Lightweight Spam Filtering Methods

Vladimir Blaskov
2014

Master (120 credits)
Master of Science in Information Security

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
Abstract

The purpose of this study is to identify and evaluate lightweight spam filtering methods, facilitating the development of high-throughput spam filtering systems. It deviates from the traditional statistical-based filters and explores different DNS and protocol analysis solutions.

A literature review is used to reveal techniques promising lower resource usage and higher throughput, and order them according to the information required to classify a message. The results are used to develop a prototype implementation and benchmark it against a popular open source spam filtering software in order to compare their performance.

Analysis of the results confirmed that the proof of concept spam filtering proxy implemented as part of this study outperforms the reference software and its performance is almost on par with Postfix, a popular Mail Transfer Agent that does not perform any filtering, especially under high load with multiple concurrent connections.
Acknowledgements

Foremost, I would like to express my deepest gratitude to Professor Tero Päivärinta for his patience, support, and great ideas without which, this study would not have been possible. Besides my advisor, I would like to also thank Professor Todd Booth for his peer review that seriously affected my view towards this project and helped me broaden my horizons.

Last but not least, I would like to express my gratitude to my family and friends who supported me during the countless hours of planning, research and coding.
Contents

Abstract ................................................................................................................................. 1

Acknowledgements ............................................................................................................. 2

Contents ............................................................................................................................... 3

List of figures ....................................................................................................................... 5

List of tables ......................................................................................................................... 6

Abbreviations ...................................................................................................................... 7

Introduction ......................................................................................................................... 8

Research problem ............................................................................................................... 8

Research questions ............................................................................................................ 8

Scope and delimitation ....................................................................................................... 9

Contribution ......................................................................................................................... 9

Methodology ....................................................................................................................... 10

Research method ............................................................................................................... 10

Research process .............................................................................................................. 12

Data collection and analysis ............................................................................................. 12

Related work ....................................................................................................................... 14

Methodology ....................................................................................................................... 14

Initial literature review ..................................................................................................... 16

DNS-based blacklists and whitelists ................................................................................. 17

DNS validation and sender authentication ......................................................................... 18
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender Policy Framework</td>
<td>18</td>
</tr>
<tr>
<td>Sender ID</td>
<td>19</td>
</tr>
<tr>
<td>DomainKeys Identified Mail (DKIM)</td>
<td>19</td>
</tr>
<tr>
<td>Criticism</td>
<td>20</td>
</tr>
<tr>
<td>Greylisting</td>
<td>20</td>
</tr>
<tr>
<td>Protocol defects / header-relay detection</td>
<td>22</td>
</tr>
<tr>
<td>Summary</td>
<td>23</td>
</tr>
<tr>
<td>Methods that do not require SMTP session to be created</td>
<td>24</td>
</tr>
<tr>
<td>Methods requiring only part of the SMTP session information</td>
<td>24</td>
</tr>
<tr>
<td>Mixed methods</td>
<td>24</td>
</tr>
<tr>
<td>Prototype implementation</td>
<td>27</td>
</tr>
<tr>
<td>Requirements</td>
<td>27</td>
</tr>
<tr>
<td>Design</td>
<td>27</td>
</tr>
<tr>
<td>Implementation details</td>
<td>29</td>
</tr>
<tr>
<td>Testing</td>
<td>33</td>
</tr>
<tr>
<td>Performance testing</td>
<td>34</td>
</tr>
<tr>
<td>Accuracy testing</td>
<td>36</td>
</tr>
<tr>
<td>Discussion</td>
<td>39</td>
</tr>
<tr>
<td>Conclusion</td>
<td>41</td>
</tr>
<tr>
<td>References</td>
<td>42</td>
</tr>
</tbody>
</table>
List of figures

Figure 1: Greylisting flowchart ........................................................................................................20

Figure 2: SMTP state machine ........................................................................................................30

Figure 3: Accuracy testing environment ..........................................................................................37
List of tables

Table 1: Keyword information .............................................................................................................. 15
Table 2: Concept map .............................................................................................................................. 26
Table 3: Performance testing results .................................................................................................... 35
Table 4: Accuracy testing results .......................................................................................................... 38
Abbreviations

CPU – Central Processing Unit

DKIM – DomainKeys Identified Email

DNS – Domain Name System

DNSBL – DNS blacklist

DNSWL – DNS whitelist

DNSxL – DNS blacklist or whitelist

MTA – Mail Transfer Agent

SMTP – Simple Mail Transfer Protocol

SPF – Sender Policy Framework

UCE – Unsolicited Commercial Email
Introduction

Nowadays Internet is an integral part of our lives and e-mail is one of the most popular methods for electronic communication. Unfortunately, the popularity of the e-mail service attracts many different types of abuse. According to Symantec [Symantec, 2012] unsolicited e-mail messages constituted 75% of all e-mail traffic for September 2012. Although their cost is not clearly visible on first sight, it is mainly measured in “lost productivity, as well as storage and carrier costs” [Edwards, 2010].

Research problem

In order to deal with unsolicited commercial e-mail messages (also known as spam), most organizations rely on dedicated devices or “external service providers” [Larsen & Meyer, 2003]. In both cases spam filtering is performed by a combination of different techniques in order to minimize the number of false positives (legitimate e-mail messages misclassified as spam). Some of these methods are pattern detection, rule-based filtering, checksum-based filtering and statistical analysis. They are based on completely different and unrelated theories, but the common trait between all of them is that all of the above-mentioned strategies are bound by CPU and memory resources. In other words, the amount of available CPU and memory resources determines the throughput of the spam filtering system.

This paper examines different strategies to reduce the number of messages reaching the CPU- and memory-bound content filters, thus effectively increasing the overall throughput of the system.

Research questions

- Is it possible to achieve comparable accuracy with lower resource usage and higher throughput than current statistical-based open source solutions?
• What is the most efficient (resource-wise) order of property-based filters?

**Scope and delimitation**

The focus of this paper is on resource usage and accuracy of different spam filtering methods. However, it is not possible to fully review all spam filtering techniques, so a literature review will be used to identify the most promising ones.

