets' Lengths)

Confusion Matrix using Minpl only

	Classified as	
Actual	Spam	Ham
Spam	48433	500
Ham	4407	1705

Detailed Accuracy By Class using Minpl only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.99	0.721	0.917	0.99	0.952	0.631
Ham	0.279	0.01	0.773	0.279	0.41	0.631
Weighted Avg.	0.911	0.642	0.901	0.911	0.892	0.631

Minttl (Minimum of seen TTL Fields)

Confusion Matrix using Minttl only

	Classified as	
Actual	Spam	Ham
Spam	48319	614
Ham	3588	2524

Detailed Accuracy By Class using Minttl only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.987	0.587	0.931	0.987	0.958	0.815
Ham	0.413	0.013	0.804	0.413	0.546	0.815
Weighted Avg.	0.924	0.523	0.917	0.924	0.913	0.815

Mintw (Minimum of TCP Window Sizes)

Confusion Matrix using Mintw only

	Classified as	
Actual	Spam	Ham
Spam	48377	556
Ham	3397	2715

Detailed Accuracy By Class using Mintw only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.989	0.556	0.934	0.989	0.961	0.895
Ham	0.444	0.011	0.83	0.444	0.579	0.895
Weighted Avg.	0.928	0.495	0.923	0.928	0.918	0.895

Sipi (Average of Inter-Packet Arrival Intervals)

Confusion Matrix using Sipi only

	Classified as	
Actual	Spam	Ham
Spam	48632	301
Ham	1851	4261

Detailed Accuracy By Class using Sipi only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.994	0.303	0.963	0.994	0.978	0.959
Ham	0.697	0.006	0.934	0.697	0.798	0.959
Weighted Avg.	0.961	0.27	0.96	0.961	0.958	0.959

Sipi2 (Average of Squares of Inter-Packet Arrival Intervals)

Confusion Matrix using Sipi2 only

	Classified as	
Actual	Spam	Ham
Spam	48682	251
Ham	1838	4274

Detailed Accuracy By Class using Sipi2 only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.995	0.301	0.964	0.995	0.979	0.971
Ham	0.699	0.005	0.945	0.699	0.804	0.971
Weighted Avg.	0.962	0.268	0.961	0.962	0.96	0.971

Sipi3 (Average of Cubes of Inter-Packet Arrival Intervals)

Confusion Matrix using Sipi3 only

	Classified as	
Actual	Spam	Ham
Spam	48615	318
Ham	2298	3814

Detailed Accuracy By Class using Sipi3 only

Class	**TP Rate**	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**ROC Area**
Spam	0.994	0.376	0.955	0.994	0.974	0.96
Ham	0.624	0.006	0.923	0.624	0.745	0.96
Weighted Avg.	0.952	0.335	0.951	0.952	0.948	0.96

Slfs (Average of Packets' Lengths with fin Set)

Confusion Matrix using Slfs only

	Classified as	
Actual	Spam	Ham
Spam	48893	40
Ham	5631	481

Detailed Accuracy By Class using Slfs only

Class	**TP Rate**	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**ROC Area**
Spam	0.999	0.921	0.897	0.999	0.945	0.655
Ham	0.079	0.001	0.923	0.079	0.145	0.655
Weighted Avg.	0.897	0.819	0.9	0.897	0.856	0.655

Slps (Average of Packets' Lengths with push Set)

Confusion Matrix using Slps only

	Classified as	
Actual	Spam	Ham
Spam	47260	1673
Ham	3263	2849

Detailed Accuracy By Class using Slps only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.966	0.534	0.935	0.966	0.95	0.922
Ham	0.466	0.034	0.63	0.466	0.536	0.922
Weighted Avg.	0.91	0.478	0.902	0.91	0.904	0.922

Slrs (Average of Packets' Lengths with reset Set)

Confusion Matrix using Slrs only

	Classified as	
Actual	Spam	Ham
Spam	48827	106
Ham	4695	1417

Detailed Accuracy By Class using Slrs only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.998	0.768	0.912	0.998	0.953	0.69
Ham	0.232	0.002	0.93	0.232	0.371	0.69
Weighted Avg.	0.913	0.683	0.914	0.913	0.889	0.69

