A Peer-to-Peer Architecture for Collaborative Spam Filtering

by

Laurent Cottereau

A thesis submitted to
the University of Dublin
for the degree of

Master of Science in Computer Science

Department of Computer Science,
University of Dublin, Trinity College

September, 2002

Declaration

Signature of Author ....

Laurent Cottereau
16 September, 2002

Acknowledgements

I would like to thank my supervisor, Mads Haahr, for all the time, help and support he gave me throughout the project. I would also like to thank my family for their constant love and support in making this year possible for me. And finally, I would like to thank the *usual suspects* among the MScNDS students, whose friendship made this year such an enjoyment.

Abstract

Over the last twenty years, the importance of electronic-mail has risen to a point where it is now a critical medium for many companies and individuals. This fact has led to the emergence of a new method of invasive marketing: *Unsolicited Commercial Email*, otherwise known as spam. Spam is costing email users an increasing amount of time and money.

It is the act of blindly mass-mailing a message that makes it spam, and not merely its content, making it harder to spot automatically. Because of this, the need for a flexible, precise, cheap and easy-to-use spam filter has yet to be met. A recent idea that has proved promising for accurate detection of spam is the collaborative sharing of knowledge about spam between users.

This thesis describes a distributed peer-to-peer architecture that allows users to store and share data about spam. This design provides a trust management model that offers both precise filtering mechanisms and protection from malevolent users. One possible implementation of this architecture is also proposed, and used for deterministic evaluation of the design.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Spamming is the scourge of electronic-mail and newsgroups on the Internet. It can seriously interfere with the operation of public services, to say nothing of the effect it may have on any individual's email system. ... Spammers are, in effect, taking resources away from users and service suppliers without compensation and without authorization. [aol97]

— V. Cerf, acknowledged as a 'father of the Internet'

1.1 Motivation

Electronic-mail has become more and more popular over the last ten years. It is now a major communication tool for companies as well as for individuals, and almost everybody has an e-mail address, and can be reached through this medium. This simple fact has led to a new trend that has become a great annoyance lately: Unsolicited Commercial Email (UCE or Spam[1] or often 'junk e-mail').

The Coalition Against Unsolicited Commercial Email is a community that has been created to advocate a legislative solution to the problem of UCE. The need for such an organization comes from the fact that it is still

[1]The term was originally defined as any annoying 'noise', which drowns out all other communication, so named for an old Monty Python skit.

Form	**Cost to Sender ($)**	**Cost to Recipient ($)**	**Cost Borne by Sender (%)**
Legal			
Telemarketing	1.00	0.10	91.00
Postal mail	0.75	0.10	88.00
Illegal			
Fax	0.03	0.10	23.00
Automated phone	0.07	0.10	41.00
Uncertain legality			
Spam	0.00001	0.10	0.01

All cost figures per contact are estimated.

Table 1.1: Cost Comparison of Unsolicited Marketing Methods

legal to send spam in most countries and states. This organization describes six reasons why spam is such an issue.

Cost Shifting Sending spam is very cheap. The figures from Table 1.1 have been gathered by SpamCon founder Tom Geller, and presented in [Gel01]. However, because a spammer can send his messages very cheaply, everyone receiving the spam must help pay the costs of it.

The ISP which processed the mail has used part of its mail servers' processing power to process the mail, storage resources to keep the message, and finally bandwidth to transport it. The ISP must pay for these resources, and either has to offer the paying customer a reduced service (e.g. less mail storage, or a lower bandwidth), or has to shift the costs of these to the user (by raising its rates).

Fraud Because the spammers know that the overwhelming majority (95% according to [cau]) of their targets do not want to receive their spam, they are using tricks to get the user to open the message, hoping that, as marketing has many time shown before, once the message is opened, the user might get interested.

In order to bypass possible filters, they often hide the origin of the message by relaying it off the mail server of a an innocent third party, which doubles the damage because the third party has to process these mails, as well as deal with the resulting complaints.

Waste of Others' Resources Because of the way mail works, a message is handled by different carriers on its way to its destination. The carriers in between are bearing the burden of carrying advertisements for the spammer.

Ronald Coarse, an economist, won the Nobel Prize in economics by tackling such situation (see [War91]). His conclusion was that it was very dangerous for the free market when an inefficient business (e.g. one that cannot bear the costs of its own activities) distributes its costs across a greater and greater numbers of victims. In economic terms, this leads to disaster.

Displacement of Normal Email In the same sense as fax was once made virtually unusable because any critical fax was disappearing in the middle of a flood of junk faxes, spams are getting so numerous that they threaten to drown the relevant, solicited e-mails.

Annoyance Factor The number of spam in almost everyone's mailbox, the lower standard of service due to spam, and finally the fact that it is simply mails you never wanted to receive, but are unable to refuse; all these reasons make the user frustrated and annoyed at receiving spam.

Ethics Spam is based on theft of service, fraud, deceit and cost-shift. Furthermore, the majority of products and services advertised by spam is generally of dubious legality.

The spam industry is getting to a point where it is close to overloading some famous ISP who have a lot of clients, like Hotmail or AOL. Table 1.2 summarizes figures from [cau] and [cnn], which give an overview of how critical the issue of spam has become for ISPs.

ISP	AOL	Hotmail
Number of messages processed per day	30 million	1.25 billion
Percentage of spam	30%	80%

all figures are approximate

Table 1.2: Amount of Spam Processed by ISPs

Nonetheless, email has become so important for everyone, that it is not considered possible to just stop using it. For this reason, in the last years, an important number of software has been designed to try and filter spam in a cheap and fast way. However, all of them have had the same problem dealing with the fact that it is not the content of spam that defines a message as spam, but the act of blindly mass mailing it.

The current mail protocols make it difficult to take this fact into account, since much of the information available to the filters can be forged by the spammers. One recent idea is to gather information about the mail that is being filtered from outside sources. The main mean of doing this is for mail servers and users to *collaborate*, in such a way that they can declare a message to be spam by trying to guess if it has been blindly mass-mailed. This thesis describes an architecture that allows such a collaboration between users in order to filter spam. This system will be called the *anti-spam network*.

1.2 Objectives

This section describes the different objectives for the collaborative spam filter. They are based on the preliminary idea that a collaborative spam filter, as with any collaborative action or system, will only really work if a lot of people are participating.

1.2.1 Scalability

If a lot of people are to be participating in the anti-spam network, then the architecture behind it should be scalable. Indeed, the service should not start breaking down as soon as the number of users increases.

1.2.2 Portability

There are two aspects to portability. First, the system should be open, so that everyone can participate in its own way (e.g. with its own piece of software). Therefore, the design should be as generic and extensible as possible. Secondly, since many different types of users should be allowed to participate, the architecture of the system should allow for heterogeneous hardware to collaborate between each other.

1.2.3 Ease of Use

Because this system should not be restrained to the most knowledgeable of users, and because a majority of the current users of electronic-mail do not understand the way e-mail works, the system should be easy to install, configure and maintain. In a more general way, it should be user-friendly. For example, for a normal user who wants to participate, it should not be necessary to convince the administrator from his ISP (whether personal or business) that the service is interesting. The decision to participate in the network should be the user's only.

1.2.4 Reliability

Another aspect to consider is that users should not be done any harm because of the system. Two main possible causes of harm have been identified.

False positives is a term often used in spam filtering systems to designate a proper and relevant message that the user really wants to receive in his mailbox (i.e. not a spam) but that has been marked and processed by the spam filter as spam. False positives are the plague of spam filtering system, because they are a worse solution than the problem for a lot of users, who care less about receiving junk e-mails than missing an important e-mail. Some users with critical e-mail will need to be given very high guarantees before they can agree to use a spam filter. As a result, if the anti-spam network is to gather as many users as possible, the false positives rate should be low.

The second issue with reliability is the resistance to attacks. Indeed, the decision to collaborate in the anti-spam network should not put the user's system in a position of weakness against electronic attacks.

1.2.5 Other Objectives

Because there are a few collaborative spam filters available already to analyze, an emphasis will be put on trying and improving the current state of the art of collaborative spam filtering.

As outlined in section 1.1, there are two main issues for a user receiving spam, the first and less obvious one is the indirect cost of this spam to himself. The second and more obvious one is the direct annoyance and loss of time. Because the goal is to convince as many users as possible to participate in the network, and because the annoyance is much more obvious to the user, if a choice must be made between these two impacts of spam, then the emphasis will be put on the later. In other words, if, in order to prevent more loss of time to the user, the cost-fighting must be sacrificed, then it will be done.

1.3 Document Structure

This section outlines the structure of this thesis. There are seven chapters in total, the order and contents of which are listed below.

Chapter 1: Introduction describes the motivation behind the project, and lists the objectives.

Chapter 2: Background introduces many important concepts and technology related to electronic-mail and spam filtering.

Chapter 3: State of the Art takes a look at related projects in the area of collaborative spam filtering systems.

Chapter 4: Design analyzes the issues at hand and discusses solutions to them. In this chapter, the generic design of the anti-spam network is also presented.

Chapter 5: Implementation describes a solution that was implemented within the scope of the project, and concludes by presenting other possible implementations.

Chapter 6: Evaluation describes a metrics for the evaluation of the anti-spam network, then evaluates the different aspects of the current implementation.

Chapter 7: Conclusion concludes the project.

Chapter 2

Background

This chapter introduces several concepts relevant to electronic-mail in general and spam fighting in particular. Then, the different popular approaches to filtering spam are outlined.

2.1 The Electronic Mail Protocols

There are many types of electronic mail systems in the world today. Most major computer vendors, such as IBM or Microsoft, offer a proprietary email system. The International Standards Organization (ISO) has established an international standard for electronic mail systems, called X.400. Because the ISO was taking a long time to develop X.400, various vendors implemented their own email systems based on its formats and protocols, believing they could migrate to standard compliance if need be. These systems were interconnected using a set of specifications defined in Requests for Comments (RFCs).

The very fast adoption of Internet by the general public made this interconnected way of messaging a *de facto* standard, although it is less integrated and complete than X.400. Still today, the main mean of sending and receiving emails is not X.400, despite the fact that it is endorsed by the ISO. Because of the objectives set forward for the anti-spam network in section 1.2, the email system described in the specifications [Pos82] and

[Cro82] will be the main concern of the project.

2.1.1 Terminology

As described in [Woo99], the Internet mail system can be easily conceptualized by introducing the key elements:

Mail user agent (MUA) is a client program used by a user to send or receive email. An MUA could also be a program or script that emulates the behavior of a typical MUA by sending or receiving email.

Mail transfer agent (MTA) is a server daemon that transfers email from one machine on the Internet to another, or mail server.

Mail delivery agent (MDA) is a small program used by an MTA to write a message into a user's mailbox.

Mail retrieval agent (MRA) is a service that retrieves messages from a mailbox on a remote server to a user's MUA.

2.1.2 Message Transport

MUAs send mail to MTAs using the Internet's Simple Mail Transfer Protocol (SMTP), and MTAs also communicate with each other using SMTP. For this reason, SMTP will be very important in the design and implementation of a collaborative spam filtering system. [Pos82] describes the SMTP protocol.

One machine acts as a client (the sender) and the other machine as a server (the receiver). The SMTP protocol describes the commands sent by the client to the server and the responses returned by the server to the client. Client commands include such things as identifying the intended recipient and the sending party of the mail message, which are similar to the To and Return addresses found on the envelope of traditional "snail mail."

There is a lack of authentication in SMTP that makes it very easy for anybody (a spammer for example) to forge many of the characteristics of a message. It must always be kept in mind that it is impossible to be sure where or who a mail is from. This is important when designing a spam filtering software, because the spammers will use these techniques to lure the user into reading the message, and it is often hard to automatically recognize that some parts of a message are fake.

2.1.3 Message Format

Another specification that will be important for the creation and implementation of a collaborative spam filtering system, [Cro82], describes the actual format and content of the message headers. These message headers include the `Date`, `From`, `To`, `Subject`, `Message-Id` and `Received` fields. These `Received` lines list each machine that processed and forwarded the message. They added, after the message is sent and one line at a time, by each MTA that processes the message on its way to its destination.

Although the headers can be forged in much the same way as seen in section 2.1.2, there is something that can be trusted in a message header. Since the spammer has no further way of forging a message that was already sent, all the lines added by the systems processing the message can be considered authentic. However this does not mean that the spammer did not add some Received lines of his own before sending the message; it only applies to the most recent Received lines. Of course, the problem that arises here is to be able to automatically distinguish between real and forged `Received` lines.

Distinguishing can be difficult and there are therefore very few spam filters which rely on these `Received` lines to identify a mail as spam. However, it may prove interesting to keep in mind for spam filtering that all the different instances of the same bulk email will all have at least one similar `Received` line in the headers: the line corresponding to the mail server from where the messages were sent. This is true even if the spammer used someone else's mail server to send his spam (see section 2.2.2).

2.2 Fighting the Spam

At the mail server administrator level, a conscious choice of policy has to be made regarding the filtering of spam going through the mail server. The administrator's reaction to his users receiving spam, to someone masquerading his IP to suggest it came from his server, or to someone using his server to send spam will affect the time and energy he can invest in implementing protection system, educating his users, or tracking down abusers.

The administrator has a choice of different approaches for fighting spam arriving in his users mailboxes. First, he can identify the spams and block them before they reach the users' mailboxes (the mechanisms for doing this are discussed in this section). Secondly, he can provide tools for his users, without actively getting involved. Finally, he can use a mix of these two methods; for example, he might want to filter the most obvious spam at the system level, and also provide tools for his users, so that they can have a more precise and customized filtering at their level. He might also want to maintain a filter that the user can choose to enable or disable (American On Line does this for example).

2.2.1 Blocking Incoming Spam

Policies

There are different strategies available for dealing with messages that have been recognized as being spam by your spam filtering system.

Dropping the message makes the whole process of spam filtering transparent to the end-user. However, there is the issue of false-positive (see section 1.2.4). When a message detected as spam is dropped, it is important to keep in mind that the message dropped might actually be a legitimate one. This solution is to be adopted only if the number of critical mail is very low.

Saving the message without delivering it to end-user allows the process of spam filtering to remain transparent to the end-user. However in case of a conflict, messages can always be recovered for users who detect that they should have received a message and did not get it. However, this results in a poor quality of service, since a user usually does not want to need to know he is supposed to receive a mail, or have to check for all of them. However this system allows for the adoption of much less strict spam filtering tools, as you never totally loose a message just because of a system error.

Marking the mail as spam and letting the end-user deal with it can be achieved by several techniques. The most common are proprietary like (such is the case of Microsoft Exchange and Microsoft Outlook), but you can also simply modify the header of the message, adding headers to the message at the MTA level, that state that the spam has been recognized as spam by the server, and then let the user decide what to do with that. This means implementing a client that will be able to filter messages depending on their headers. This might cause a problem to the system administrator, especially if the network is not homogeneous (for example, an important ISP). Although most main e-mail clients nowadays have means to do that, they do not work the same way, and it would be difficult to maintain a database of how to take into account the information put into the header at the MTA, depending on your system. Carolyn Duffy Marsan, in [Mar02], explains that Sieve[1], a proposed standard for a mail filtering language, is a possible solution to this issue.

The most widely used policy nowadays is apparently the latter, because it provides with the better quality of service, although the spam filtering is

[1][Sho01] describes Sieve as a language for filtering e-mail messages at time of final delivery. It is designed to be implementable on either a mail client or mail server. It is meant to be extensible, simple and independent of access protocol, mail architecture, and operating system. It is suitable for running on a mail server where users may not be allowed to execute arbitrary programs, such as on black box Internet Message Access Protocol servers, as it has no variables, loops, or ability to shell out to external programs.

not transparent to the user anymore. However, as much as possible, spam filtering software should give the user (or the administrator) the opportunity to choose between any of those policies.

At the Mail Transport Agent

There are several places where incoming spam can be blocked. The most obvious one is the Mail Transport Agent (MTA) because the MTA has a complete copy of the message, and hence, has access to all the relevant information that could help decide if the message is spam.

At the Router Level

Another approach to filtering incoming spam is to place the filtering directly at the router level. The obvious and major interest of blocking the spam at the router level comes when the router is the exterior router, because this router represents a bottleneck for data. This means that if some filtering system is implemented directly on the router, there will not be any need for anything else anywhere else on the network, and the network is protected from spam.

However, at the router level, all the information from the packets cannot necessarily be accessed. The main information available concerns the destination port and address, as well as the sender port and address. From these the fact that it is an e-mail can be deduced, and then, filtering can be achieved depending only on these informations. This is why only the type of filtering presented in section 2.3 can be performed at the router.