**Contribution**

The goal of this research is to identify the right combination and order of existing spam filtering techniques, allowing for more effective message classification in terms of system resources. The emphasis on resource utilization is not frequently observed in existing academic literature, as demonstrated by in the “Related work” chapter, and this paper is seeking to fill the gap in the knowledge of resource-wise unsolicited e-mail filtering.

On the one hand, this development may be interesting for business as it will reduce the loss from unsolicited e-mail messages and reduce the cost of spam filtering. On the other hand, this research may set base for further development in this direction, as effectiveness of spam filters is currently measured only by their accuracy, but we should also put emphasis on their resource usage in order to be able to effectively fight against spam.
Methodology

This chapter documents the methodology used to perform the research described in this thesis. It begins by describing the selected research method, continues with an outline of the research plan and provides information about the data collection and analysis process.

Research method

According to Sekaran [Sekaran, 2009] research is “a systematic and methodical process of inquiry and investigation that increases knowledge and/or solves particular problem”, and Walliman [Walliman, 2010] gives a rather broad definition of research methods – “they represent the tools of the trade, and provide you with ways to collect, sort and analyze information so that you can come to some conclusions”. In other words, the research method gives us the framework to conduct a research, and the right selection of a research method is of utter importance as it directly influences the conclusions of the study.

Information systems researchers aim to develop and communicate knowledge related to both the management and use of information technology within the organization. According to Hevner, March, Park and Ram [Hevner et al., 2004] two distinct paradigms are available to acquire and systematize such knowledge – behavioral science and design science. Behavioral science is oriented towards development and justification of theories related to organizational or human behavior in the context of information systems.

Design science, on the other hand, “seeks to create innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, management, and use of information systems can be effectively and efficiently accomplished” [Hevner et al., 2004]. It shares common traits with behavioral science, but overall, its focus is heavily shifted towards design (as suggested by the name) and implementation. The process of designing useful artifacts is complex because of the amount of innovation involved – existent theory is often insufficient therefore they push the boundaries of accumulated knowledge to
Design-science method was selected for this research, as it perfectly fits the description of the method – the objective is to create and evaluate an information technology artifact solving an already identified problem.

Hevner et al. identify four main types of information technology artifacts:

- **Constructs** – vocabulary and symbols
- **Models** – abstractions and representations
- **Methods** – algorithms and practices
- **Instantiations** – implemented and prototype systems

While behavioral research focuses mainly on instantiations, design science represents much more holistic approach by providing means to research any of the above-mentioned types.

The framework for information systems research proposed by Hevner et al. defines the following seven guidelines for design science in information systems research:

- **Design as an artifact** – result of a design research should be a purposeful artifact addressing a specific problem
- **Problem relevance** – objective of the research is to collect and interpret knowledge allowing the development of a technology-based solution solving a problem that is relevant to the target community
- **Design evaluation** – design of an artifact is an iterative process by its nature and the evaluation is an important phase of the research, providing a feedback needed for the evolution of the artifact
- **Research contributions** – effective design research must provide clear contribution in at least one of the areas of the design artifact, foundations, and methodologies
- **Research rigor** – application of rigorous methods in both the construction and evaluation of the designed artifact
• **Design as a search process** – multiple iterations leading to an optimal design
• **Communication of research** – results must be properly communicated to different target audiences

**Research process**

• Perform a literature review to identify related work, analyze the results and produce a list with lightweight spam filtering methods
• Develop a spam filtering solution, implementing the methods identified at the previous phase of the research
• Deploy a popular spam filtering solution and the prototype implementation on a publicly accessible mail server and collect metrics for 20 days
• Perform a throughput benchmark, comparing a popular spam filtering solution and our prototype implementation
• Analyze metrics and compare both solutions

It should be noted that the literature review was the most time consuming activity in this research and it took more than 1 month to be completed. Another important aspect was the prototype implementation design and development process. It was not as straightforward as it looks in this paper – a lot of time was invested to select the right architecture, the right language and to implement the prototype. Initially, 2 weeks were planned for the whole development process, but at the end it took almost a month to implement the solution. One of the major problems was that selected a programming language that I was not familiar with, and I had to learn it during the development process. On the other hand, Go turned out to be an excellent language with almost flat learning curve and a syntax that is very similar to C and other C-like programming languages.

**Data collection and analysis**

The starting point of this research was a literature review identifying related work and analyzing
its importance in the light of the specified objectives. This allowed me to produce a list with already existing lightweight spam filtering solutions and stack them together in a prototype implementation.

Due to the nature of the problem, a controlled experiment was used to perform design evaluation of the proposed solution. This experiment consisted of two phases:

- Throughput benchmark of a popular open source spam filtering solution and our prototype implementation
- Real-world test of a popular open source spam filtering solution and our prototype implementation

For the real world test, both solutions were deployed on a publicly accessible mail server for 20 days and processed e-mail messages for the same domain name, in order to minimize external influence over the experiment. Metrics were collected and analyzed using quantitative methods, in order to compare observed solutions by several traits:

- Total number of classified spam messages
- Relative difference in the number of classified spam messages between the two observed solutions
Related work

The goal of this chapter is to identify already existing spam filtering methods, different from statistical and pattern-based filters, evaluate their performance and classify them according to stage of communication, when they act.

Methodology

According to Okoli and Schabram [Okoli & Schabram, 2010] a literature review may seek multiple different purposes:

- Providing a theoretical background for subsequent research
- Finding the scale of a research topic
- Answering practical questions by evaluating existing research literature

This paper represents a combination of all of the above. Primarily, it will be used as a base for further development of a Master’s thesis, but on the other hand spam prevention is a broad topic, so this literature review will be used as a topic and scope delimiter.

A stand-alone review of research literature, according to Fink [Fink, 2005], must be systematic by following a methodological approach, explicit by explaining the procedures by which it was conducted, comprehensive by including all relevant material, and reproducible by others who would follow the same approach in reviewing the topic. However, with the imposed time limits, it is not possible to perform a comprehensive literature review, so this chapter will be used to give insight on the searching and screening processes.

