ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed
Computing Group

Spamato

A Collaborative Spam Filter System

Nicolas Burri

Diploma Thesis

November 4, 2003 – March 3, 2004

Supervising Professor: Prof. Dr. Roger Wattenhofer
Supervising Assistant: Keno Albrecht

Preface

Abstract

This diploma thesis had the main goal to develop and implement a collaborative spam filter tool. A group of users should be connected to an anti-spam network allowing them to block spam messages cooperatively.
The implemented filter uses a server which accepts spam reports of all connected clients to collect information about newly spread spam mails. For every reported message the server generates an identifier (often referred to as "fingerprint") that is stored in a database for later lookups. To use the filter, a client needs to generate a corresponding identifier of all incoming messages and check if these fingerprints can be found in the database. If so, the mail was reported as as spam before and can be removed from the mail box. Simplified, a spam mail has to be reported once, to remove it from all users' mailboxes.
The main problem of this approach is the fact that spam mails often contain personalized or randomized parts making it difficult to generate a fingerprint suitable to identify copies of the same advertisement, received by different users. Even though the whole text of a spam mail may be randomized it usually tries to make its recipient visit a linked website. Therefore, our filter uses the domains linked within the message to generate a fingerprint of the mail ignoring the remaining text.
To allow future extensions and a higher level of portability, a framework with the name *Spamato* was developed, which serves as a container for spam filters. Since Spamato offers a standardized mail client interface, a mail program does not need to know about the registered filters to check if a message is spam. The client simply sends the message to the Spamato system, which internally uses all registered spam filters in parallel to test the message.
Finally, an add-in for Microsoft Outlook was developed which connects to the local Spamato system. The add-in uses Spamato to spam check incoming mails and also offers a graphical user interface to the filter system.
Once these three components had reached a certain stage of development we were able to successfully run a beta test with a small group of users utilizing Spamato to check their everyday email traffic.

Acknowledgment

First of all, I would like to thank my parents and my brother for their support during all the years of my studies.
A big “thank you” goes to the brave Spamato testers who made it possible to proof that the system actually works. I am also thankful to all people who proof read this thesis and gave helpful ideas.
I am highly grateful to my advisor Keno Albrecht for his extensive support during the last four months. Working together with him was a lot of fun. I am also grateful to Prof. Wattenhofer and his team for giving me the possibility to write this thesis in a very pleasant environment.
Last but not least, I thank my “open space” companions for our refreshing afternoon meetings.

Contents

1 Introduction

1.1 Problem Description

Over the last years, unsolicited bulk mail, better known as spam, has become one of the most annoying problems of the internet. Practically every email user gets some unwanted bulk mails on a regular basis and there is no way to protect an active email address from becoming a spam target. Even though the careful use of the email system may help to keep the number of incoming spam messages low for a while, there is always the risk that the wrong people get to know about the address. A link on a website, the subscription to a newsletter, the participation in a contest, a virus or worm on the machine of a friend... There are dozens of possible situations where an address can get into the hands of a bulk mailer, and once it is out, there is no way to prevent those advertisers from sending a flood of "great deals". While it is annoying for home users to clean their personal in-boxes from spam, companies suffer in two ways from the bulk mails they receive: Their employees need a lot of time to scan through in-boxes full of spam, extracting the good mails. Even worse than the time lost in this process is the risk of missing an important message, which increases with the number of spam mails present. On the technical side, due to the waste numbers of spam mails the mail servers need to handle a much higher workload, which can lead to system slowdowns and therefore to a reduced efficiency of the company's workflow.

1.2 Spam In Numbers

The constantly increasing bandwidth and calculation power in combination with good profits allow the spammers to send more and more messages. In 2001, Ronald Scelson, one of the worst spammers at that time, confessed in an interview [1] to send about 18 million messages each day. In 2003, Shane Atkinson was exposed [2] to be one of the key players in the spam business. He admitted to have sent up to 100 million messages on good days. Even though it is impossible to verify the accuracy of such numbers it is clear that the amount of spam messages is exploding and there is no end of this development to come. America Online (AOL) claims [3] to stop up to 2.4 billion spam mails each day, which means about 15'000 messages per user and year. Spam watchers

all over the planet recently reported that spam makes up more than 50% of all mails sent now. According to statistics published by Brightmail [4], the largest commercial anti spam company, in January 2004, even 60% of all mails they checked were spam! These incredible numbers make clear that spam has become more than a simple nuisance and that something needs to be done.

1.3 Solution Proposals

Since practically everybody is affected by this problem, there is a wide public interest in getting the situation under control.

1.3.1 Laws

Especially politicians and the yellow press call for laws to forbid spamming. While the United States already have enacted such a law [5] the European Union is working on one [6]. Most experts highly doubt though that such laws will have any effect on the spamming business, since the email system is an international service and local laws can be avoided easily. Spammers may either place their mail servers in countries with a low interest in prosecuting foreign investors or rely on badly configured or hacked third party servers which allow anonymous users to send mails.
Also the fact that there are only a few lawyers and politicians with a sound knowledge of computer science makes it improbable that an effective anti spam law will be presented any time soon.

1.3.2 A New Mail Protocol

Another proposed approach is to change the mail protocol. The currently used Simple Mail Transfer Protocol (SMTP) [7] has been in use for more than twenty years and was not designed to deal with spam. It does not offer any way to verify the identity of the sender of a message, since the whole mail header information can be faked. It is obvious that especially spammers like to take advantage of this weakness. While a new protocol might offer better security and perhaps even stop spam, it will take years to develop a successor for SMTP that reaches a comparably high level of acceptance as the current standard. There are countless applications that include mail functionality and not all of them will easily be adapted to a new standard. A long migration time will be necessary, during which both protocols must be supported to allow a smooth email communication at all times.
Big email providers like Microsoft, AOL or Yahoo have started to develop their own proprietary protocols [8] which should make it impossible to fake the senders identification, but all of them have failed to gain widespread adoption so far.

1.3.3 Spam Filters

Today the common way to deal with spam is to use spam filter tools. These programs try to detect and remove spam messages automatically before the user gets to see them. Filters can either run directly on the mail server or on the client side depending on the way they identify spam. While filtering obviously does not help to reduce the amount of spam received, it is currently the only working approach to reduce the number of messages a user otherwise has to delete manually.
Spam filter systems have become sophisticated pieces of software, which often include clever algorithms and heuristics to identify bad messages. However, the perfect filter that catches all spam messages but never marks an innocent mail does not exist yet and probably never will. While it may be impossible to reach perfection, there is still a lot of room for improvements. Especially collaborative systems that let different users work together to filter their spam have a great potential which is only used to a limited degree so far.

1.4 Goals Of This Work

The main goal of this diploma thesis has been to design and implement a collaborative spam filter, which benefits from the potential that lies in this approach. After some evaluation of the available systems it became clear that one filter would never be able to identify every incoming spam message. This is why we have decided to extend the initial task and also created a simple framework that allows to embed other Java based spam filters, which can be developed as part of future works.

2 Spam Filter Systems

2.1 Overview

Before we started to design our own spam filter, we analyzed existing anti spam tools and tried to assess their individual strengths and weaknesses. We evaluated the filters by comparing the percentage of incoming spam mails caught to the number of wrongly marked innocent mails. While it usually does not matter if one spam mail slips through the filter uncaught, even a few so called "false positives", where good mails gets marked as spam, lead to a mostly useless filter tool. A high spam detection rate alone is worth nothing if the user has to check the spam folder every day to make sure no important message was accidentally removed as spam.
All working filters we have found can be classified in four groups:

- Blacklists
- Rule based filters
- Bayesian filters
- Collaborative filters

The next section gives a short overview over these four approaches. After that, our filter designed as part of this work is presented. The last section of this chapter gives a short introduction to hybrid spam filter tools, which avoid the limitations of individual filters by combining several different spam filters in one tool.