2.2.2 Stopping Outgoing Spam

One of the main way for a spammer to be able to send a lot of mail without having its account canceled by its ISP is to use somebody else's mail server to send its bulk email. For historical reasons, relaying[2] was usually allowed

[2]Relaying is the action of transmitting mail between MTAs which do not relate to the mail domain of either the sender of the recipient of the mail.

by default. For example, Sendmail 10 (see [Cos97]) is the first version of Sendmail that forbids it by default. This means that anybody can connect to port 25 of your mail server and send a message to any email address without authentication. Of course, the spammers use this to their benefit. Therefore, one of an important mean of stopping outgoing spam is to deny relaying on the mail server.

2.2.3 Fight the Spammer

There is also a significant amount of software available on the Internet that are trying to fight the spam by actually fighting the spammer as an individual physical person. They all rely on an assumption that anybody who is trying to come up with a design for spam filtering has to make: the spammers will not stop mass-mailing bulk email until it becomes prohibitively expensive to do so. Taking this into account, different softwares will use different tactics:

Vipul's Ricochet, described in [ric], is a tool that gives the user all the information required in order to protest to the administrator of the email server that sent the email. The idea here is that if the spammers are not able to get easy access to mass-mailing services, they will not be able to spam anymore. This is not really realistic, and the tool is actually used more for getting information in order to create filtering rules or edit filtering lists.

SpamBait (see [spac]) and CgiSpy Spam (see [cgi]) are quite interesting because, in a way, they are a collaborative action against spamming. The idea is to host a dynamic web page displaying a lot of random email addresses (that is an address of the form xxx@yyy.zzz) where xxx, yyy and zzz are random strings. Since the spammer's main source of addresses is harvesting the Internet with robots, these addresses pollute their databases of addresses, forcing them to send much more mails in order to reach as many people.

2.3 List-based Filtering

Maybe the simplest way of filtering spam is to try and identify which mail servers spam are coming from, and deny any access to your mail server to messages coming from these servers. Because of its simplicity, this system is very widely used nowadays and several such lists (called blacklists) of spammer sites and spam relay sites are being maintained up-to-date. One of the best known such effort is the Mail Abuse Protection System (MAPS) Realtime Blackhole List (RBL) presented in [map]. As an example of its popularity, it can be said that since version 8.9, Sendmail features direct support of the MAPS RBL. Apart from the MAPS RBL, there are a few other lists that are kept up-to-date and quite complete: among these we can mention the Distributed Sender Boycott List and the Open Relay Database, which list insecure servers by testing if relaying is allowed on them (see section 2.2.2).

Although this mean of filtering or blocking spam is widely used, its draconian approach makes it unappropriate. A platform like Yahoo! Mail is often used by spammer because it is simple and anonymous to create an account. For this reason, the domain yahoo.com is in the MAPS RBL. However, if a mail server administrator decides to block messages according to the MAPS RBL, the user will not be able to receive any mail from anybody using the Yahoo! Mail platform, including friends who are obviously not trying to spam them.

This limitation has been addressed by adding the concept of whitelist. It is a list of people that the user has specified to be trustworthy. When a message arrives at the server, it will be processed only if it is in the whitelist *or* it is not in the blacklist. Although this is more flexible than blacklists, it is still not really practically usable. The issue lies in the difficulty for a user to identify all the persons from which he might receive mail. Except for very particular cases, it is nearly impossible to come up with an exhaustive list of such emails.

To solve this simple problem, a widely used solution is to have some kind of confirmation system that will automatically add active senders (which includes friends and no spammer) to the user's whitelist. For example, SpamBar, described in [spab], will automatically send a confirmation message to anybody who is trying to mail the user and is not yet on his whitelist. If the sender replies to this confirmation message, he will be added to the whitelist. This will work because spammers rely on heavy automation to send their email, and will not read incoming mails in the accounts they use to send messages (plus, the account might be already canceled by the time you are trying to send the confirmation message). The same strategy is used by the Tagged Message Delivery Agent (TMDA), described in [tmd].

Although these list-based means of filtering spam is not really relevant to collaborative spam filtering in itself, these methods are interesting for a couple of other reasons; in a way, these systems represent the state of the art of trust management applied specifically to deciding if an email address is a spammer or a legitimate user. So far, no system has been designed that is as easy and as efficient in that regard. This is the reason why lists are used in almost all the systems which will be discussed in this chapter. The whitelists especially are often used as a first protection against errors (for example, to make sure that a mail from the CEO of the company is not mistakenly seen as spam and deleted.)

Furthermore, these systems are, however different they may be from collaborative filtering, also spam filtering systems, and thus are making some assumptions regarding spam that are quite interesting and relevant to collaborative spam filtering. For example, the following assumptions regarding spam are described in [tmd]:

- An email address cannot be kept secret from spammers.
- Content-based filters cannot distinguish spam from legitimate mail with sufficient accuracy.
- To maintain economies of scale, bulk-mailing is generally:
  - An impersonal process where the recipient is not distinguished.

    – A one-way communication channel (from spammer to victim).

- Spam will not cease until it becomes prohibitively expensive for spammers to operate.

As much as possible, these assumptions should be taken into account when designing a spam filtering system.

2.4 Rule-based Filtering

As explained in section 1.1, one major problem of spam is that it is non-trivial to automatically detect that a particular message is an unsolicited bulk e-mail. This is the reason why most systems end up using white lists and black lists, which are not really adequate, as argued in section 2.3. Nonetheless, there have been attempts at solving the problem of automatic detection of spam content.

At first glance, this problem seems theoretically quite impossible, since there are no distinct difference between a spam and a normal message; you have the same structure of the message (based on [Cro82]), the same encoding is used for characters, nothing is really different. However, it is obvious that it does not take a human a long time to recognize a message as spam, without knowing that this message has been mass-mailed. So there has to be some kind of fingerprint that differentiates a spam from a normal message.

A number of spam filtering systems have taken this as granted and have tried to represent these marks as rules, thus filtering emails that would have such marks. A good example of such piece of software is Blackhole, described in [bla]. This software uses white lists and black lists (on hosts and email addresses), but also can check the header, subject and body of the message for custom rules (basically regular string expression matches against patterns). It also has some features like blocking depending on the type of character encoding. However, this is difficult to configure, since it is still hard to come up with the rules you want to use to filter spam and not create any false positive. This is even made harder by the extreme simplicity of the language for rules. The problem of this piece of software is it tries to do too many tasks at the same time (it filters spam, provide virus checking, white lists, logging functionalities, etc...) and thus can do none of these as efficiently as if it was focusing on spam filtering. In conclusion, this system is only efficient in very particular situations and only for users who are very knowledgeable of spammers techniques and are willing to spend a lot of time configuring their spam filter.

Another system using the same kind of technique to filter spam is called SpamAssassin, and is described in [spaa]. It currently seems to be the most widely used and is one of the most accomplished free spam filters, because it seems to be very efficient, filtering most of the spam coming in a mail server, uses less processing power than many other systems, is fast and is quite easy to configure. SpamAssassin has been designed so that is would be easy to extend or customize, by adding or modifying rules. The spam identification tactics include these four following types of filtering:

Header Analysis this is the same kind of analysis as the one featured by Blackhole.

Text Analysis this is trying to spot spam according to content.

Blacklists this is the same kind of filtering as seen in section 2.3.

Razor and DCC (see chapter 3) this is quite original. SpamAssassin is using the outputs of collaborative spam filters to filter spam. We will discuss these systems in details in later paragraphs.

Table 2.1 presents an extract from the quite impressive list of tests performed on incoming mail. The goal of these tests is to come up with a spam score, and tag the message with it (by adding a specific header). The actual filtering is left to the user, so that he has a liberty of action. This is especially interesting for ISPs who want to provide their users with a spam filtering tool, but do not want to force any specific action, or take any responsibility in what happens to messages that have been identified as spam.

Area Tested	**Description of test**
body	Tells you how to stop further
body	Claims to be legitimate email
body	I wonder how many emails they sent in error...
body	Suspect you might have received the message by mistake
body	Claims not to be SPAM
body	Claims SPAM helps the environment
body	Plugs Viagra
body	"You have been selected as a finalist", sure
body	There's no such thing as a free shipping
body	Offers a limited time offer
body	Tells you about a strong buy
header	Received via a relay in orbs.dorkslayers.com
header	DNSBL: sender is Confirmed Spam Source
header	DNSBL: sender is a Spamware site or vendor
header	Received via RBLed relay, see http://www.mail-abuse.org/rbl/
uri	Stainless Steel Network Spam
uri	URL of CGI script called "unsubscribe" or "remove"
uri	URL of page called "unsubscribe"
rawbody	JavaScript code
rawbody	Auto-executing JavaScript code
rawbody	JavaScript code which can easily be executed

Table 2.1: Extract from the tests performed by SpamAssassin

The idea of being able to easily add, modify and delete rules is important, because it provides a solution to two main problems of fighting spam. First, it can never be assumed that the perfect way of detecting spam has been discovered: the spammer will always find a way around it (this is especially true in open source environment, since the spammer has easy access to what the software is looking for in a message). Secondly, it provides the user (or here, the administrator) with a way to adapt the system to its special environment, thus making it more flexible. This refers to the objective of portability set in section 1.2.2.

However, the fact that SpamAssassin is also using the spam signatures from two collaborative spam filters (information on those is available in chapter 3) still shows that these rules cannot be used by themselves for the only reason that it is impossible to come up with rules so strict that they will not create false positives, or flexible enough that they will filter all spam without creating false positives.

2.5 Artificial Intelligence

As has been seen above, it is usually easy for a human being to detect spam, though he cannot always say exactly why such or such message is spam. This is why there has been some research done is the area of spam filtering through artificial intelligence. It may be surprising that text categorization can be effective in anti-spam filtering, because, unlike other text categorization tasks, it is the act of blindly mass-mailing a message that makes it spam, not its actual content. Still, it seems that the language of spam constitutes a distinctive genre, and that spam messages are often about topics rarely mentioned in legitimate messages, making it possible to train a text classifier for anti-spam filtering.

In particular, it has been argued that a Naive Bayesian classifier can be used to filter spam. Such a system would use some machine learning algorithm applied to text categorization. These algorithms learn to classify documents into fixed categories, based on their content, after being trained on manually categorized documents. In 1998, Sahami et al, in [SDHH98], trained a Naive Bayesian classifier on manually categorized legitimate and spam messages, reporting impressive precision and recall on unseen messages.

However, in 2000, Androutsopoulos conducted, in [AKC$^+$00], a cost-sensitive evaluation of naive bayesian anti-spam filtering, focusing on this classifier. Their conclusions were that these types of systems were positive only when the cost of filtering a legitimate message was not too high; they are not viable if a message classified as spam is automatically deleted or in an environment where speed of mail delivery is critical (these issues have been already mentioned in section 2.2.1).

Chapter 3

State of the Art

Although all of these methods are achieving a certain level of success (especially systems using several of these methods at the same time), there is yet a way that has only received recent attention; collaborative spam filtering. Instead of focusing on the content of a message, a collaborative spam filter tries to gather information about the message using collaboration throughout the network. This information will allow the system to decide if the message is blindly-sent bulk email.

This chapter discusses in detail several different collaborative spam filtering systems. The main aim of this discussion is to find what can and needs to be improved so that the anti-spam network is as effective as possible.

3.1 Kolathur's and Subramanian's

Satheesh Kolathur and Subha Subramanian, in [KS], have discussed such a collaborative spam filtering system. Their system relies on a threshold number of users to mark an email as spam. The threshold value is defined as the number of users in the system who are required to mark an email as spam so that it is registered as such in the system. Once an email is marked as spam, the server halts the delivery of any matching email to other users in the system. The filter plugs the email delivery system in between the Message Transfer Agent and the Local Delivery Agent. Figure 3.1 is an

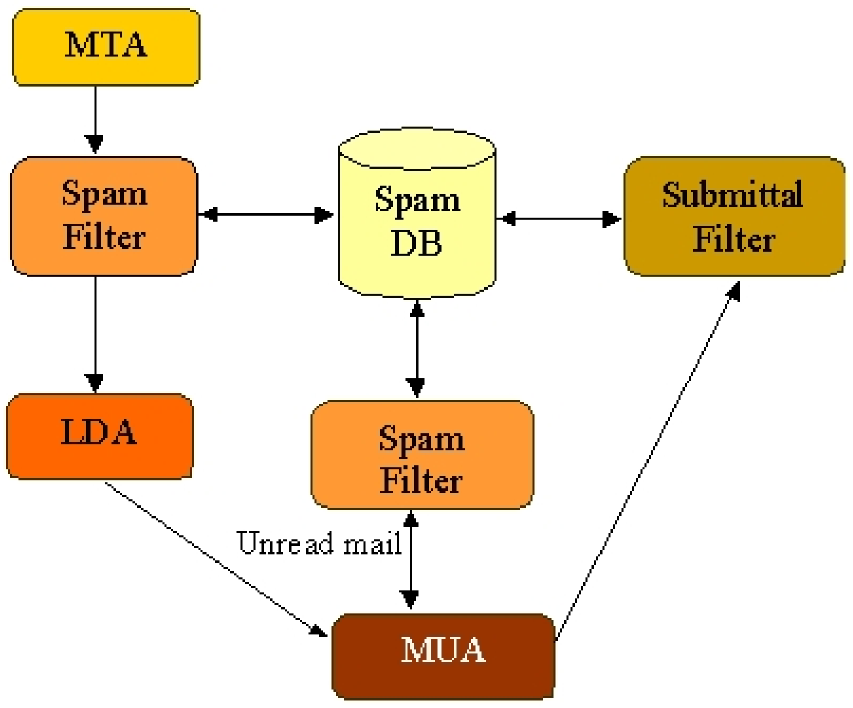


Figure 3.1: Overview of a Collaborative Spam Filter

overview of the way their filter works:

Since the spam filter plugs in between the MTA and LDA, it picks up the incoming emails (from the mqueue folder) and checks for a matching entry or a similar entry in the spam database. If it finds a matching entry that is classified as spam for the system, a spam factor is assigned to the email and placed in the filtered mqueue directory which is a new folder created for the purpose of this filter. Only then does the LDA process the mail and appends it to the relevant user mail file. The second pass of the filter which is triggered by the MUA ensures that any mail marked as spam between the time the mail reached the users mail folder and the time it was read by the user is marked as spam. This mechanism is referred to as real-time filtering. Figure 3.2 presents the filtering process.

Another important function is the learning process; this is the process

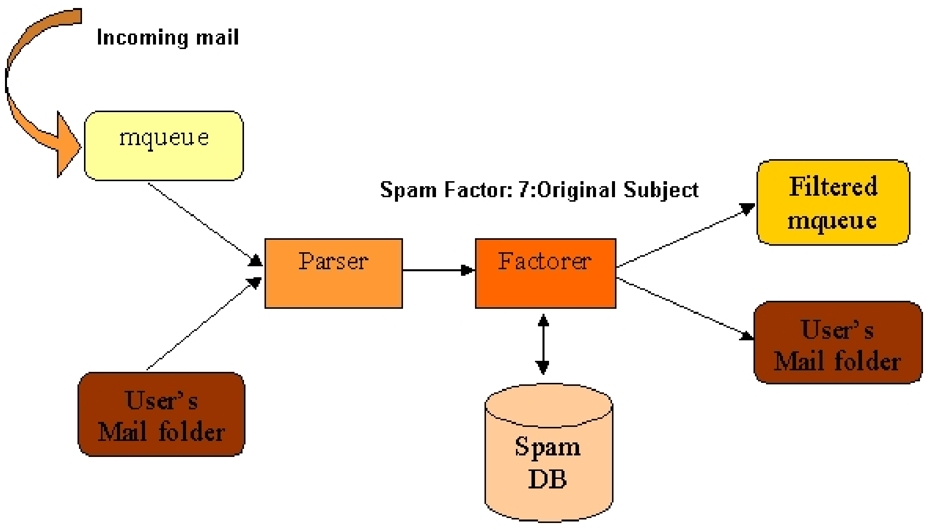


Figure 3.2: The Filtering Process

of registering of spam by the users of the system. The user who recognises a mail as spam will forward it to a pre-defined adress. There, the email goes through a submittal filter. This filter checks if this forwarded mail is from a legitimate user. It then checks for an entry in the spam database that matches the properties of this email. If there is a match, the count on that spam entry is incremented and checked against the threshold value. If the threshold value is reached, it is classified as spam. The submittal filter also prevents the users from marking valid and important emails as spam. This could be an email from the system administrator or the CEO of the organization. Figure 3.3 shows the learning process.

The advantage with this arrangement is that minimum changes need to be made to the existing scheme of things since the filter plugs into the current system (for e.g. between MTA and LDA). The user only needs to decide if the email received is spam and mark it as one, and the rest of the process would be virtually transparent. Though a few users would initially get the spam, it would prevent the rest of the users from receiving the same.

