System Oriented Techniques For High-Performance Anti-spam Solutions

by

Zhenyu Zhong

(Under the direction of Kang Li)

Abstract

Email has become a crucial part of life as the Internet has developed. However, a massive influx of spam emails has threatened the usefulness of email communication. Many techniques have been developed, such as machine learning, authentication, collaboration, etc. However, little has been done from a systems perspective to provide an effective, robust and efficient anti-spam solution. The arms race between spammers and anti-spam researchers has brought new challenges to the design of modern anti-spam systems.

This dissertation focuses on the systems aspect of the challenges that the anti-spam researchers face in designing various anti-spam approaches. the system aspects. In particular, we attempt to provide solutions to the challenges in the collaborative approach, stand-alone approach and sender-based approach. These challenges are 1) preserving privacy of email content in collaboration, 2) achieving both high accuracy and high processing speed, and 3) selectively punishing email senders without exact knowledge of whether the email sender is a spammer or a normal user.

We design a novel technique for message transformation to preserve the privacy of email content and derive resemblance information for collaborative email classification. We also carefully design a communication protocol to ensure email privacy during information

exchange among the collaborative entities. The experimental results demonstrate a comparable accuracy and greater robustness compared to Bayesian and Distributed Checksum Clearinghouse approaches. This dissertation proposes a new metric for privacy evaluation and demonstrates a system with excellent privacy preservation.

This dissertation continues to explore the tradeoff between spam filtering accuracy and speed by using approximate classification. It demonstrates about one order of magnitude of speed improvement over two well-known spam filters, while achieving identical false positive rates and similar false negative rates.

For cost-based approaches, we propose to push the spam filter to the early stage of the SMTP conversation, and determine the cost based on the email quality and spam behavior. The experimental results show that under state-of-the-art hardware, the proposed technique can effectively limit the ability of the spammer effectively and significantly even if he possesses more CPU resources than the normal sender.

INDEX WORDS: SPAM, Security and Privacy, Performance Evaluation, Approximation, Bloom Filter, Data Distribution, Computational Cost

System Oriented Techniques For High-Performance Anti-spam Solutions

by

Zhenyu Zhong

B.E., The East China Normal University, 1997

A Dissertation Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2007

System Oriented Techniques For High-Performance Anti-spam Solutions

by

Zhenyu Zhong

Approved:

Major Professor: Kang Li

Committee: David K. Lowenthal
Eileen T. Kraemer
Khaled Rasheed
Jaxk Reeves

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2007

Dedication

This dissertation is dedicated to my beloved wife, Hua Lu and my family.

Acknowledgments

Pursuing a Ph.D is like converting myself from a ragged shape to a piece of "art". Whether this "art" is a real art does not depend solely on my initiative, and hard work. This real art cannot be achieved without the people I would like to thank at this moment.

I would like to thank my advisor Dr. Kang Li for directing me on my research. He not only helps me from the technical perspective, but helps me construct the philosophy on how to do the research as well. More importantly, he directs me on how to face the pressure and keep faith on what I am doing.

I would also greatly thank Dr. Eileen T. Kraemer, who has been sacraficing her spare time, days and nights on giving me suggestions and improving my writing skills and quality of this dissertation. I would also like to express my thanks to my other committee members: Dr. David Lowenthal, Dr. Khaled Rasheed and Dr. Jaxk Reeves for their precious time and great efforts.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Email spam is known as unsolicited bulk email and good email that people are willing to accept is called ham. Since 1978, email spam has become ubiquitous primarily because it is very inexpensive to deliver large amounts of email in a short period. Email is naturally adopted by mass-marketers as a tool to conduct marketing. Spam has been deluging users' email systems, eating up the Internet bandwidth, slowing down mail servers, and consuming storage. According to Securing Computing Inc., by the year of 2008, the daily email volume will reach 100 billion messages everyday, and more than 50 percent of these messages will be spam.

Initially, individuals tried to block spam emails by keyword matching. For example, if the message contained the words "drugs", "mortgage", etc., it was put in the spam folder. However, this naive approach introduced more errors. *The false negative rate* - the percentage of spam emails misclassified as ham messages - is not improved because the spammer adapts to these countermeasures, and develops more sophisticated ways to circumvent the spam filters. *The false positive rate* - the percentage of ham messages misclassified as spam - does not improve either because ham messages might contain those keywords. (For example, people who really need medical care might not be able to receive email as they expect if it contains the word "drugs".) Other more sophisticated anti-spam approaches have been developed, such as DNSBL [11], Greylisting [12], and the Bayesian approach [7] etc., but they are insufficient. More seriously, identity thieves have adopted email. Phishing [1] emails attempt to fraudulently acquire sensitive information, such as usernames, passwords, and bank account information. Another malicious abuse of email occurs through the spread of

viruses. For example, Sober worm [2] is spread through the attachments in an email. Thus, the usefulness of email is threatened, and it becomes a weapon for other Internet attacks as well.

1.1 Challenges

The arms race between the spammer and anti-spam researchers has continued ever since the birth of spam. Initial anti-spam approaches that used to be effective have become inadequate because spammers adapt, causing the decline of the filter accuracy and leading good messages to be misclassified as spam, which creates significant problems for the recipients. New alternative approaches have been built on previous techniques, because a single defense is no longer sufficient. We will provide background on existing anti-spam approaches in chapter 2.2.

In this dissertation, we focus on three different challenges:

- Providing a robust, scalable, large-scale, collaborative anti-spam solution while simultaneously ensuring the privacy of the emails among distrusted email entities.
- Achieving both high filtering accuracy and high processing speed for anti-spam systems.
- Selectively punishing email senders without exact knowledge of whether the email sender is a spammer or a normal user.

1.1.1 Challenge for Privacy-Preserving Collaboration

The economics of spam dictates that the spammer must target several recipients with identical or similar email messages. This makes collaborative spam filtering a natural defense paradigm, wherein a set of email clients shares their knowledge about recently received spam emails and provides a highly effective defense against a substantial portion of spam attacks.

However, any large-scale collaborative anti-spam approach faces a fundamental and important challenge: *ensuring the privacy of the emails among distrusted email entities*. Unlike email service providers such as Gmail or Yahoo mail, which utilize spam/ham classifications from all their users to classify new messages, privacy is a major concern for cross-enterprise collaboration, especially on a large scale.

To protect email privacy, a digest approach has been proposed in collaborative anti-spam systems to both provide encryption for the email messages and to obtain useful information (*fingerprint*) from spam email. Ideally, the digest calculation should be a one-way function so that it is computationally hard to generate the corresponding email message. It should embody the textual features of the email message so that if two emails have similar syntactic structure, their fingerprints should also be similar. A few distributed spam identification schemes, such as Distributed Checksum Clearinghouse (DCC) [74] and Vipul's Razor [29] have different ways to generate fingerprints. However, these systems are not sufficient to handle two security threats: 1) *Privacy breaches*, where an incoming message conveyed among a group of collaborating entities is actually a ham message; 2) *Camouflage attacks*, such as character replacement and good-word appending, make it hard to generate the same email fingerprints for highly similar spam emails.

1.1.2 Challenge to Balance Trade-Off between Accuracy and Speed

It is very challenging to achieve both high accuracy and high speed in spam filtering. Anti-spam systems used to be deployed at the end-user level at the beginning stage of spam. As spam became more serious and the spam volumes became huge, the situation called for a high-performance and accurate anti-spam system be deployed at the enterprise level mail server.

Another challenge facing current anti-spam systems is the significant processing resources required at the enterprise-level mail server. Usually recipients expect the email to arrive without delay. However, the processing overhead introduced by the anti-spam appliance

often ranges from several seconds to even hours. For example, greylisting could temporarily block a message and delay its delivery. Another source of delay is the bursty nature of spam. When a receiving mail server is under spam attack, the mail server must allocate system resources to process this bursty email traffic. Thus a majority of the system resources such as CPU, memory, available ports, and bandwidth etc., are consumed by the spam attacks. The email system can be overloaded and operate slowly. Thus, it delays the normal emails.

To speed up message processing, some native approximate spam filtering can be conducted, where some resource-consuming rules or time-consuming rules are skipped. However, not only does the false negative rate worsen because of this erroneous optimization, but the false positive rate also deteriorates, which leads to more losses and complaints from recipients. In addition to the enormous spam traffic burstiness, some anti-spam systems are not well designed for the high-performance, real-time spam filtering purpose. Many researchers claim to have high accuracy on a certain data corpus, but the speed issue is usually overlooked.

1.1.3 Challenge for Sender-Based Spam Filtering without Exact Knowledge

Achieving both high accuracy and high processing speed does not prevent an increase of spam volume, which becomes another serious problem. Sender-based spam filtering is a promising approach to reduce spam volume.

Currently, most of the anti-spam approaches take place during or after the email communication. Most resources are consumed by the receiving mail server. The spammers' delivery capabilities are not restrained. They can still inject as many spam emails as possible into the Internet without severe punishment or cost. As the volume of spam email jumps, anti-spam systems become more focused on achieving high speed and high accuracy. Some counterattack mechanisms must be applied to reduce spam volumes. A cost-based approach has proved to be effective in resisting network abuses and could be applied against spam. The idea is to reduce spam volume by exhausting spammers' resources. However, this approach can be a

two-edged sword. It can limit spammers' delivery capabilities, but at the same time, it can exhaust the normal users' resources as well. Without distinguishing spam from ham, it is hard to determine how much cost is affordable by normal senders and sufficient to suppress the spammers.

1.1.4 Our Approaches to the Challenges

First, we design **A L**arge-scale **P**rivacy-**A**ware **C**ollaborative **A**nti-spam **S**ystem (ALPACAS),[1] an anti-spam system that simultaneously achieves the conflicting goals of effectively harnessing the power of collaboration for countering spam and ensuring the privacy of the participating entities.

Second, we design a **Hash-based Approximate Inference** spam filter that adopts approximation techniques to speed up Bayesian filters while keeping high classification accuracy. We conduct a comprehensive study on existing Bayesian filters for acceleration purposes. We propose to improve three different stages of Bayesian filters including the pruning, the query, and the scoring stages, by applying approximation techinques respectively.