2.2 Spam Filter Concepts

2.2.1 Blacklist

Description

Blacklists like the MAPS realtime blackhole list [9] are the oldest approach to filter spam messages. A central list, usually managed and altered by an operator, is used to store information about the addresses of mail servers sending spam. Especially ISPs, hosting mail accounts for many users, used to apply this filter technique and simply blocked all messages coming from a server listed on such a blacklist.

Advantages

The main advantage of blacklists is that they do not only stop a current spam message but also every future spam from the same server. Spammers are therefore forced to change the IP addresses of their servers regularly and cannot continously send spam from the same server over longer periods.

Disadvantages

Blacklists have a lot of disadvantages and only the worst ones are presented here. The worst problem about blacklists is that they produce a lot of false positives. Since spammers often use hacked or open mail servers to send their advertisements, these abused servers can get blacklisted. As a result, the good mails sent over these servers will get blocked as well by any ISP using the blacklist. A study from 2001 claims that the MAPS RBL has a false positive rate of more than 30%, which is absolutely unacceptable.
Since blacklists only stop spam from known servers, they miss a lot of unwanted messages, because the sending servers may not be known yet. The study mentioned above also reports a spam detection rate of less than 25% for the MAPS list — every other spam filter approach offers a drastically better performance. Also, the users of a blacklist usually do not have a chance to influence which servers are blacklisted but have to trust the list owner. In the past, several scandals became public where blacklist operators abused their power and put mail servers on their blacklist which never sent out any spam but belonged to someone who fell in disgrace with the list owner.

Conclusion

Blacklists have so many severe disadvantages that they are not regarded as a feasible solution any more. While there are still several blacklists in use, most providers do not solely rely on them anymore but use modern hybrid systems, which may include a blacklist amongst other filters.

2.2.2 Rule Based Filter

Description

Rule based filters like the widespread Spamassassin [10] system analyze the content of an incoming message according to a given set of filter rules. Header, subject and body of the mail are parsed and a spam probability is calculated depending on how many spam rules match.

Advantages

Rule based filters are very versatile and can be used directly on the mail server or the client side. Since the efficiency of these filters heavily depends on the quality of the rule set used, a centralized system has the advantage that one

administrator can keep the system up to date for all users. A client side installation offers more flexibility though, since each user may set up individual rules.

Disadvantages

Since spam messages are constantly adapted to pass as many filters as possible, it is necessary to keep the filter rules up to date. Especially novice users may find this task difficult and can be forced to rely on external sources to get new rule sets.
Another problem is that rule based filters tend to block mails with exotic text layouts. For example, most of these filters have a rule that marks a message as spam if it contains too many words in capital letters. While in most of the cases a mail triggering this rule really is spam, there are also innocent messages that are written in capitals and therefore get blocked. Hence, to reduce the number of false positives, a rule based system usually requires a mail to match several conditions before it gets marked as spam. The perfect tradeoff between filter efficiency and a minimum amount of false positives is hard to find and needs constant adaptation of the system.

Conclusion

Rule based spam filters are superior to blacklists, but when configured to catch a high percentage of incoming spams they also mark a certain number of innocent mails as spam. As a result, tools like Spamassassin started to migrate into hybrid systems and allow to include other spam filters as rules. While this trick is rather simple to implement, it leads to a much better filter performance. Nevertheless, the constant need to keep the rules up to date makes rule based tools inferior to the bayesian filters presented in the next section.

2.2.3 Bayesian Filter

Description

Bayesian filters are a sub-species of rule based filters and are currently often named as the best spam filters available. Bayesian filters do not use a static set of rules to identify spam but rely on dynamically generated token lists. A token can be anything from a letter to whole phrases, depending on the algorithm used. The filter keeps two lists: one for good tokens, often seen in normal mails, and one for bad tokens, which appear in spam messages frequently.
Every incoming message is parsed and the tokens it contains extracted. The filter then decides depending on the good to bad token ratio found, if the message is spam or not.
Due to the high popularity of this approach there are a lot of different implementations for various mail clients and operating systems available. Paul Graham, one of the leading heads in bayesian filter development, maintains a very informative website [11] where most of the working filters are linked.

Advantages

One of the best things about bayesian filters is that they do not need to be configured but learn from the user's behavior. If the user removes a message the filter identifies the important tokens within the mail and adds them to the bad tokens list. After a short time, this list contains most of the suspicious tokens and incoming spam is detected at a high rate.
Good mails, which are not removed as spam, are also parsed and the detected tokens are added to the good tokens list. This list is responsible for the low rate of false positives bayesian filters are known for. A mail containing several good tokens will not be marked as spam, even if there are some bad tokens present. Since the token lists are compiled individually, the filter adapts to the user's environment, allowing an employee of a bank to receive mails containing text about loans and credits while the same messages are rated with a high spam probability if they are received by another user.

Disadvantages

One weak point of the bayesian filters is their need to identify tokens within a mail, since the spammers can use various techniques to cloak their messages. The simplest way to do so is to misspell suspicious words or to insert additional characters. For example, the classic amongst the spam words "sex" can also appear as "sèx" or "s.e.x". If the message is encoded as HTML, there are even more ways to confuse the filter. For example HTML comments may be inserted, which are not visible to the user but to the filter. While the user sees "free cash" the HTML code may look like this:

```
fr<!-- com1 -->ee c<!-- com2 -->as<!-- com3-->h
```

While it is possible to deal with simple attacks like these there are ways to write a message which leave no tokens detectable. Figure 2.1 shows an easy example how an HTML table and a monospaced font may be used to write an undetectable spam message. An HTML table with one row and invisible borders is generated. The corresponding letters of each line are put together in one table field and html line breaks (`<br>`) are used to split them up again. While the user sees the text as usual from the left to the right, the characters appear top down in the HTML code. Even a good parser will have trouble to decode this kind of message correctly.

Another weakness are very short mails which only contain a link to an image with the advertisement. The bayesian filters completely fail here, since these messages do not contain any useful tokens at all.
Finally, the spammers may also abuse the good tokens list to make their messages look better. The names of friends and business partners become good tokens, since they appear in good mails frequently. Therefore, the spammers have started to add hundreds of names to their mails hoping to include at least some of the names the receiver has on the good tokens list. In certain environments, like the banking business mentioned before, this effect may even lead to problems without any modification of the spam messages. The terms "credit"

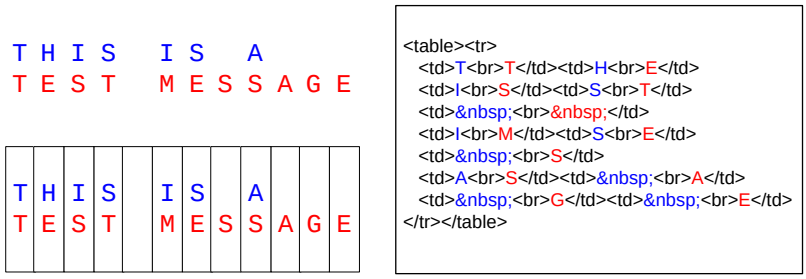

Figure 2.1: Table with one column and HTML line breaks to separate the characters in the cells. The resulting HTML code does not contain any readable strings and is hard to parse.

and "loan" are surely on the good tokens list of a bank employee for obvious reasons. As a result, the spam mails concerning financial topics will not be filtered for these users.