Slss (Average of Packets' Lengths with syn Set)

Confusion Matrix using Slss only

	Classified as	
Actual	Spam	Ham
Spam	46759	2174
Ham	3004	3108

Detailed Accuracy By Class using Slss only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.956	0.491	0.94	0.956	0.948	0.789
Ham	0.509	0.044	0.588	0.509	0.546	0.789
Weighted Avg.	0.906	0.442	0.901	0.906	0.903	0.789

Sp (Average Number of Packets)

Confusion Matrix using Sp only

	Classified as	
Actual	Spam	Ham
Spam	47517	1416
Ham	3115	2997

Detailed Accuracy By Class using Sp only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.971	0.51	0.938	0.971	0.954	0.92
Ham	0.49	0.029	0.679	0.49	0.57	0.92
Weighted Avg.	0.918	0.456	0.91	0.918	0.912	0.92

Spas (Average of Packets with ack Set)

Confusion Matrix using Spas only

	Classified as	
Actual	Spam	Ham
Spam	47920	1013
Ham	3671	2441

Detailed Accuracy By Class using Spas only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.979	0.601	0.929	0.979	0.953	0.835
Ham	0.399	0.021	0.707	0.399	0.51	0.835
Weighted Avg.	0.915	0.536	0.904	0.915	0.904	0.835

Spfs (Average of Packets with fin Set)

Confusion Matrix using Spfs only

	Classified as	
Actual	Spam	Ham
Spam	48933	0
Ham	5864	248

Detailed Accuracy By Class using Spfs only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	1	0.959	0.893	1	0.943	0.587
Ham	0.041	0	1	0.041	0.078	0.587
Weighted Avg.	0.893	0.853	0.905	0.893	0.847	0.587

Spl (Average of Packet Lengths)

Confusion Matrix using Spl only

	Classified as	
Actual	Spam	Ham
Spam	47031	1902
Ham	1728	4384

Detailed Accuracy By Class using Spl only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.961	0.283	0.965	0.961	0.963	0.962
Ham	0.717	0.039	0.697	0.717	0.707	0.962
Weighted Avg.	0.934	0.256	0.935	0.934	0.934	0.962

Spps (Average of Packets with push Set)

Confusion Matrix using Spps only

	Classified as	
Actual	Spam	Ham
Spam	48600	333
Ham	4059	2053

Detailed Accuracy By Class using Spps only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.993	0.664	0.923	0.993	0.957	0.933
Ham	0.336	0.007	0.86	0.336	0.483	0.933
Weighted Avg.	0.92	0.591	0.916	0.92	0.904	0.933

Sprs (Average of Packets with reset Set)

Confusion Matrix using Sprs only

	Classified as	
Actual	Spam	Ham
Spam	48864	69
Ham	5478	634

Detailed Accuracy By Class using Sprs only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.999	0.896	0.899	0.999	0.946	0.612
Ham	0.104	0.001	0.902	0.104	0.186	0.612
Weighted Avg.	0.899	0.797	0.899	0.899	0.862	0.612

Spss (Average of Packets with syn Set)

Confusion Matrix using Spss only

	Classified as	
Actual	Spam	Ham
Spam	48933	0
Ham	6094	18

Detailed Accuracy By Class using Spss

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	1	0.997	0.889	1	0.941	0.507
Ham	0.003	0	1	0.003	0.006	0.507
Weighted Avg.	0.889	0.886	0.902	0.889	0.838	0.507

Sttl (Average of Seen ttl Fields)

Confusion Matrix using Sttl only

	Classified as	
Actual	Spam	Ham
Spam	47685	1248
Ham	3143	2969

Detailed Accuracy By Class using Sttl only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.974	0.514	0.938	0.974	0.956	0.804
Ham	0.486	0.026	0.704	0.486	0.575	0.804
Weighted Avg.	0.92	0.46	0.912	0.92	0.914	0.804