The search for relevant literature started with rather broad search terms, such as unsolicited commercial e-mail and spam filtering. These keywords were used to gain knowledge about the scale of the explored field and each of them returned more than 30000 results in Google Scholar. Message filtering is a fast-paced niche and as we are interested in the latest trends, the result set was further reduced by searching only for articles published between 2010 and
2012. At this point it was much easier to review the titles of the articles for relevant information and read the abstract where needed. All articles focused only on content filtering were excluded from further examination as they were out of scope. The whole process was used as a preliminary literature review that helped us identify keywords for further searches.

The preliminary literature review provided insight over the spam protection field and identified several lightweight filtering methods that needed further investigation. After carefully evaluating articles describing these methods, they were documented and classified in several categories. The table below shows the timeline of performed searches and the number of returned results. It should be noted that these numbers represent search results only for articles published between 2010 and 2012.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Date searched</th>
<th>Number of results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsolicited commercial email</td>
<td>1 December 2012</td>
<td>10200</td>
</tr>
<tr>
<td>Spam filtering</td>
<td>1 December 2012</td>
<td>11700</td>
</tr>
<tr>
<td>Lightweight spam filtering</td>
<td>2 December 2012</td>
<td>969</td>
</tr>
<tr>
<td>DNS blacklist whitelist spam filtering</td>
<td>8 December 2012</td>
<td>342</td>
</tr>
<tr>
<td>DNS validation spam filtering</td>
<td>8 December 2012</td>
<td>651</td>
</tr>
<tr>
<td>Email sender authentication</td>
<td>15 December 2012</td>
<td>8780</td>
</tr>
<tr>
<td>Greylisting</td>
<td>16 December 2012</td>
<td>212</td>
</tr>
<tr>
<td>Adaptive greylisting</td>
<td>16 December 2012</td>
<td>44</td>
</tr>
<tr>
<td>Protocol defects spam filtering</td>
<td>22 December 2012</td>
<td>212</td>
</tr>
<tr>
<td>Header-relay detection</td>
<td>22 December 2012</td>
<td>13</td>
</tr>
</tbody>
</table>

Table 1: Keyword information

The results of the above searches were sorted by relevance and crawled until no more relevant articles were found, which means that different numbers of articles were skimmed for each keyword – categories with less than 50 results were fully reviewed, while the most generic search terms produced lots of results, only the first 40-50 of which were on topic. Most of the papers were ruled out after reading their title and/or abstract, but more than 150 articles in
total were considered relevant and therefore were reviewed. However, only a small fraction of them made it into the final paper because of:

- **Overlapping information** – discussion of the same strategies, but from different perspective, or in different time frame
- **Misclassification** – some articles were considered to be relevant on the first pass, but were found to be out of scope after a careful review
- **Poor scientific value** – vague methodology, not well tested, or some other deficiencies

The rest of this chapter represents an overview of the remaining articles, which both had an academic value and were relevant to the performed research.

**Initial literature review**

Nowadays most spam filters examine message headers and contents, in order to be able to classify them by using pattern or probabilistic analysis, according to Lieven [Lieven, 2006]. He agrees that this method works very well for most types of spam, but on the other hand it has some serious drawbacks:

- Content filters require **full message to be received before being able to analyze it** – this late point of classification leads to **high resource usage**
- Spam senders easily **adapt** their messages to **cheat** content filters
- Unable to deal with **image-based spam**

Esquivel, Akella and Mori [Esquivel et al., 2010] classify spam filtering approaches in two categories:

- Filters based on **properties of the sending SMTP server**
- Filters based on the **contents of the e-mail**

Content filters (using pattern or probabilistic analysis) are clearly from the second type, but in
this paper the focus is on the first type. The same article enumerates the following filter based on properties of the sending SMTP server:

- **DNS-based blacklists and whitelists**
- **DNS validation and sender authentication**
- **Greylisting**
- **Protocol defects**

Additionally, Alberto Treviño and J.J. Ekstrom [Treviño & Ekstrom, 2007] describe one more method falling in the same category – **header-relay detection**.

One particular project caught my attention because its stated goals overlapped to great extent with the objective of the current research. Wirebrush 4 SPAM [Pérez-Díaz et al., 2012] is a plug-in based C framework designed for the development of fast spam filters. In order to achieve this it relies on multithreaded design, lazy filter evaluation and caching. However, it still heavily uses traditional techniques such as statistical-based Bayesian filter, so the speed is achieved as a result of better system design and the use of compiled, statically typed language.

Aside from the Wirebrush 4 SPAM research, all other examined papers focused mainly on accuracy rather than resource usage and throughput, which is understandable, but not in line with the recent levels of spam traffic.

**DNS-based blacklists and whitelists**

DNS-based blacklists and whitelists were recently documented by John Levine in RFC 5782 [Levine, 2010]. The idea behind this mechanism is to use the Domain Name System (DNS) to publish lists of IP addresses of known spam-sending servers (**blacklists**) or legitimate SMTP servers (**whitelists**). Each address listed in a DNSxL (DNS blacklist or DNS whitelist) should have a corresponding DNS entry, which is created by reversing the octets of the IP address and appending the domain name of the DNSxL provider. DNS blacklists are also required to provide
a DNS TXT record with the reason for blacklisting, while this is not mandatory for whitelists.

Additionally, Esquivel, Akella and Mori [Esquivel et al., 2010] describe an IP reputation system based on DNS lists and claim that this is the most lightweight and scalable solution for spam filtering. In their paper, three categories of e-mail senders are identified:

- **Legitimate servers**: “privately owned infrastructure server which has been setup with the goal of allowing legitimate users to send email”
- **End-hosts**: end-user machines which are not supposed to send e-mails, but most of the spam today originates from such nodes, most likely infected by malware with built-in SMTP capabilities
- **Spam gangs**: “sophisticated spammers [who] have set-up elaborate mechanisms to provide a false sense of legitimacy to their actions”

A different DNSxL is constructed for each of the above-mentioned types, in order to provide a way to differentiate them and the final conclusion reached is that “effective IP reputation filtering can significantly lower the load on e-mail delivery systems” and can cover up to 90% of all spam and ham (non-spam) messages if the lists are regularly updated.