Conclusion

Bayesian filters are surely among the best filter concepts currently available. They are flexible, offer a good detection rate and produce a low number of false positives. However, there are still some mails they will never be able to detect (e.g. image only mails) and it has yet to be seen if the spammers will not be able to come up with another way to hide the critical words and phrases within their mails, rendering bayesian filters a lot less effective.

2.2.4 Collaborative Filter

Description

Collaborative filters do not try to identify spam mails automatically but rely on user feedback. The main idea is to connect a large number of users who all report their spam mails to a well known hub. This central entity generates an identifier for each reported message (based on its content) and stores this so called "fingerprint" for later lookups. Whenever a client receives a mail, the corresponding identifier of that message is calculated and compared to the central list of known spams. If the message has already been reported by any user, its identifier will be found on the list and the mail will be marked as spam.

Advantages

The clear advantage of collaborative spam filters lies in the fact that they use the best spam detector possible — the human user himself. If there are enough users cooperating, the individual participant does not have to report a lot of messages any more, since only the first few recipients of a new spam mail need to report it and all other receivers will have the message removed automatically. Also, the risk of false positives is low, since they only happen if malicious users wrongly report a good message. Complex trust systems can be used to reduce

the rate of false positives by malicious users even further, creating the probably most efficient spam filter concept available.

Disadvantages

The efficiency of collaborative spam filter tools directly depends on the quality of the fingerprints generated for the reported messages. It has been a long time since the spammers have started to include randomized and personalized parts in their mails, which look different for every recipient. Therefore, it is difficult to calculate an accurate identifier for a message that also matches a slightly different looking version of the same mail received by another user.
Collaborative systems also have to deal with malicious users reporting good mails as spam. While it does not matter if an individual message is reported as spam (since its fingerprint will not match any other mail) a reported newsletter leads to problems. Therefore, most collaborative systems offer a client side whitelist which allows the user to stop the system from checking certain mails. While these whitelists work pretty well in most of the cases, they still have to be seen as a workaround and not as a solution to the problem.

Conclusion

Collaborative spam filter systems have a great potential since they use the human user as a spam detector and therefore don'tdo not care about the content cloaking mechanisms used by the spammers. The best known collaborative spam filters, the Linux tool Vipul's Razor [12] and its Microsoft Windows counterpart SpamNet from Cloudmark [13], currently connect about 900'000 users to the largest anti spam network existing. The main problem of this network is that it uses a cluster of servers to store the list of reported spams. If these servers fail due to a denial of service attack or other technical problems the whole system may become inoperative.
An interesting approach was recently presented at Berkeley university. As part of a diploma thesis [14] a collaborative tool was designed which does not require a central server architecture but is built on top of the P2P system Tapestry [15]. The system splits the fingerprint of a message in several parts and spreads these fragments over the P2P network. A later lookup only needs to find a subset of the components of the corresponding fingerprint to identify a spam message. While the algorithm generating the fingerprints needs some modification to make it more reliable, this system is a very promising step in the direction of a fully decentralized collaborative spam filter. A very similar approach was also suggested in [16].

2.3 Our Approach: The Collaborative URL Filter

Description

As part of this work we developed a collaborative spam filter which identifies mails by the URLs they contain. This approach is reasonable, since the majority of all spam mails try to make the reader visit a certain website and therefore contain clickable links. The filter is using the domains of all links included in the mail to create an identifier of the message, which is then treated like in other collaborative systems.
A speciality of our filter is its self-learning client side whitelist for good domains. The filter learns from the user's behavior about trustworthy domains and excludes these from future spam checks. If a message does not contain any links at all or only links to fully trusted domains, the filter will ignore this mail.

Advantages

The common approach to use the full text to generate the identifier of a mail is vulnerable to random and personalized text inserted into the spam message. The same spam mails received by different users may differ in more than 90% of their content, making it hard to detect their relation automatically. Since the messages still want to sell the same product though, it is supposable that they contain links to the same domains. Therefore, the filter extracts the domains included in the messages and calculates an identifier based on this information. Only the domains and not the whole links may be used for this process, since the full URLs usually contain randomized and personalized parameters or even sub-domains, which differ for every recipient.
The self-learning client side whitelist the filter includes is our approach to inherit the good tokens list, known from bayesian filters. The filter observes the user's behavior and tries to identify trustworthy domains seen in received mails. These domains are put on the whitelist and will not be included in future spam checks anymore. Because of this exclusion of trusted domains, most of the repeatedly received good mails (e.g. company internal newsletters), which only contain references to known, trusted domains, will not be checked any more. Summarized, the whitelist helps to reduce the communication overhead and also increases the privacy of the user, since the server does not get to know about at least a part of the good messages.

Disadvantages

Obviously, the system cannot deal with messages that do not contain any links. While the majority of spam mails do, especially scam mails like the infamous "Nigerian Investment Scam" [17] often do not contain any URLs, since the scammer just wants the addressee to answer the mail.
In addition, messages which have added randomized domains for each recipient cannot be detected, since they all produce different fingerprints. Such mails are very rare because domain based filtering is still uncommon. Naturally, it is to be expected that the spammers will accommodate by using additional

randomized links as soon as this technique becomes more popular.
Finally, the filter also needs to deal with the user trust problem. Since there is no off-the-shelf solution to create a dynamic trust system available, we decided to exclude this aspect from the current work and to make it a priority task of a future thesis.

2.3.1 Conclusion

The URL filter clearly should not be used alone, since it is unable to detect certain types of mails but in combination with other filters a good performance can be expected. For example a bayesian filter might be used to block the majority of the incoming spam messages and the URL filter could take care of those messages that cannot be identified by the bayesian tool, since they do not include visible tokens.

2.4 Hybrid Filter Systems

The previous sections showed that there are already various good spam filter concepts available. However, none of them is perfect but each has its own unique set of advantages and disadvantages. This leads to the basic idea of hybrid filter systems, namely to overcome the limitations of a single filter by combining several different systems in one tool. While it is easy to write a spam mail that avoids one given filter type, it is very hard to come up with a message that deceives a set of different filters, since a hiding mechanism to get a message past one filter often makes the mail look suspicious to another one.
In an enterprise environment, the commercial, hybrid system Brightmail [4] enjoys a good reputation. Companies rely on Brightmail, since it offers a good performance and the necessary professional support to keep the system up to date. Since Brightmail does not offer a home user solution, the freeware Spampal [18] is better suited for individual users. There are several different, free filters available on the internet which can be included in Spampal, and the user has the free choice which of them to use. Unfortunately, Spampal is only available for Microsoft Windows operating systems. Linux users can work with Spamassassin [10] though, which is also evolving into a full featured hybrid tool.

3 Spamato and URL Filter Design

3.1 Overview

The main goal of this work was to implement a collaborative spam filter, but soon after the start of the work we recognized, that a single filter would never be able to stop all spam messages. Therefore, we decided to extend the task and additionally designed a simple framework, called *Spamato*, which allows the development of further filters and extensions. Figure 3.1 shows the system we designed and implemented in this work. It consists of the framework Spamato, a collaborative URL spam filter, and an Outlook add-in used for client interaction.