Third, we adopt a cost-based approach that is the most promising general solution for resisting network abuse to reduce the volume of outgoing spam, especially for Email Service Providers (ESP). We customize this approach in the context of anti-spam and propose a novel mechanism to adaptively assign computational costs to the senders based on their behaviors and the *quality* of their message content as analyzed by the spam filter at the email delivery time. We define the quality of an email as the likelihood of that message being accepted by the recipient as a useful message.

1.2 Dissertation Roadmap

The remainder of the dissertation is organized as follows. Chapter 2 provides a study of spam. We also introduce existing anti-spam approaches adopted widely in the world and

[1] The word "alpaca" refers to a domesticated species of South American camelid developed from the wild alpacas. It resembles a sheep in apperance, but is larger and has a long erect neck.

point out our unique difference from these approaches. Chapter 3 discusses our work in privacy-preserved, collaborative, anti-spam systems. We continue to analyze the tradeoff between accuracy and speed for the Bayesian approach in Chapter 4. Also, we present our cost-based technique to reduce spam volumes in Chapter 5 and show its ease of deployment. Finally, Chapter 6 concludes the dissertation.

Chapter 2

Background of Spam vs. Anti-spam

Email Spam is mostly known as unsolicited bulk emails (UBE), which means the email is unsolicited by the recipients and it is delivered to a large number of recipients in identical or highly similar form. Organizations or individuals send spam emails for different purposes. For example, well-known companies deliver advertisements directly or through third parties to people's inboxes for marketing purposes; Jeremy Jaynes, the most notorious spammer in the world, becomes a millionaire by sending out 10 million emails every day. In year 2003, the US government enacted legislation associated with UBE. It is called "Controlling the Assault of Non-Solicited Pornography and Marketing Act of 2003" (CAN-SPAM). However, spam hasn't been restrained by this law. More seriously, spammers use various mutations to avoid being caught by spam filters. Before the discussion on existing anti-spam approaches, we give background knowledge for spam techniques in the following section.

2.1 Spam Techniques

In this section, we describe the techniques adopted by spammers in three aspects: 1) email addresses harvesting, 2) obscuring the email content, 3) spam email delivery.

2.1.1 Email Address Harvesting

To deliver spam email to a huge number of recipients, the spammers must obtain a large list of email addresses. They can either purchase this information from other spammers, or use "harvesting bots" to crawl Web pages, postings on mailing lists, and other online materials to search for email addresses.

Dictionary attack is another common approach that spammers take to find actual email addresses. The spammer can look up common user names in the dictionary, for example, alex@yahoo.com, adam@yahoo.com, etc. Spammers can add the email addresses to their list if the outgoing emails destined to these addresses are successfully delivered.

Another existing trick played by spammers is to include a fake "unsubscribe" link in the message. Normal users usually click on that link to refuse the next delivery. However, this doesn't happen as the users expect, but rather indicates the validity of the email address to the spammers. This trick is similar to the dictionary attack, but it provides information on which existing email addresses are "alive".

2.1.2 Obscuring Email Content

Spammers craft the email content to confuse the spam filters. By observation and other researcher's work [6], there appear to be many ways to obscure the email content. Here we list a few.

- **Source Forgery:** Spammers have tried many ways to hide their identities. For example, they can fake the "From:" and "Reply to:" addresses, or insert fake mail server records to the return path of the email.

- **Good Word Attack:** To confuse statistical filters, spammers modify their messages by inserting or appending words found in legitimate email.

- **Word Obfuscation:** Spammers deliberately misspell words or phrases in the messages. For example, *viagra* is written as *v1@gra*, *mortgage* is written as *m0rt gage*, etc. The recipient can still recognize the email content. However, the computer cannot tell that these mutated words are actually highly related.

- **Content Chaff:** Spammers confuse statistical filters by randomly picking paragraphs from novels and inserting them in the messages. For instances, the message might

contain paragraphs from Harry Potter books, or Charles Dickens' "David Copperfield" in the spam emails.

- **Picospam:** These emails do not contain many words or phrases except for HTTP links. Although the technique requires the user to click on the link to reach the spammer website, it circumvents content-based spam filters by not exposing much information.

- **Hidden Text in HTML:** Since most current email clients can display Web pages, the spammer hides random text by using HTML techniques so that they don't present any trouble for the recipient to read, but confuses content-based spam filters.

- **Character Encoding:** In HTML, a spammer can encode character to a different representation. For example, *Pharmacy* can be written as *Pharmacy*. A normal content-based spam filter might only trap *Pharmacy* but not *Pharmacy*, thus it cannot capture the spam-sensitive keywords.

- **Image Spam:** Spammers scan the text into images and attach those images to the messages, which makes it harder for traditional spam filters to identify or classify.

There may be even more approaches adopted by spammers beyond the ones listed above; the point is that spammers will tirelessly continue to develop various strategies to circumvent existing content-based anti-spam techniques.

2.1.3 Spam Email Delivery

Spammers are capable of sending a huge number of messages through webmail, open proxies, open relays and spam zombies. We discuss them in turn as follows.

- **Webmail:** It is easy and free for spammers to sign up for online email accounts from email service providers such as Hotmail, Yahoo, etc., and start sending spam. According to Goodman [67], webmail is a substantial source of spam.

- **Open Proxies:** This is a proxy server that allows client connections from any IP address and makes connections to any Internet resources for the clients. It provides Internet services to those who don't have direct access to the services. However, since anyone can access the proxy, it is susceptible to abuse. Spammers can install open proxies on end users' machines through computer viruses and send emails to these proxies. Then the proxies forward the emails to the intended destinations. Proxies enable spammers to hide their true IP addresses. To warn the machines that accidentally become open proxies, anti-spam researchers or organizations maintain the open proxy blacklists to block emails sent from the machines on the list.

- **Open Relays:** Email relay is defined as an email which is not intended for a local user in the mail server and can be relayed by the mail server to its destination. Spammers use automated tools to uncover these open relay mail servers. Then, they can inject large amounts of spam in a very short time. This damages the reputation of the mail servers that have open relay enabled, and more seriously, the huge volume of spam could lead to a denial-of-service for the mail server or even crash the server. Thus, most experts recommend not enabling the relay feature on a mail server.

- **Spam Zombies:** This is the most dangerous spamming strategy. Spammers install viruses or trojans on an unsuspecting user's PC when he browses some websites, such as an adult website. Once activated, these viruses or trojans launch SMTP client application that allow the spammers to send email directly from the compromised PCs. Spam zombies are hard to trace because the victims might have dynamic IP addresses, or the trojans might take over the mail client application installed on the computer (for example, Microsoft Outlook), and send spam on behalf of the owner of the computer. The spammers let the email service provider take the blame. To date, many spam messages have been sent in this manner. The difference between open proxies and spam zombies is that, with spam zombies, spammers do not need to initiate the SMTP traffic themselves.

In general, a spammer thrives by not only collecting email addresses and obscuring the message content, but also in delivering emails in a systematic way. In the next section, we introduce state-of-the-art anti-spam approaches combatting these spam techniques.

2.2 Anti-Spam Techniques

A significant amount of work has been done in the anti-spam area. In this section, we provide necessary background information regarding the conventional state-of-the-art anti-spam techniques.

2.2.1 Protection from Email Address Harvesting

People publish their email addresses on web pages. These addresses are easy for the web crawlers to obtain. One of the common techniques to protect an email address from harvesting is by obfuscation. For example, "johndoe@hotmail.com" can be written as "johndoe AT hotmail DOT com", or "Johndoe@hotmail.com" etc. Another strong defense is to publish a picture containing the email address instead of plain text.

Another way to thwart the harvesters is to analyze the web crawler and distinguish the suspicious email addresses harvesters from the well-behaved ones. Project Honey Pot [17] is a distributed network of web pages that website administrators can include on their sites to collect information about crawlers. These web pages are generated randomly making honey pot hard to recognize for the crawlers. These web pages also have links to the honey pot pages. The invisible links are formatted to be accessible to the crawlers. The honey pot sets a trap by providing fake email addresses and domains so that it can record the information such as IP address and timestamp about the crawlers every time they harvest these fake email addresses. Once the spammers send emails to the fake email addresses, the honey pot can associate the information such as recipient email address and spam email IP addresses with previous records collected. Once the crawler is identified as an email addresses harvester, the honey pot notifies the distributed network so that each member can avoid these crawlers.

To avoid dictionary attacks, users should use complicated usernames, for example, the username should contain characters $[a-zA-Z]$, digits $[0-9]$, and other symbols [._]. By adding complexity to the username, it is less likely that the spammers can figure out the valid email addresses in a short period by dictionary attack.

2.2.2 Initial Anti-Spam Techniques

Whitelist/Blacklist

A whitelist is a user's personal list of pre-approved email addresses or domains. These addresses or domains are permitted to send email to him without checking those messages for spam. Similarly, a blacklist is the user's list of email addresses or domains from which he will not accept any mail under any circumstances. Most spam filters and online email service providers permit users to set up whitelists and blacklists manually. For example, Microsoft Hotmail provides "Safe List" and "Block Senders" options for its users. Each user adds trusted email addresses or domains to the "Safe List" to prevent emails sent by them from being filtered as junk emails. Similarly, known spammer email addresses or spam domains can be added to the "Block Sender" list so that all messages delivered from them are marked as junk email immediately. In the early stages of spam, these two simple approaches were effective, because spammers used real email addresses to deliver bulk email. However, spammers soon got around the blacklist by forging email addresses, because SMTP does not authenticate the email sender. The spammer can even send emails on behalf of someone on the whitelist maintained by the recipient; thus the whitelist involuntarily opens the door for the spammer.

2.2.3 Sender Authentication

One prominent characteristic of spam email is source forgery. In this section, we introduce simple techniques such as DNS-based Blocklist, Greylisting. Then, we continue to dis-

cuss several novel authentication approaches such as Sender Policy Framework, SenderID, DomainKey, etc.

Conventional Sender Authentication

- **DNS-based Blocklist:** Although the spammer can forge the email address, he cannot hide the IP address, because once the TCP connection is established, the receiver gets the source IP address. Some organizations like Spamhaus construct databases of IP addresses of verified spam sources or proxies. The database can be queried through a DNS request, so it is called DNS Blocklist(DNSBL). There are many ways to gather the IP addresses of spam sources or proxies. For example, the blocklist can be filled manually by the administrator, or tested and submitted by users, or populated by Spamtrap, etc. Listed IP addresses don't expire until they receive delisting requests. The administrator will accept the request only if the open proxies or open relay problem is solved by the requester.