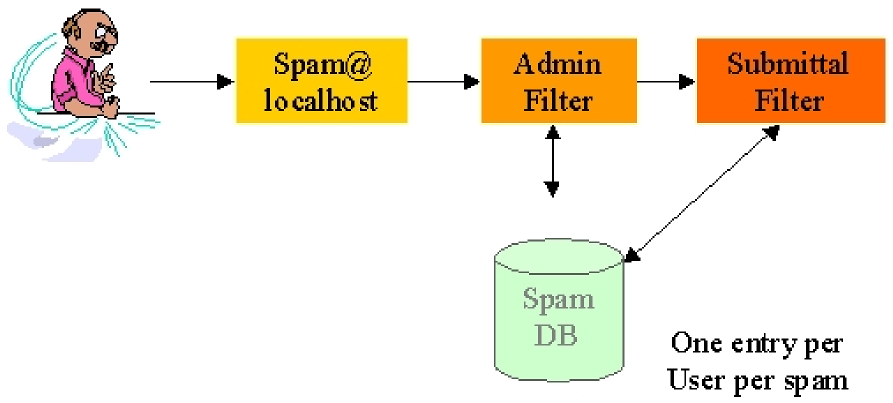


Figure 3.3: The Learning Process

Because this system has apparently not been actually implemented (only the design is presented), there are no ways to get any idea of its effectiveness in fighting spam, although they claim that this effectiveness would grow (maybe exponentially) with the number of users.

3.2 Vipul's Razor

Razor is defined by Vipul Ved Prakash as as

> distributed, collaborative, spam detection and filtering network. [Pra]

Razor maintains a catalogue of spam. Clients can use that catalogue to automatically filter out known spam. Spam is reported using the Razor Reporting Agent. For scalability, only a 20-character unique identifier (a SHA Digest, see [NIS02]) is sent to the closest Razor Catalogue Server, which then echoes this signature to other trusted servers. The actual filtering (or denying) of the messages is done by Razor Filtering Agents which check incoming mail (at the user end or at a MTA) for a signature match against a Catalogue Server. As in section 3.1, the idea is that once a message is catalogued as spam, all the Filtering Agents on the network can filter or block this spam.

This system works especially well in conjunction with other spam-detection systems; for example, SpamAssassin, discussed in section 2.4, in addition to rule-based filtering and blacklists, uses Razor's Filtering Agent, in order to benefit from the collaborative network set by Vipul's Razor.

3.2.1 Razor-check

Razor-check is the component mostly used by system administrator and can be used without taking part in the collaborative reporting of spam. Razor-check will check a mail against the distributed Razor Catalogue. It is usually invoked before the mail is delivered or processed by a human (thus transparently preventing the user from every seeing the spam message). The process is usually launched against every incoming mail by mail processors (like procmail, see [pro01]), or MTAs (like sendmail, see [Cos97]). The result of calling razor check is binary: either the signature already exists in the database, which means that the message is a spam and can be discarded, or it is not and razor will consider it as non-spam.

3.2.2 Razor-report

Razor-report is the Razor Reporting Agent. This is the process used to report spam message to a Razor Catalogue Server.

There are several ways that the razor-report agent can be used, mainly depending on the user's mail client. For example, if it is possible to use key bindings (for example, mutt, described in [Elk], a customizable shell email client written by Michael Elkins allows this), a particular key can be set which, when pressed, will pipe the current message through razor-report.

If a simpler mail client like pine is used, there are still ways to pipe the message through razor-report. A tip described in [raz] consists of setting the razor-report as a particular printer (instead of a printing command line, you can set the razor-report process to be launched). Then, in order to report a message as spam, all that is required is to print the message, and select the appropriate message.

Finally, the other main way to report a spam is to set what is called a troll box: they are mailboxes that are particularily set for gathering spam. This is mainly done by replying to spammers from an address not used for legitimate communications. The reason why it works is that as we have described above, spammers are particularly interested in 'live' e-mail addresses, e-mail addresses they are sure are being read. So when a reply is made to them, they usually add the From address in the reply to their mailing lists. Eventually, when setting up a troll box, this address will start getting spammed. The troll box can also be set by posting to USENET from troll addresses, or embed them in webpages. Then, using procmail (or a similar mail processor) any mail sent to this address is filtered directly to razor-report. A troll box can also be set to report spam from a machine where razor-report is not available. Bouncing the message to the troll box will report the spam to a Razor Catalogue Server.

3.2.3 Razor-discover

Razor-discover is used regularly for razor agents to automatically discover the closest server.

3.2.4 Efficiency

Although it is hard to estimate the efficiency of such a system (since it is hard to test all the messages that are marked as spam to be sure they are not normal messages), table 3.1 presents a certain number of figures that show the efficiency of Razor. It shows that it is not really great yet. The reason for this is that there is a need for a different way of making the signature, or storing the spam messages.

As of now, the best way to use Razor is still to use it in conjunction with another spam detection system. One of the most efficient combinations today is the SpamAssassin system that was described in section 2.4. However this puts too much emphasis on the use of signature and not enough on the actual generation of the signatures. This leads to a lack of signatures being

Date	Spam Count	Jammed Count	Razored Count	Blacklisted Count	Trollbox Count
07/04/02	74	43 (58.11%)	37 (50.00%)	5 (6.76%)	35 (47.30%)
08/04/02	109	82 (75.23%)	29 (26.61%)	10 (9.17%)	41 (37.61%)
09/04/02	194	157 (80.93%)	34 (17.53%)	16 (8.25%)	63 (32.47%)
10/04/02	140	104 (74.29%)	32 (22.86%)	6 (4.29%)	64 (45.71%)
11/04/02	165	143 (86.67%)	26 (15.76%)	36 (21.82%)	48 (29.09%)
12/04/02	168	133 (79.17%)	43 (25.60%)	19 (11.31%)	57 (33.93%)
13/04/02	84	71 (84.52%)	20 (23.81%)	18 (21.43%)	45 (53.57%)
14/04/02	101	74 (73.27%)	39 (38.61%)	10 (9.90%)	56 (55.45%)
15/04/02	117	91 (77.78%)	29 (24.79%)	11 (9.40%)	48 (41.03%)
16/04/02	159	121 (76.10%)	32 (20.13%)	19 (11.95%)	57 (35.85%)
17/04/02	151	123 (81.46%)	29 (19.21%)	18 (11.92%)	45 (29.80%)
18/04/02	136	107 (78.68%)	33 (24.26%)	13 (9.56%)	54 (39.71%)
19/04/02	135	121 (89.63%)	24 (17.78%)	10 (7.41%)	38 (28.15%)
20/04/02	111	97 (87.39%)	33 (29.73%)	10 (9.01%)	43 (38.74%)
21/04/02	75	57 (76.00%)	22 (29.33%)	6 (8.00%)	37 (49.33%)
22/04/02	177	156 (88.14%)	25 (14.12%)	5 (2.82%)	41 (23.16%)
23/04/02	199	176 (88.44%)	39 (19.60%)	9 (4.52%)	53 (26.63%)
24/04/02	212	174 (82.08%)	47 (22.17%)	6 (2.83%)	57 (26.89%)
25/04/02	189	166 (87.83%)	25 (13.23%)	4 (2.12%)	63 (33.33%)

Table 3.1: Efficiency of Vipul's Razor

generated because the administrators of the central database do not trust (and they might be right, since they have no trust management system built inside the program) the general user.

A final criticism to the Razor system would be its centralization. It has been said that not enough signatures were stored on the system, but the centralization of the database means that there is an important scalability issue that arises with the increase of the number of users, and the number of signatures. There has been a report in [raz] recently about this issue. The answer to this was providing the users who were willing to host it a program allowing them to act as proxy server for the database. Although this solves the problem in the short term, it still does not solve the problem of the database being overcrowded with old spam, as the administrators seem reluctant to get rid of old spam; they argue that many spam keep being sent for months. They seem to think that a solution will be found when it is needed, without offering any hints of what they intend to do.

The second version of Razor should have been out in the beginning of June 2002, but apparently, technical issues have delayed it. However, there has been information about it in [raz]; the three main improvements will concern the hashing algorithm, the introduction of a trust management system, and the possibility of deleting a signature from the database (in case of an error for example.)

3.3 Cloudmark's Folsom

Folsom is, in a way, an extension to Razor : Vipul Ved Prakash has teamed up with Jordan Ritter, who was Chief Architect for the Napster Peer-to-Peer file-sharing software. They intend to sell the result of their research (currently code named Folsom) to companies, through their start-up, Cloudmark. The main filtering idea is the same as that of Razor, but Folsom relies more heavily on artificial intelligence than on actual users for spam recognition. Although the system is not yet available for real-life testing, the authors are saying that they have run tests that show that the use of Folsom can reduce the flow of spam to a near-zero rate, with very few false-positives. They argue that they are implementing proprietary techniques that prevent spammers from circumventing the system by adding little modifications to their spams (this problem is caused by the use of the hash signatures, and is one of the main current issues with Razor). The efficiency of this system remains to be seen.

3.4 Distributed Checksum Clearinghouse (DCC)

The DCC or Distributed Checksum Clearinghouse, presented in [dcc], is very interesting because it is different from any other system discussed so far. Its main goal is not necessarily to identify unsolicited bulk email, but to identify bulk email and leave it up to the administrator to say which bulk emails are allowed, through the use of whitelists.

This system consists of a number of clients and servers that collect and count checksums[1]. DCC servers are exchanging the checksums which are seen most often, since it means these messages are bulk emails. However they are not necessarily unsolicited bulk email. The checksums include values that are constant across common variations in bulk messages, including *customizations*.

A clearinghouse server totals reports of checksums of messages from clients and answers queries about the total counts for individual checksums. Each recipient decides independently how to handle each bulk message. The recipient of a message or a DCC client can report and ask about the total counts for several different checksums for each message. If one or more totals for a message are higher than thresholds set by the client, a DCC client that is part of a SMTP server can log, discard, or reject the message. DCC clients that are parts of mail user agents can discard, file, or score messages based on their "bulkiness."

One use of this system is to have an isolated DCC server fed by troll boxes. This ensures that all mails stored in the clearinghouse are actually unsolicited. However, this way, you are not using the full power of the system, that is the cooperation between the servers. If you open the server to other servers, you no longer know if the emails are solicited or not. That usually mandates the use of white lists to ensure that solicited bulk mail is not rejected.

As seen in the first version of Razor, the simplistic checksums (i.e. hash algorithms) of spam are not very effective, because they do not take into account small variations and customizations. A better approach to hashing of spam messages needs to provide the following two properties:

non-propagation In most hash algorithms a change in any part of a plain text will either change the resulting hash completely or will change all parts of the hash after the part corresponding with the changed plain text. Non-propagation means that small changes in any part of the

[1]What is called a checksum in DCC is the equivalent of a signature in the previous two systems.

Type of Checksum	**Details**
IP	address of SMTP client
env_From	SMTP envelope value
From	SMTP header line
Message-ID	SMTP header line
Received	Last Received: header line in the SMTP message
Substitute	SMTP header line chosen by the DCC client, pre-fixed with the name of the header
Body	SMTP body ignoring white-space
Fuz1	filtered or "fuzzy" body checksum
Fuz2	another filtered or "fuzzy" body checksum

Table 3.2: Types of Checksums used in DCC

plain text will leave most of the signature the same.

alignment robustness Most hash algorithms are very alignment sensitive. If the plain text is shifted by a byte (say by inserting a character at the start) then a completely different hash is generated. An algorithm robust to alignment changes will automatically re-align the resulting signature after insertions or deletions. This works in combination with the non-propagation property for telling with accuracy if two emails are "similar".

Taking this into account, DCC currently implements a fuzzy checksum and ignores various aspects of the message. Furthermore, it makes it easy to change the fuzzy checksum as spam evolves. Since the DCC started being used in production in late 2000, new checksum schemes have been developed and distributed several times. In Table 3.2[2] are the different types of checksum used in DCC.

Unless used with isolated servers and thus wasting part of the interest of the system, the DCC does cause some additional network traffic. However, the client-server interaction for a single mail message consists of exchanging

[2]source http://www.rhyolite.com/anti-spam/dcc/dcc-tree/dcc.html .

a single pair of UDP/IP datagrams. That is often less than the several pairs of UDP/IP datagrams required for a single DNS query. SMTP servers commonly make at least one DNS query for every message to check the envelope Mail_From value and often several more. As with the Domain Name System, DCC servers can be placed near active clients to reduce the DCC network costs. DCC servers should exchange reports of checksums for the service to be most effective, but they should exchange only the checksums of messages that are frequently seen. Since the vast majority of mail is sent to only a few people, only a small fraction of reports of checksums need to be exchanged by servers. That significantly reduces the network bandwidth and disk storage needed by a DCC server.

The DCC system is composed of six different programs. dccd is the actual server, the DCC daemon. To keep a server's database of checksums from growing without bound, checksums are erased when they become old. Checksums with large totals can be kept longer. This process is made by a program called dbclean. The utility program cdcc is used to maintain the shared file of server names, addresses, passwords, and so forth as well as to check the health of servers. On systems running servers, this program is also used to maintain the database of checksums. Dccm queries a DCC server, add header lines to incoming mail, and reject mail whose total checksum counts are high. Dccm is intended to be run with SMTP servers using sendmail. Dccproc is the second major client program. It adds header lines to mail presented by file name or stdin, but relies on other programs such as procmail to deal with mail with large counts. Dccsight(8) is similar but deals with previously computed checksums.

This system has many advantages. First, on the opposite of Razor, it focuses on the *bulkiness* of messages to detect spam, and thus does not have to store as much information as a Razor database. Secondly, there is no particular action that a client needs to take in order to declare a message as bulk; this is done automatically. This is an attractive feature, but the drawback is that user involvement is required for the generation of the whitelists, more intrusive to the mail reading process than merely clicking on a button when identifying a message as spam. However it prevents the whole community from being worried about manipulation errors. There can be no false negative this way (except the practically impossible problem of two messages having the same hash).

However, there might be a problem in terms of additional traffic on the network. Although it seems that the algorithm does limit traffic between DCC servers, the traffic on the local network is still high.

Although these last two collaborative spam filters are being used today with a certain success, they have one important problem; the efficiency of both is directly related to the number of users (this is what Folsom is trying to address), even if troll boxes can lower that impact, while none of them is really portable (for example, none of them is really usable under Microsoft Windows).

Chapter 4

Design

This chapter details the design of the anti-spam network. After an outline of the way the spam filtering is done and a description of all the features of the system, the three major aspects of the system are tackled separately. Finally, different threats to the network are identified and addressed.

4.1 Scenario

The filtering of the anti-spam network is following the general lines of the spam filtering systems described in sections 3.1 and 3.2. The three steps are shown in Figure 4.1.

A — Spam is sent to ASN user A The spammer sends a spam message to the first user. This user recognizes it as spam, and stores it in its list of spam messages.

B — ASN users exchange lists of spam The two users exchange their signatures of spam. This means that ASN user B now knows about the message the spammer sent to A earlier.

C — Same Spam is sent to user B The spammer sends the same spam message to the second user. Since this spam is on the list of spams, the system recognizes and blocks the message. ASN user B never actually sees the message.