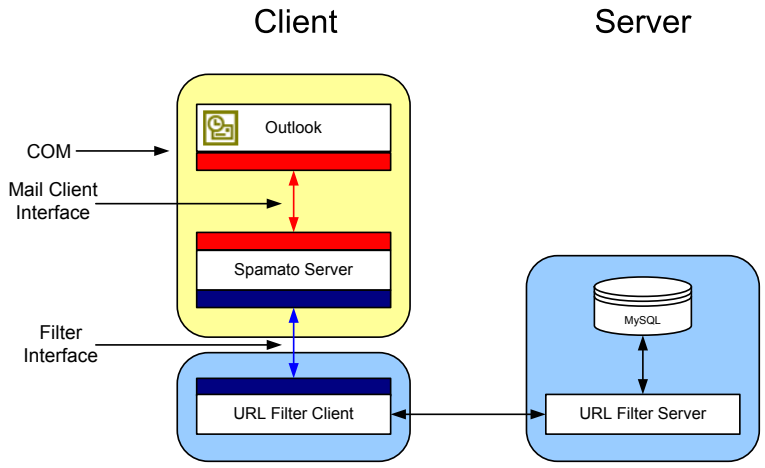

Figure 3.1: Setup of the implemented system containing the Spamato middleware, the collaborative URL filter and the Outlook add-in.

3.2 Local Spamato Server

The local Spamato server is a Java tool that serves as a simple middleware connecting a mail client to a set of spam filters. As can be seen in Figure 3.2, Spamato offers two interfaces, one for mail clients and one for spam filters. Mail clients can send requests for high level operations to the Spamato server, which maps them to calls to the registered filters.
Since the system was written in pure Java, it is highly portable and easy to extend. The following two sections give an overview over the Spamato interfaces.

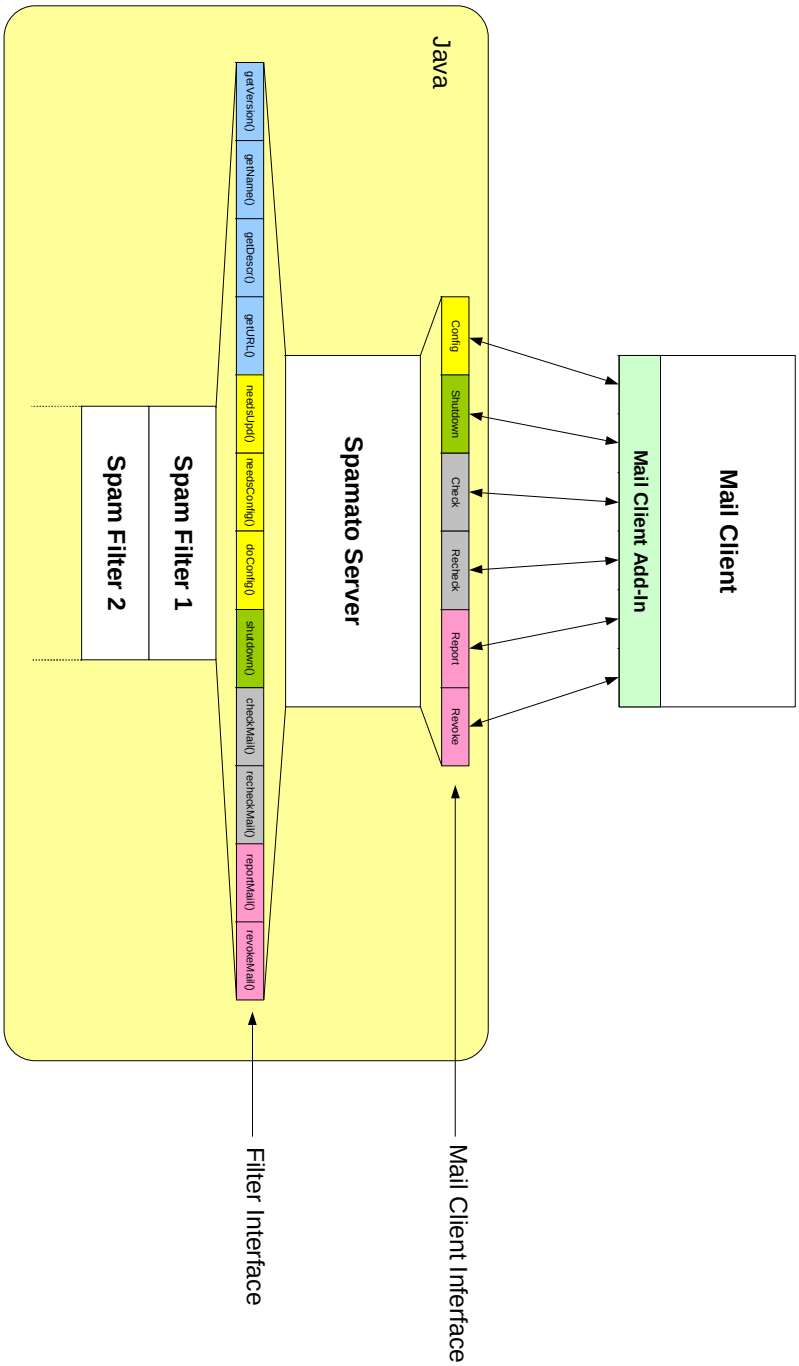

Figure 3.2: Spamato server with two interfaces for filters and mail clients.

3.2.1 Filter Interface

A Spamato-compatible spam filter must implement a given Java interface. The methods defined by the interface can be grouped as follows:

Startup and Configuration On startup, Spamato asks each registered filter if it needs to be configured. If the filter decides that some configuration work is necessary, Spamato calls the filter's configuration method, defined by the filter interface. The operation is split in two steps, because the configuration method may also be called at a later time, if the user directly requests to configure the filter.

Spam Check Spamato informs the different filters whenever the mail client wants to check if a message is spam or not. Each filter examines the mail individually and reports its decision. The response of the filters is a floating point number between 0 and 1, where 0 means the message is good and 1 means the filter is absolutely sure that the mail is spam. A special case is the situation when a filter cannot decide if a message is spam or not. In this case the filter returns -1. Spamato sums up the responses of all registered filters, ignoring those that answered with -1. If the average of all valid responses lies above a certain threshold, the client will be informed that the message is spam.

User Feedback The user has the possibility to help the Spamato system by reporting or revoking mails. The report function can be used to notify the system about a spam mail that made it past the filters without being detected. The revoke operation is exactly the opposite and allows the user to inform the filters about a mail that was wrongly marked as spam. These two methods are the key functions of every collaborative filter system, since these filters only work on the base of user feedback. Additionally, also other spam filters, which contain self learning functions, depend on this kind of information to optimize their efficiency.

Shutdown Before the Spamato server turns off, each filter is informed about the shutdown and gets an opportunity to save its settings and to clean up.

Administrative Operations Finally, the interface defines a set of administrative functions, which can be used to get some information about the filter.

For a detailed description of the Java methods and the parameters defined by the filter interface, see Appendix B.2.

3.2.2 Mail Client Interface

The mail client interface serves as connection point between the mail client and the Spamato server. To allow a platform and mail program independent use of the system, we decided to use XML messages over a TCP connection to setup the communication between the Spamato server and the mail client. (Also see Appendix B.4.1). Since the filter functionality is encapsulated in the

Spamato server, the mail client interface allows the mail program to use a couple of high level operations without the need of further knowledge about the functionality of the spam filters present.

Report The report operation is directly mapped to the corresponding methods of the filter interface. The Spamato server extracts the email data contained in the XML message received from the mail client and hands this information to all registered spam filters.

Revoke The revoke operation works analogously to the report operation. It only differs by calling the revoke instead of the report methods of the filters.