  Mail server's DNSBL features can be set to query the database. For example, Sendmail with the DNSBL feature enabled will extract the IP address from the incoming email and make a reverse DNS query to the DNSBL database. If the source is flagged by the DNSBL database, the message is blocked. DNS Blocklist effectively defeats source forgery if the IP address is in the database. Also it alleviates whitelist/blacklist maintenance costs for the end user. However, a legitimate sender sometimes might have trouble delivering email if the spammer compromises his mail server and his IP address is on the DNS Blocklist. The Legitimate sender must make request to the DNS Blocklist host to remove him from the list. This could take hours or days during which the legitimate email sender is unable to deliver email.

- **GreyListing:** Another supplement to the whitelist, blacklist, and DNS blocklist is greylisting. Greylisting is implemented on the receiving mail server. When an incoming message arrives, the server looks at the IP address of the host attempting the delivery,

the sender address, and the recipient address. If this information has never been seen before, the mail server refuses this delivery temporarily and sends back a message with an error code of either "450", "451", or "452" indicating that the requested action is not taken because the mailbox is unavailable, the requested action is aborted due to an error in processing, or the requested action is not taken due to insufficient system storage, respectively. According to the SMTP protocol, error code "4xx" is defined as "Transient Negative Completion reply", which means the receiving mail server can't accept the email at that moment. It requests the sender to retry for acceptance. Thus a normal sending mail server queues the refused messages and makes some reasonable number of attempts to deliver later. Greylisting doesn't impact the legitimate emailers. This simple approach is effective based on an assumption that the spammer never retries. Greylisting doesn't require maintenance at the end-user or administrator level. However, it introduces delays for legitimate emails and the major cost of processing spam is on the receiver side rather than the spammer side.

All the approaches in this section are attempts to avoid spam by investigating the email sender behaviors. Some techniques such as DNSBL can detect open proxies or open relays and alert the victims. The next section describes the latest techniques in authenticating email senders, which formally evaluate the sender identities.

Novel Sender Authentication

Sender authentication is very important not only because spam emails can sneak into recipients' mailboxes by using fake sender addresses, but also because the victims whose email addresses are being abused suffer from a bad reputation from these spam emails. For example, if spammers forge a sender address with an AOL domain, recipients of those spam emails would have an impression that AOL supports spammers.

- **Sender Policy Framework(SPF):** To validate the identity that is associated with a message, the SPF approach has been proposed. A domain that supports SPF must

Table 2.1: Interpret the SPF Check Result

SPF Check Result	Explanation
None	No record is published by the domain or no checkable sender domain could be determined from the given identity.
Neutral	The domain owner explicitly states that he doesn't want to assert whether or not the IP address is authorized. It should be treated exactly like "None" result.
Pass	This indicates that the sender is authorized to deliver emails.
Fail	It explicitly states that the sender is not authorized to use the domain in the given identity. The mail receiver can reject the email.
SoftFail	This can be treated somewhere between "Fail" and "Neutral". The domain believes the host is not authorized but is unwilling to make strong statement. The mail receiver should not reject the message solely on this result, but may put the message to colser scrutiny.
TemError	This means that the SPF client encounters a transient error while performaing the check. The mail receiver may choose to accept or temporarily reject the message. For example, a DNS query timeout could introduce TemError
PermErro	This means that the domain's published records could not be correctly interpreted.

publish SPF records to a domain name server (DNS) as a DNS resource record to authorize the use of the domain name by the mail servers. In an SMTP session, the receiving mail server checks the sender's identity derived from "MAIL FROM" command, because this command provides the sender email address. It is also recommended that SPF checks the "HELO" identity separately, because the identity of email sender can also be derived from "HELO" command. To conduct an SPF identity check, the mail receiver makes a *check_host*() function call with three parameters: IP address of the mail sender, domain name of the "MAIL FROM" or "HELO" identity, and sender name of the "MAIL FROM" or "HELO" identity. The return result is explained in Table 2.1. For detailed information, please refer to RFC 4408 [14]. The *check_host*() function invokes a DNS lookup to the domain that is claimed to be responsible for

the message. If the domain is verified "alive", the SPF records are retrieved on that domain; otherwise the domain is most probably invalid. After the mail receiver gets results, it makes a classification decision according to its local policy.

- **Sender ID:** Derived from SPF, Microsoft designed the Sender ID framework to counter email domain spoofing. Instead of examining fake domains and addresses in "HELO" and "Mail From" SMTP commands, Sender ID validates one of the message's address header fields that is purported to be the responsible address for the email. Sender ID uses the same syntax for SPF records and domain administrators publish SPF records in the DNS for authorizing outbound email servers. Upon receiving the incoming message, the mail server verifies which domain claims to send the message and checks the DNS for an SPF record of that domain. The receiving mail server compares the email sender IP address to the IP addresses published in the SPF record. If there is a match, the mail server accepts the email; otherwise the email is rejected.

  SPF doesn't require outgoing messages to identified. It assigns the source domain the responsibility for the outgoing messages. A different authentication approach known as Domain Key Identified Mail(DKIM) focuses on authenticating the identity for each email when it is transferred through the Internet.

- **Domain Key Identified Mail(DKIM):** DKIM lets the sender sign each outgoing email with a cryptographic signature. The entity that signs the message is called "Signer", while the entity that verifies the signature is called "Verifier". The signature is appended in the "DKIM-Signature:" header field in the email. Figure 2.1 shows an example of the DKIM signature. The signature is composed of several tags and their associated values. The tags are separated by ";". Table 2.2 describes the meanings of the tags mostly used in DKIM and defined in the Internet draft [15].

  To sign a message, two hash values are generated. One is computed over the body of the message, and the other is computed over the selected header fields of the message.

```
DKIM-Signature: a=rsa-sha1; q=dns;
    d=example.com;
    i=user@eng.example.com;
    s=jun2005.eng; c=relaxed/simple;
    t=1117574938; x=1118006938;
    h=from:to:subject:date;
    b=dzdVyOfAKCdLXdJOc9G2q8LoXSlEniSb
    av+yuU4zGeeruD00lszZVoG4ZHRNiYzR
```

Figure 2.1: An example of DKIM signature header

The body hash is computed by a one-way function that is computationally hard to reverse, such as sha1 algorithm [16] specified in "a=" tag and inserted after "bh=" tag in the "DKIM-signature:" header. The header fields selected for hash are specified in "h=" tag. The example shown in Figure 2.1 computes the header hash value of the "from:", "to:", "subject:" and "date:" header fields by using sha1 algorithm. Then a signature is calculated by applying the rsa [3] algorithm on the header hash value with a private key provided by the domain. This signature is inserted after "b=" tag in the "DKIM-signature:" header.

To verify a message, the verifier computes two hash values in the same way as the signer - one hash value for the body and another for the selected header fields. Then the verifier looks at the "DKIM-signature:" header and retrieves the public key according to the information in this header. Usually, the verifier makes a DNS query to the domain specified in the "d=" tag to retrieve the DNS TXT record for public key. Once the verifier receives the public key, he decrypts the signature and compares it to the header hash value. At the same time, he compares the body hash value to the value specified

Table 2.2: Tags on the DKIM-Signature header field

Tag Name	Tag Meaning
a=	The algorithm used to generate the signature. It MUST support "rsa-sha1" and "rsa-sha256". The "rsa-sha256" is recommended.
b=	The actual signature data.
bh=	The hash of the canonicalized body part of the message as limited by the "l=" tag.
d=	The domain that will be queried for the public key.
h=	Signed header fields. A colon-separated list of header field names that identify the header fields presented to the signing algorithm.
q=	A colon-separated list of query methods used to retrieve the public key. By default, it is "dns/txt", which defines DNS TXT record lookup algorithm.
t=	Signature timestamp, which is the time that the signature is created. The format is an unsigned integer indicates the number of seconds since 00:00:00 on January 1, 1970.
x=	Signature expiration. The format is the same as "t=" tag. It is an absolute value indicating the current verification time.

in the "bh=" tag. If both match, the message is authenticated; otherwise the message is improperly signed or spam.

So far, DKIM has been adopted and enhanced by Yahoo's DomainKeys and Cisco's Identified Internet Mail specifications. In addition, other industry players including IBM, American Online, Microsoft, Sendemail, and others are jointly developing an open-standard email authentication specification.

SPF and DKIM are two techniques that attempt to authenticate the identity of the email. They don't require major changes to existing mail systems and are compatible with the mail systems without authentication components. However, one problem of these two approaches is forwarding. For SPF, when the mail is forwarded to a third party, the forwarder may not change the return-path, if the mail server of the third

party checks SPF, the message could be rejected. For DKIM, the signature could be no longer valid through forwarding, thus the valid message is rejected as well.

2.2.4 Contented-based Approaches

We have observed many obfuscations in spam emails. Obfuscations can happen both in email headers and email bodies. Much research has been done to examine the email content for useful information such as spam sensitive keywords or phrases, occurrence frequencies of tokens, etc. In this section we mainly discuss rule-based content filtering and the machine learning approach.

Rule-based Filtering

Usually spam email contains certain patterns. For example, the email content may contain "Herbal viagra", the "To:" field and the "Cc:" field may both be empty, the message body may use small font size, sender address contains number and character sequences, etc. These patterns can be evaluated by a set of rules in the form of regular expressions. Each rule is associated with a score. Every time the message is processed, the set of regular expressions are matched to the email header and body. For example, "$/V(?:agira|igara|iaggra|iaegra)/i$" is a rule to find the misspelled viagra variants. Once a rule is executed, the score is accumulated. Eventually, the overall email score is compared to a pre-defined threshold. The message is classified as spam if the overall score exceeds the threshold.

The most popular rule-based spam filter is SpamAssassin [10]. It maintains a set of rules that is updated periodically to adapt to the latest spam. These rules can be employed to identify spam obscuring techniques such as word obfuscations, hidden text, character encoding, etc. However, the rules are not user friendly. It is not easy for a normal user to define his own rule because it requires a normal user to understand regular expressions. To check a rule, the message content is scanned from beginning to end, which introduces

significant overhead. As the size of the rule set increases, the processing overhead limits the spam filtering performance.

Machine Learning Techniques

Machine learning is a popular technique that can automatically classify email based on knowledge obtained from the email content. This technique eliminates the need for people to manually scrutinize the content and evaluate every email they get.