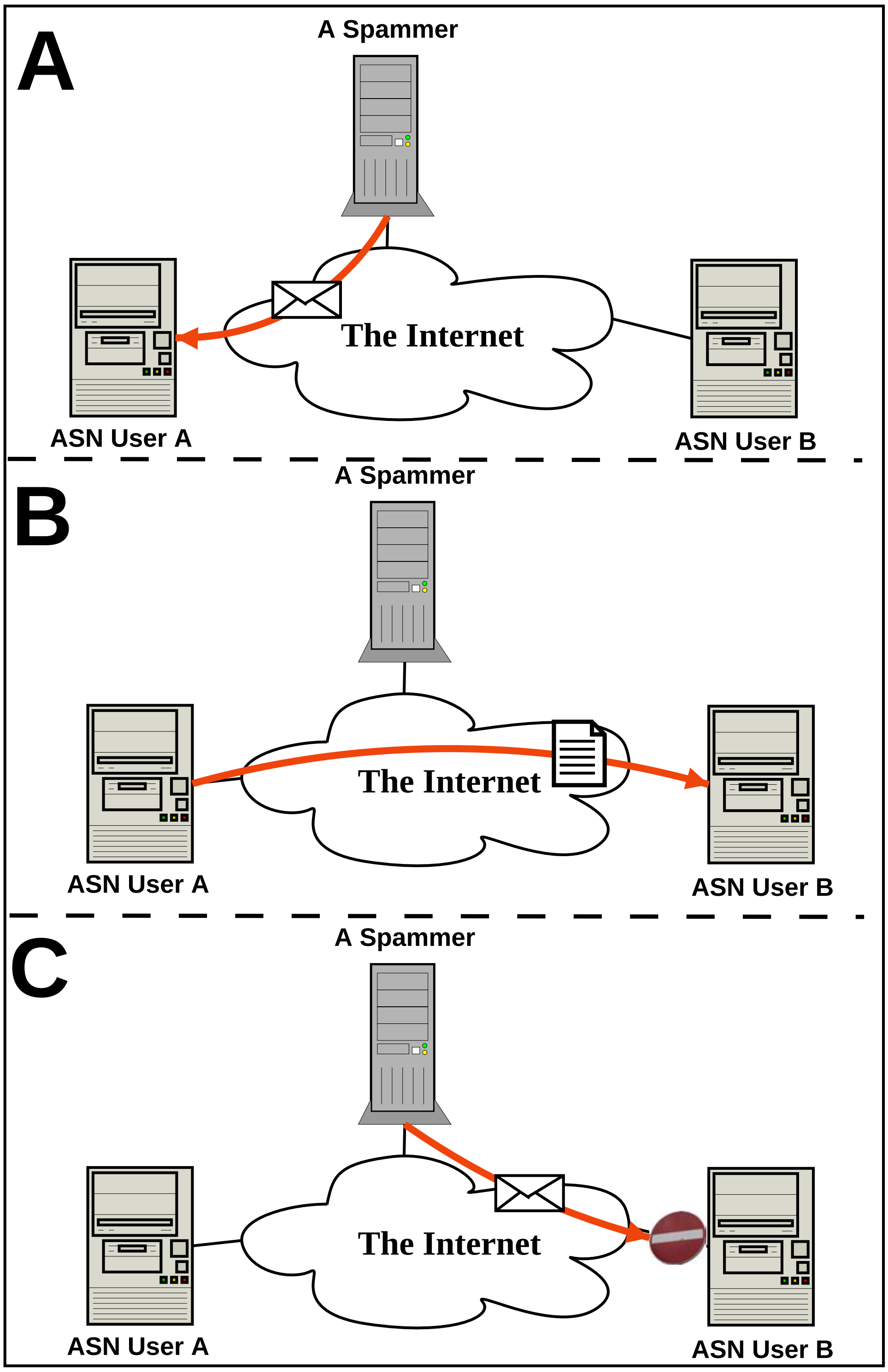
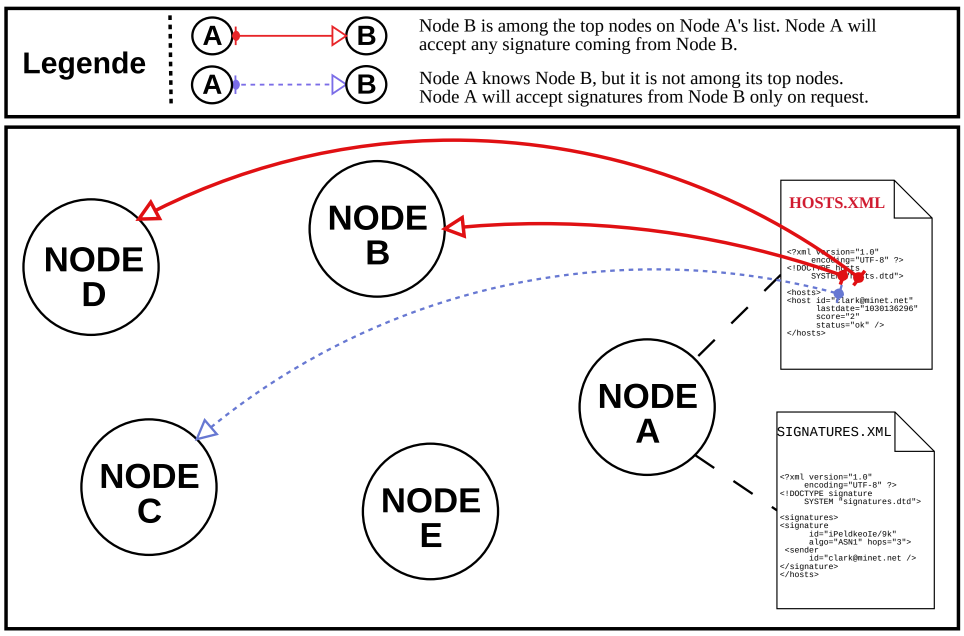


Figure 4.1: Basic Scenario of Collaborative Spam Filtering


Legende
A
B
Node B is among the top nodes on Node A's list. Node A will accept any signature coming from Node B.
A
B
Node A knows Node B, but it is not among its top nodes. Node A will accept signatures from Node B only on request.
NODE D
NODE B
NODE A
NODE C
NODE E
HOSTS.XML
SIGNATURES.XML

```
<?xml version="1.0"
     encoding="UTF-8" ?>
<!DOCTYPE hosts
     SYSTEM "hosts.dtd">

<hosts>
<host id="clark@minet.net"
      lastdate="1030136296"
      score="2"
      status="ok" />
</hosts>
```

```
<?xml version="1.0"
     encoding="UTF-8" ?>
<!DOCTYPE signature
     SYSTEM "signatures.dtd">

<signatures>
<signature
      id="iPeldkeoIe/9k"
      algo="ASN1" hops="3">
 <sender
      id="clark@minet.net />
</signature>
</hosts>
```

Figure 4.2: Network Topology of the Anti-Spam Network

4.2 Features

The network topology of the system is shown in table 4.2 The anti-spam network is a decentralized peer-to-peer network, where each peer corresponds to an email user identified by a unique address. Each peer's knowledge of the rest of the network lies in the peers file, where other peers' identifiers are listed. All the peers on the peers file are ranked according to usefulness, availability and proximity (this is explained below in section 4.2.3). The idea here is to have on top of your list emails that are on the same spamming lists as the user is. By communicating with the other peers, each peer is able to maintain a list of spam signatures that are relevant to its mail. The spam signatures are compared to any incoming mail before it is presented to the reader, and if it is recognized as spam, an appropriate action is taken.

The following functionalities represent the main features of the anti-spam network. However, most of them have a local scope and have no impact on the other peers on the network. Therefore, certain implementations might decide to leave them out, or to alter them. The idea here is to be more resistant to attacks (since the system place very little trust on the other peers), and also for the system to be adaptable to different needs.

4.2.1 Signatures and Peers Data Structure

Because we are building a decentralized system, each peer has to maintain its own database of peers and signatures of spam. In order to fulfill our objective of user-friendliness, the system must be easy to install. That immediately rules out any database that would necessitate a server to be launched and maintained on top of the system. The obvious answer to this problem is to use files as data support.

Since it has also been stated that cross-platform availability was an objective (see section 1.2.2), we will need a cross-platform data structure to hold the data. Furthermore, this data structure should be open and extensible, so that the whole system could be extensible. This would also mean that it would be easy for external programs to interact with the data.

4.2.2 Report Spam

As seen previously in the basic scenario, the user needs to be able to report a message in his mailbox as spam. All the processing should be done transparently once the user has pointed to the message. For ergonomic purposes, this action should be as simple and straight forward as possible; this might mean typing on a key, or maybe drag-and-drop in graphical interfaces. The problem with this is that it will obviously depend on the mail client, because different mail clients may use different mail formats and also because depending on the mail client, you might not be able to customize key strokes for example. We have seen the same problem while studying the way Vipul's Razor works in section 3.2. It has do be adapted according to the email client. Another, more appropriate, approach is to make it a command line process, which can be called by the mail clients This can be done through a customized key mapping if available, like with mutt (see [Elk]), or maybe through special tricks, like setting a printer with pine (see

[Uni01]). For graphical interfaces, and because drag-and-drop would be nice, an interface could easily be built on top of the process.

For the details on the signature algorithm, see section 4.5 below.

4.2.3 Ranking Peers

From the point of view of the system, close nodes are primarily nodes that are on the same spamming lists. In order to take that into account, if a node sent the user a spam signature with which a message is recognized as spam, its score will increase. After this main priority, network proximity and latency between two nodes are taken into account. And finally, because the system will not necessarily be dealing with nodes that are always available on the network, this will have to be taken into account when ranking nodes. Even if a node is exactly on all the same spamming lists as the user, if it is always connected to the network at a different time than him, it should have a lower rank on his list than a node which is always available, and only on several same spamming lists as the user.

4.2.4 Exchange Signatures

The heart of the anti-spam network is to provide a mean of exchanging signatures between the users of the system. An important mechanism is that a signature must be spread as long as it is alive, i.e. as long as the spam is still being sent and received. In order to determine whether or not this is the case, it is decided that as long as a signature finds a match in a mailbox, the spam is still being sent and the signature should therefore still be propagated. In addition to that, a hops-to-live value, coupled with a maximum age for a signature on one single node, will ensure that a signature that is dead – i.e. the spam is not sent anymore – will not be kept on the network, thus not wasting resources on the nodes.

Exchanging signatures with the top nodes on one's list is important because, considering how they are ranked (see section 4.2.3), these signatures should be the ones that are relevant to a given user. Exchanging peers with the top nodes is important because there is a probable transitivity between peers on top of peers lists. If peer A is at the top of the list of peer B, and peer C is at the top of the list of peer A, then signatures of peer C are probably relevant to the mail of peer B.

It is also very important to try to communicate with other nodes than those on top of the list in order to determine if some of them might be better suited to be on top of the list. If a peer on top of the list is not providing any relevant signatures anymore, its rank should decrease. At the same time, if a peer is on the same spamming lists as the user, it should be given a chance to make its way to the top of the list. Therefore, a mechanism should be implemented that can track the last time a peer sent the user a signature which helped him filter a spam message. This could then be used to decrease the score of the peer if this date is growing too old.

In order to replace a node that would drop out of the top list with a node that is already relevant to the user (e.g. instead of picking a new top node randomly), the system must make sure to rank also the nodes which are not on top of the list.

4.2.5 Exchange Peers

An important aspect of such decentralized peer-to-peer networks is the knowledge of the network. Because a centralized server to hold the nodes addresses is not desirable, there is a need for a way for each node to know the rest of the network. This will be done by exchange of emails between peers. It will be done at the same time as the exchange of signatures in order to try and avoid doubling any communication overhead.

Since each node will contact more frequently high-ranked nodes (see section 4.2.3) on its peers file, if a node is contacted by another, chances are the sender is on the same spamming lists as the receiver. Obviously, a node is trying to learn of other nodes that are on the same spamming lists as him. So it makes sense for a receiver of a request for nodes to answer with the nodes on top of its list of peers.

4.2.6 Revoke Signature

As was seen with Razor in section 3.2, a main problem with the idea of collaborative spam filtering is that a human being can make mistakes quite easily. Because the declaration of a message as spam must be as straight forward as possible, confirmation every time somebody creates a new signature is not necessary. This might lead to a genuine error from a legitimate user of the system. It would of course be an interesting feature if this user was then allowed to repair any mistake he has done.

However, it seems that there are several risks involved with this idea: for example, a malevolent user might intercept any spam (or create the signatures for its own spam if he is a spammer), send it to you and then decide to revoke the signatures, thus hoping that this would force the spam in question into your sight, instead of having it deleted.

There is still something that can be done; since the signatures are stored independently on each node, and because they will not spread out until another node asks for signatures, a great deal of the problem is solved by providing a simple revocation mechanism that works only locally. This will be enough because in most cases, the genuine user will realize he has made a mistake as soon as he makes it – this fact has been demonstrated by users of Razor, in [raz] – and using the revocation mechanism immediately will prevent the generated signature from being sent outside the node, except for really unfortunate bad luck.

4.2.7 Whitelist

One of the big issues with spam filtering is the issue of false positives. The solution to fulfilling the objective of reliability set in section 1.2.4 lies in the use of a whitelist, as explained in section 2.3.

This whitelist consists of a list of email addresses of senders from which any mail will be accepted even if the message appears to be spam according to the anti-spam network. For example, a company will not let its users participate in the anti-spam network unless they have a certain number of guaranties regarding critical emails. Critical emails would include emails from the CEO or the system administrator, or maybe certain mailing lists. Anyway, a whitelist provides with a very inexpensive and straightforward way of avoiding most false positives. However, it should be kept in mind that this might be a problem for usability, as the whitelist has to be customized by the user.

The same cautious users who want to make sure that critical email is not going to be recognized as spam will probably argue that a spammer can easily forge the From: header of a spam message to make it look as if it comes from somebody on your whitelist. However, two facts show that a whitelist is still relevant. First, it is quite clear that although it is very easy for a spammer to associate an email address with a name for customization purposes, it is definitely much more difficult to associate an email address with the email address of somebody that would surely be on the associated whitelist although not unheard of. Secondly, the reason why spammers are so hard to stop is because they most often can not be sued or attacked legally in any way for what they are doing. However, forging an identify is illegal in most countries. It can be assumed that this risk is the reason why most spammers have chosen not to forge real email address when customizing spam over the last years especially considering that they can never be sure that the forged email is actually on the receiver white list.

Because of the objective of ease-of-use set in section 1.2.3, it should be easy to add and remove emails from the whitelist. For this reason, and because of the need for a cross-platform system, what needs to be defined primarily is the format of the whitelist file. From then on, it should be easy to implement user-friendly front-ends for customization of the whitelist.

4.2.8 Ban Peer

Because a much-needed functionality to the system would be an easy and quick way to avoid any peer that appears malevolent, or really clumsy, a banning mechanism needs to be put into place. The most obvious use of this function would be when the user realizes somebody is repeatedly sending false positives reports to him. This would allow him to not consider this user anymore.

Different levels of banning are necessary though, to differentiate between the user that made a mistake, and thus is not as trustworthy as before, and the actual attacker who is trying to send you false positives. This could be done quite simply by having two levels of banning; the first one is temporary, and the second is permanent. Temporary banning will mean that the banned peer is removed from the peers file (nothing prevents it from reappearing there again if the peer's address is sent to the user again). However, a permanently banned peer stays in the peers file, with a special status that will ensure the user will not deal with this peer again.

4.2.9 Customize Security

Once again, because the anti-spam network needs to be accessible to as many users as possible, the range of potential users must be very wide. This means as many patterns of use as possible must be available. For example, both a child who has no critical mail whatsoever, and receives only a couple of spam messages per day, as well as a businessman, who frequently receives critical email that cannot be delayed, and receives about twenty spam messages per day must be able to adapt the system to their need. More so, the system must apply to all types of users between those two extremes.

For this reason, the system needs to be highly customizable in the actions it takes against what has been detected as probable spam, and this customization must be as easy to do as possible. Some example will include a choice between direct action as soon as a message is classified as spam, or necessity for a confirmation, or maybe the choice between deletion of a spam message, and moving to a particular folder. These details will be addressed during implementation.

4.2.10 Update Algorithm

There is a need to address the spammers' ability to create messages that circumvent filtering mechanisms. For the anti-spam network, this reactivity will come from being able to change the signature algorithm at any time, with backward compatibility. This last need can be easily dealt with by inserting somewhere in the signature the id of the signature algorithm. However, there is a problem with the distribution of the new algorithms since it must not be possible for a malevolent user to distribute a tricked algorithm. Not only would this prove bad for the anti-spam network, but it could also have much more dire consequences for the user and his computer.

This problem can be solved with the use of a public key encryption scheme. A private key is used to sign the file containing the new algorithm, and the public key is distributed such that the user can verify the new algorithms.

Provided that the public key is available to the user of the anti-spam network, it is then easy to distribute a new algorithm throughout the network. This can be achieved by using a Public Key Infrastructure (PKI) and having certificates with the new algorithms.

Since the id of the algorithm used to create the signature is added to the actual signature, the system will realize that he does not know the appropriate algorithm as soon as it receives a signatures with an id it does not know.

When this happens, the system from which the signature was received will be contacted for the encrypted file representing the algorithm. This means a new protocol command is necessary. The actual signature will not be added to the local signatures file until the algorithm file has been received and verified with the public key. This way, a user should not receive a signature created with an algorithm unknown to the sender, only somebody aware of the private key can inject a new algorithm, and the new algorithm should propagate through the network quickly.

Whenever a node has more than one algorithm available, any one of them (probably the youngest) can be used for creating new signatures. The old algorithm files will still be kept for backward compatibility.

4.3 Node Communication

4.3.1 Request/Response

The anti-spam network will work according to a scheme of request/response from a peer to another, one peer acting in turn as the sender and as the receiver. There is no need to keep a connexion open at all times since all that is needed is an exchange of peers and signatures. The requester sends a message to the other node. The message contains a certain number of peers relevant to the sender and trustworthy (this means they rank high on the sender's peer file) as well as a command asking for the receiver's spam signatures and peers.