Sttl2 (Average of Squares of Seen ttl Fields)

Confusion Matrix using Sttl2 only

	Classified as	
Actual	Spam	Ham
Spam	48880	53
Ham	5643	469

Detailed Accuracy By Class using Sttl2 only

Class	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
Spam	0.999	0.923	0.897	0.999	0.945	0.561
Ham	0.077	0.001	0.898	0.077	0.141	0.561
Weighted Avg.	0.897	0.821	0.897	0.897	0.856	0.561

Sztwp (Average Number of Packets with TCP Window Size of Zero)

Confusion Matrix using Sztwp only

	Classified as	
Actual	Spam	Ham
Spam	48897	36
Ham	5660	452

Detailed Accuracy By Class using Sztwp only

Class	**TP Rate**	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**ROC Area**
Spam	0.999	0.926	0.896	0.999	0.945	0.617
Ham	0.074	0.001	0.926	0.074	0.137	0.617
Weighted Avg.	0.897	0.823	0.9	0.897	0.855	0.617

Annex II

Source code for the Ranking Algorithm

```
import string
import math

def inductioncbr():
  minii=raw_input("Enter minii")
  maxii=raw_input("Enter maxii")
  sipi=raw_input("Enter sipi")
  sipi2=raw_input("Enter sipi2")
  sipi3=raw_input("Enter sipi3")
  sp=raw_input("Enter sp")
  spl=raw_input("Enter spl")
  spl2=raw_input("Enter spl2")
  spl3=raw_input("Enter spl3")
  minpl=raw_input("Enter minpl")
  maxpl=raw_input("Enter maxpl")
  spss=raw_input("Enter spss")
  slss=raw_input("Enter slss")
  spas=raw_input("Enter spas")
  slas=raw_input("Enter slas")
  spfs=raw_input("Enter spfs")
  slfs=raw_input("Enter slfs")
  sprs=raw_input("Enter sprs")
  slrs=raw_input("Enter slrs")
  spps=raw_input("Enter spps")
  slps=raw_input("Enter slps")
  sttl=raw_input("Enter sttl")
```

```
sttl2=raw_input("Enter sttl2")
minttl=raw_input("Enter minttl")
maxttl=raw_input("Enter maxttl")
stw=raw_input("Enter stw")
mintw=raw_input("Enter mintw")
maxtw=raw_input("Enter maxtw")
sztwp=raw_input("Enter sztwp")


infile=open("inductionrule.txt",'r')
ruleline=infile.readlines()
infile.close()
nr=len(ruleline)
indexrule=0
solved="false"
while (indexrule<nr and solved=="false"):
  items=ruleline[indexrule]
  item=[]
  items=items.rstrip('\n')

  for val in items.split(','):
    item.append(val)
  size=len(item)-1
  clas=item[size-1]
  percentage=item[size]

  index=0
  flag=1
  #print size,item[0],item[3],item[6],item[9],item[12],item[15],item[18]
```

```
while (index<size and flag==1):
  #print item[index]
  if item[index]=="minii":
    if item[index+1]=="<=":
     if not float(minii)<=float(item[index+2]):
       flag=0
    elif item[index+1]=="<":
     if not float(minii)<float(item[index+2]):
       flag=0
    elif item[index+1]==">=":
     if not float(minii)>=float(item[index+2]):
       flag=0
    elif item[index+1]==">":
     if not float(minii)>float(item[index+2]):
       flag=0
    elif item[index+1]=="=":
     if not float(minii)==float(item[index+2]):
       flag=0
  elif item[index]=="maxii":
    if item[index+1]=="<=":
     if not float(maxii)<=float(item[index+2]):
       flag=0
    elif item[index+1]=="<":
     if not float(maxii)<float(item[index+2]):
       flag=0
    elif item[index+1]==">=":
     if not float(maxii)>=float(item[index+2]):
       flag=0
    elif item[index+1]==">":
```

```
    if not float(maxii)>float(item[index+2]):
      flag=0
   elif item[index+1]=="=":
    if not maxii==float(item[index+2]):
      flag=0
elif item[index]=="sipi":
  if item[index+1]=="<=":
   if not float(sipi)<=float(item[index+2]):
     flag=0
  elif item[index+1]=="<":
   if not float(sipi)<float(item[index+2]):
     flag=0
  elif item[index+1]==">=":