Machine learning techniques, in general, search a large space of possible hypotheses to determine one that best fits the observed data or previous knowledge held by the learner. The observed data or previous knowledge refer to a set of training examples. In anti-spam, the training examples are a group of emails including spam and ham messages. The hypothesis space can be represented in the form of decision trees, artificial neural networks, etc. Also, a probability can be calculated for each hypothesis, and new instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities. We will mainly discuss the Bayesian approach in the following paragraphs.

The Bayesian probability combination has been widely used in various message classifications. To make a classification, a message is first parsed into tokens (words or phrases), and the frequencies of tokens shown in previously known types (spam or ham) are obtained. Based on the combination of each token's frequency statistics, the message is classified into one or more categories. Here, only two categories are necessary: spam or ham. Almost all the statistic-based spam filters use the Bayesian probability calculation [48] to combine individual tokens' statistics into an overall score and make filtering decision based on the score.

Usually, these filters undergo a training stage in which they gather statistics of each token. The statistic we are mostly interested in for a token T is its *spamminess*, calculated as follows:

$$\mathbf{S}[T] = \frac{C_{spam}(T)}{C_{spam}(T) + C_{ham}(T)} \tag{2.1}$$

where $C_{spam}(T)$ and $C_{ham}(T)$ are the number of spam or ham messages containing token T, respectively.

To calculate the probability of a message M with tokens $\{T_1, ..., T_N\}$ being spam, one needs to combine the individual token's spamminess to evaluate the overall message spamminess. A simple way to make classifications is to calculate the product of individual token's spamminess ($S[M] = \prod_{i=1}^{N} S[T_i]$) and compare it with the product of individual token's hamminess ($H[M] = \prod_{i=1}^{N} (1 - S[T_i])$). The message is considered spam if the overall spamminess product $S[M]$ is larger than the *hamminess* product $H[M]$.

The above description illustrates the idea of statistic-based filters using Bayesian classifications. In practice, various techniques are developed for combining token probabilities to enhance the filtering accuracy. For example, many Bayesian filters, including Bogofilter and QSF [89], use a method suggested by Robinson [49]: Chi-squared probability testing. The Chi-squared test calculates $S[M]$ and $H[M]$ based on the distribution of all the tokens' spamminess ($S[T_0], S[T_1], ...\}$) against a hypothesis, and scale $S[M]$ and $H[M]$ to a range of 0 to 1 by using an inversed chi-square function. Here we give the quations 2.2 and 2.3 to calculate $S[M]$ and $H[M]$, where $C^{-1}()$ is the inversed chi-square function, $2n$ is the degree of freedom and n is the number of distinct tokens in the email. Details of this algorithm are described in [7, 49],

$$H[M] = C^{-1}(-2\ln(\prod_{i=1}^{n}(1 - S[T_i])), 2n) \tag{2.2}$$

$$S[M] = C^{-1}(-2\ln(\prod_{i=1}^{n} S[T_i]), 2n) \tag{2.3}$$

To avoid making filtering decisions when $H[M]$ and $S[M]$ are very close, several spam filters [8, 89, 7] calculate the following indicator instead of comparing H[M] and S[M] directly

$$\mathbf{I}[M] = \frac{1 + S[M] - H[M]}{2} \tag{2.4}$$

When $I > 0.5$, it indicates the corresponding message has a higher spam probability than ham probability, and should be classified accordingly. In practice, the final filter result

is based on $I > thresh$, where $thresh$ is a user-selected threshold. For conservative filtering, $thresh$ is a value closer to 1, which will filter fewer spam messages, but is less likely to result in false positives. As $thresh$ gets smaller, the filter becomes more aggressive, blocking more spam messages but also at a higher risk of false positives. It is first trained with known spam and ham to gather token statistics and then classifies messages by looking at its token's previously collected statistics. A more detailed description of the Bayesian spam filter algorithm can be found in several recent publications [50, 73, 8, 7].

2.2.5 Collaborative Anti-spam System

Effective collaboration among email recipients presents a natural barrier against spam since spammers generally tend to target a large number of users. Recently, there have been many efforts on coordinated real-time spam blocking. Examples include Distributed Checksum Clearinghouse (DCC) [74], Vipul's Razor [29], SpamNet [18], Cyphertrust's Ironmail system [19], P2P spam filtering [20, 21], and SpamWatch [22, 30]. We use DCC as an example to illustrate the mechanism and drawbacks of the collaborative approach, because to the best of our knowledge, DCC is the only open-source, completely non-proprietary, and fully functional collaborative anti-spam system.

The DCC system can be hosted on a centralized server or distributed among a number of participating servers. It maintains spam email information in the digest form, which is computed by hashing functions such as MD5 over email headers and bodies. The DCC servers exchange their spam knowledge periodically and frequently so that the entire DCC system has a consistent and updated overall spam knowledgebase. Each participating DCC server is subscribed to by a number of email clients. The email clients can be either mail servers or mail recipients. Take the mail server as the example. When an email arrives at a mail server, it queries one of the DCC servers with the email digest. The DCC server replies back with the recent statistics about the digest (such as the number of instances of this digest

being reported as spam). The mail server can classify based on the statistics and update the knowledgebase in the DCC server.

The DCC system suffers from ***two*** major drawbacks: ***First***, since hashing schemes like MD5 generate completely different hash values even if the message is altered by a single byte, the DCC scheme is successful only if exactly the same email is received at multiple collaborative servers. DCC tries to improve its robustness by selecting and hashing parts of the messages based on a predefined dictionary. But spammers can easily get around this technique by attaching one or two different words to each email. ***Second***, the DCC scheme does not completely address the privacy issue. Consider the scenario wherein an email server EA_i received a ham email EM_a. Suppose another email server, EA_j too has received the same email EM_a and sends its digest to EA_i. Since EA_i has seen this email before, it immediately discovers that EA_j too has received the same email EM_a. Hence, EA_i learns the content of EM_a that is being queried. Further more, the email sender might have intended to maintain the confidentiality of the recipients. Clearly, this requirement by the sender is violated. We refer to this type of privacy compromise as *inference-based privacy breaches*. These two drawbacks highlight the need for all the existing collaborative mechanisms that to not only accommodate minor differences among messages, but also protect against inference-based privacy compromises.

2.2.6 Cost-based Technique

A cost-based approach is the most promising general solution for resisting network abuse, such as spam [68, 69] and network DoS attacks [75, 76]. Cost takes many forms, including monetary payments [77], "hashcash" [78], and computational puzzles [79]. By requiring the remote peer to consume some computational resources before granting the service, the protected side can reduce the risk of network abuse.

Proof-of-work system

Dwork and Naor [79] proposed a general mechanism that requires a sender to compute a moderately hard pricing function or cryptographic puzzle for each message. The cost to compute the pricing function is negligible for normal users, but high for mass mailers. One of the most famous proof-of-work systems is "hashcash" [78] that requires the sender to produce a string whose cryptographic hash matches to a pre-calculated value. To apply the "hashcash" into an email system, a computational puzzle must be passed to the email sender. This could happen when the receiver responses to the SMTP commands issued by the sender such as HELO, MAIL FROM, RCPT TO, DATA, etc.

This proof-of-work can limit the spammers' capability to send spam by exhausting their resources in the form of CPU power or the number of memory accesses. Various forms of cost have been proposed for the proof-of-work system.

Researchers have searched extensively for a proper form of cost. One question that is still not answered is how much cost should be given to the sender. Also it requires changes to both the senders' and recievers' mail systems.

Tarpits

In addition to the cost in the form of cpu power or memory access times, another approach is to introduce delays to the email sender [85]. For example, during the email greeting stage, the receiver deliberately delays a few seconds. This delay doesn't hurt the normal email sender, but it matters to the spammer sending millions of messages. Marty Lamb [86] extends the tarpits approach in his work by setting up a proxy between the sender and receiver, so that the proxy forwards sender data to the receiving mail server, and returns the response data from receiving mail server back to the sender. This proxy controls the forwarding byte rate and response returning byte rate in different stages of SMTP including HELO, MAIL FROM, RCPT TO, DATA. This doesn't introduce any changes to the current systems. However, it is unclear how long the delay should be.

Other forms of cost

Other cost-based mechanisms have already been applied to the existing mail service. For example, Hotmail has daily message limits that prevent users from sending large volumes of messages; In many systems, users must answer a simple visual recognition problem to prove they are human. However, a legitimate mailing list could suffer from the daily message limits. Also recent research [87] shows that computers can easily accomplish single character visual recognition, thus enabling spammers to thwart those mechanisms.

Chapter 3

Privacy Preserved collaboration against spam

Statistical filtering (especially Bayesian filtering) has long been the bedrock of anti-spam systems, but spam continues to be a serious problem to the Internet society. Recent spam attacks pose strong challenges to the statistical filters, and highlight the need for a new anti-spam approach.

The economics of spam dictates that the spammer has to target several recipients with identical or similar email messages. This makes collaborative spam filtering a natural defense paradigm, wherein a set of email clients share their knowledge about recently received spam emails, and provide a highly effective defense against a substantial fraction of spam attacks. Also, knowledge sharing can significantly alleviate the burdens of frequently training stand-alone spam filters.

However, any large-scale collaborative anti-spam approach is faced with a fundamental and important challenge, namely *ensuring the privacy of the emails among distrusted email entities*. Different from email service providers such as Gmail or Yahoo mail, which utilize spam/ham classifications from all their users to classify new messages, privacy is a major concern for cross-enterprise collaboration, especially in a large scale. The idea of collaboration implies that the participating users and email servers have to share and exchange information about the emails (including the classification result). But, emails are generally considered as private communication between the senders and the recipients, and they often contain personal and confidential information. Therefore, users and organizations are not comfortable sharing information about their emails until and unless they are assured that no one else (human or machine) would become aware of the actual contents of their emails. This genuine

concern for privacy has deterred users and organizations from participating in any large-scale collaborative spam filtering effort.

To protect email privacy, a digest approach has been proposed in the collaborative anti-spam systems to both provide encryption for the email messages and to obtain useful information (*fingerprint*) from spam email. Ideally, the digest calculation has to be a one-way function such that it should be computationally hard to generate the corresponding email message. It should embody the textual features of the email message such that if two emails have similar syntactic structure, then their fingerprints should also be similar. A few distributed spam identification schemes, such as Distributed Checksum Clearinghouse (DCC) [74], and Vipul's Razor [29] have different ways to generate fingerprints. However, these systems are not sufficient to handle two security threats: 1) *Privacy breach* as discussed in detail in section 3.1, and 2) *Camouflage attacks*, such as character replacement and good-word appendant, which make it hard to generate the same email fingerprints for highly similar spam emails.