The receiver registers the peers in its peer file (with a null rank) and then replies with its own top-ranked peers, as well as all the signatures that have been stored in the signatures file since the sender was last contacted, given their hop-to-live value is positive.

If the sender has not received a response after a certain timeout period, it will consider the peer was not available. This means that it might decide to contact another peer for more spam signatures, and it also means that the receiver's score will go down in the peers file.

There is no need for any confirmation that a message has been lost or not, since the other peer will be considered unavailable in this case, resulting in a decreased score (see section 4.2.3). All that is needed is integrity of the data, and this will be dealt with by the lower transport layers of the network.

4.3.2 The Transport Protocol

There are two main choices for the transport protocol for the anti-spam network; HTTP and SMTP. Other protocols (e.g. TCP) cannot be considered because the system must be usable by as many users as possible. This means that a user should be able to participate in the anti-spam network from behind a firewall. A quick survey of the current situation of ISPs shows that the only ports that are open for most firewalls (through a proxy or not) are those of HTTP and SMTP. This situation is especially true for students or employees of companies; these users are usually behind very restrictive firewalls.

The HTTP protocol has the main advantage of being very versatile, and it can be adapted to almost any need. Furthermore, it allows a direct link to be established between the client and the server, and this allows for fast operations, compared to an asynchronous link. However, because of the usual use of HTTP — the transfer of web content – the firewalls are usually set to let pass only outgoing communication. Since we are interested in a peer-to-peer use of HTTP, this fact will prove too restrictive.

The second alternative, though, proves much more adequate. First of all, the use of SMTP seems to be a natural choice for a spam filtering service, as there is a logical relation between a user and an email address, which can be used to identify a node's address on the system. Then, as described above, we need to be able to communicate through a proxy, and SMTP has the advantage over HTTP that all firewalls will let it pass both ways. In fact, if the user's mail server is cut from the Internet, you can hardly imagine how the user is able to receive spam messages. Thirdly, using SMTP as the transport layer allows for the use of the already widely installed infrastructure of mail servers networks, which provide with high reliability. Finally, a very important point considering any user should be able to join the anti-spam network is the fact that the asynchronous quality of SMTP and the mail infrastructure make sure a user does not need to be connected to the network at all time for him to get his messages. This implies that the anti-spam network could be joined by users with transient connections, such as dial-up connections or the interfaces typically used with PDAs. In fact, any user that receives spam will be able to join the anti-spam network.

These four facts make the choice for SMTP reasonable. However, there are a few drawbacks to this that need to be taken into account. The first problem comes from the very advantage of using email addresses as identifiers. Because the nodes addresses will be stored in each node, these email addresses are public. Therefore, somebody willing to participate in the anti-spam network will have to make his address public. Of course, there are ways to come around this problem, the most obvious one being a 'screen' address used only for the anti-spam network, but this is not perfect for our objective of user-friendliness since it adds a step in installing the anti-spam software and also a level of overall complexity. Furthermore, spammers will not necessarily be willing to swamp their database of email addresses with addresses of users who are trying to fight spam. They might still want to use them, but their values will probably be lower than that of a 'live' address of someone who actually reads the spam, since the spammer knows these users will not be well inclined toward spam.

The second potential problem is the speed of the process. On the contrary of a direct connexion, there are here several third parties involved. On each of them, a time is spent processing the request. Furthermore, there is no way to verify if the recipient received the message, or when he received it.

4.4 The Filter

As with every spam filtering system, there is a need for a filter plugged in somewhere in the system, which will look at all the incoming mail, and determine if it is a spam or not. The design of the anti-spam network does not force the choice of a particular type of filter. This section describes different possible solutions for implementing the filter.

Whatever the solution is, a delicate aspect of the protocol communication has to be kept in mind; when an administration mail arrives in the mailbox, it cannot be just simply displayed in the user mailbox. It needs to be intercepted, recognized for what it is, and processed according to the protocol.

4.4.1 Centralized Mail Processor

The *centralized mail processor* is a filter plugged in between the MTA and the MDA, on the mail server. Several such processors are available. A good example would be procmail (see [pro01]), which is very popular and widely available on Unix systems.

The good point with this email processor is that it can easily deal with the protocol communication messages as well. By simply adding the relevant rules to the user's configuration of the mail processor, this final functionality is enabled, allowing the communication to work.

4.4.2 Local Mail Proxy

Although procmail, which works on the mail server, has been given as an example, an important point to stress is that there is no need for this mail processor to be actually on the mail server. The user can also — but it might prove harder to install, configure and maintain — use a very basic and simple mail proxy on the client. This proxy will intercept any incoming mail (POP) or any incoming mail folder listing (IMAP) and then perform the necessary operations.

4.4.3 Extension of the Mail Client

Finally, another solution is to bundle everything in the email client. This solution would be very easy for the user to install and configure, since only one package would have to be installed. Furthermore, there is less chance of having the filter break down and the user start receiving the protocol messages in the mailbox. However, this solution has a major drawback: each email client requires a different implementation, which in turn requires a lot of time. This might not be compatible with the objectives of a lot of users defined in section 1.2.

4.5 Signature Algorithm

For someone who receives spam, the most obvious customization of spam is the display of the user's name in the body of the subject of the email. Of course, this is not pure coincidence. If two users receive the same bulk email, each will have his name in the message. This means that technically, the two messages are different. However they should be recognized by our spam filter as the same spam. In order to do that, the current state of the art is to use a fuzzy hashing algorithm, as seen in chapter 3.

But in order to further reduce the possibilities of differences between occurrences of the same bulk email, an idea is to track the potential area of the message where such customizations are most likely to occur, and then get rid of these parts on all messages, before hashing them. In an unpublished study, available in appendix A, Gabriel Matthews has tried to identify such areas. They include numbers, URLs, Punctuation, white spaces, javascript, HTML. His results were that more than 52% of spam have customizations in these different areas.

This means that it could prove viable to remove these (e.g. remove any extra white spaces in the message) before hashing. The cost of this is mainly computer processing power and time. Although this would prove too costly for systems like Razor where the checking is often done on centralized high-duty servers, the anti-spam network is distributed and the load can be spread among the individual clients, which makes the issue less relevant.

However, fuzzy hashing is really effective only when the size of the messages to be hashed are big enough. More precisely, the idea behind the fuzzy hash is to be able to compare two hashes. However, this could lead to numerous collisions[1] as the sizes get smaller. Getting automatically rid of so broad a range of text parts might make all messages (including non spam messages) so small that the fuzzy hashes and the spam signatures comparisons will be irrelevant.

[1]A collision is said to occur if two messages are recognized as similar when they are not.

This fact made a study on the compositions of mail messages necessary. This study is displayed in appendix B. The conclusions to this study is that, for most hashes, stripping the message of all its customization-prone part will prove useful and should not be too dangerous. However a further rigorous mathematical study will be required to be more precise about the fuzzy hash. One last thing that can be said is that using a fuzzy hash in this way might be a conscious trade-off between a better detection of spam, and a higher risk for false positives.

4.6 Possible Threats

In order to be valid for use in the real world, the design must be able to cope with different threats. The main threats identified for the anti-spam network are: false positives, corruption of data, peer overload, and mail server overload.

4.6.1 False Positives

One of the main issues with collaborative spam filtering is one that has been outlined many times above: false positives. Collaborative spam filtering is supposed to lessen the number of false positives compared to a traditional spam filtering technique by removing the need for an automated recognition of spam. However, this introduces the possibility for wrong reports, with a user declaring a message as spam by mistake (or malevolently).

The worst impact here is for users who receive critical mail that cannot be lost because of the mail filtering system. For users who do not care that they may get an important message late, the solution is quite simple; since the user can choose to move a message marked as spam to a special folder instead of deleting it, he will just need to set this in his configuration and check regularly in this special folder for any mail that would stand out as not spam. This method has been proved effective by many email systems like Hotmail. For the users who are not willing or able to afford receiving a message late, there is the possibility of increasing the threshold of matching signatures required to classify as spam. This would mean that a few spam messages might slip through the anti-spam network filter, but this would also mean that the number of false positives could be virtually null (and most spam would still be filtered).

4.6.2 Corruption of Data

Since the data is stored on each node, there is no way to make sure every step is taken to prevent data corruption on another node, not to speak of voluntary corruption. In earlier stage of the design (see 4.2.6), such a threat was identified and a decision was taken not to provide revocation because of the way a spammer could use them to compromise the system.

However, with this design, there is no particular trust given to the information received from a node, except for the signatures; the email addresses of other nodes will have to be tested before they are trusted. As shown in [NIS02], because hashes functions are complex one-way functions, it is virtually impossible to – randomly or purposely – receive a relevant signature that has not been produced from an actual message. This means that the only thing corruption of data will do is create useless and irrelevant signatures. Furthermore, as shown in Figure 4.2, signatures not coming from peers on top of the user's list are discarded unless explicitly requested. This fact makes it hard for malevolent users to try to flood a user with random signatures in order to generate false positives.

4.6.3 Overload of a Peer

Since there is a background process on the node that answers to incoming queries from the anti-spam network, there might be a temptation for someone who wants to deter the user from participating in the anti-spam network to send a lot of queries to him. Although the bottleneck created by the mail servers is probably enough to shield the user from any flooding, it is important to have a value set somewhere in the configuration of the software that represents a maximum number of requests that the background process will answer per hour.

This kind of malevolent attack is probably not likely to take place. However, there is a real risk for a user to become a central point in the network (e.g. if he reports a lot of spam, and numerous users start relying on him). This is not a good idea in a centralized network (e.g. if this peer goes down). However, this threshold for the number of answered requests will also have a long-term impact: the load will be more widely spread on the anti-spam network. Because the ranking of peers also depends on the availability, if a peer answers only to 10 requests per hour, the eleventh peer to connect to him will end up querying another peer, thus spreading the load throughout the network.

4.6.4 Overload of the Mail Server

An issue that arises from the use of SMTP as the transport protocol is the increased load on the mail server. If a great number of users of the mail server are participating in the anti-spam network, and if the mail server is already getting near its maximum load, the anti-spam network communications might prove to have too great an impact on the load of the server. However, in any case, since only two mails are exchanged for a connection, the amount of mails exchanged for the anti-spam network should prove to be only a small portion of the mail server load.

There is always the possibility for the mail server administrator who thinks the anti-spam network is using too much processing power and bandwidth to set a few rules that will only allow a certain number of anti-spam network messages to be exchanged per hour. This would prove easy since the anti-spam messages follow strict rules and these rules are freely and easily available. The effects of that choice would be the same as the ones outlined in section 4.6.3 although possibly on a much bigger scale, since an entire mail domain might get affected. Nonetheless, since this service might mean a decreased number of spam for the end user, it is reasonable to think that mail administrators will try and keep the system running unrestricted.

Chapter 5

Implementation

This chapter describes an implementation of the design presented in the previous chapter. This implementation uses a backend of perl scripts and XML files. It also uses procmail for the filtering. However, as emphasized in chapter 4, different implementation are possible, and a number of others are outlined at the end of the chapter.

Due to time constraints, the current implementation is not completely functional yet for a wide range of users. However, it serves as a proof of concept for the design presented in chapter 4.

5.1 Programming Language

In order to fulfill the objectives of a cross-platform solution presented in section 1.2.2, the programming language chosen for the backend has to be platform-independent. The main tasks performed by this backend would be string analysis and manipulation, as well as SMTP communication.

These considerations and the fact that the programming language should be widely and easily available narrow the choice of language to perl (see [WCO00]) and java (see [GM96]). Because of the power of perl concerning string manipulation, and the weight of java when all that is needed is simple console scripts, it was decided to implement the backend of the anti-spam network using perl.

5.2 Data Storage

As outlined in section 4.2.1, the file format used for the storage of data should be platform-independent. For this reason, the obvious solution is to use XML, which also has the advantage of being both human and machine-readable. The discussions in section 4.2 define the need for five main types of data in order for the anti-spam network to be fully operational: configuration, signatures, peers, whitelisted peers and algorithms.

5.2.1 Configuration File

The configuration file is quite simple, consisting of a series of key/value pairs. The following is the simple DTD describing its format:

```
<!ELEMENT configs (config)*>
<!ELEMENT config (#PCDATA)>

<!ATTLIST config name CDATA #REQUIRED>
<!ATTLIST config value CDATA #REQUIRED>
```

The following is an example of a configuration file for a Windows platform, which uses the above DTD:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configs SYSTEM "config.dtd">
<!-- Start of data -->
<configs>
    <config name="maindir" value="D:\asn\"/>
    <config name="algorithms"
      value="D:\asn\algo\algorithms.xml" />
    <config name="whitelist"
      value="D:\asn\data\whitelist.xml" />
    <config name="peers" value="D:\asn\data\peers.xml" />
    <config name="signatures"
      value="D:\asn\data\signatures.xml" />
    <config name="version" value="1.0" />
    <config name="defaultalgo" value="ASN1" />
    <config name="nbpeertosend" value="3" />
    <config name="email" value="clark@minet.net" />
    <config name="smtpserver" value="mail.cs.tcd.ie" />
    <config name="spamthreashold" value="2" />
    <config name="sighoptolive" value="3" />
    <config name="siglifetime" value="604800" />
    <config name="nbpeertocontact" value="10" />
</configs>
<!-- End of data -->
```

These values will be used by the different scripts to decide how the software should behave. For example, the value `nbpeertocontact` corresponds to the number of peers to contact every time the script is trying to renew its signatures. These values will be explained when describing the actions of the different scripts, in section 5.3.

5.2.2 Peers File

The peers file consists of a list of peers that represent the view of the network for the user. The DTD for the peers file is as follows:

```
<!ELEMENT peers (peer)*>
<!ELEMENT peer (#PCDATA)>

<!ATTLIST peer lastdate CDATA #REQUIRED>
<!ATTLIST peer id CDATA #REQUIRED>
<!ATTLIST peer score CDATA #REQUIRED>
<!ATTLIST peer status CDATA #REQUIRED>
```

`id` is the email address of the user, which represents its unique id on the anti-spam network.

`lastdate` is a timestamp corresponding to the last time the peer was contacted. This is used to decide whom to contact between two peers who are not on top of the list; the one that was not contacted for the longest time will be contacted.

`score` is an index representing how useful this peer has been in the past. As explained in section 4.2.3, the higher the number, the more spam the node has recognized thanks to signatures coming from this peer. Also, the higher the number, the more available the peer is. These features are implemented in the current version. Whenever a message is recognized as spam by comparing it to a particular signature, the score of all peers associated with this signature — meaning they sent you this signature — are raised by a certain number of points.

`status` is used to decide availability of a peer. When a request is sent to the peer, the status changes from `ok` to `waiting`. When a response is received and the status is `waiting`, it goes back to `ok`. After a certain amount of time, if the status is still `waiting`, the score of the peer is decreased and its `status` is reset. A banned peer (see section 4.2.8) will have a status of `banned`.

The following is an example of a very small peers file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE peers SYSTEM "peers.dtd">
<!-- Start of data -->
<peers>
    <peer id="laurent.cottereau@int-evry.fr"
      lastdate="1030136296" score="2" status="ok"/>
    <peer id="clark@minet.net"
      lastdate="1030100102" score="0" status="waiting"/>
    <peer id="cotterel@cs.tcd.ie"
      lastdate="1030100102" score="0" status="ok"/>
</peers>
<!-- End of data -->
```

The cleaning of the peers file is done regularly by getting rid of any peer whose score has become negative. In addition to this, when a peer is temporarily banned, it is removed from the peers file (see section 4.2.8).

5.2.3 Signatures File

The signatures file is slightly more complicated than the peers file. In addition to the characteristics of each signature, the peers from which this signature was received must be recorded. Since there could be more than one peer sending the same signature, the sender cannot just be recorded as an attribute of the signature. Instead, each sender is an element inside the signature element. The following is the DTD for the signatures file:

```
<!ELEMENT signatures (signature)*>
<!ELEMENT signature (sender)*>
```

```
<!ELEMENT sender (#PCDATA)>

<!ATTLIST signature id CDATA #REQUIRED>
<!ATTLIST signature algo CDATA #REQUIRED>
<!ATTLIST signature hops CDATA #REQUIRED>

<!ATTLIST sender id CDATA #REQUIRED>
<!ATTLIST sender date CDATA #REQUIRED>
```

`id` of the `signature` is the actual signature of the spam. As of the current implementation, this is be a SHA-1 digest, but this could change with the introduction of new signature algorithms. This means that the final signature is limited by the rules of XML attributes; especially concerning the use of single (`'`) and double (`"`) quotes.

`algo` of the `signature` is a four-character string identifying the algorithm used to produce the signature of the spam. Such an algorithm identifier is necessary in order to be able to compare a message with this signature.