   if not float(sipi)>=float(item[index+2]):
     flag=0
  elif item[index+1]==">":
   if not float(sipi)>float(item[index+2]):
     flag=0
  elif item[index+1]=="=":
   if not float(sipi)==float(item[index+2]):
     flag=0
elif item[index]=="sipi2":
  if item[index+1]=="<=":
   if not float(sipi2)<=float(item[index+2]):
     flag=0
  elif item[index+1]=="<":
   if not float(sipi2)<float(item[index+2]):
     flag=0
  elif item[index+1]==">=":
```

```
    if not float(sipi2)>=float(item[index+2]):
      flag=0
   elif item[index+1]==">":
    if not float(sipi2)>float(item[index+2]):
      flag=0
   elif item[index+1]=="=":
    if not float(sipi2)==float(item[index+2]):
      flag=0
elif item[index]=="sipi3":
  if item[index+1]=="<=":
   if not float(sipi3)<=float(item[index+2]):
     flag=0
  elif item[index+1]=="<":
   if not float(sipi3)<float(item[index+2]):
     flag=0
  elif item[index+1]==">=":
   if not float(sipi3)>=float(item[index+2]):
     flag=0
  elif item[index+1]==">":
   if not float(sipi3)>float(item[index+2]):
     flag=0
  elif item[index+1]=="=":
   if not float(sipi3)==float(item[index+2]):
     flag=0
 elif item[index]=="sp":
  if item[index+1]=="<=":
   if not float(sp)<=float(item[index+2]):
     flag=0
  elif item[index+1]=="<":
```

```
    if not float(sp)<float(item[index+2]):
      flag=0
   elif item[index+1]==">=":
    if not float(sp)>=float(item[index+2]):
      flag=0
   elif item[index+1]==">":
    if not float(sp)>float(item[index+2]):
      flag=0
   elif item[index+1]=="=":
    if not float(sp)==float(item[index+2]):
      flag=0
  elif item[index]=="spl":
   if item[index+1]=="<=":
    if not float(spl)<=float(item[index+2]):
      flag=0
   elif item[index+1]=="<":
    if not float(spl)<float(item[index+2]):
      flag=0
   elif item[index+1]==">=":
    if not float(spl)>=float(item[index+2]):
      flag=0
   elif item[index+1]==">":
    if not float(spl)>float(item[index+2]):
      flag=0
   elif item[index+1]=="=":
    if not float(spl)==float(item[index+2]):
      flag=0
  elif item[index]=="spl2":
   if item[index+1]=="<=":
```

```
        #print item[index+2]
        if not float(spl2)<=float(item[index+2]):
          #print "wrong float(spl2)"
          flag=0
      elif item[index+1]=="<":
       if not spl2<float(item[index+2]):
         flag=0
      elif item[index+1]==">=":
       if not float(spl2)>=float(item[index+2]):
         flag=0
      elif item[index+1]==">":
       if not float(spl2)>float(item[index+2]):
         flag=0
      elif item[index+1]=="=":
       if not float(spl2)==float(item[index+2]):
         flag=0
    elif item[index]=="spl3":
     if item[index+1]=="<=":
      if not float(spl3)<=float(item[index+2]):
        flag=0
     elif item[index+1]=="<":
      if not float(spl3)<float(item[index+2]):
        flag=0
     elif item[index+1]==">=":
      if not float(spl3)>=float(item[index+2]):
        flag=0
     elif item[index+1]==">":
      if not float(spl3)>float(item[index+2]):
        flag=0
```

```
    elif item[index+1]=="=":
      if not float(spl3)==float(item[index+2]):
        flag=0
  elif item[index]=="minpl":
    if item[index+1]=="<=":
      if not float(minpl)<=float(item[index+2]):
        flag=0
    elif item[index+1]=="<":
      if not float(minpl)<float(item[index+2]):
        flag=0
    elif item[index+1]==">=":
      if not float(minpl)>=float(item[index+2]):
        flag=0
    elif item[index+1]==">":
      if not float(minpl)>float(item[index+2]):
        flag=0