Check/Re-Check Spam The check and re-check operations allow the mail client to test if a message is spam or not. The Spamato server extracts the mail from the received XML message and feeds it to all registered filters. Depending on the responses of the filters, it will eventually send a simple `good` or `spam` message to the mail client, which deals with the message accordingly.
The re-check operation is used to examine a mail which was checked already before. While some filters will always come to the same decision if a mail is checked more than once, especially collaborative filters may have learnt about a spam message since the last check and can mark a mail as spam, which was not detected before. The re-check operation can often be mapped to a call of the normal spam check method, but filters which collect statistical information need to know if a message is checked for the first time or repeatedly.

Configuration With this operation, the Spamato server can be forced to open its configuration window, which gives the user access to the configuration options of all registered filters.

Shutdown Once the mail client shuts down it must call the shutdown command to make Spamato end its process. As soon as the Spamato server receives this command it informs all registered spam filters by calling their individual shutdown methods, before terminating itself.

3.3 Outlook Add-In

3.3.1 Overview

As part of this work we implemented a Spamato add-in for Microsoft Outlook. The main function of this add-in is to offer the user a graphical interface to interact with the Spamato server and of course to make Spamato check the incoming mails.
Outlook offers a very powerful extensibility interface which makes it easy to extend its functionality by means of an add-in. Unfortunately, this COM-Interface has some severe bugs and problems which made the theoretically easy

task to write an add-in in Visual Basic .NET a rather complicated and annoying exercise.

3.3.2 Startup and Shutdown

Once the add-in is installed, it is loaded automatically as soon as Outlook is launched. During its initiation, the Spamato add-in also starts the Spamato server. Thus, no memory and cpu time is wasted on the Java virtual machine (running the Spamato server) while Outlook is not started.
Consequently, the add-in is also responsible to stop the Spamato server on Outlook shutdown. As soon as the add-in receives the shutdown event from Outlook, it sends the corresponding command to the Spamato server and waits for some seconds. After a certain delay, the add-in checks if the Java process was stopped as requested or terminates it forcefully to guarantee no Java ghost processes remain loaded once Outlook is closed.

Graphical User Interface and User Interaction

Since the Spamato system was designed to be easy to use, the whole user interface consists of four toolbar buttons only.

Figure 3.3: The Spamato toolbar buttons

Spamato If the button with the caption "Spamato" is clicked, the Outlook add-in sends the configuration command to the Spamato server, which opens the requested dialog.

Report As it can be expected, a click on the report button leads to one or more spam reports. Every selected mail is encapsulated in an XML message and sent to the local Spamato server, one by one. As soon as the message has been transmitted to the Spamato server it is marked as read and moved to the Spamato Outlook folder, generated by the add-in.

Revoke The revoke button works identically as the report button with the difference that the selected messages are reported to be good. Mails that have been revoked, get moved to the default Outlook in-box.

Email Address Option Since some editions (and patch levels) of Microsoft Outlook show a security warning if an add-in tries to access fields of a message which contain email addresses (from, to, cc, bcc), we added this toggle button to give the user a choice to include or exclude this information. If the bulb is activated, the additional information is included in the XML message sent to the Spamato server and the user must confirm that the add-in is allowed to access these message fields. Unfortunately, Outlook does not allow to trust an add-in for more than 10 minutes, which means

that the users needs to click away the warning message several times per hour.

3.3.3 Spam Check

Whenever a new email is detected in the Outlook in-box the add-in encapsulates the mail in an XML message and sends it to the Spamato server, to check if it is spam or not. Depending on the reply of the server (`good` or `spam`) the mail will either be ignored or marked as read and moved to the "Spamato" Outlook folder.
As described before, some spam filters may not detect a spam message at the moment it comes in but only some time later. To deal with this problem the Spamato add-in has a background thread running which re-checks all unread mails in the Outlook in-box periodically. It is sufficient to check unread messages, since the user has already seen all read messages and would have reported them if they were spam.

3.4 URL Filter

Since the concept of the filter has already been described in Section 2.2.4, the following sections will focus on the most interesting aspects of its implementation.

3.4.1 URL Filter Client

The URL filter client consists of several Java classes of which the main class implements the Spamato filter interface. To access the filter's functionality, the Spamato server calls the methods defined by this interface and passes a `MailObject` containing the actual mail data as parameter. Since the filter logic is part of the URL filter server component, the client's primary task is to encapsulate and send the information the server requires to analyze a mail.

URL extraction

As described before, the filter identifies a mail by the domains linked within the message. To reduce the workload on the server, the links are already extracted from the mail body on the client side. As a result, the client does not need to send the whole message to the server, but only the list of extracted links. If no domains are available (e.g. because the mail does not contain any links), the filter will not test the message at all. Besides the bandwidth and server cpu time saved, this approach also increases the privacy of the client, since the server does not get to see the full mail but only a couple of links.

Local Whitelist

The client side whitelist helps to reduce the amount of unnecessary traffic by removing all links to known, good domains before the URLs (extracted from a mail) are sent to the server. The trust in a domain, listed on the whitelist,

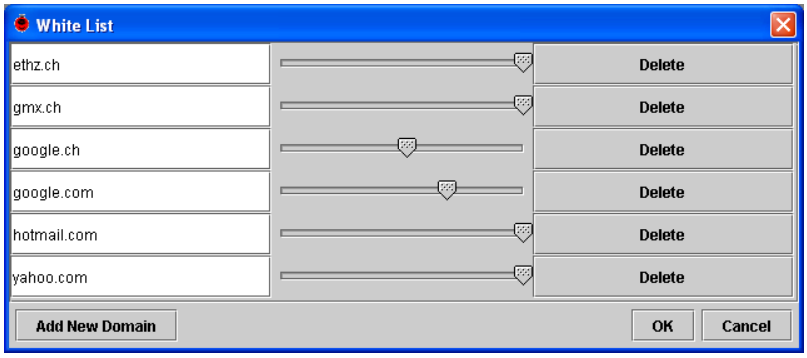

Figure 3.4: Whitelist manipulation dialog. The sliders in the center column represent the trust level of the corresponding domains.

is represented in a fuzzy way. The current implementation offers 50 different levels of trust, but only the domains with maximum trust are not included in the communication with the URL filter server anymore.
Whenever an incoming mail is not detected as spam, the domains within this message get extracted and their trust level is increased by one. In case that a domain is not yet on the whitelist, it is automatically added at the lowest trust level.
Whenever a message is detected or reported as spam, its included domains get a severe penalty and loose several levels of trust (if they are on the whitelist at all).
This combination of slowly increasing and rapidly falling trust guarantees that only domains which often appear in good mails but hardly ever in spams, have a chance to be omitted from the communication.
Of course, the user can also manipulate the whitelist manually. To do so, the URL filter client offers a graphical interface that can be accessed over the Spamato configuration dialog. The user may either manipulate the trust of an already listed domain or add a new one. Figure 3.4 shows a sample snapshot of this whitelist manipulation dialog. In this example, the domain `ethz.ch` has full trust and will be omitted from any communication with the server. The domain `google.ch` has collected some trust already but has not reached full trustworthiness yet and therefore will still be included in the communication.

3.4.2 URL Filter Server

The server component of the URL filter consists of a Java programm implementing the filter logic and a MySQL database, storing the collected spam information.

Domain Extraction

As mentioned before, our filter identifies messages according to the linked domains they include. The links of a message are already extracted on the client side, but the server still needs to identify the domain part of the received URLs.

This step is done on the server side, because there are numerous ways to encoded a URL. For example all of the following links lead to the same website:

- `http://www.ethz.ch`
- `http://129.132.202.79`
- `http://www%2eethz%2ech`
- `http://&#119;&#119;w.&#%101;th%7A.ch`

These are only a few examples of possible encodings and it will take a while to write a decoder that is capable of extracting the domain from all possible link representations. As long as the decoder runs on the server side, it can easily be adapted if a new perfidious way to cloak a link is detected.