`hops` of the `signature` represent the hops-to-live value of this signature. When sending this signature to another peer, the `hops` value is decremented by one. A signature with a `hops` value of 0 is not propagated. However, if a spam is found that matches this signature, the `hops` is reset to the maximum value. This value is defined in the configuration. The idea behind this feature is to try and ensure that only live signatures will propagate, and when a spam is no longer being sent, its signature will disappear from the signatures files after a certain time.

`id` of the `sender` is the email of the peer which sent the signature.

`date` of the `sender` is a timestamp representing the date at which the signature was received from the `sender`. If a spam is found by matching this signature, the `date` is set to the current time, thus representing the fact that the signature is alive and should be kept for a little while longer.

Following is an example of a small signature file:

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE signatures SYSTEM "signatures.dtd">

<!-- Start of data -->
<signatures>
    <signature id="PPdfg+U5uSz/o9pudCWCmjjDrUw"
      algo="SHA1" hops="3">
        <sender id="clark@minet.net" date="1030474044" />
    </signature>
    <signature id="eD67R-51uSzkj8Ueelkjsjj87/i"
      algo="SHA1" hops="2">
        <sender id="cotterel@cs.tcd.ie" date="1030474231" />
    </signature>
</signatures>
<!-- End of data -->
```

The cleaning of the signatures is done regularily by deleting old signatures from the file. There is not one date per signature, because it is important to distinguish between the senders. Once one of the dates is older than the signature lifetime (`siglifetime` as defined in the configuration file), the corresponding sender is deleted from the file, thus effectively decreasing the weight of the signature. Once a signature has no more senders, it is deleted from the file.

5.2.4 Whiltelist File

The whitelist file is a list of the email addresses that should not be considered potential spammers. This means that the anti-spam network should not suspect a message coming from a sender that appears in the whitelist. Following is the DTD of the whitelist file:

```
<!ELEMENT whitelist (peer)*>
<!ELEMENT peer (#PCDATA)>

<!ATTLIST peer id CDATA #REQUIRED>
<!ATTLIST peer priority CDATA #REQUIRED>
```

`id` is the email of the sender whose mail is critical and should not be compared to signatures by the system. This value will be compared to the email from the `From:` header of any incoming message.

`priority` is not used in the current implementation. However it has been added to the DTD because certain implementations might decide to make use of it. The idea of the `priority` attribute is to associate a peer with an importance that denotes how critical the mail from this sender is. This priority score can be compared to the spam score that is generated for each incoming message and that represents its likelihood of being spam. In effect, this value is equivalent to the `spamthreashold` of the configuration file, except that it is customized for each different email address, instead of being a system-wide value. The idea behind this feature is to meet any possible situation's needs in terms of balance between reliability and performance. A priority of 0 means that mail coming from this sender will not be processed at all. In the current implementation, all whitelist entries have a priority of 0.

Following is an example of a small whitelist file:

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE whitelist SYSTEM "whitelist.dtd">

<!-- Start of data -->
<whitelist>
    <peer id="mads.haahr@cs.tcd.ie" priority="0" />
    <peer id="vinny.cahill@cs.tcd.ie" priority="0" />
</whitelist>
<!-- End of data -->
```

5.2.5 Algorithms File

Each algorithm available to the system locally is linked to a file where are implemented all the perl functions necessary to use the algorithm. The algorithms file is a repository where all the algorithms are listed, along with the full path to the file. The following is the DTD for the algorithms file:

```
<!ELEMENT algorithms (algorithm)*>
<!ELEMENT algorithm (#PCDATA)>
```

```
<!ATTLIST algorithm id CDATA #REQUIRED>
<!ATTLIST algorithm url CDATA #REQUIRED>
```

`id` is the four-character unique identifier of the algorithm which was refered to in section 5.2.3.

`url` is the full path to the perl file containing all the functions necessary to use the algorithm in perl.

The following is an example of the current implementation for a Windows platform:

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE algorithms SYSTEM "algorithms.dtd">

<!-- Start of data -->
<algorithms>
    <algorithm id="SHA1" url="D:\asn\algo\sha1.pl"/>
    <algorithm id="ASN1" url="D:\asn\algo\asn1.pl"/>
</algorithms>
<!-- End of data -->
```

The algorithms file is updated when receiving a new algorithm. Although there does not seem at present to be any reason for suppressing an algorithm, should the need appear, the only thing to do is to delete the relevant line in the file.

5.3 The Different Scripts

There are six main backend scripts used for this implementation of the anti-spam network. Each of them can be called by different pieces of software, such as email clients, or mail processors.

5.3.1 asn-com

This script processes the incoming ASN-specific messages arriving in the mailbox. These messages are recognised by the presence of a special `X-ASN:`

header in the message. Following the header is the command. Four commands are defined:

EXC-I `EXC` means that this message is part of an exchange of signatures and peers. The `I` — I for Incoming — means that this is a request for exchange. In receiving this command, the system will add (if not already present) the sender and the peers in the body to the peers file, with a score of 0. If the peer is trusted (i.e. has a high score), the receiving peer will also add the signatures provided in the message body to its own. Finally, it will return a message containing a set of signatures and its top list of peers.

EXC-R `R` means that the message is the response to a previous request. In receiving this command, if the status of the peer is `waiting`, the system will add the peers and the signatures in the body of the message to its data files, as well as reset the sender's status.

ALG-I `ALG` means that this message is a request for an algorithm. It is sent when the sender has received a signature from this node signed with an algorithm it does not know and therefore needs to be provided with. On receiving this command, the receiving system sends back (in the form of an `ALG-R` message) the signed file containing the different functions necessary for full use of the new algorithm. If the system does not know this algorithm, nothing is done. In this case, the sender will not receive anything, and will ignore any signature created using the algorithm. This feature is not currently implemented. However, as seen in section 5.2.5, the system already supports several algorithms.

ALG-R A response to a request for an algorithm update which has an attached file digitally signed containing all the functions necessary for normal use of this algorithm. Upon reception, the system will verify the digital signature of the file, and if correct, add this algorithm to its set of algorithms (see section 5.2.5). If the digital signature of the file is not correct, the new algorithm, as well as all the signatures using this algorithm are deleted.

The script will perform the actions outlined above if and only if the maximum number of actions defined in the configuration file has not been exceeded. As seen in chapter 4, this value is used to prevent flooding of both the local user and the mail server, as well as spread the load across the network.

5.3.2 asn-check

This script takes a full message (including headers) as input, and adds a header stating whether the message is spam or not. In order to decide if the message is spam, the script first checks if the sender is on the whitelist. If the sender is not on the whitelist, the script compares the message to each signature in the signatures file. In order to do that, it checks which algorithm was used to create each signature and uses the comparing function from the appropriate algorithm. This function is different for each algorithm. This computation issues a score for this message. Comparing this score with the `spamthreashold` value from the configuration file, the script will add a `X-ASN-OK` or `X-ASN-WARNING` header, depending on which value is highest.

In the current implementation, this script, as well as asn-com, is launched by procmail (see [pro01]), using a .procmailrc file like the following:

```
SHELL=/bin/sh
VERBOSE=on

MAILDIR=${HOME}/mail
INBOX=${HOME}/mbox
LOGFILE=${MAILDIR}/log

:0
* ^X-ASN:
| perl /home/clark/asn/asn-com.pl

0:fw
| perl /home/clark/asn/asn-check.pl
```

```
:0:
* ^X-ASN-WARNING
/home/clark/mail/spam
```

This set of rules will first check if the message is an ASN message. If it is, the mail is sent to the asn-com script. If it is not, the message is piped to asn-check, which adds a header stating whether or not the message is recognized as spam. The message then goes through the last rule, which checks to see if the header stating spam is there. In case it is, the mail is finally delivered to a spam folder, thus never reaching the inbox, but without being deleted.

5.3.3 asn-exc

This script initiates the exchange of peers and signatures with the other nodes on the network. When invoked, it sends a request for exchange to the top nodes on its list, as well as a few others (determined using the `lastdate` attribute of the peers file, as seen in section 5.2.5). The numbers are determined in the configuration file, and thus can be customized to meet different needs. In the current implementation, there are 10 top nodes on a single peers file, and one other node on the list of available peers is also contacted. The script will send these peers a list of its top peers, as well as the signatures that have arrived since last time the peer has been contacted (this can be calculated with the use of the `date` attributes from signatures and the `lastdate` from the peers).

It has been seen in section 4.2.4 that the system must verify that the top nodes really deserve their high score. One of the ways to accomplish this, as explained in section 4.2.3, is for the system to take into account the availability of the nodes. This is done by the asn-exc script (along with the asc-com script). Every time a request for exchange is sent to a peer, the status of this peer goes from `ok` to `waiting`. Every time a response to an exchange is received, the asc-com script will change back the status of the peer to `ok`. Finally, every time the asn-exc is launched, it will go through the whole list of peers, looking for peers with a `waiting` status. Each one of them will then be reset to `ok` and their score is lowered to illustrate the fact that they are not available enough. Whether the reason for this is a lost message or a user with a transient connexion does not really matter.

The asn-exc, in the current implementation, is launched from a `cron` task on the server.

5.3.4 asn-clean

The other way (see section 4.2.3) of making sure that a top node really deserves to stay a top node is to lower its score if it does not provide relevant signatures for a long period. Every time a signature matches an incoming mail, the `date` of each of its senders is updated to the current time. This value can be used to detect if a peer on top of the list has not been providing relevant signatures for a long period of time. This duration can be set in the configuration file, and every time asn-clean is launched it will check this for every top peer. The peers that have not provided a relevant signature for the specified duration see their score being lowered.

Another function of asn-clean is to clean the peer files of peers with negative scores (and which are not banned) and the signature files of signatures that have expired.

The asn-clean, in the current implementation, is launched from a `cron` task on the server.

5.3.5 asn-sign

This script is invoked by the user when he recognizes a mail in his mailbox as spam. This script takes a mail as input, computes the relevant signature and writes it into the signatures file. The algorithm used to create the signature of the message depends on the implementation. In the current implementation, the default algorithm is set in the configuration file.

The asn-sign is launched from the mail client. For example, using pine, a printer was set (using the command line `perl ~/asn/asn-sign.pl`, for more details see [Uni01]). Once this was done, a simple click on the `%`

key (followed by validation) would declare the message as spam and add a signature relevant to this spam in the signatures file.

5.3.6 asn-revoke

This script takes a messages as input. It goes through the signatures file, trying to figure out if the input message appears. Since it has no way of knowing with which algorithm the message might have been signed, it must create all the possible signatures with the known algorithms and look for any of these in the signatures file. If such a signature is found, then it is deleted from the file.

The asn-revoke script, in the current implementation, is launched from the mail client, in much the same way as the asn-sign script, explained in section 5.3.5.

5.4 Signature Algorithms

The current implementation includes implementations of two algorithms, although update of the algorithm is not yet available. This means that different signature algorithms can live on the same system.

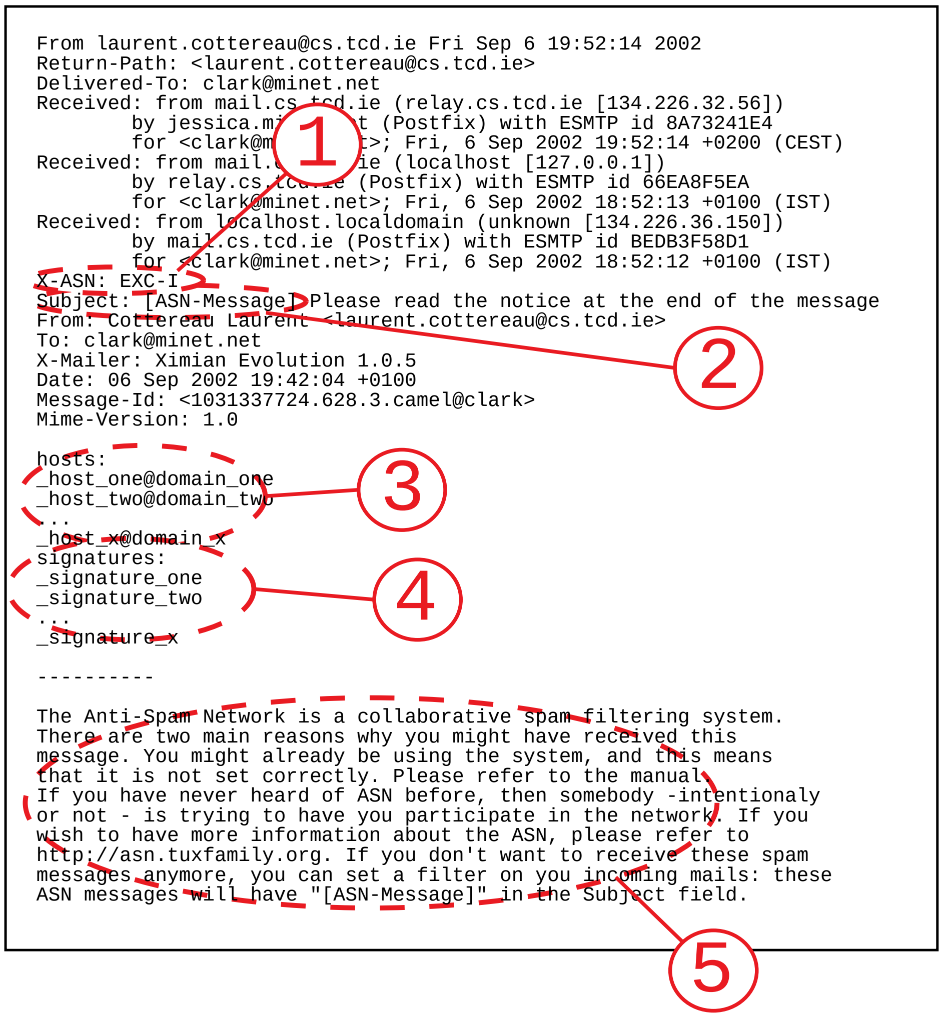
5.4.1 SHA1

The SHA-1 digest (code `SHA1`) simply takes the body of the message, and hashes it using the SHA-1 message digest algorithm in base 64 from the National Institute of Standards and Technology. This returns a 27 characters-long string. For more information on this digest, see [NIS02].

5.4.2 ASN1

The first ASN signing algorithm (code `ASN1`[1]) is also a SHA-1 signature in base 64. But before being hashed, the body is stripped of all the elements pointed out by Gabriel Matthews as being customization-prone (see appendix B).

[1]This stands for Anti-Spam Network first signature algorithm. It should not be confused with the Abstract Syntax Notation ASN.1.