**DNS validation and sender authentication**

Current SMTP specification [Klensin, 2008] allows any host sending mail to identify itself as any domain name it wants, without any further verification of this claim – this “feature” facilitates the propagation of spam, so several different projects claim to solve the problem.

**Sender Policy Framework**

Sender Policy Framework (SPF) is one of the DNS validation methods. It is documented in RFC 4408 [Wong & Schlitt, 2006] and allows a domain to “explicitly authorize the hosts that are allowed to use its domain name, and a receiving host may check such authorization”. This is achieved by checking SMTP server authorization to use given domain at two different
checkpoints:

- **HELO/EHLO** command: it is *optional*, but *recommended*, to check the domain given by the remote SMTP server as an argument of the HELO/EHLO command
- **MAIL FROM** command: it is *mandatory* to check if the remote SMTP server is authorized to send e-mail for the domain specified in the MAIL FROM address

Authorization is checked by using the DNS for publishing and querying of specially crafted SPF records. Current specification allows SPF information to be published in two different DNS Resource Records (RRs): SPF or TXT.

**Sender ID**

Sender ID is a sender authentication scheme described in RFC 4406 [Lyon & Wong, 2006] and RFC 4407 [Lyon, 2006], and allows SMTP servers to “determine whether an e-mail address in a received message was used with the permission of the owner of the domain contained in that e-mail address”. This scheme is based on the SPF specification, with several minor additions:

- Purported Responsible Address (PRA): while SPF verifies only the MAIL FROM address, Sender ID introduces the notion PRA by using heuristics to establish this address from the headers of an e-mail message, and subsequently checks the authorization of this address
- Positional modifiers: introduced in the Sender ID specification, but not used as of the moment of writing of this paper

**DomainKeys Identified Mail (DKIM)**

DKIM is a method for relating domain name to e-mail message, outlined in RFC 6376 [Crocker et al., 2011] and based on the idea that “assertion of responsibility is validated through a cryptographic signature and by querying the Signer’s domain directly to retrieve the appropriate public key”. This system is based on public key cryptography and its main advantage is that it prevents spammers from forging the source address of their messages.


**Criticism**

Esquivel, Akella and Mori [Esquivel et al., 2010] claim that DNS SPF resource records (RRs) are not only utilized by legitimate e-mail operators, but also by some sophisticated spammers. Due to their nature, this claim is also valid for Sender ID and DKIM, therefore, in order to be effective against spam, these methods cannot be used stand-alone, but only on conjunction with other methods.

**Greylisting**

This method is initially described by Evan Harris [Harris, 2003]. It allows SMTP servers to temporarily reject e-mail from unknown sender, expecting that the remote server will retry to send the e-mail in a per-specified interval. This technique takes advantage of the fact that most spam senders do not use RFC-compliant servers, so once they receive a “soft-fail” response code, they never retry to send the message.

![Greylisting flowchart](image)

Greylisting is based on only three chunks of information (often referred as a triplet) for any incoming message:

- **IP address of the host attempting the delivery**
- **Envelope sender address**
• Envelope recipient address

This triplet is used to uniquely describe a relationship between sender and recipient. It is processed by following a simple rule: “If we have never seen this triplet before, then refuse this delivery and any others that may come within a certain period of time with a temporary failure” [Harris, 2003]. In this case, any standards-compliant SMTP server should attempt retries after certain, pre-specified, interval.

Harris properly documents the timer values used during his testing, but he also recommends changing these values as this will impede spammers from guessing them:

• Initial delay for an unknown triplet: 1 hour
• Lifetime of a triplet that has not yet allowed a mail to pass: 4 hours
• Lifetime of a verified triplet: 36 days

Peter Lieven [Lieven, 2006] notes that while fixed timers keep the algorithm very simple, a lot of useful information may be lost because of them. So he proposes an improved strategy called adaptive greylisting that uses variable timers that adapt according to the amount of messages sent by a given SMTP server. The prototype implementation was stopping 93% of all connections before they even reached the mail queue and thus effectively reduced the load of the SMTP server.

However, it should be noted that greylisting has some deficiencies, documented by Janecek, Gansterer & Kumar [Janecek et al., 2008]:

• Introduces delays to the mail delivery process
• No reliable way to determine to decide if a session refers to a previous delivery attempt
• Spammers can adapt and bypass the filters by sending potentially different messages matching already whitelisted triplets.
• Large organizations often use server farms to handle their outgoing e-mail traffic, therefore attempts to resend a temporarily reject mail may come from different IP address and fail
• It is not clear how should messages with multiple recipients be handled

In order to overcome the above-mentioned problems, Janecek et al. propose two-level greylisting architecture, complemented by a reputation system. This method is based on a slightly different triplet consisting of:

• The last part of the domain name of envelope sender address
• Envelope sender address
• Envelope recipient address

The first component of the triplet is changed to accommodate organizations with outgoing SMTP server farms where temporarily rejected e-mail may come from a different IP on subsequent delivery attempts. The novelty introduced with this method is the second level of greylisting which takes action right after the message data is received allowing for e-mail headers to be analyzed. However, two temporary rejections increase the delay in the mail delivery process and that is why a reputation system is used to deal with known hosts – SMTP servers with reputation above certain threshold do not pass through the greylisting process and their messages are delivered immediately.

**Protocol defects / header-relay detection**

A careful investigation proved that although different terms were used, spam filtering through protocol defects and header-relay detection are basically the same. This method described by Alberto Treviño and J.J. Ekstrom [Treviño & Ekstrom, 2007], relies on strict enforcing of RFC standards and analysis of message headers in order to “identify over 90% of current spam with less than 1% false positives, requiring only about 1ms of processor time per message”.

A major advantage of this method is the low percentage of false positives, because legitimate SMTP implementations are required to strictly adhere to the standards.
Summary

DNS-based blacklists and whitelists can be queried even before fully establishing a SMTP session and yield relatively good results. However, it should be noted that the detachment of the publishing entity is critical for the results, as a malicious organization may publish DNSxL records leading to a very high rate of false positives.