Fingerprint Generation

Once the domains are identified, a fingerprint of the message can be calculated. We currently use the built-in hash-method of Java to generate a number for each domain, and simply sum these up. While there are certainly better algorithms to generate a fingerprint based on domains, this simple approach did not produce any collisions between identifiers of different mails during the beta test.

Database

The MySQL database stores the fingerprints of the reported spam mails together with their respective number of reports, in a table. To check for spam, a simple lookup on this table is enough to see, if the message has already been reported by any user.
Since we have been interested in learning more about the methodology of the spammers, the client does not only include the links of a message in a spam report, but also the full mail body and all header information available. This additional information is only transmitted if the user willingly reports a spam message. It is not included if the mail is checked for spam. This guarantees that no good mails, possibly containing personal data get logged. While the so collected auxiliary data does not have a direct influence on the filter efficiency, it is very valuable when it comes down to evaluate the strengths and weaknesses of the system.

4 Beta Test

4.1 Overview

The combination of the Outlook add-in, the Spamato server, and the URL filter represents a fully applicable spam filter system. Once all components had reached a certain level of development, we decided to run a beta test to see if the system behaved as we expected. Due to the missing trust component and the early stage of development, an open beta test with a wide user base was not practicable. Therefore, we had to rely on a small, trustworthy group of testers. The fact that only an add-in for Microsoft Outlook existed turned out to be a problem. Especially inside ETH many people rely on other mail clients and understandably didn't want to start using Outlook just to participate in our beta test. Therefore, it was rather difficult to find suitable testers, but eventually we managed to recruit about ten active Outlook users who were willing to test Spamato.

4.2 Setup

The server component was installed on a simple IBM notebook running Windows XP Professional. Due to the low number of users, this machine was more than sufficient and never produced any problems during the whole beta phase. The client component was distributed as self-extracting zip file including a Windows Installer to set up the Outlook add-in and the Spamato server plus URL filter client.

4.3 Results after 20 days

- The filter has received about 1'500 spam reports during the first 20 days of our beta test. Additionally we have also inserted 10'000 older mails from a spam archive.
- About one third of all stored identifiers have been reported more than once. Considering the low number of testers, this is a very good result and our assumption about the constancy of the domains within spam messages seems to be correct.
- The detection rate for incoming spam mails lies between 25% and 50%.
- One spam mail reported was containing a large number of links to innocent domains.
- No report of a false positive is known.

4.4 Conclusion

The test can be seen as a success. For a first beta version, the system ran quite stable and the filter efficiency fulfilled our expectations. While 25% - 50% of the incoming spam caught does not seem to be very impressive at first sight, two important points need to be kept in mind when evaluating the result:

- Practically all beta testers had server side spam filters running which blocked the majority of spam before it reached the client. This means all spam mails our system caught, had already been checked and bypassed the server-side filter.
- The efficiency of the filter is highly dependent on the number of its users. The fact that it actually did filter incoming messages with as few as ten active users proofs that our approach to identify spam messages is reasonable and should produce good results for a larger number of users.

5 Summary

The work on the spam topic was interesting and challenging. Both sides, the spammers and the anti spam community, are using clever approaches to prevail over the other side. Therefore, it was difficult to come up with a reasonable tool in the short time of this diploma thesis. While it has been clear from the start, that the result of this work would never be able to keep up with the sophisticated spam filter tools available, it was nice to see that our tool actually caught spam messages which had already passed other filters.
Especially the first month of this thesis was very exciting, since the whole topic was still new to me and we learned about new dirty tricks of the spammers on a daily base. It took us almost two months to evaluate the existing approaches and to design our own system. Once our filter design was finished, the actual programming phase could begin. Writing the Java components was a rather straight forward task, although debugging was difficult at some times, due to the distributed nature of the tool. Unlike the development of the Java programs, the implementation of the Outlook add-in in Visual Basic .NET turned out to be very tedious. In the beginning it did not seem to be more than a few days of work to write the add-in, since the Microsoft Visual Developer Studio automatically generates a skeleton program with all important methods predefined when a new add-in project is started. Unfortunately, the COM interface used to create Microsoft Office add-ins turned out to be buggy and seems to have some severe design flaws. As a result, it was necessary to include different workarounds to make the code behave as expected. Since an official documentation of known problems was not available most of the times, a lot of Google Group searches were necessary to collect the information required to fix the critical problems.
Once we were ready to start a beta test, the lack of testers for our system was frustrating to a certain extend. Nevertheless, even with the low number of testers we could recruit, the system showed its potential and can therefore be seen as a success.
Summarized, I had a very good time working on this thesis, since not only the work was interesting, but also the personal environment was very enjoyable. My advisor Keno Albrecht gave me all the support I could ever hope for and was always willing to answer my questions. Also Prof. Wattenhofer and the rest of his team helped working on the Spamato project by reporting their spam messages and showing interest in the progress of the work. For this participation I am very grateful to them.

6 Future Work

Since the time to write a diploma thesis is limited to four months we had to skip many interesting points. This chapter gives an overview over some possible future tasks

6.1 URL Filter

6.1.1 Improved Domain Extraction

The currently used method for domain extraction from a URL uses a simple pattern matching to identify the domain. The terms before and after the last "." in the URL string are interpreted as a domain. Unfortunately, there are several cases where the last three terms need to be taken, since the top level domain consists of two parts (e.g. co.uk), or a redirection service is used (e.g. ch.vu). Another problem of the current domain extractor are exotic URL encodings (also see Section 3.4.2) which are not parsed correctly. In a future work, a better domain extraction mechanism might be implemented, which deals with those problems.

6.1.2 Better Combo-URL Handling

The current implementation of the filter treats all domains included in a mail equally. Since the spammers may include innocent links in their spam messages, this is not always the best approach. A statistical analysis on the distribution of the reported domains might be used to identify the domains that have a higher probability to be spamvertized.

6.1.3 User Trust

The filter is missing a user trust mechanism which automatically decides on the trustworthiness of a user. This function is essential, if a large number of unknown users shall be allowed to use the system, because it is the only protection mechanism against malicious users. Since there is no off-the-shelf solution available, it will not be easy to find a good solution.

6.1.4 Server to P2P

Currently, the filter uses a central server to store all spam reports and to check incoming messages. Since a server always means a central point of failure, it would be nice to have a P2P solution or at least a more reliable, replicated system replacing this component. There are two prototypes of collaborative spam

filters, working on P2P networks [14][16] available and it might be interesting to combine our filter with the distribution mechanisms used in these systems.

6.2 Spamato Extensions

6.2.1 Additional Mail Client Support

At the moment there is only a Spamato add-in for Microsoft Outlook available. As a future work add-ins for other mail clients which offer a plug-in mechanism (e.g Mozilla) might be developed. Since the Spamato server offers a simple interface to attach email clients, this task should not be too hard.

6.2.2 Local Mail Server Proxy

Since there are some mail clients which do not support plug-ins a mail proxy tool might be written which acts like a mail server towards the mail program. This tool would have to open a connection to the mail server and forward the received data to the client. Features like IMAP or SSL make it difficult to write such a tool.

6.2.3 More Filters

Since Spamato was designed as a container for several filters, other filters than the existing one can be written an included in the system. Especially a bayesian tool would be helpful since the combination of such a tool with the collaborative URL filter would lead to a very good filter performance.