```
From laurent.cottereau@cs.tcd.ie Fri Sep 6 19:52:14 2002
Return-Path: <laurent.cottereau@cs.tcd.ie>
Delivered-To: clark@minet.net
Received: from mail.cs.tcd.ie (relay.cs.tcd.ie [134.226.32.56])
        by jessica.mi[illegible]t (Postfix) with ESMTP id 8A73241E4
        for <clark@m[illegible]t>; Fri, 6 Sep 2002 19:52:14 +0200 (CEST)
Received: from mail.[illegible]ie (localhost [127.0.0.1])
        by relay.cs.tcd.ie (Postfix) with ESMTP id 66EA8F5EA
        for <clark@minet.net>; Fri, 6 Sep 2002 18:52:13 +0100 (IST)
Received: from localhost.localdomain (unknown [134.226.36.150])
        by mail.cs.tcd.ie (Postfix) with ESMTP id BEDB3F58D1
        for <clark@minet.net>; Fri, 6 Sep 2002 18:52:12 +0100 (IST)
X-ASN: EXC-I
Subject: [ASN-Message] Please read the notice at the end of the message
From: Cottereau Laurent <laurent.cottereau@cs.tcd.ie>
To: clark@minet.net
X-Mailer: Ximian Evolution 1.0.5
Date: 06 Sep 2002 19:42:04 +0100
Message-Id: <1031337724.628.3.camel@clark>
Mime-Version: 1.0

hosts:
_host_one@domain_one
_host_two@domain_two
...
_host_x@domain_x
signatures:
_signature_one
_signature_two
...
_signature_x