    elif item[index+1]=="=":
      if not float(minpl)==float(item[index+2]):
        flag=0
  elif item[index]=="maxpl":
    if item[index+1]=="<=":
      if not float(maxpl)<=float(item[index+2]):
        flag=0
    elif item[index+1]=="<":
      if not float(maxpl)<float(item[index+2]):
        flag=0
    elif item[index+1]==">=":
      if not float(maxpl)>=float(item[index+2]):
        flag=0
```

```
  elif item[index+1]==">":
    if not float(maxpl)>float(item[index+2]):
      flag=0
  elif item[index+1]=="=":
    if not float(maxpl)==float(item[index+2]):
      flag=0
elif item[index]=="spss":
  if item[index+1]=="<=":
    if not float(spss)<=float(item[index+2]):
      flag=0
  elif item[index+1]=="<":
    if not float(spss)<float(item[index+2]):
      flag=0
  elif item[index+1]==">=":
    if not float(spss)>=float(item[index+2]):
      flag=0
  elif item[index+1]==">":
    #print "right spss"
    if not float(spss)>float(item[index+2]):
      flag=0
  elif item[index+1]=="=":
    if not float(spss)==float(item[index+2]):
      flag=0
elif item[index]=="slss":
  if item[index+1]=="<=":
    if not float(slss)<=float(item[index+2]):
      flag=0
  elif item[index+1]=="<":
    if not float(slss)<float(item[index+2]):
```

```
        flag=0
    elif item[index+1]==">=":
      if not float(slss)>=float(item[index+2]):
        flag=0
    elif item[index+1]==">":
      if not float(slss)>float(item[index+2]):
        flag=0
    elif item[index+1]=="=":
      if not float(slss)==float(item[index+2]):
        flag=0
  elif item[index]=="spas":
    if item[index+1]=="<=":
      #print "right spas"
      if not float(spas)<=float(item[index+2]):
        flag=0
    elif item[index+1]=="<":
      if not float(spas)<float(item[index+2]):
        flag=0
    elif item[index+1]==">=":
      if not float(spas)>=float(item[index+2]):
        flag=0
    elif item[index+1]==">":
      if not float(spas)>float(item[index+2]):
        flag=0
    elif item[index+1]=="=":
      if not float(spas)==float(item[index+2]):
        flag=0
  elif item[index]=="slas":
    if item[index+1]=="<=":
```

```
    #print "right slas"
    if not float(slas)<=float(item[index+2]):
      flag=0
  elif item[index+1]=="<":
    if not float(slas)<float(item[index+2]):
      flag=0
  elif item[index+1]==">=":
    if not float(slas)>=float(item[index+2]):
      flag=0
  elif item[index+1]==">":
    if not float(slas)>float(item[index+2]):
      flag=0
  elif item[index+1]=="=":
    if not float(slas)==float(item[index+2]):
      flag=0
elif item[index]=="spfs":
  if item[index+1]=="<=":
    if not float(spfs)<=float(item[index+2]):
      flag=0
  elif item[index+1]=="<":
    if not float(spfs)<float(item[index+2]):
      flag=0
  elif item[index+1]==">=":
    if not float(spfs)>=float(item[index+2]):
      flag=0
  elif item[index+1]==">":
    if not float(spfs)>float(item[index+2]):
      flag=0
  elif item[index+1]=="=":
```

```
    if not float(spfs)==float(item[index+2]):
      flag=0
  elif item[index]=="slfs":
   if item[index+1]=="<=":
    if not float(slfs)<=float(item[index+2]):
      flag=0
   elif item[index+1]=="<":
    if not float(slfs)<float(item[index+2]):
      flag=0
   elif item[index+1]==">=":
    if not float(slfs)>=float(item[index+2]):
      flag=0
   elif item[index+1]==">":
    if not float(slfs)>float(item[index+2]):
      flag=0
   elif item[index+1]=="=":
    if not float(slfs)==float(item[index+2]):
      flag=0
  elif item[index]=="sprs":