A System Requirements and Installation

A.1 Outlook Add-In And Spamato

A.1.1 System Requirements

- Windows operating system
- Outlook 2002, XP or 2003 (Outlook 2000 will *not* work)
- Java runtime 1.4.x or newer installed (Version 1.3.x will *not* work)
- Microsoft .NET runtime installed

A.1.2 Installation

1. As a first step it must be checked if an up to date Java runtime is installed on the system. To do so, the command "`java -version`" can be used from the command line. If this leads to an error message or an older version than 1.4 is present, a new Java runtime [19] needs to be installed.
2. The Spamato system and the Outlook add-in come as self extracting zip file. After a double click on the `SpamatoClient.exe` file, the windows installer will open and start with the Spamato installation. If the .NET runtime is missing, the installer will print a corresponding message and abort the installation process. In this case, the .NET runtime must be installed either over the `Windows Update` or from a downloadable version [20] before the Spamato setup can be restarted.
3. When the installation process terminates successfully the Outlook add-in and the Spamato server are ready to use. After the first start of Outlook a Spamato window will pop up (see Figure A.1) asking for a username and password for the URL filter. After a click on the "`New User`" button another dialog will be opened where a username and password can be chosen. As soon as the new user was generated the installation is complete.

A.2 URL Filter Server

A.2.1 System Requirements

- Java runtime 1.4.x or newer installed (Version 1.3.x will *not* work)
- MySQL Server installed and running

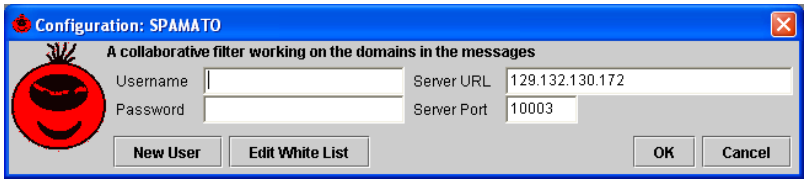

Figure A.1: The URL filter configuration dialog. The user can set his username and password and the address of the URL filter server. Also the whitelist can be accessed from here.

A.2.2 Installation

The URL filter server consists of a Java program available as jar file and a MySQL database. It is suggested to run both components on the same machine to allow fast database access, but it is also possible to run them on different machines.

Database

The database schema for the filter is available as SQL files and can be installed as follows:

1. MySQL [21] must be installed and started.
2. A command line connection to the MySQL server must be opened with

   ```
   mysql -h HOSTNAME -u USERNAME -p
   ```

   HOSTNAME is the name of the machine running the database server and USERNAME a valid MySQL username. The parameter `-p` indicates that the user has a password set.
3. The file `setup_spam_db.sql` contains a script to create the database necessary. To start it the following command must be entered on the open MySQL command line.

   ```
   SOURCE setup_spam_db.sql;
   ```

   The script creates a database called `spamfilter` containing all tables required by the URL filter.
4. As a next step, a MySQL user, with limited access rights to the `spamfilter` database, should be created. This user needs the priviliges `SELECT`, `UPDATE` and `INSERT` on the `spamfilter` database and should be limited to the machine running the Java component of the server. To create such a user the following command must be entered on the MySQL command line:

   ```
   GRANT Select, Insert, Update ON spamfilter.* TO 'USER'@'HOST'
       IDENTIFIED BY 'PASSWORD';
   ```

`USER` is the username of the new MySQL user, `HOST` the name or IP address of the machine running the Java component, and `PASSWORD` the password for the newly generated user.

Java Component

The Java component consists of a jar file (`URLFilterServer.jar`) containing the whole server code. Before the server can be started, a config file called `connectionstring.txt` needs to be created in the same directory as the `URLFilterServer.jar`. This config file contains the connection URL to the MySQL server, which depends on the individual system setup. The connection string is built as follows:

```
jdbc:mysql://HOST/spamfilter?user=USER&password=PASS&autoReconnect=true
```

`HOST` is the address of the machine running the MySQL server, `USER` the username and `PASS` the password of the MySQL user. The option `autoReconnect=true` tells the JDBC driver to reopen closed connections, which may happen if the server did not receive any client requests over a longer period of time.

After the installation the URL filter server can be started by calling

```
java -jar URLFilterServer.jar
```

from the command line. The server will also generate a file called `Redirect.err` which contains a log entry for all exceptions that are thrown during server execution. Since this file is overwritten whenever the server is restarted, it should be saved under a different name if problems with the server occur.

B Technical Reference

B.1 Mail Object

The `MailObject` class is the Spamato internal representation of an email. It does not contain any methods but offers a consistent way to store a message.

```
1  public class MailObject implements Serializable {
2
3      public String header;
4      public String sentOn;
5      public String receivedOn;
6      public String from;
7      public String to;
8      public String cc;
9
10     public String subject;
11     public String body;
12
13     public Vector urls;
14
15     public String hasAttachement;
16     public String encoding;
17 }
```

Listing B.1: MailObject.java

While most of the fields are self-explanatory, the following entries need some additional explanations:

- The `header` field stores the full mail header.
- The fields `sentOn` and `receivedOn` store the corresponding date and time stamps from the mail header.
- The field `urls` is a vector of `java.net.URL` objects, which stores all links that are included in the body part of the mail.
- `hasAttachment` is a value indicating the number of attachments.
- The field `encoding` indicates if the message was encoded as plain text, html or rich text.

B.2 Filter Interface

Every Spamato compatible filter must implement this interface, since it allows the Spamato server to access the filters functionality in a standardized way.

```
1  public interface FilterInterface {
2      public final static int NO_UPDATE_AVAILABLE = 1;
3      public final static int UPDATE_POSSIBLE = 2;
4      public final static int UPDATE_NECESSARY = 3;
5
6      public String getVersion();
7      public String getName();
8      public String getDescription();
9      public URL getURL();
10
11     public int needsUpdate();
12
13     public boolean needsConfiguration();
14     public boolean doConfiguration(JPanel p,
           ActionListener al);
15
16     public float checkMail (MailObject mail);
17     public float recheckMail(MailObject mail);
18     public void reportMail(MailObject mail);
19     public void revokeMail(MailObject mail);
20
21     public void shutdown();
22 }
```

Listing B.2: Filter Interface for Spamato Filters

Constants The three constants `NO_UPDATE_AVAILABLE`, `UPDATE_POSSIBLE` and `UPDATE_NECCESSARY` allow a filter to indicate if an update is available.

- **NO_UPDATE_AVAILABLE** There is no newer version of this filter available.
- **UPDATE_POSSIBLE** There is a newer version of this filter available but the old one still works.
- **UPDATE_NECESSARY** There is a newer version of this filter available and the old version will *not* work any more.

getVersion(), getName(), getDescription(), getURL() These four methods can be used to give some user readable information about the filter (e.g. how it works and where its homepage can be found)

needsUpdate() On Spamato startup this method is called to give the filter a chance to check if it needs an update. Only the three constants defined above are valid return values.

needsConfiguration() Spamato calls this method during its initiation phase. If the filter returns `true` Spamato will open the configuration dialog and call the filter's `doConfiguration()` method.

doConfiguration(JPanel p, ActionListener al) This method is called if the filter's `needsConfiguration()` method returned `true` or if the user forces Spamato to show its configuration dialog.
The `JPanel`, which is passed as a parameter, can be used to give the user a graphical interface to the filter's configuration options. It will be embedded in the Spamato configuration dialog as soon as the method returns.
The `ActionListener` can be used to close the whole Spamato configuration dialog. If this listener receives an `ActionEvent` from any source (e.g. an OK button on the config panel of the filter), it will close the Spamato configuration window.