----------

The Anti-Spam Network is a collaborative spam filtering system.
There are two main reasons why you might have received this
message. You might already be using the system, and this means
that it is not set correctly. Please refer to the manual.
If you have never heard of ASN before, then somebody -intentionaly
or not - is trying to have you participate in the network. If you
wish to have more information about the ASN, please refer to
http://asn.tuxfamily.org. If you don't want to receive these spam
messages anymore, you can set a filter on you incoming mails: these
ASN messages will have "[ASN-Message]" in the Subject field.
```

Figure 5.1: Format of the ASN Message

5.5 Message Formats

Figure 5.1 details the format of an ASN Message. There are five main parts that describe this format:

1. The header X-ASN: *command* as described in 5.3.1.

2. The subject is always *[ASN-Message] Please read the notice at the end of the message.* The idea here is in case somebody receives the message, which should never reach the mailbox, this subject will make it easy for the user to filter these types of messages.

3. The peers part of the body of the message consists of the line *peers:* and then one email per line.

4. The signatures part of the body of the message consists of the line *signatures:* and then a signature per line. Each signature line consists of three parts:

   (a) The hops-to-live value. This value is not necessarily to be trusted. What this means is that when a user receives this signature, it will not necessarily take this hops to live value for granted, and will use it only if it does not exceed its own maximum hops-to-live value defined in the configuration file.

   (b) The code for the algorithm used to create this signature. This is the same four-character string that was found in the signatures file.

   (c) The actual signature.

   Following is an example of a line from a ASN message:

   `2 SHA1 eD67R-51uSzkj8Ueelkjsjj87/i`.

5. A explanatory paragraph about the different reasons possible for receiving such email into the mailbox. Anything can go there since everything after the first empty line of the body will be ignored by the ASN parser.

5.6 Alternative Implementations

Sections 5.1 to 5.5 presented a functional but somewhat minimal implementation of the architecture presented in chapter 4. the important aspect to keep in mind when considering this implementation of the anti-spam network is that the current implementation is really nothing more than one of the possible implementations. One of the objectives of the project, presented in section 1.2.2, was to design a portable architecture that could be implemented differently depending on the intended use, the underlying software, or other reasons.

One of the main ideas of the anti-spam network is to provide a way to exchange in a peer-to-peer manner signatures of spam. The architecture has been built on the assumption that what is received is not to be trusted fully. This means that it should be hard to attack, but it also means that most mechanisms can be adapted to meet different needs. As long as the protocol for communicating with other nodes is respected, what happens locally has no influence on what happens on the other nodes.

In order to solve the issue of the localization on the mail server, another way of filtering could be to implement a mail proxy on the local machine, which would be plugged between the MUA and the MDA (and not on the server). Although this would prove much trickier to implement than using the widely installed procmail, it would allow for reducing the heavy task of mail processing and filtering on the mail server, without losing the portability of the whole architecture.

Instead of backend scripts that are launched from the mail client in various and sometimes tricky ways, another possibility would be to extend an existing mail client. In the best of cases, the mail client is open-source, and this can be done in a straight-forward manner. Even if it is not open-source, many popular mail clients will accept plug-ins (like Eudora, or Microsoft Outlook). This approach would allow the filtering, signing and revocation to be much more integrated with the email client. It can even be imagined that the user would not even have to install the anti-spam network; he would just need to upgrade his normal email client, and the feature would become available. Of course, this option would be much more user-friendly than the current implementation, although a new implementation would be necessary for each email client.

Chapter 6

Evaluation

In order to provide a full and thorough evaluation of the architecture proposed in chapter 4 as well as of the implementation described in chapter 5, the only definite way is to run a full-scale test of the system. This test can prove that such objectives as scalability, portability, ease-of-use and reliability that have been described in section 1.2 have actually been achieved.

However, because of time constraints, this test is not possible within the scope of the project. Therefore, this chapter addresses each objective from section 1.2 individually in a non-empirical way and discusses the extent to which each one has been achieved. After this, the signature feature is discussed in more details.

6.1 Scalability

6.1.1 What is Scalibility?

Mads Haahr points out that

> There is no generally accepted scientific definition of what exactly scalability *is*, and people tend to rely on an intuitive understanding of the concept instead. [Haa98]

However, a definition, although vague, about scalability can be found. For example, George Coulouris explains that

> A system is described as *scalable* if it will remain effective when there is a significant increase in the number of resources and the

number of users. [CDK01]

This vision of scalability will be used in this section. What stays to be determined is what exactly is meant by *remains effective*. In the optic of the anti-spam network, the metrics will be functionality of the system. As the anti-spam network is a spam filtering system, what will be looked at is whether or not scale changes the way spam is filtered.

6.1.2 Scalability in the Anti-Spam Network

In this section, the scalability of the anti-spam network will be discussed. In order to do that, the scale parameters must be outlined first, as well the the metrics for the efficiency of the system.

Scale Parameters

There are several parameters than can vary with scale. The following is a list of the major ones:

Number of peers This is what is considered as the scaling of the anti-spam network.

Number of messages sent and received Since each user cannot receive or send a negative number of emails, when the number of users on the network grows, so does the number of messages sent and received throughout the network.

Number of spam received Since a user cannot receive a negative number of spam, when the number of users on the network grows, so does the number of spam received throughout the network.

Number of spam reported Since a user cannot report a negative number of spam, when the number of users on the network grows, so does the number of spam reported.

Number of false positive ports Since a user cannot submit a negative number of spam reports, when the number of users on the network grows, so does the number of spam reported.

Efficiency Metrics

The following is a set of metrics that will be used in order to discuss the efficiency — or functionality — of the anti-spam network.

Administration Overhead The amount of protocol messages exchanged between peers. Should this grow too important, the anti-spam network cannot be considered functional anymore.

The mail server load The message load on the server. If the anti-spam network generates a stream of messages that overloads the server, then the system cannot be considered functional.

The processing power required The total power required from the machines on the network. If this power grows in unmanageable proportion, then the anti-spam network cannot be considered functional.

The success rate The number of spam detected and filtered thanks to the system. If this number grows too small, the system is not functional. This metrics is subjective, since it depends on what the user is expecting from the network. At the same level as the success rate is the number of false positives wrongly filtered by the system.

Discussion

The aim of this discussion is to figure out the impact of the variation of the scale parameters on the efficiency metrics. The only direct impact of the increase in the number of users is the increase in administration overhead. Indeed, each user adds his own load of protocol messages to the total, since each user has his own channels of communication to the other peers (instead of, for example, using existing channels). This fact, possibly leading to overload or Denial of Service, has been recognized since design (see section 4.6.3 and section 4.6.4) and tackled by adding a threshold on the number of communications a single peer will handle before refusing any further one. This feature means that each peer will see its administration overhead increase with the number of users, rapidly reaching a maximum plateau, thus preventing overload and DoS.

The number of messages sent and received will have a direct impact on the load of the mail server. Although precise figures could not be gathered about a potential maximum load of mail servers, it is though that the upgrade of the hardware (memory, bandwidth, ...), or the duplication of the mail server, can solve any such problem. Indeed, this issue is really one of the mail infrastructure, which is considered scalable.

The increase of the message traffic also has an impact on the total processing power required by the system. Indeed, each incoming message needs to be compared to the spam signatures stored on the recipient peer. On mail servers where the filtering is similar as the one in the current implementation (see section 5.3), this might create a bottleneck on the server, as has been pointed out in section 4.4. However, if the filtering of the incoming mail is distributed on different hosts (e.g. remote users with extended mail client which support the system, as described in section 4.4.3), then the added load is globally distributed among the added users.

The increase of the number of spam received does not have any major impact on the functionality of the anti-spam network, since the increase of spam received does not necessarily mean more spam is reported. However, the increase in the number of spam reported will have an impact on the system. The administration overhead will slightly increase, as the number of signatures to be exchanged between peers will rise. However, the number of protocol messages does not increase, so the impact on the administration overhead should not prove to be too relevant.

With the increase in the number of signatures to which each incoming mail must be compared, there should be an increase in the total processing power needed to filter mail. However, as seen above, this should not be an issue, except for mail servers where the filtering is done locally. Indeed, an increase in the number of spam reported will result in an increase in the number of spam filtered. Considering this is the reason why people are using the system, and considering what was said in section XXXX about an increase of the hardware available on the network, this makes the anti-spam still functional in spite of an increase of the number of spam reported.

The last parameter that may increase with scale is the number of false reports. This might be the hardest to address without full-scale test, because it is very hard to reproduce the actions of a beginner or a resourceful spammer. What can be said is that, with trust management among the known peers, the design has provided a set of features that will lower the impact of false positives (see section 4.6.1), and potentially get rid of them (e.g. with such features as `spamthreshold`). There is a balance between success with spam and number of false positives and, hopefully, the success rate can still be significant with scale.

To conclude, without extensive deployment, the scalability of the anti-spam network is difficult to prove. However, the architecture of the system has features that address the main scalability issues, and fine customization should make scalability attainable. A point to keep in mind is that this fine customization might actually be hard to obtain or maintain.

6.2 Portability

This section will try to determine how well the objective of portability expressed in section 1.2.2 was tackled. In order to evaluate the portability of the anti-spam network, two issues are assessed: openness and heterogeneity.

6.2.1 Openness

George Coulouris defines openness as :

> The openness of a computer system is the characteristic that determines whether the system can be extended and reimplemented in various ways. (...) Openness cannot be achieved unless the specification and documentation of the key software interfaces of the components of a system are made available to software developers. [CDK01]

As the anti-spam network has been specifically designed with these ideas in mind, openness is quite well supported. What makes this possible is the fact that there is no implicit trust between peers. The only thing that needs to be respected to allow communication is the format of the emails. Apart from that, most features can be adapted depending on the needs, without major impact on the service.

Furthermore, the documentation for this system will be made available to everyone, probably through a website. And the current implementation will be released under the GNU General Public License (GPL, see [Fou91]).

6.2.2 Heterogeneity

The great asset of the anti-spam network is that it relies on the mail infrastructure for its communication. Given well-defined message format, SMTP forms a platform-independent transport protocol, therefore the transport layer of the anti-spam network is platform-independent.

The current implementation of the anti-spam network is not fully cross-platform. Although the scripts are written in Perl, and thus are available on all platforms to which perl was ported, the problem comes from the filtering, which requires procmail, which in turn requires a Unix mail server. However, the other implementations outlined in section 5.6 support a heterogeneous network.

To conclude, while the implementation described in chapter 5 does not support a heterogeneous network, the generic design of the anti-spam network, as well as its use of email as the transport protocol make it portable across many platforms.

6.3 Ease of use

The current implementation does not fulfill all the objectives of ease of use that had been set in 1.2. Although the reporting and revocation is quite easy and fast (two keys when using pine), there is a real improvement to be made regarding installation and configuration.

In the current implementation, because of the use of procmail for filtering, there is a need to use the anti-spam network server-side. This means that the user must have shell access to the server. Then, the user will probably need root access (or possibly good relations with the administrator) in order to install the few perl modules that are needed[1] or procmail if it is not already available on the server. Furthermore, some tasks must be run from cron, which adds another layer of complexity. Finally, all this is to be done on the command-line, which obviously does not fit most of the potential users of the anti-spam network.

The configuration file is text-based, using XML. At the moment, it needs to be edited by hand. The problem is that it is not obvious for a normal user how to write an XML file. This means that a wrapper is required to make it user-friendly, possibly a graphical interface. This wrapper has not yet been implemented.

The second issue with configuration comes from the desire for a design that could adapt to everyone's needs. This desire has led to the availability of many customizations. With all the possibilities offered, it might become hard for a new user to decide what values to use for items such as `spamthreshold`, without losing the balance between spam detected and false positives. The solution to this would come from a set of standard configuration files, fitting different standard types of users. But in order to come up with these, a full-scale testing of the system must be realized.

In all these regards, the current implementation fails to achieve the ease of use aimed at. However, once again, the anti-spam network provides with a generic design. This design can and will be used to implement much more ergonomic solutions. In particular, extending popular email clients might make the installation and use much more user-friendly. In particular, there would be no need to have any kind of access to the mail server, which makes it usable with the regular ISPs. Finally a long full-scale testing should provide means of solving the remaining issue outlined above.

[1]XML::Simple and Net::SMTP

6.4 Reliability

The reliability issue of the anti-spam networks covers two main aspects. The first is the possibility for the system to classify a relevant message as spam and the other is the ability to resist attacks from people attempting to make the system unusable (possibly spammers).

6.4.1 False Positives

There are two main means of having a proper email detected as spam by the anti-spam network. The first one is a collision of the signature algorithms. It is possible (however highly rare with the current implementation) that two messages will get the same hash. If one of those message is a spam and get reported, and the other is a proper mail, then the proper mail will be considered spam too. The only solution to this problem is to come up with signature algorithms which keep the probability of collision low. This is the case in the current implementation, since a SHA-1 Digest is used (see [NIS02]).

The other reason for false positives is much harder to prevent, it is a reporting of a proper message as spam (by mistake or not). If the reported mail is a bulk email (for example, from a mailing list), then it might be detected as spam for another user of the system. It was decided in 4.2.6 that a global revocation functionality was not to be implemented, however, the local revocation, according to the experience of other collaborative spam filtering systems (see [raz]), would be enough to restrict the risk of false positives mainly to the malevolent reporting of proper mailing lists.

Furthermore, the trust management built at the core of the anti-spam network design should make that malevolent reporting very hard to achieve. Indeed, a malevolent user would have to report proper spam, in order to get a high rank, and only then would he have a chance to report for sure a false-positive. Even then, if it gets to that, a few functionalities make sure that matters do not get out of hands. Whitelisting (see 4.2.7) will ensure that no critical email gets detected as spam, and banning (see 4.2.8) will ensure that a malevolent user who came around the trust management system will not be able to keep making false reports.

Finally, the last way to fight false positives can be to raise the threshold for spam. This will make getting false positives much less probable, but in the same time, less spam will get detected. The remaining problem, of course, is to be able to find out the good balance between spam reported and false positives. This is likely to require actual development, and even then will still depend on individual users' preferences.

It is hard to get a definite idea on the ability of the anti-spam network to cope with false-positives without deployment, because false-positives, as shown above, come from conditions that are hard to reproduce: incapable or clumsy users, or malicious and resourceful spammers. However, the ability to upgrade the software and the signature algorithms transparently from the rest of the network is an important safeguard.

6.4.2 Resistance to Attacks

It would be very naive to believe that every trick a person might come up with to attack the anti-spam network has been imagined and cared for. However, the main threats to the network have been tackled (see 4.6). And once more, the possibility for the user to upgrade the software and the signature algorithms, as well as features reducing the impact of attacks on critical email, are the main defenses of the system against attacks.

6.5 Signature

What has been identified in 1.2 as the main possible weakness of the system is the signature process. There are three main aspects to consider when trying to evaluate the signature process: the impact on performances of the system (local and global), the success rate, and finally the update of the algorithm.

6.5.1 Performance

The current implementation is more centralized as the ideal one. The main issue with that is that with such an implementation, the mail server might become a bottleneck for the system, especially as signature algorithms will grow more and more complicated. Of course, and once more, the solution lies in a higher degree of distribution.

If the load is distributed, there should be no real issue with performance, as personal computer power is increasing. Furthermore, processing power on personal computers is often unused, as the success of Internet computing applications such as the SETI@home project (see [KWA$^+$01]) has shown.

Should the anti-spam network be widely adopted by the community, the load from servers might actually be relieved, since some of the current spam filters on the server might be removed. Most mail servers have an impressive set of such filters (see Table 3.1), that are applied to each incoming mail, thus using a large percentage of the available processing power. Because the anti-spam network does not specialize on one type of spam detection (as the filters seen in 2.3 and 2.4 do), it can potentially replace all the filters, thus preventing the server from wasting power on spam filtering.

The remaining issue that some companies would argue about is the fact that removing the spam at the server level prevents the waste of resources used to deliver and save the spam in the mailboxes until its filtering. As explained in 1.2.5, this issue was not an objective tackled during the project design and implementation. Furthermore, it can be argued back that resources such as hard disk space (needed for copy of the spam until delivered locally) is by far cheaper than the price of a potential mail server breakdown.

6.5.2 Sucess Rate

The two signature algorithms already available with the current implementation should be quite effective with the current state of the spam, as shown by G. Matthews' work. However, several spam filters editors have made the same kind of conclusion over the last year, and customized their filters to take them into account.

This has forced the spammers to adapt their messages, their main tactic involving the addition of series of random characters at the beginning or end of a message. This kind of customizations is not dealt with by the current algorithms. The possible solution is the use of fuzzy algorithms, and although this has not been implemented, the anti-spam network can be considered to deal with such changes because of its update feature.

6.5.3 Update of the Algorithm

The main issue with the update of the algorithms feature is, as with any such decentralized distribution of a piece of software, the security of the procedure. However, the use of a PKI to certify the new signature files should be enough to keep the risks near-zero. There are several free open-source PKI currently available, such as NewPKI (see [new]), and their power has been demonstrated by practice.

An important point to keep in mind about the update of the algorithm is that only algorithms in perl are supported by the current architecture. The reason for this lies in the way the algorithm is ditributed. When a peer receives a signature whose algorithm it does not know, it asks the sender of the signature for the algorithm, and if it cannot get the algorithm through this peer, then the signatures are dropped. Then if it can, the algorithm is assumed to be working, without any other checks about programming language for instance. The use of another language is not necessarily impossible to incorporate, but it requires more controls and data structures.

Chapter 7

Conclusion

This thesis has described the analysis, design and one implementation of a peer-to-peer architecture for collaborative spam filtering. The architecture relies on human detection of spam and on the existing mail infrastructure for transport of administration messages. It also defines a set of commands for inter-communication between peers.

7.1 Objectives Fulfilled

Although full-scale deployment and tests were not possible within the scope of the project, chapter 6 has shown that the architecture of the anti-spam network presented in chapter 4 addresses all the objectives that were outlined in section 1.2.

However, it has also shown that the current implementation proposed in chapter 5, in particular in the area of ease-of-use, described in section 1.2.3, in not yet fully functional. Nonetheless, other characteristics of the architecture, such as the extensibility of the system to other implementations presented in section 5.6 have been outlined to balance this issue.

To conclude, it can be said that some more work in implementation is all that is needed for the anti-spam network to be ready to be tested in a full-scale real-life environment.

7.2 Future Work

Although the architecture of the anti-spam network is ready to be tested in full-scale, some possible work could be conducted to further improve the design or architecture. They are described here, as they constitute examples of possible future work on this project:

- XML is currently used for the storage of data, and a proprietary-format is used for exchange of data. Some work could be done in this area. Firstly, it might be interesting to try and exchange data in XML format. This might pose some problems (like size of the messages), but the advantage would be certain, which makes these problems worth a study.

  Secondly, the proposed DTD for each different types of data is not necessarily optimum, in the sense that it uses a lot of XML attributes, which might be a problem for future implementations. Indeed, XML attributes cannot contain multiple attributes, are not easily expandable, and cannot describe structures.

- An interesting work on the algorithm update could be to try and find a way to allow for scripts written in other languages. This would definitely improve the openness of the architecture.

- There is a need for a study of the different possibilities of configuration, in regard to the expected reliability and success rate. Such a study could solve the issue tackled in section 6.3, by possibly offering standard configurations for different types of users.

- There is an important work to be done in finding new algorithms that will create signatures more adapted to spam filtering. The distributed aspect of the anti-spam network allows for much more complex and processor-intensive algorithms than can be used with current collaborative spam filters. This extra available power should be used.

Appendix A

Study on the Customization of Spam

This appendix is based on an unpublished study by Gabriel Matthews, carried out in the year 2001. His study focuses on the analysis of the customization of spam messages. Two messages representing the same spam (i.e. the main part of it is similar), are rarely exactly the same. The little differences, introduced by the spammmer to disturb spam filters, are called customizations. This study is trying to analyze which types of customizations are most frequent.

A.1 Types of Customizations

Most of the customizations that are useful to spammers are customizations that will not alter the sense for the reader. A good example would be a *chaNge Of cAse*; this change will not prevent the reader from understanding the mail, but, for a computer, it will make this message different from one displaying *cHange of Case*. The Following is a list of the different types of customizations taken into account by this study.

Numbers These are digits in the mail. They can be fax or telephone numbers, as well as random series of numbers created to bypass a potential spam filter.

URLs URLs and mailtos with javascript embeded are not counted in this type. In order to count them only once, they are counted as javascript.

Punctuation These include periods, series of more than one star (*****), series of more than one underscore (_____). This type also includes the change of case.

White Spaces They include blank lines, spaces, tabulations.

Javascript This is all javascript tags and content, as well as embedded javascript in mailtos and URLs.

HTML This is made of the HTML tags that might appear in a message.

Word Order This is a change in the order of the word in the email. As this is harder to achieve automatically, this is much more rare.

A.2 Test Plan

A series of thousands of spam is passed through a filter that removes Numbers, URLs, Punctuation, White Spaces, Javascript and HTML. The remaining texts are compared (using the unix function `diff`) in order to find the spams that are similar. This method is not necessarily optimum, in the sense that it certainly lets a few similar spam go undetected. However, it has the advantage of making sure that spam declared similar with this method are indeed similar. After this step, there were a series of 902 pairs of similar spams.

The second part of the process is to come back to the unstripped version of the message, and then manually (by using `diff` once more) check what kind of differences appear between two similar spam.

A.3 Results

The results of the study are summarized in table A.1 and presented in a pie chart in figure A.1. They show that the two most important repositories of customizations in spam messages are white spaces and javascript. This makes sense since white spaces will easily go unnoticed to the reader's eyes, and javascript is often not displayed. Therefore, a spammer can safely make

	Java-script	URLs	Numbers	White Spaces	Punc-tuation	HTML	Word Order
Difference	159	69	112	197	36	30	2
Percentage	34%	15%	24%	42%	8%	6%	0%

Total comparisons with zero squared differences	902
Total of those with actual differences	471
Percentage with actual differences	52%

Table A.1: Study on Customization of Spam

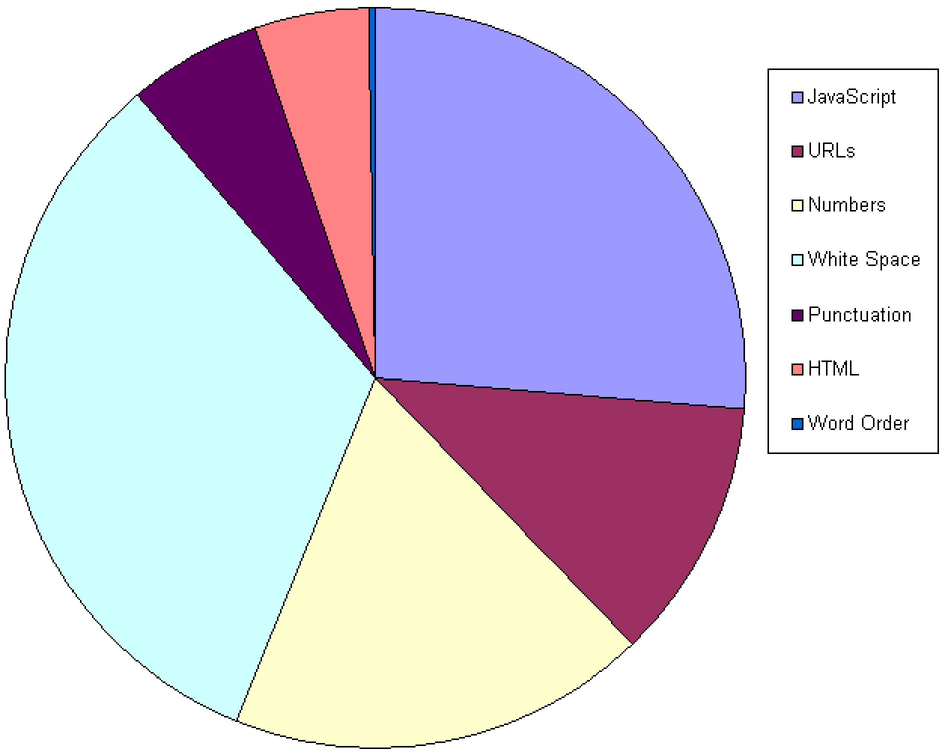


Figure A.1: Study on Customization of Spam

slight changes to these parts without fear of changing the meaning of his message.

Appendix B

Analysis of the body of spam messages

In this appendix, an analysis of the average composition of spam is conducted. This study uses the conclusion from the study conducted by Gabriel Matthews and reported in appendix A. It consists in the analysis of a set of 5846 spam messages collected from different mail accounts between 2001 and 2002.

B.1 Test Plan

Each spam message is processed by a perl script that strips each different customization-prone part (see section A.1 for explanation on the different types of customizations). The size of the message after each step is recorded. This allows for a precise idea of the repartition of the message between these different types, as well as the size of the actual content of the message (the rest).

B.2 Average Composition of Spam

The average values for the whole set are then computed and shown in figure B.1. They show that, even after all the stripping is done, an average of more than two-thirds of the message is left. The second conclusion is that the most important type of customization-prone part is HTML.

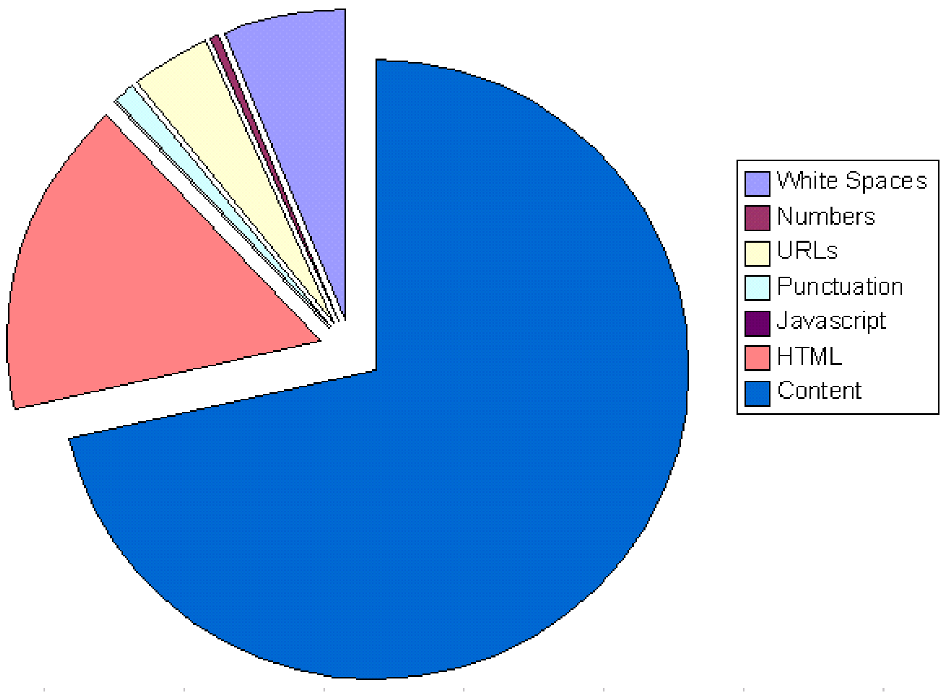


Figure B.1: Average Spam Composition

B.3 Size of Stripped Messages

The previous results are not enough. This study has been conducted because in order to choose a signature algorithm for the messages, a better knowledge of the spam messages was needed. The problem here is that the average size is not necessarily enough. What matters if the signature algorithm is supposed to use a fuzzy hash is not to use it on too small a message, because the risks of collision are getting too high.

In order to get an idea of this risk, the results shown in figure B.2 were computed. This figure shows the number of spam message whose size is lower than a certain size after being stripped down of the customization-prone parts. It shows that the number is insignificant under 256 bytes. It also shows that a significant part (3%) of the number of messages stripped down weights less than 512 bytes.

Getting an idea on the exact probability of collisions with fuzzy hashes

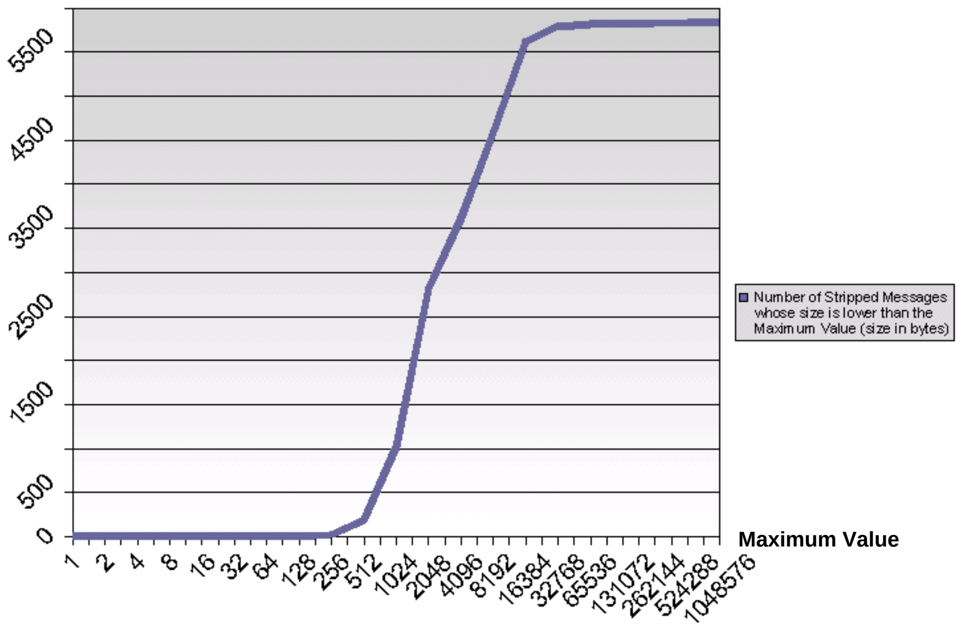


Figure B.2: Size of stripped message

from these values requires a mathematical study which is beyond the scope of the present work.

Bibliography

[AKC$^+$00] Ion Androutsopoulos, John Koutsias, Konstantinos V. Chandrinos, George Paliouras, and Constantine D. Spyropoulos. An evaluation of naive bayesian anti-spam filtering. In *Proceedings of the workshop on Machine Learning in the New Information Age*, number 11, pages 9–17, 2000.

[aol97] Parker v c.n. enterprises press release, September 1997. AOL. http://legal.web.aol.com/decisions/dljunk/parkerp9.html.

[bla] Blackhole. http://the.groovy.org/blackhole.shtml.

[cau] Cauce web site. http://www.cauce.org/.

[CDK01] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed System, Concepts and Design.* Addison-Wesley, third edition, 2001.

[cgi] Cgi spy. http://www.cgispy.com/.

[cnn] Cnn.com science and technology. http://www.cnn.com/TECH/.

[Cos97] Bryan Costales. *Sendmail.* O'Reilly, second edition, 1997.

[Cro82] David H. Crocker. Standard for the format of arpa internet text messages. Request for Comments 822, Internet Engineering Task Force, August 1982.

[dcc] Distributed checksum clearinghouse frequently asked questions. http://www.rhyolite.com/anti-spam/dcc/dcc-tree/FAQ.htm.

[Elk] Michael Elkins. *The Mutt E-Mail Client.* version 1.4.

[Fou91] Free Sofware Foundation. *GNU General Public License*, June 1991. version 2.

[Gel01] Tom Geller. Spamcon foundation news, August 2001. Issue 8.

[GM96] James Gosling and Henry McGilton. *The Java Language Environment, A White Paper.* Sun microsystems, May 1996.

[Haa98] Mads Haahr. Implementation and evaluation of scalability techniques in the eco model. Master's thesis, University of Copenhagen, 1998.

[KS] Satheesh Kolathur and Subha Subramanian. Collaborative spam filter. http://www.cs.uh.edu/ kolathur/comp527project.html.

[KWA$^+$01] Eric Korpela, Dan Werthimer, David Anderson, Jeff Cobb, and Matt Lebofsky. Seti@home: Massively distributed computing for seti. *Computing in Science and Engineering*, 3(1):78–83, 2001.

[map] Mail abuse protection system. http://mail-abuse.org/.

[Mar02] Carolyn Duffy Marsan. Standard may bring order to e-mail chaos. *Network World Fusion*, July 2002.

[new] Newpki. http://www.newpki.org.

[NIS02] NIST. Secure hash standard. *Federal Information Processing Standards Publications*, 180-2, August 2002.

[Pos82] Jonathan B. Postel. Simple mail transfer protocol. Request for Comments 821, Internet Engineering Task Force, August 1982.

[Pra] Vipul Ved Prakash. Razor. http://razor.sourceforge.net/.

[pro01] *Procmail FAQ*, April 2001. v1.358.

[raz] Archives of the razor-users mailing list. http://www.geocrawler.-com/archives/3/2539/.

[ric] Ricochet, automated agent for tracing and reporting internet junk email. http://www.vipul.net/ricochet/.

[SDHH98] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk email. July 1998.

[Sho01] T. Showalter. Sieve: A mail filtering language. Request for Comments 3028, Internet Engineering Task Force, January 2001.

[spaa] Spam assassin. http://spamassassin.taint.org.

[spab] Spam bar. http://www.spambar.com/.

[spac] Spambait. http://www.unicom.com/spambait/.

[tmd] Tagged message delivery agent. http://tmda.net.

[Uni01] University of Washington. *Pine Man Pages*, October 2001.

[War91] David Warsh. Nobel winner coase blends theories of economics law. *The Boston Globe*, page 63, October 1991.

[WCO00] Larry Wall, Tom Christiansen, and Jon Orwant. *Programming Perl*. O'Reilly, third edition, 2000.

[Woo99] David Wood. *Programming Internet E-mail*. O'Reilly, 1999.