To simultaneously achieve the conflicting goals of ensuring the privacy of the participating entities and effectively and resiliently harnessing the power of collaboration for countering spam, we design a particular framework and name it *"***A L***arge-scale* **P***rivacy-***A***ware* **C***ollaborative* **A***nti-spam* **S***ystem"* (**ALPACAS**)

In designing the ALPACAS framework, the work in this chapter makes two unique contributions: 1) We present a resilient fingerprint generation technique called *"feature-preserving transformation"* that effectively captures the similarity information of the emails into their respective encodings, so that it is possible to perform fast and accurate similarity comparisons without the actual contents of the emails. Further, this technique also ensures that it is computationally infeasible to reverse-engineer the contents of an email from its encoding. 2) For further enforcing the privacy protection, a privacy-preserving protocol is designed to control the amount of information to be shared among the collaborating entities and the manner in which the sharing is done.

We evaluate the proposed mechanisms through series of experiments on a real email corpus. The results demonstrate that the ALPACAS framework has a comparable overall filtering accuracy to the traditional stand-alone statistical filters. Furthermore, ALPACAS resists various kinds of spam attacks effectively. For good-word attack, ALPACAS has 10 times better false negative rates than both DCC and BogoFilter [7], a well known Bayesian-based spam filter. For character replacement attack, ALPACAS shows a 30 times better false negative rate than DCC and 9 times better false negative rate than BogoFilter. ALPACAS also provides strong privacy protection. The probability of a ham message to be guessed correctly by a remote collaborating peer is well controlled below 0.001.

3.1 Prior Work

Prior efforts on coordinated real-time spam blocking include distributed checksum clearinghouse (DCC) [74], Vipul's Razor [29], SpamNet [18], P2P spam filtering [20, 21] and SpamWatch [22, 30]. We discuss the drawbacks of the existing collaborative anti-spam schemes using DCC as a representative example.

The DCC system attempts to address the privacy issue by using hash functions. Here, the participating servers do not share the actual emails they have received and classified. Rather they share the emails' *digests*, which are computed through hashing functions such as MD5 over the email body. When an email arrives at a mail server, it queries the DCC system with the message digest. The DCC system replies back with the recent statistics about the digest (such as the number of instances of this digest being reported as spam). DCC suffers from two major drawbacks: First, since hashing schemes like MD5 generate completely different hash values even if the message is altered by a single byte, the DCC scheme is successful only if exactly the same email is received at multiple collaborative servers. DCC develops fuzzy checksums to improve the robustness by selecting parts of the messages based on a predefined dictionary. But, spammers can get around this technique by attaching a few different words to each email.

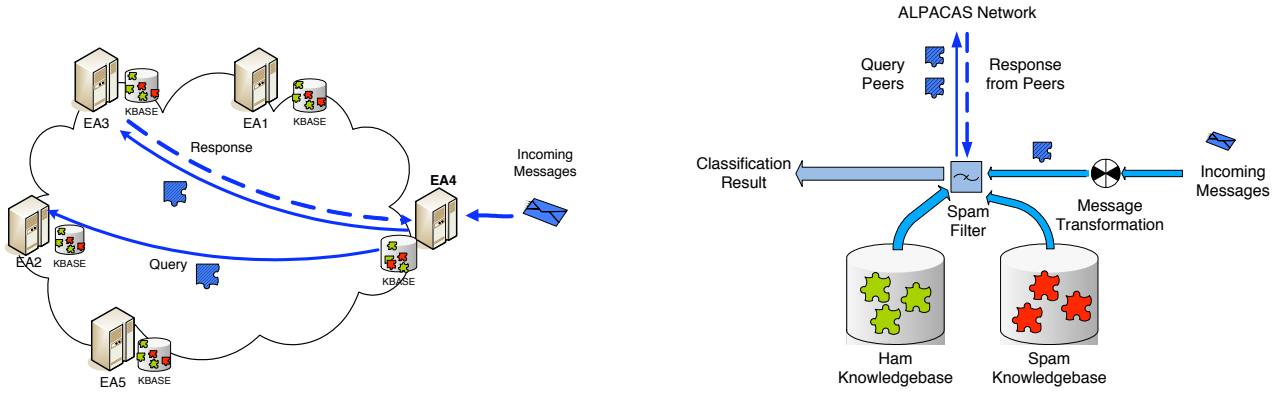
Second, the DCC scheme does not completely address the privacy issue. A closer examination reveals that the confidentiality of the emails can be compromised during the collaboration process of DCC. Thus, it violates the privacy requirement from the email sender for maintaining the confidentiality of the recipients when he wants to deliver emails to multiple recipients by using 'Bcc:'. In particular, one DCC server can possibly infer who else receives the same email by comparing the querying fuzzy checksum. Assuming DCC uses perfect hash function, consider the scenario wherein an email server EA_i received a ham email M_a. Suppose another email server, say EA_j, receives an identical email later, and sends its fuzzy checksum to EA_i. Since EA_i had seen this email before, it immediately discovers that EA_j too has received the same email M_a. We refer to this type of privacy compromise as *inference-based privacy breaches*.

These two drawbacks, namely vulnerability toward camouflage attacks and potential risk of privacy breaches, highlight the need for better collaborative mechanisms that are not only resilient towards minor differences among messages, but are also robust against inference-based privacy compromises.

3.2 The ALPACAS Anti-spam System

We present the ALPACAS framework to address the design challenges of the collaborative anti-spam system.

- **Challenge 1**: To protect email privacy, it is obvious that the messages have to be encrypted. However, in order for the collaboration to be effective, the encryption mechanism has to satisfy two competing requirements: a) The encryption mechanism has to hide the actual contents for privacy protection. b) It should retain important features of the message so that effective similarity comparisons can still be performed on the encrypted messages.



(a) ALPACAS Network (b) Internal mechanism of $EA4$

Figure 3.1: ALPACAS System Overview

- **Challenge 2**: To avoid inference-based privacy breaches, it is necessary to minimize the information revealed during the collaboration process. However, the lesser the information conveyed, the harder it is to perform meaningful similarity comparisons.

Accordingly, the ALPACAS framework includes two unique components, namely *feature-preserving fingerprint* and *privacy-preserving protocol* to address the above challenges respectively. In addition, in the interests of scalability, we design a *Distributed Hash Table(DHT)-based architecture* for distributing ham/spam information among the collaborating entities. DHT is usually defined as a class of decentralized distributed system that privide an efficient lookup service that any participating node can retrieve the value associated with a given name.

The ALPACAS framework essentially consists of a set of collaborative anti-spam agents. An email agent can either be an entity that participates in the ALPACAS framework on behalf of an individual end-user, or it may represent an email server having multiple end-users. Without loss of generality, in this work, we assume that the email agents represent individual end-users. Each email agent of the ALPACAS framework maintains a spam knowledgebase and a ham knowledgebase, containing information about the known spam and ham

SPAM Sample Message No. 1

Hello,

We tried contacting you **a while** ago about your low interest **mortgage** rate. you have been selected for our lowest rate in years… You could get over $420,000 for as **LOW** as $400 a month! **Bad** credit, **Bankruptcy**? Doesn't matter, low rates are fixed no matter what! To get a **no cost**, no **obligation** consultation click below:
http://www.**re-f1nanc3.com/signs.asp**

Best Regards,
Kathie Banks
To be remov(ed: http://www.**re-f1nanc3.com/deletion.asp)**

ALPACAS Feature Set: (297475 384769 555671 743293 798044 1085012 1107317 1243401 **1701456 1783248**)
DCC Digest:
Body: **f23a4d65 f6513269 2ec02108 18de6efe**
Fuz1: **81e889e3 63967036 de719a24 6c65a635**
Fuz2: **abd336ae 2d6fbc1b 69bdc0a6 792389f9**
Vipul's Razor Fingerprint:
1) **hHdm8wvQnv8tt44O8_2cmnW-Y1UA** 2) **QB0M4cGx1qEA**

SPAM Sample Message No. 2

Hello,

We tried contacting you **awhile** ago about your low interest **mort(age** rate. you have been selected for our lowest rate in years… You could get over $420,000 for as **little** as $400 a month! **Ba(d** credit, **Bank*ruptcy**? Doesn't matter, low rates are fixed no matter what! To get a **free**, no **obli,gation** consultation click below:
http://www.**nxshrq.com/i/LzMvaW5kZXgvYXJuLzdhOWoyaTQ0ZGFn**

Best Regards,
Elsa Simons
To be remov(ed: http://www.**nxshrq.com**

ALPACAS Feature Set: (**153049** 297475 384769 555671 **650358** 743293 798044 1085012 1107317 1243401)
DCC Digest:
Body: **ac02a0a8 703ba1ff 1a226388 ba345cc3**
Fuz1: **efacfdc1 a3b1de56 66d9245b 4b69dcd0**
Fuz2: **effdb71e 7212829e 6e4184d6 d61e5339**
Vipul's Razor Fingerprint:
1) **SGvtcOqKomr8QCghbTrUzilRFX0A** 2) **YJG-Dgei1qEA**

Figure 3.2: ALPACAS Feature Sets, DCC and Razor Digests for 2 spam emails (Texts in bold font indicate differences)

emails. Figure 3.1(a) shows the email agent EA_4 querying two other collaborative agents with *partial* information of an incoming message for the purpose of classification. Figure 3.1(b) illustrates the internal mechanism of each email agent: Upon receiving an email, the respective email agent transforms the message into a feature digest. It then uses *part* of the feature digest to query a few other email agents to check whether they have any information that could be used for classifying the email. Based on the responses from these agents and its local knowledgebase, a simple method to classify email is presented in section 3.2.2.

3.2.1 Feature-Preserving Fingerprint

In our approach, the fingerprint of an email is a set of digests that characterize the message content. The set of digests is referred to as the *transformed feature set (TFSet)* of the email. The individual digests are called the *feature elements*. The transformed feature set of a message M_a is represented as $TFSet(M_a)$. In the following sections, we will discuss how to generate $TFSet$ and how to further enforce the privacy preservation.