   if item[index+1]=="<=":
    if not float(sprs)<=float(item[index+2]):
      flag=0
   elif item[index+1]=="<":
    if not float(sprs)<float(item[index+2]):
      flag=0
   elif item[index+1]==">=":
    if not float(sprs)>=float(item[index+2]):
      flag=0
   elif item[index+1]==">":
```

```
    if not float(sprs)>float(item[index+2]):
      flag=0
  elif item[index+1]=="=":
    if not float(sprs)==float(item[index+2]):
      flag=0
elif item[index]=="slrs":
  if item[index+1]=="<=":
    if not float(slrs)<=float(item[index+2]):
      flag=0
  elif item[index+1]=="<":
    if not float(slrs)<float(item[index+2]):
      flag=0
  elif item[index+1]==">=":
    if not float(slrs)>=float(item[index+2]):
      flag=0
  elif item[index+1]==">":
    if not float(slrs)>float(item[index+2]):
      flag=0
  elif item[index+1]=="=":
    if not float(slrs)==float(item[index+2]):
      flag=0
elif item[index]=="spps":
  if item[index+1]=="<=":
    #print "right spps"
    if not float(spps)<=float(item[index+2]):
      flag=0
  elif item[index+1]=="<":
    if not float(spps)<float(item[index+2]):
      flag=0
```

```
  elif item[index+1]==">=":
    if not float(spps)>=float(item[index+2]):
      flag=0
  elif item[index+1]==">":
    if not float(spps)>float(item[index+2]):
      flag=0
  elif item[index+1]=="=":
    if not float(spps)==float(item[index+2]):
      flag=0
elif item[index]=="slps":
  if item[index+1]=="<=":
    #print "right slps"
    if not float(slps)<=float(item[index+2]):
      flag=0
  elif item[index+1]=="<":
    if not float(slps)<float(item[index+2]):
      flag=0
  elif item[index+1]==">=":
    if not float(slps)>=float(item[index+2]):
      flag=0
  elif item[index+1]==">":
    if not float(slps)>float(item[index+2]):
      flag=0
  elif item[index+1]=="=":
    if not float(slps)==float(item[index+2]):
      flag=0
elif item[index]=="sttl":
  if item[index+1]=="<=":
    if not float(sttl)<=float(item[index+2]):
```

```
            flag=0
        elif item[index+1]=="<":
            if not float(sttl)<float(item[index+2]):
                flag=0
        elif item[index+1]==">=":
            if not float(sttl)>=float(item[index+2]):
                flag=0
        elif item[index+1]==">":
            if not float(sttl)>float(item[index+2]):
                flag=0
        elif item[index+1]=="=":
            if not float(sttl)==float(item[index+2]):
                flag=0
    elif item[index]=="sttl2":
        if item[index+1]=="<=":
            if not float(sttl2)<=float(item[index+2]):
                flag=0
        elif item[index+1]=="<":
            if not float(sttl2)<float(item[index+2]):
                flag=0
        elif item[index+1]==">=":
            if not float(sttl2)>=float(item[index+2]):
                flag=0
        elif item[index+1]==">":
            if not float(sttl2)>float(item[index+2]):
                flag=0
        elif item[index+1]=="=":
            if not float(sttl2)==float(item[index+2]):
                flag=0
```

```
elif item[index]=="minttl":
  if item[index+1]=="<=":
    if not float(minttl)<=float(item[index+2]):
      flag=0
  elif item[index+1]=="<":
    if not float(minttl)<float(item[index+2]):
      flag=0
  elif item[index+1]==">=":
    if not float(minttl)>=float(item[index+2]):
      flag=0
  elif item[index+1]==">":
    if not float(minttl)>float(item[index+2]):
      flag=0
  elif item[index+1]=="=":
    if not float(minttl)==float(item[index+2]):
      flag=0
elif item[index]=="maxttl":