On the other hand, DNS validation and sender authentication methods are controlled directly by the organizations owning the respective domains, so they should be trusted. The drawback of these methods is that this should be widespread, in order to be effective, and this is not an easily achievable goal.

Greylisting, especially in its late incarnations such as adaptive greylisting and multi-level greylisting, has great potential, if the organizations employing it are ready to accept its major disadvantage – delay in the mail delivery process. This delay is theoretically observed only on first contact between two entities, but there are different situations when this may not be acceptable.

Spam identification rate of the protocol defects monitoring solutions is not high, compared to other methods, but the low false positives rate makes this technique particularly appealing to antispam filter implementers.

Based on the stage of communication, when the methods act, they could be separated in three categories:

- Methods that do not require SMTP session to be created
- Methods requiring only part of the SMTP session information
- Mixed methods
Methods that do not require SMTP session to be created

Both DNS-based blacklists and whitelists and DNS validation and sender authentication methods require only an IP address of the sending server, therefore they can act only upon a fully established TCP session.

Methods requiring only part of the SMTP session information

The greylisting method requires the arguments of the MAIL FROM and RCPT TO commands, therefore an established SMTP session is required.

Mixed methods

Protocol defects / header-relay detection is a common name for multiple different checks, all of which require an established SMTP session, but some of them may also need the full session information.

The following table represents a concept map with the methods explored above and the respective sources of information:

<table>
<thead>
<tr>
<th>Concept</th>
<th>Source</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS-based blacklists and whitelists</td>
<td>Levine, 2010</td>
<td>“A DNSxL is a zone in the DNS. The zone containing resource records identifies hosts present in a blacklist or whitelist.”</td>
</tr>
<tr>
<td></td>
<td>Esquivel et al., 2010</td>
<td>“Effective IP reputation filtering can significantly lower the load on e-mail delivery systems”</td>
</tr>
<tr>
<td><strong>DNS validation and sender authentication</strong></td>
<td>Wong &amp; Schlitt, 2006</td>
<td>“E-mail on the Internet can be forged in a number of ways [...] This document describes version 1 of the Sender Policy Framework (SPF) protocol, whereby a domain may explicitly authorize the hosts that are allowed to use its domain name, and a receiving host may check such authorization.”</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>Lyon &amp; Wong, 2006</td>
<td>“Internet mail suffers from the fact that much unwanted mail is sent using spoofed addresses [...] This document describes a family of tests by which SMTP servers can determine whether an e-mail address in a received message was used with the permission of the owner of the domain contained in that e-mail address.”</td>
</tr>
<tr>
<td></td>
<td>Crocker et al., 2011</td>
<td>“DomainKeys Identified Mail (DKIM) permits a person, role, or organization that owns the signing domain to claim some responsibility for a message by associating the domain with the message.”</td>
</tr>
<tr>
<td><strong>Greylisting</strong></td>
<td>Harris, 2003</td>
<td>“[...] method of enhancing the abilities of mail systems to limit the amount of spam that they receive and deliver to their users.”</td>
</tr>
<tr>
<td></td>
<td>Lieven, 2006</td>
<td>“[...] a simple spam filter that enters the mail delivery process at a very early stage. It is able to keep a huge amount of spam out while being very lightweight [...]”</td>
</tr>
</tbody>
</table>
In contrast to existing greylisting approaches, our method successfully handles messages originating from server farms and it cannot be bypassed by sending identical messages repeatedly. It has been illustrated that this technique allows for achieving very high spam blocking rates of 99.9% and higher.

“The tests outlined here are capable of detecting over 90% of current spam with less than 1% false positives. Additionally, these tests require very little training and very little processing power. Furthermore, since the tests work only on the headers, messages that can trick statistical filters (such as phishing scams or image spam) are still easily detected and eliminated.”

Table 2: Concept map
Prototype implementation

The previous chapter identified and classified several lightweight spam filtering methods that will be used as a base for further development in the direction of a high throughput spam filtering system. This chapter focuses on the requirements, design and implementation of an initial version of such system, and the obstacles overcome in the process.

Requirements

The primary goal of this prototype is to reach higher throughput than a traditional statistical-based spam filtering implementation, and in order to achieve this, it will employ an array of techniques to lower its CPU and memory resources usage.

As for platform support, the long term goal is to support POSIX-compliant platforms, such as Linux, FreeBSD and Mac OS X, but the bare minimum for this initial implementation is to run on Linux. POSIX is an acronym for Portable Operating System Interface. It represents a set of standards defining operating system interface and environment, including a command interpreter and common utility commands [Josey, 2011]. POSIX ensures compatibility and interoperability between operating systems, and POSIX-compliant software should run without modifications on a multitude of different platforms.

Last but not least, our aim is not to create a fully functional MTA (Mail Transfer Agent), but a simple, lightweight SMTP proxy that will filter out spam messages and forward the rest to a standards-compliant mail transfer agent that will deliver the messages to their recipients.

Design

As it was mentioned earlier in the paper, design is an iterative process representing the search for optimal solution. Hevner [Hevner et al., 2004] goes further and lists three essential components of the design process: means, ends and laws. Means are the resources available to
construct a solution, ends are the goals and laws represent the constraints of the environment. While the proposed methodological design process involves the specification of all possible solutions and then selecting the optimal, it is mentioned that this is not possible, nor feasible, in many situations. Spam filtering is relatively complex field with multiple variables, especially when resource constraints are taken into consideration, so a satisfactory solution was sought. As described by Hevner et al., this task “involves the creation, utilization, and assessment of heuristic search strategies”. The literature review performed in the previous chapter was used as a foundation to construct an initial implementation of a spam filtering SMTP proxy that was later improved through several benchmarks (more information related to the benchmarking techniques used may be found later in this chapter).