Shingle-based Message Transformation

Our feature-preserving fingerprint technique is based upon the concept of Shingles [4], which has been used in a wide variety of web and Internet data management problems, such as redundancy elimination in web caches and search engines, and template and fragment detection in web pages [33, 35].

Shingles are essentially a set of numbers that act as a fingerprint of a document. Shingles have the unique property that *if two documents vary by a small amount their shingle sets also differ by a small amount.*

Figure 3.2 presents an example to illustrate the strength of this feature-preserving fingerprint technique. The figure shows two real spam emails that are very similar to each other. The spammers have deliberately mutated one of the emails through word and letter substitutions to obtain the other. The figure shows the *TFsets* of the two emails. For comparison purposes, we also indicate the results of the MD-5 , Vipul's Razor and the DCC transformations on the two emails. For MD-5, Vipul's Razor and DCC, the hash digests of the two emails are totally different from each other whereas the shingle sets of the two emails retain a high degree of similarity that 80% of the *TFsets* of both spam emails are the same.

To generate a *TFset* of a message M, we use a sliding window algorithm, in which a window of some pre-determined length (W) slides through the message. At each step the algorithm computes a Rabin fingerprint [36] of W consecutive tokens (a token could be either a single word or character, and we use character-based token throughout the work in this chapter) that fall within the window. Each fingerprint is in the range $(0,\ 2^K - 1)$, where K is a configurable parameter. For a message with X tokens, we obtain a set of $X - W + 1$ fingerprints. Of these, the smallest Y are retained as the (W,Y) *TFset* of M. We represent (W, Y) *TFset* of a message M as $TFSet_{(W,Y)}(M)$. The similarity between two messages M_a and M_b can be calculated as $\frac{|TFSet_{(W,Y)}(M_a) \cap TFSet_{(W,Y)}(M_b)|}{|TFSet_{(W,Y)}(M_a) \cup TFSet_{(W,Y)}(M_b)|}$.

In consideration of the privacy preservation, the message transformation uses a Rabin fingerprint algorithm, which is a one-way hash function such that it is computationally

infeasible to generate the original email from its $TFset$. However, it is possible to infer a word or a group of words from an individual feature value. The privacy protection requires multiple levels of defenses. In the next subsection, we present our privacy enhancement.

Term-level Privacy Preservation

Term-level privacy breach is defined as a feature element that uniquely identifies a word or a group of words, and an email agent could infer a phrase or a sentence out from a feature with a reasonable probability if the agent had come across a previous message whose $TFset$ contained the same feature value. For example, a term "$99,999" corresponds to a shingle value 16067109. If a recipient of message M_a knows that the encryption of message M_b contains a common shingle value 16067109, he can immediately infer that M_b also contains the term "$99,999".

One approach to mitigate the possibility of inferring a word or a group of words is to shuffle the tokens of the original email and compute $TFset$ on the shuffled email. Though this is expected to accomplish term-level privacy compromise, arbitrary and large-scale shuffling can destroy the email features thereby affecting the spam filtering accuracy.

To shuffle the email content in an acceptable manner, our feature-preserving fingerprint scheme adopts a *controlled shuffling* strategy wherein the tokens are shuffled in a pre-determined format. Further, the position of a token after shuffling is always within a fixed range of its original position.

Specifically, the controlled shuffling scheme works as follows. The email text is divided into consecutive *chunks* of tokens. Each chunk consists of z consecutive tokens of the email text, where z is a configurable parameter. The tokens in each chunk are shuffled in a pre-determined manner, whereas the ordering of the chunks within the email text remains unaltered. Concretely, each chunk is further divided into y *sub-chunks* (we assume that y is a factor of z). The tokens within an arbitrary chunk CK_h are shuffled such that the token at

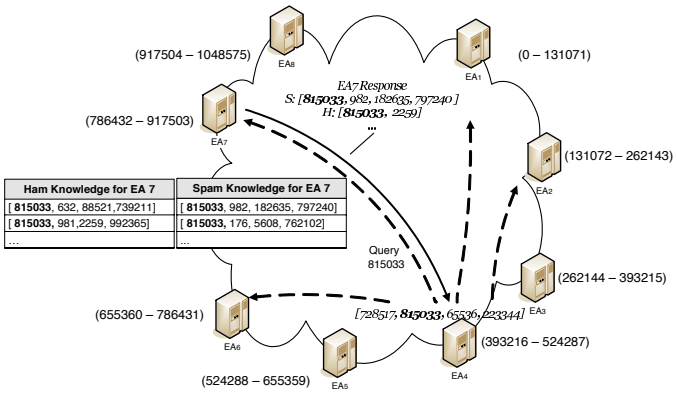


Figure 3.3: ALPACAS Protocol: Query and Response

r^{th} position in the s^{th} sub-chunk (this is the token at the index $(s \times \frac{z}{y}) + r$) in the chunk CK_h) is moved to $(r \times y + s)^{th}$ position within CK_h.

Suppose two messages contain an identical term, by shuffling the term, the rendered text could be different. Thus, it could make the feature element generated from the shuffled text different. We expect this controlled shuffling scheme to reduce the term-level privacy breach. A comprehensive study on this subject will be done in our future work.

3.2.2 Privacy-preserving Collaboration Protocol

Feature-preserving fingerprint is just one level of privacy protection; the amount of information exchanged during collaboration can be further controlled for stronger privacy protection. In particular, we design the collaborative anti-spam system equipped with privacy-aware message exchange protocol based on the following spam/ham dichotomy that *revealing the contents of a spam email does not affect the privacy or confidentiality of the participants, whereas revealing information about a ham email constitutes a privacy breach.*

Our protocol works as follows: When an agent EA_j receives a message M_a, EA_j computes its $TFSet$: $TFSet(M_a)$. It then sends a query message to other email agents in the system to check whether they can provide any information related to M_a. However, instead of sending

the entire $TFSet(M_a)$ as a part of the query message to all agents, EA_j sends very small subsets of $TFSet(M_a)$ to a few other email agents (the email agents to which the query is sent is determined on the basis of the underlying structure (please see Section 3.2.3)). The subsets of $TFSet(M_a)$ included in the queries sent to various other email agents need not be the same (our architecture optimizes the communication costs by sending non-overlapping subsets to carefully chosen email agents).

An email agent that receives the query, say EA_k, checks its spam and ham knowledgebases looking for entries that include the feature subset that it has received. A feature set is said to *match* a query message if the set contains all the feature elements included in the query. Observe that there could be any number of entries in both spam and ham knowledgebases matching the partial feature set. For each matching entry in the *spam knowledgebase*, EA_k includes the *complete* transformed feature set of the entry in its response to EA_j. However, for any matching *ham entries*, EA_k sends back a small, *randomly selected part* of the transformed feature set. Figure 3.3 illustrates our privacy preserving collaboration protocol. In this figure, the agent EA_4 sends a query with the feature element 815033 to EA_7, which responds with a complete feature set of a matching spam email and a partial feature set of a matching ham email.

At the end of the collaboration protocol, EA_j would have received information about any matching ham and spam emails (containing the feature set of the query) that have been received by other members in the collaborative group. For each matching spam email, EA_j receives its complete TFSet. For each matching ham email, EA_j receives a subset of its transformed feature set. EA_j now computes the ratio of $MaxSpamOvlp(M_a)$ to $MaxHamOvlp(M_a)$ and decides whether the M_a is spam or ham. In this work, we use a simple classification strategy that is described in equation 3.1.

$$Score = \frac{1 + MaxSpamOvlp(M_a) - MaxHamOvlp(M_a)}{2} \tag{3.1}$$

If the score is greater than a configurable threshold λ, M_a is classified as spam. Otherwise it is classified as ham.

3.2.3 System Structure

We design an efficient and scalable structure for the ALPACAS prototype which also minimizes the chances of inference-based privacy breaches. Our prototype structure is based upon the following design principle: *A query should be sent to an email agent only if it has a reasonable chance of containing information about the email that is being verified. Contacting any other email agent not only introduces inefficiencies but also leads to unnecessary exposure of data.*

The proposed prototype structure is based on the distributed hash table (DHT) paradigm [38, 40]. In this DHT-based structure, each email agent is allocated a range of feature element values. An email agent EA_j is responsible for maintaining information about all the emails (received by any email agent in the system) whose TFSet has at least one feature element in the range allocated to it. Specifically, if there are N email agents in the collaborative group, the range $(0, 2^K - 1)$ (recall that the all feature elements lie within this range) is divided into N non-overlapping consecutive regions represented as $\{(MinF_0, MaxF_0), (MinF_1, MaxF_1), \ldots, (MinF_{N-1}, 2^K - 1)\}$, where $(MinF_j, MaxF_j)$ denotes the sub-range allocated to the email agent EA_j. EA_j maintains information about every spam and ham email that has at least one feature element between $MinF_j$ and $MaxF_j$ (inclusive of both end-points). For each such spam email, EA_j stores the entire TFSet in its spam knowledgebase. For ham emails, EA_j stores a subset of the email's $TFSet$. If the feature element value Ft falls within the sub-range allocated to EA_j (i.e., $MinF_j \leq Ft \leq MaxF_j$), then EA_j is called the *rendezvous agent* of Ft. The set of rendezvous agents of all the feature elements of M_a is called M_a's rendezvous agent set. The spam and ham knowledgebases at a rendezvous agent is indexed by the feature element that falls within the agent's sub-range. Figure 3.3 illustrates a ALPACAS prototype with eight agents and feature elements in the range of (0,1048575).

The presented DHT structure is only for proof of concept. The work in this chapter focuses on the feasibility of collaboration with transformed messages and we expect that a more sophisticated and robust P2P structure is applied in a real deployment.

3.3 Experiments and Results

In this section, we compare ALPACAS with two popular spam filtering approaches, namely Bayesian filtering and simple hash-based collaborative filtering. We use BogoFilter [7] and DCC as the representatives of these two approaches respectively. As most other Bayesian filters, BogoFilter calculates a score (spamminess) for each message. The message is classified as a spam if its spamminess is greater than or equal to a preset threshold (μ), and vice-versa. On the other hand, the DCC bases its decision on the number of times the email corresponding to a particular hash value have been reported as spam. If this *spam count* of the hash value corresponding to in-coming email exceeds a threshold, the email is classified as spam, and otherwise it is classified as ham.