  if item[index+1]=="<=":
    if not float(maxttl)<=float(item[index+2]):
      flag=0
  elif item[index+1]=="<":
    if not float(maxttl)<float(item[index+2]):
      flag=0
  elif item[index+1]==">=":
    if not float(maxttl)>=float(item[index+2]):
      flag=0
  elif item[index+1]==">":
    if not float(maxttl)>float(item[index+2]):
      flag=0
```

```
    elif item[index+1]=="=":
     if not float(maxttl)==float(item[index+2]):
       flag=0
  elif item[index]=="stw":
    if item[index+1]=="<=":
     if not float(stw)<=float(item[index+2]):
       flag=0
    elif item[index+1]=="<":
     if not float(stw)<float(item[index+2]):
       flag=0
    elif item[index+1]==">=":
     if not float(stw)>=float(item[index+2]):
       flag=0
    elif item[index+1]==">":
     if not float(stw)>float(item[index+2]):
       flag=0
    elif item[index+1]=="=":
     if not float(stw)==float(item[index+2]):
       flag=0
  elif item[index]=="mintw":
    if item[index+1]=="<=":
     if not float(mintw)<=float(item[index+2]):
       flag=0
    elif item[index+1]=="<":
     if not float(mintw)<float(item[index+2]):
       flag=0
    elif item[index+1]==">=":
     if not float(mintw)>=float(item[index+2]):
       flag=0
```

```
    elif item[index+1]==">":
      if not float(mintw)>float(item[index+2]):
        flag=0
    elif item[index+1]=="=":
      if not float(mintw)==float(item[index+2]):
        flag=0
  elif item[index]=="maxtw":
    if item[index+1]=="<=":
      if not float(maxtw)<=float(item[index+2]):
        flag=0
    elif item[index+1]=="<":
      if not float(maxtw)<float(item[index+2]):
        flag=0
    elif item[index+1]==">=":
      if not float(maxtw)>=float(item[index+2]):
        flag=0
    elif item[index+1]==">":
      if not float(maxtw)>float(item[index+2]):
        flag=0
    elif item[index+1]=="=":
      if not float(maxtw)==float(item[index+2]):
        flag=0
  elif item[index]=="sztwp":
    if item[index+1]=="<=":
      if not float(sztwp)<=float(item[index+2]):
        flag=0
    elif item[index+1]=="<":
      if not float(sztwp)<float(item[index+2]):
        flag=0
```

```
        elif item[index+1]==">=":
          if not float(sztwp)>=float(item[index+2]):
            flag=0
        elif item[index+1]==">":
          if not float(sztwp)>float(item[index+2]):
            flag=0
        elif item[index+1]=="=":
          if not float(sztwp)==float(item[index+2]):
            flag=0
      index=index+3


    if flag==1:
      #print "soln",item
      solved="True"
    indexrule=indexrule+1
  if solved=="True":
    action=0;
    if(clas=="dnsbl"):
      if(int(percentage)==100):
        action=5
      elif(int(percentage)<100 and int(percentage)>=95):
        action=4
      elif(int(percentage)<95):
        action=3
    elif(clas=="nspam"):
      if(int(percentage)==100):
        action=1
      elif(int(percentage)<100 and percentage>=95):
        action=2
```

```
        elif(int(percentage)<95):
            action=3
        print "The solution is ",clas,"Score",action,percentage,"%"

if __name__ == "__main__":
    inductioncbr()
```

Declaration

I, the undersigned, declare that this thesis is my original work and has not been presented for a degree in any other university, and that all sources of materials used for the thesis have been duly acknowledged.

Declared by:

Name: Zelalem Hailu

Signature: [signature]

Date: Nov 18th 2011

Confirmed by advisor:

Name: Mulugeta Libsie

Signature: [signature]

Date: 18/11/11