When considering high throughput designs, one of the most commonly referred articles is “The C10K problem” [Kegel, 2003]. The original article addresses certain shortcomings in web servers, but discussed solutions are applicable in multiple areas, including SMTP servers and spam filtering proxies. The 5 most popular network I/O strategies are:

- Serve multiple clients with each thread, and use non-blocking I/O and level-triggered readiness notification
- Serve multiple clients with each thread, and use non-blocking I/O and readiness change notification
- Serve multiple clients with each thread, and use asynchronous I/O
- Serve one client with each thread, and use blocking I/O
- Build server code into the operating system kernel

Due to its complexity, the last option was immediately ruled out. Moreover, it is not feasible from security point of view, as execution of code within the kernel address space imposes severe security implications.

The first option is the most often implemented in network servers, but that doesn’t mean it scales well. As mentioned by Kegel, if the system lacks asynchronous disk I/O layer, the first time
a server needs disk I/O, its process blocks and all clients must wait.

All other options were carefully considered and two emerging technologies combining them were evaluated:

- **Node.js** is a platform for easily building fast, scalable network applications. It uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive, real-time applications that run across distributed devices [node.js, 2013].
- **Go** aims to be an expressive, concise, clean, and efficient programming language that compiles to machine code. Its concurrency mechanisms make it easy to write programs that get the most out of multi-core and networked machines, while its novel type system enables flexible and modular program construction [Go Programming Language, 2013].

Node.js seemed like the better option on first sight, because of its event-driven architecture with non-blocking I/O model. However, Rob von Behren et al. argue that most problems often associated with the threaded model are related to specific implementations, not to the model in general [von Behren et al., 2003]. They go further and claim that threads provide more natural abstraction for high-concurrency systems. Due to the highly concurrent nature of a spam filtering system, Go was thoroughly investigated and determined to be better-suited for multi-core systems where it is able to make most of their advantages out of the box. It is based on a lightweight threading model, where goroutines are managed in user space, not by the operating system kernel. Go was selected as a programming language for the prototype implementation because of its ability to fully utilize modern multi-core systems in order to achieve highly concurrent processing.

**Implementation details**

In order to keep the prototype implementation as simple and straightforward as possible, only the mandatory commands specified in RFC 5321 [Klensin, 2008] are supported:

- **HELO/EHLO** – “hello” or “extended hello” commands are used to introduce and identify a
client to the server. These commands accept exactly one argument and that is the **fully qualified domain name** (FQDN) of the client or its IP address.

- **MAIL FROM** – initiate a mail transaction. Accepts one argument – the e-mail **address of the sender**, also referred as “**reverse-path**”.
- **RCPT TO** – identify an individual recipient of an e-mail. Multiple recipients are specified by multiple uses of this command. Accepts one argument – the e-mail **address of the recipient**, also referred as “**forward-path**”.
- **DATA** – send the actual data of an e-mail message. The end of data is marked with the following 5 symbol sequence – “<carriage-return><line-feed>,<carriage-return><line-feed>”.
- **RSET** – this command specifies that the current **mail transaction** should be **aborted**. Any stored **sender**, **recipient** and **mail data** information should be **discarded**.
- **HELP** – return helpful information to the client.
- **NOOP** – this command does not affect any parameters or previously entered command, and it specifies no action.
- **QUIT** – end of connection, server must send acknowledgement of this command and close the transmission channel.

---

*Figure 2: SMTP state machine*
Two commands specified in RFC 5321 are not implemented as they may be used by malicious entities to disclose and enumerate existing e-mail accounts:

- **VRFY** – asks the server to confirm that the e-mail address, used as an argument of this command, actually exists.
- **EXPN** – asks the server to confirm that the e-mail address, used as an argument of this command, identifies a mailing list, and if so, to return the members of this list.

It should be noted here that there are multiple SMTP extensions, documented in different RFC documents that are supported by other spam filtering solutions, but were not included in this reference implementation, in order to limit the initial development time, but may be considered for future inclusion.

Next important decision was the most efficient, in terms of throughput, order of the spam filters. Earlier performed literature review identified and classified several methods that will be stacked in the following order:

- **DNSxL** – DNS checks do not need any other session information, aside from the IP address of the client, and depend only on network I/O, so they will be performed first. Also, higher priority is given to the DNS whitelist checks, because if an IP address is present in a DNSWL, there is no need to perform an additional DNSBL check.
- **Protocol defects** – the SMTP proxy is checking for strict protocol conformance on each received command (starting with HELO/EHLO), and in case of deviations the spam score of the message is increased.
- **DNS validation and sender authentication** – another set of DNS checks that are performed after MAIL FROM command is received. Currently only SPF and DKIM checks are implemented.
- **Greylisting** – adaptive greylisting with variable timers are implemented, but this technique requires both MAIL FROM and RCPT TO commands to be received, and spends some CPU and memory resources on database checks, in order to verify if a triplet is present on the
Each connection is internally represented with the following data structure:

```go
type Session struct {
    connection net.Conn
    bufferIn *bufio.Reader
    bufferOut *bufio.Writer
    state State
    score float
    sender string
    recipients []string
    message []byte
}
```

Below you may find an explanation about the purpose of all fields:

- **connection** – stores information about the connection – socket descriptors, local and remote addresses, and timeouts.
- **bufferIn** – input buffer.
- **bufferOut** – output buffer.
- **state** – state of the connection, more information about the possible values may be found further in this chapter.
- **score** – current spam score of the message. Each filter may increase (or decrease) this field, based on the checks it performs. Once the value of this field reaches certain configurable threshold, all further processing is stopped and an error code is returned to the client trying to submit the message.
- **sender** – e-mail address of the sender.
- **recipients** – an array storing multiple recipients for a single message.
- **message** – message content.

The state variable contains information about the current state of the connection. This information is used internally by the SMTP proxy to determine what commands are accepted in the current context, and to evaluate if the client is abiding by the protocol. The State type is internally defined like this:

```go
type State int
```
const (  
haveConnection State = iota  
haveHello  
haveMailFrom  
haveRcptTo  
haveData  
)

These states have the following meanings:

- **haveConnection** – a connection has just been initialized, no commands were sent yet.
- **haveHello** – the client has already sent HELO/EHLO command.
- **haveMailFrom** – the client has sent a MAIL FROM command.
- **haveRctpTo** – the client has sent a RCPT TO command.
- **haveData** – the client has sent a DATA command.