We conduct a comprehensive study on the accuracy comparison between ALPACAS and BogoFilter for the entire range of the threshold. For other performance measurements, the default threshold for both is set to 0.5. Since DCC is strongly biased to a low false positive rate, we set the DCC threshold to 1, which gives the best false negative rate as shown in Figure 3.5.

3.3.1 Experimental Setup

The datasets used in our experiments are derived from two publicly available email corpora, namely TREC email corpus [41] and the SpamAssassin email corpus [10]. To simulate the collaboration among recipients, we categorize the emails in the TREC corpus, which are the real emails from Enron Corporation according to their target addresses ('To:' and 'cc:' fields) to obtain 67 email sets, each corresponding to the emails received by one individual. Half of each email set including ham and spam are used for training, and the remainder is

used for testing. In the experiment, we also assume that each individual can have a pre-classified email corpus (spamAssassin corpus) as the initial knowledgebase. Each individual incrementally feeds the knowledgebase with a fraction of his email set (TREC) categorized for the training purpose. We apply BogoFilter, DCC and ALPACAS on each individual's email set and measure the overall accuracy results.

3.3.2 Performance Metrics

We use the standard metrics to measure the spam filtering accuracy. A ham email that is classified as spam by the filtering scheme is termed as a *false positive*. The false positive percentage is defined as the ratio of the number of false positive emails to the total number of actual ham emails in the dataset used during the testing phase. The false negative percentage is analogously defined.

Currently there are no available metrics to measure the privacy of collaborative anti-spam systems. In this work, we first define the *message-level privacy breach percentage* as follows. A ham email M_a is said to have suffered a privacy compromise if an email agent that is not a recipient of M_a discovers its contents. *Message-level privacy breach percentage is defined as the ratio number of ham messages suffering privacy compromises to the total number of test messages.*

The communication overhead of the system is quantified through the *per-test communication cost* metric, which is defined as the total number of messages circulated in the system during the entire experiment.

3.3.3 SPAM Filtering Effectiveness

In the first set of experiments we study the effectiveness of the ALPACAS approach in filtering traditional spam messages (as captured by the testing datasets). Figure 3.4 shows the false positive percentages of the BogoFilter, the ALPACAS and the DCC schemes when

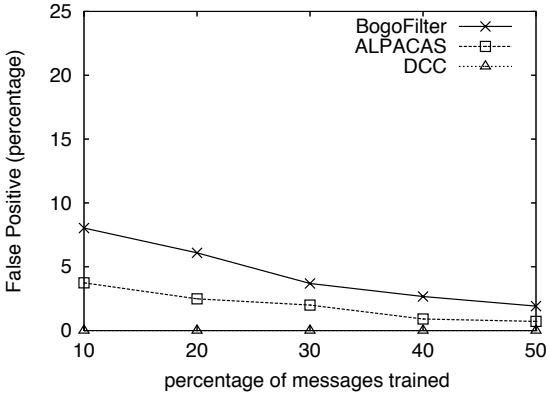


Figure 3.4: False Positive Percentages of ALPACAS, BogoFilter and DCC

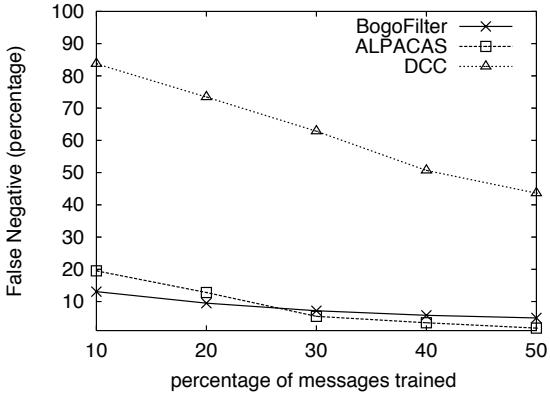


Figure 3.5: False Negative Percentages of ALPACAS, BogoFilter and DCC

the size of the training set employed by each agent increases from 10% to 50% of the total messages in its email set. Figure 3.5 indicates the false negative rates for the same experiment.

In general, as we expect, ALPACAS has a strong feature preserving capabilities and demonstrates a better accuracy than BogoFilter when there are enough email resources shared in the network. Figure 3.4 shows that ALPACAS always performs a better false positive percentage than the BogoFilter. For the false negative percentage shown in Figure 3.5, ALPACAS is better than BogoFilter after around 27% of the messages in the email sets are employed during the training phase. And ALPACAS shows about 60% lower false negative percentage than that of the BogoFilter when 50% of the messages in the email sets are used for training.

The results also indicates that the essence of the collaboration is knowledge sharing. When the size of the training sets employed at the individual agents is small, ALPACAS doesn't demonstrate a better false negative rate than the BogoFilter. It is also natural that the transformed message is less effective than the original message. Furthermore, DCC performs much worse for the false negative percentage than the other two schemes. Note that the false negative percentages of DCC are an order of magnitude higher than our approach.

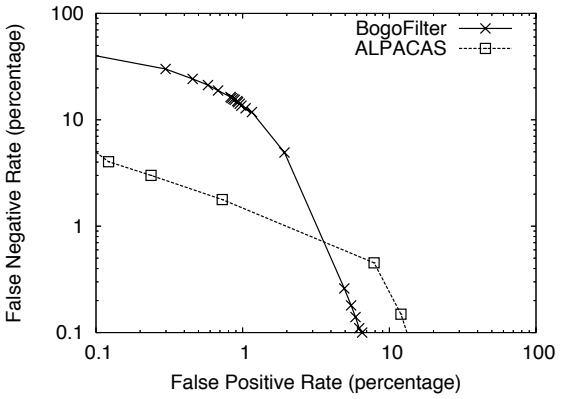


Figure 3.6: System Overall Accuracy

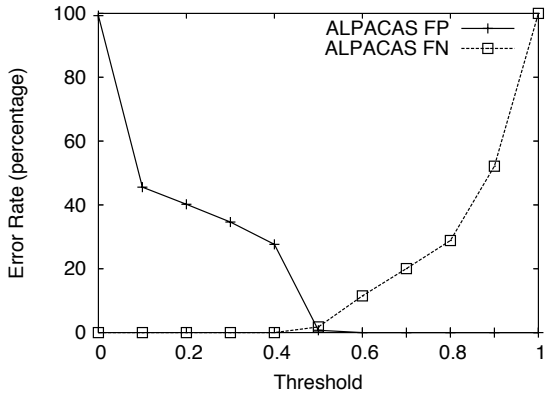


Figure 3.7: Effects of Threshold

All the ALPACAS, DCC and Bayesian schemes are threshold-based approaches, so finding the appropriate threshold to achieve both low false positive and low false negative rates is the key to the success of these approaches. We obtain results from the previous experiment in which 50% of the emails in its email set are used during the training phase. We vary the threshold parameters of the two schemes and collect the false positive and false negative percentages. In Figure 3.6 we plot the results of the experiment with false positive percentages on the X-axis and the false negatives on the Y-axis.

The results show that neither of the approaches outperforms the other at all false positive percentage values. However, the ALPACAS approach yields significantly better false negative results than the BogoFilter for the normally preferred false positive range. Generally, users have a much lower tolerance of false positives than false negatives, and anything more than 1% percent false positives is usually considered unacceptable.

Figure 3.7 shows the effects of threshold parameter (λ) on the false positive and the false negative percentages of the ALPACAS approach. As we expect, the false positive percentages decrease with increasing values of λ, whereas the false negative percentages show a corresponding increase. From this figure, we conclude that the ALPACAS approach yields the best performance when λ is around 0.5.

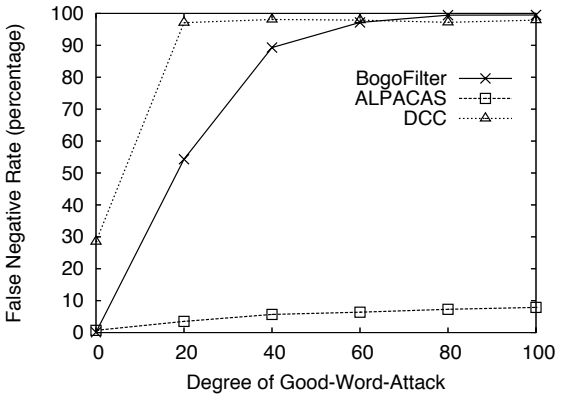


Figure 3.8: System Robustness Against Good-Word Attacks

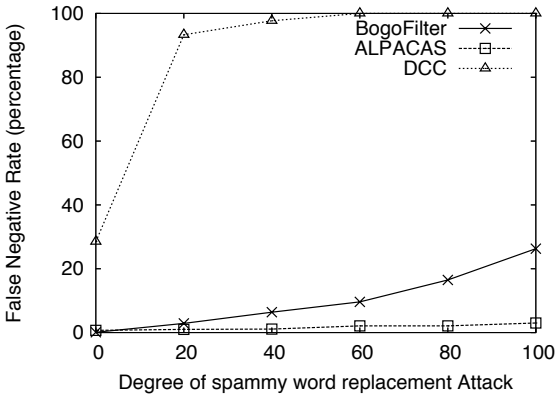


Figure 3.9: System Robustness against Character Replacement Attacks

In summary, ALPACAS has an overall comparable accuracy to the current approaches such as BogoFilter. It has advantages over BogoFilter when a low false positive rate is preferred. Notice that, even with the same accuracy results, a collaborative filter is often preferred because of its resistance to the camouflage attacks, which is presented in the next subsection.

3.3.4 Robustness Against Attacks

In this section we evaluate the robustness of the ALPACAS approach against two common kinds of camouflage attacks, one is *good-word attack* and the other is *character replacement attack*. We compare the results with those of Bayesian and DCC approaches.

In the first experiment of this series, we emulate the good-word attack by appending words that generally appear in ham messages in the test set. The good words are selected randomly from a good word database created from the labeled ham data. We vary the amount of appended words in the range of 0% to 100% of the original emails' word count and we call it *degree of attack*. The experimental setup consists of 67 agents with each agent employing 50% of the messages in its email set during the training phase.