checkMail(MailObject mail) This method is called whenever a new email must be checked. Its return value represents the filter's decision if the message is spam or not. The filter must return a value in the range of 0 to 1 where 0 means the mail is certainly *no* spam and 1 means the mail definitively is spam. If the filter encounters a mail it cannot analyze it also has the option to answer with -1. In this case, Spamato will exclude the filter from the global decision if the message is spam and only decide on the results of the other available filters.

recheckMail(MailObject mail) This method is called if a mail needs to be checked again some time after its initial test. Most filters can directly map a call to this method to a call of the method `checkMail(MailObject mail)` but if a filter collects statistical information or dynamically learns from the user behavior it must know if a message was recorded already.

reportMail(MailObject mail) This method tells the filter that the user marked the email within the parameter `mail` as spam. The filter can implement this method as a simple `return` if it does not learn dynamically from user feedback.

revokeMail(MailObject mail) This method is called whenever a user revokes a message, previously marked as spam. Most of the times, this method is called if the user wants to undo an accidental spam report but it could also be that Spamato wrongly marked an innocent message. Since the decision to mark the mail as spam was made based on the result of all available filters, it does not necessarily mean *this* filter made a mistake, but it still receives a notification.

shutdown() This method is called when Spamato is about to shut down. The filter can use this method for example to save its configuration data to disk or to close open connections.

B.3 Adding a Spam Filter to Spamato

The current implementation of the Spamato system does not yet offer a dynamic filter loading mechanism. New filters need to be directly included in the

Spamato main class `LocalServer`. The registration of a new filter is a matter of one line of code though. All that needs to be done is to add an instance of the new filter to the `Vector filterList`. The best place in the code to do this is marked with the comment `ADD NEW FILTERS HERE`.

B.4 Mail Client Interface

The mail client interface must be used to connect a mail client to the Spamato system. The interface defines six high level functions, which can be called by sending an XML message formatted as described in the following section.

B.4.1 XMLMessage

To allow a high level of portability the mail client needs to communicate with the Spamato server over XML messages formatted as follows:

```
1  <message>
2      <type>TYPE</type>
3      <senton>SENTON</senton>
4      <receivedon>RECEIVEDON</receivedon>
5      <from>FROM</from>
6      <to>TO</to>
7      <cc>CC</cc>
8      <header>HEADER</header>
9      <subject>SUBJECT</subject>
10     <body>BODY</body>
11     <hasattachement>HAS_ATTACHEMENT</hasattachement>
12     <encoding>ENCODING</encoding>
13 </message>
```

Listing B.3: XML Message

TYPE The type field is used to indicate which operation the client requires from the Spamato system. Valid values are: `request`, `rerequest`, `report`, `revoke`, `shutdown` and `setup`.

SENTON, RECEIVEDON These fields contain the sending and receiving times of the message as they appear in the mail header.

FROM, TO, CC These fields contain the sender and receiver addresses. The `bcc` field does not need to be included, since it is always empty for incoming messages.

HEADER This field stores the full mail header if available.

SUBJECT The subject of the mail.

BODY The body part of the email.

HAS_ATTACHEMENT This field indicates how many attachments the mail had.[1]

[1] The attachments are not included in the XML message.

ENCODING This field shows if a message was encoded as `html`, `plain` or `richtext`.

Fields that cannot be filled due to missing information should be initialized with a dummy value like `N/A`.

B.4.2 Operations

The Spamato system currently offers the following six operations to a connected mail client.

Spam Check

If the client wants to make Spamato check an email, it has to generate an XML message with the type field set to `request`. The remaining fields of the message must be filled with the corresponding values gathered from the mail. The whole XML message must then be sent to the Spamato server which will check the included mail and finally answers with `good` or `spam`[2].

Spam Re-Check

An email that passed the first spam check but should be tested again must be sent in an XML message with the type field set to `rerequest`. Spamato treats a re-request like a normal spam check but filters that collect statistical information get informed that this is not a new message but one that was already checked before.

Report

To report a message as spam the client needs to generate an XML message of the type `report`. As for a spam check the remaining XML message fields need to be filled with the corresponding data gathered from the email and the whole XML message must then be sent to the Spamato system. The server responds with `ok` as soon as the message has been received and parsed.

Revoke

To inform Spamato about a mail which has been wrongly marked as spam, the user can send an XML message of type `revoke` including the innocent message. Spamato will then respond with an `ok` again, as soon as the XML message has been parsed.

Setup

To make Spamato open its configuration window the client can send an XML message with the type field set to `setup`. The remaining fields of the XML

[2] The Spamato server does not reply with an XML message, since we had some problems with the string parsing in the Outlook add-in, and there are only two response types necessary.

message do not need to be set, since the server will not try to access any further values. Also, the server will not send any response to a setup message.

Shutdown

The client can shutdown the whole Spamato process by sending an XML message of type `shutdown`. Since the server will not try to access any other fields of the XML message the remaining fields do not need to be set. The server does not send any response to a shutdown message but silently terminates after shutting down all filters.

Generally, if a communication problem happens and the server receives an illegally formatted XML message, it responds with `illegalmessage` and ignores the received data.

3 Bibliography

[1] Brian Livingston. Inside the spammer's world. http://news.com.com/2010-1071-281499.html, 2001.

[2] Juha Saarinen. Spammers hit below men's belts. http://www.nzherald.co.nz/storydisplay.cfm?storyID=3518097, 2003.

[3] Matt Hines. AOL filters spam tools down to users. http://news.com.com/2100-1032_3-5083980.html, 2004.

[4] Brightmail. http://www.brightmail.com, 2004.

[5] Spam Laws: CAN_SPAM Act of 2003. http://www.spamlaws.com/federal/108s877.html, 2004.

[6] Spam Laws: European Union. http://www.spamlaws.com/eu.html, 2004.

[7] Jonathan B. Postel. Simple Mail Transfer Protocol. http://www.ietf.org/rfc/rfc0821.txt, 1982.

[8] Stefanie Olsen. AOL tests caller id for e-mail. http://zdnet.com.com/2100-1104_2-5145065.html, 2004.

[9] Mail abuse protection system. http://mail-abuse.org, 2004.

[10] Spamassassin. http://www.spamassassin.org, 2004.

[11] Paul Graham. Anti spam website. http://www.paulgraham.com, 2004.

[12] Vipul's razor. http://razor.sourceforge.net, 1998.

[13] Cloudmark. Spamnet. http://www.cloudmark.com, 1998.

[14] F. Zhou, L. Zhuang, B. Zhao, L. Huang, A. Joseph, and J. Kubiatowicz. Approximate object location and spam filtering on peer-to-peer systems. http://citeseer.nj.nec.com/zhou03approximate.html, 2003.

[15] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[16] Jörg Metzger, Michael Schillo, and Klaus Fischer. A multiagent-based peer-to-peer network in java for distributed spam filtering. http://citeseer.nj.nec.com/574576.html, 2004.

[17] Sophos hoax description: Nigeria letter.
http://www.sophos.com/virusinfo/hoaxes/nigerian.html, 2004.

[18] Spampal. http://www.spampal.org, 2004.

[19] Sun Mycrosystems. Java 2 Platform, Standard Edition.
http://java.sun.com/j2se/index.jsp, 2004.

[20] Microsoft. .NET Framework 1.1 Redistributable Package.
http://www.microsoft.com/downloads/details.aspx?FamilyID=262d25e3-f589-4842-8157-034d1e7cf3a3&displaylang=en,
2004.

[21] MySQL: The World's Most Popular Open Source Database.
http://www.mysql.com, 2004.