The result of the development is a minimalistic, standards-compliant SMTP daemon, that is able to filter e-mails in real-time, during transmission, based on different properties of the sending entity and of the message itself. The code base consists of only 4800 lines, thanks to Go’s terse syntax and rich standard library. The difference between this daemon and other solutions is the order in which filters are executed, and the complete lack of filters based on statistical analysis.

**Testing**

Different techniques were used during development process and after that to evaluate the performance of the prototype implementation and compare alternative solutions.

Initially it was planned to use a spam feed to perform the tests – this would have guaranteed that each time the tests are performed with exactly the same input data. However, it turned out that it is not possible to find spam feed data with full, non-sanitized, mail headers. As it is evident from the implementation details above, the prototype implementation of a lightweight spam filter heavily depends on DNS and requires unaltered data in the `Received` headers of an e-mail message, in order to determine the IP address of the server that is sending it. This data is not present in any feed I have been able to find, so other test solutions were found.
Performance testing

Performance benchmarks are used to determine the maximal throughput of a system. The goal of this paper is to prove that it is possible to filter spam messages with less CPU and memory resources than regular statistical-based filters and thusly increase the overall throughput of the system. In order to determine how the prototype implementation compares to already existing solutions, it was benchmarked against several other implementations:

- **Postfix** – an open source mail transfer agent.
- **SpamAssassin** – widely used open source spam filtering solution; it cannot work as a stand-alone proxy, so it was tested as an after-queue content filter for Postfix.
- **smtp-sink** – a tool, distributed with Postfix MTA, that just receives messages on a network port and discards them. It is used to determine the maximal throughput of a system.

All solutions were tested on the same system, in order to guarantee consistent environment between the different attempts.

Configuration of the test system:

- Type: Virtual Private Server
- CPU: 8 x 2.00 GHz
- Memory: 1 GB
- Disk: 24 GB
- OS: CentOS 6.4

Our prototype implementation is working as a SMTP proxy, so it doesn’t actually deliver any messages – it simply forwards them after processing. The following configuration was used, in order to achieve similar setup for Postfix, and force it to work in proxy mode:

```bash
# Redirect all messages to 127.0.0.1:8025
relayhost = [127.0.0.1]:8025
```

Other than that change, default configuration was used for the Postfix test. For the
SpamAssassin test, several lines in /etc/postfix/master.cf were changed.

Old version:

```
smtp    inet  n  -n  -  -  smtpd
        -o content_filter=spamassassin
        spamassassin unix  -n  -n  -pipe
        user=spamd argv=/usr/bin/spamc -f -e /usr/sbin/sendmail -oi -f ${sender} ${recipient}
```

New version:

```
smtp    inet  n  -n  -  -  smtpd
        -o content_filter=spamassassin
        spamassassin unix  -n  -n  -pipe
        user=spamd argv=/usr/bin/spamc -f -e /usr/sbin/sendmail -oi -f ${sender} ${recipient}
```

Another tool shipped with Postfix that is called smtp-source was used to generate dummy traffic. This program accepts several options allowing us to specify the size of the generated messages, their total number and the number of parallel connections. It was invoked with the following parameters:

```
smtp-source -s $CONCURRENCY -l 512000 -m 10000 -f root -t root 127.0.0.1:25
```

Several tests with 10000 messages, but different number of parallel connections were performed, in order to simulate different types of load, while the size of the messages was always fixed to 512 kilobytes.

The following table represents the distribution of the results gathered using the technique described above:

<table>
<thead>
<tr>
<th>Concurrency</th>
<th>1</th>
<th>10</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>smtp-sink</td>
<td>10076 messages/min</td>
<td>10256 messages/min</td>
<td>10197 messages/min</td>
</tr>
<tr>
<td>Postfix</td>
<td>2103 messages/min</td>
<td>2181 messages/min</td>
<td>1865 messages/min</td>
</tr>
<tr>
<td>SpamAssassin</td>
<td>7 messages/min</td>
<td>6 messages/min</td>
<td>4 messages/min</td>
</tr>
<tr>
<td>Prototype</td>
<td>1117 messages/min</td>
<td>1322 messages/min</td>
<td>1640 messages/min</td>
</tr>
</tbody>
</table>

Table 3: Performance testing results

The concurrency number indicates the number of simultaneous connections initiated. The smtp-
sink tool was used as a baseline, in order to determine the maximal throughput of the system, but it is not a standard-compliant implementation, so it cannot be meaningfully compared to the other solutions.

Clearly, Postfix has the best performance from all tested servers. It did not perform any filtering and that explains its higher throughput. However, the result with 100 concurrent connections is visibly lower and that could possibly mean that the resources on the test server were not sufficient.

Conversely, SpamAssassin with all possible checks enabled cannot compare to any of the other solutions and its throughput decreases further with the increase of the number of concurrent connections. It is true, that the test environment was more on the low end of hardware specification, but with the rise of cloud computing many organizations have to deal with such constrained environments.

On the other hand, our prototype implementation performed quite well and, what is more important, its throughput increased along with the number of concurrent connections. In the case of 10 simultaneous connections it is 40% slower than Postfix, and with 100 simultaneous connections it is only 12% slower than Postfix.

**Accuracy testing**

Evaluation of spam filters is not related only to their performance, but to their accuracy as well. The goal of the developed lightweight solution was to filter at least 75% of all spam messages, before they reach statistical-based filters, so it is considered acceptable if it misses up to 25% of the messages classified as spam by SpamAssassin. However, it is not acceptable if the number of false positives is much higher than SpamAssassin.

Small multiplexer daemon was developed for the purpose of this test. Its only goal was to accept incoming connections on port 25 and multiplex the traffic in real-time to 2 servers on the local network: our prototype implementation and Postfix/SpamAssassin, both running on
Virtual Private Servers with exactly the same configuration:

- Type: Virtual Private Server
- CPU: 2 x 2.00 GHz
- Memory: 512 MB
- Disk: 20 GB
- OS: CentOS 6.4

Both Postfix/SpamAssassin and the prototype implementation were configured to accept messages for the same domain, and the IP address of the box running the multiplexer was set as a MX record for that domain in DNS for 20 days. Other than the domain configuration and Postfix/SpamAssassin integration, no other changes were made to the default Postfix configuration.

![Diagram showing multiplexer and servers](image)

Figure 3: Accuracy testing environment

As stated before, accuracy is just as important in spam filtering as performance. That is why another test was performed – two servers, one with Postfix/SpamAssassin and one with our prototype implementation, were running in parallel for 20 days and processing e-mail traffic for a private previously-registered domain. A multiplexer running in front of the two servers,
guaranteed that each incoming connection was properly proxied to both of them.

The total number of processed e-mails for the test period was 49425 messages, or 2471.25 messages per day.

<table>
<thead>
<tr>
<th></th>
<th>Spam messages</th>
<th>Ham messages</th>
<th>% spam</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpamAssassin</td>
<td>45663</td>
<td>3762</td>
<td>92.38%</td>
</tr>
<tr>
<td>Prototype</td>
<td>42836</td>
<td>6589</td>
<td>86.66%</td>
</tr>
</tbody>
</table>

Table 4: Accuracy testing results

As expected, SpamAssassin performs better than our prototype, because of the benefits provided by the statistical-based filters. However, the prototype implementation was able to catch 93.81% of the messages classified as spam by SpamAssassin. Such performance is clearly encouraging, because even though our prototype was not able to reach SpamAssassin’s accuracy, with such performance it can be used in front of any statistical filter and remove almost 94% of all spam messages before reaching it.
Discussion

Performance and accuracy tests from the previous chapter are indicative that lightweight spam filtering methods are viable alternative to statistical filters, but without proper analysis and discussion, they would be no different than a simple software comparison benchmarks. As stated by Gregor and Jones design research may be used as “a knowledge-building activity, rather than the structural nature of the knowledge or theory that results” [Gregor & Jones, 2007]. This statement fits with the seven guidelines proposed by Hevner [Hevner et al., 2004], and it should be noted that the use of a well tested and proven framework for design science and information systems research greatly facilitated the process, and allowed to plan in advance and anticipate eventual problems. Most of the seven guidelines were previously discussed in the relevant chapters, but in this chapter I will try to synthesize the knowledge gathered during this research and extend existing theories by proposing a new direction for the development of spam filters.

It is evident by the benchmarks in previous chapter that our prototype solution is much faster than SpamAssassin, but they do not answer the most important question – why? I would argue that the main reason for that is the filtering strategy used in the prototype implementation:

- Early filtering – spam messages are filtered as early as possible, in order to not waste resources
- Filter stacking – filters requiring least information for the message should be stacked first

Filtering messages earlier allows us to free resources that can be used to process another mail. So if a filter reaches certain pre-specified level of confidence that a given message is spam, it should immediately reject it, instead of continuing to process it and passing it to the next filter. This principle, however, is not very useful if filters are not “properly” stacked.

Proper stacking could be a rather arguable topic – are the most accurate filters deployed first, or the lightest on system resources should act on the message before the others? In the prototype
implementation, I used a completely different approach: filters were ordered by the amount of information they need to process an e-mail. Therefore, filters requiring only the IP address of the remote server were ordered first, followed by filters that need some header information such as sender or recipients, and filters that need full message contents act last. It should be noted that statistical-based filters fall in the last category, so it is possible to combine both solutions for better accuracy of results, but this is out of the scope of this paper and could be used as a base for future research and development.

A possible concern with the thesis is that the source code of the prototype implementation is not provided for future verification of the results, but the reason for that is the undergoing effort to productionize it, and turn it into a software-as-a-service startup. However, I would argue that the lack of source code is not that important for two reasons:

- Precisely replicating the results is next to impossible due to the nature of the unsolicited commercial e-mails – they are constantly changing and evolving, and therefore it is unreasonable to expect to achieve exactly the same results
- Sticking to the filter stacking order should yield similar results, even considering the volatile nature of spam

Therefore, it is possible to verify the results of this paper, and conduct further research, e.g. explore the combination between property-based and statistical spam filters.

Another important aspect of the solution, proposed in this paper, is its scalability as it is intended for very demanding (or even worse – restricted) environments with limited amount of resources or high amount of traffic. Luckily, scalability is part of the SMTP protocol, so the solution can be scaled by adding additional MX records pointing to as many servers as needed to deal with the traffic. However, it is also possible to scale the solution by using only one MX record pointing to a load balancer, installed in front of a server farm with several spam proxies, filtering traffic in real time and redirecting the cleaned up stream of messages to servers performing the deliveries to users’ inboxes.
Conclusion

The aim of this study was to determine if it is possible to achieve comparable accuracy with lower resource usage and higher throughput than current statistical-based open source solutions.

In order to find the answers of the research questions, I started by exploring existing literature and identifying known lightweight spam filtering methods, that were later classified into several categories by the amount of information required for their actions. This knowledge was used to construct a prototype implementation embodying the ideas developed in the paper, and compare it to a popular open source spam filtering solution, based on statistical methods.

Results from tests show that the prototype implementation performed between 160 and 410 times faster than SpamAssassin, depending on the overall load of the system, while maintaining almost 94% of its accuracy. The lower accuracy may not be acceptable in most situations, but it gives us the possibility to run a solution implementing the ideas explored in this paper in front of a statistical filter, in order to reduce the amount of traffic reaching this filter, thusly increasing its throughput.

It should be noted that while this chapter is the conclusion of this thesis, the work is not completed and more than a year after I began my research, I am still improving the prototype implementation, which already has evolved into a real-world solution that will be the foundation of a new software-as-a-service company providing spam-filtering services for Internet Service Providers or other high-volume environments. Additionally, different components of the whole stack already are, and will be open sourced on my GitHub account, https://github.com/blaskov.
References


Wirebrush4SPAM: a novel framework for improving efficiency on spam filtering services.
Software: Practice and Experience, DOI: 10.1002/spe.2135