Figure 3.8 shows the false negative rate of BogoFilter, DCC and the ALPACAS approach at various degrees of attack. False positive results are not presented because they are not affected by the attacks. The false negative percentages of the ALPACAS and BogoFilter are very low when the degree of attack is less than 5%. However, the performance of the BogoFilter degrades drastically as the degree of attack increases, whereas the false positive percentage of the ALPACAS approach increases by very small amounts. For example, when the amount of good words introduced is around 80%, the false negative rate of BogoFilter is close to 100%, whereas it is around 7% for the ALPACAS scheme. The performance of DCC is very bad for all its different forms of checksums even at very low degrees of attack. This is because of the nature of its hashing mechanism, which maps similar (but not identical) messages into two totally different hash values.

In the second experiment of this series, we study the resilience of the ALPACAS, BogoFilter, and DCC schemes towards another common type of attack, which we call *character replacement attack*. In this attack the spammer replaces a few characters of a certain fraction of words that are highly likely to be present in spam emails (henceforth, we refer to these words as "spammy words"). The spammer attempts to reduce the spam weight (weight indicating the probability that the email is a spam) assigned by filters to the email. Emails containing "Vi@gra" instead of "Viagra" are examples of character replacement attacks. In order to emulate this attack, we first create a spam dictionary. For each email in the corpus, we extract the words that appear in the spam dictionary. We then replace a few characters of a certain randomly selected fraction of the words in the spam list. The ratio of the number of changed words to the total number of words in the email that appear in the spam dictionary is called the *degree of attack*.

We then measure the filtering effectiveness of the three anti-spam schemes. The setting is similar to that of the previous experiment. Figure 3.9 shows the false negative percentage of the three schemes when the percentage of spam words that are modified in each email varies from 0% and 100%. As the degree of attack increases, the effectiveness of BogoFilter

deteriorates. When 100% of spammy words are modified, the false negative percentage is as high as 27%. In contrast, the false negative percentage of the ALPACAS system is 3% even when 100% of spammy words are modified. The DCC again performs very poorly even at low degrees of attack.

3.3.5 Privacy Awareness of the ALPACAS Approach

One major design consideration of the ALPACAS approach is preserving the privacy of the emails and their recipients. To measure the privacy breaches, we emulate the following model for privacy compromises. When a rendezvous agent EA_i gets a part of the transformed feature set of an email M_a (either for querying or for publishing), EA_i collects all the ham emails received by it that match the part of the feature set that has been sent to it. In the absence of any further information EA_i selects one of these matching ham emails, say M_b as its guess. In other words, EA_i guesses the contents of the email M_a to be similar to that of M_b. If the guess is correct (the contents of M_a are indeed similar to those M_b) then we conclude that a privacy breach has occurred. We count such privacy breaches to calculate the message-level privacy breach percentage.

The privacy breach also relates to how much information is conveyed during the collaboration. We consider three different query policies in our experiment: 1) query with minimal feature set, 2) query with full feature set, 3) query with partial feature set. To further reduce the content breach possibility, we only share spam knowledge across the collaborative network.

In this work, we introduce two metrics to measure the privacy: 1) **Metric 1:** A rendezvous agent EA_i gets a part of the transformed feature set of an email M_a, EA_i takes a guess no matter whether it has matched feature sets in its ham emails. We regard the number of total guesses in the ALPACAS network as G. If EA_i has m matched feature sets in its ham emails, and EA_i has an exact ham message as M_a, the probability p_i to guess that it has the same message as M_a is $\frac{1}{m}$, otherwise p_i is 0. Suppose the number of the agents in the ALPACAS

Table 3.1: Privacy Breach(**Metric 1**): Effectiveness of smaller sets for various number of agents

Privacy Breach		Number of Shingle Values in the Feature Set						
		1	2	10	20	30	40	50
AgentNum	100	0.17%	0.25%	0.28%	0.30%	0.31%	0.32%	0.33%
	200	0.09%	0.13%	0.16%	0.17%	0.18%	0.19%	0.20%
	300	0.06%	0.07%	0.08%	0.09%	0.095%	0.102%	0.107%
	400	0.05%	0.06%	0.065%	0.068%	0.0692%	0.0698%	0.071%
	500	0.06%	0.07%	0.078%	0.083%	0.086%	0.087%	0.089%
	600	0.04%	0.048%	0.057%	0.064%	0.068%	0.073%	0.077%

is N, the overall privacy breach rate is measured as $\frac{\prod_{i=1}^{N} p_i}{G}$. 2) **Metric 2:** The privacy is measured in a similar way with the only difference that EA_i only takes guesses whenever there is a match for the feature set of M_a being queried. So the total necessary guesses in the ALPACAS network is G', $G' \leq G$. And the overall privacy breach rate is measured as $\frac{\prod_{i=1}^{N} p_i}{G'}$. In the rest of this subsection, we present experimental results for these two metrics respectively.

Privacy Awareness using Metric 1

Table 3.1 shows the message-level privacy breach percentages of the ALPACAS approach as the number of collaborating agents vary from 100 to 600 for the three query policies. Since the TREC dataset only contains emails received by 67 individuals, we split the email set corresponding to each user into 10 equi-sized trace files. Each of these trace-files drives an email agent. The number of feature elements in the TFSet of each email is 50, and 50% of the emails in each trace is used during the training phase.

The results show that the privacy breaches are very rare for all three modes of the ALPACAS approach. For metric 1, the privacy breach rate depends on the chances that a querying message in the form of a partial feature set can find an identical feature set in the peer neighbors. Normally, the privacy breach percentages go down as the number of agents

Table 3.2: Privacy Breach(**Metric 2**): Effectiveness of smaller sets for various number of agents

Privacy Breach		Number of Shingle Values in the Feature Set						
		1	2	10	20	30	40	50
AgentNum	100	0.89%	12.3%	42.5%	62.7%	75.5%	83.4%	100%
	200	0.80%	11.8%	38.1%	58.3%	73.6%	83.5%	100%
	300	0.64%	8.6%	33.8%	53.1%	70.4%	78.6%	100%
	400	0.56%	9.7%	39.2%	58.1%	77.8%	86.3%	100%
	500	0.95%	15.27%	51.8%	72.8%	86.1%	90.4%	100%
	600	0.62%	8.4%	29.0%	47.5%	68.5%	78.3%	100%

in the system increases. This can be explained as follows. When the number of email agents in the system increases, the range of DHT values allocated to each email agent decreases. Thus, a rendezvous agent is unlikely to have received a similar email in the recent past. There can be exceptions, for example, in Table 3.1, when the node number is 400, it shows a lower privacy breach rate. It is because when we split the datasets into 10 equal-sized trace files and assemble them into 400 separate agents, it happens to have some identical files in the same agent, in which case it reduces the possibility of privacy breach.

Although with an overall low privacy breach, the reduction of privacy breach by using smaller sets is not as significant as we expected. We ascribe this behavior to the small number of email instances in our testing set when compared to the large feature set space. To demonstrate the reduction of privacy breach by using smaller sets, we presents results using our second metric in the following subsection.

Privacy Awareness using Metric 2

Table 3.2 demonstrates privacy results using our second metric. The result shows that when the full feature set is used as the query, 100% privacy breach is introduced whereever there is a privacy breach threat. However, by using only 1 shingle value in the feature set, the privacy breach is well controlled below 1%. The privacy breach rate increases as the amount

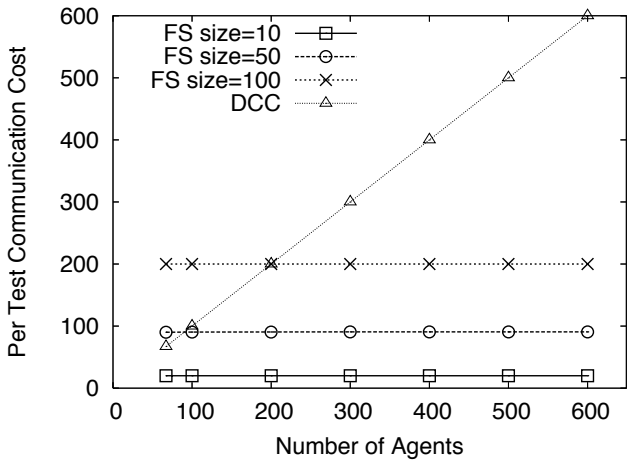


Figure 3.10: Communication Overheads of the ALPACAS and the DCC

of the feature set exposed in the query increases. This is because the more information exposed, the greater the chance the message is guessed correctly. We also find that when 2 shingle values are used, the privacy breach rate increases sharply. We ascribe it to the small number of email instances compared to a large feature set space.

In general, for our second metric, the impacts to the privacy breach rate come from the number of identical messages delivered to multiple recipients. To further control the privacy breach, we plan to continue our study by two means: one is to experiment with various sizes of datasets and feature set spaces; the other is to use feature range in the query rather than the exact feature value, with the hope to further hide the real feature value for the purpose of privacy protection.

3.3.6 Communication Overheads of the ALPACAS approach

Communication overhead is a major factor affecting the performance of collaborative anti-spam systems. We compare the ALPACAS approach with the replicated DCC approach. Figure 3.10 indicates the per-test communication cost of both schemes when the number of agents in the system increases from 67 to 600. We conducted experiments with the size of

$TFSet$ being set to 10, 50, and 100. The training phase employed 50% of the emails in the trace files.

The graph indicates that the per-test communication costs of the DCC approach increases rapidly with increasing number of email agents, whereas the per-test communication costs of the ALPACAS approach essentially remains constant. This result can be explained as follows. In the DCC system, the spam digest database is replicated at each participating agent. Hence, any update to this database has to be reflected at all replicas, which results in high communication overheads. In the ALPACAS approach, the query and publish messages are sent to only the rendezvous nodes of the corresponding emails. The number of rendezvous nodes is directly dependent upon the cardinality of the transformed feature set being employed. Thus, in this scheme the per-test communication costs depend on the number of feature elements in $TFSet$s and not upon the number of participating agents. The results also show that the ALPACAS approach is highly scalable with respect to number of participating agents.

3.3.7 Message Transformation Algorithm Analysis

In this set of experiments, we study the effects of various configuration parameters on the effectiveness of the ALPACAS approach. We first study the effects of feature set size and window size on the accuracy of ALPACAS approach.

Figure 3.11 and 3.12 respectively show the false positive and the false negative percentages of the ALPACAS approach at various settings of the feature set size and the window size parameters. The results show that employing larger number of feature elements yields better classification accuracies. This is because larger feature sets capture more information about the characteristics of individual emails. We also observe that the ALPACAS approach performs best with medium sized windows (windows containing 8-10 characters). This observation can be explained as follows. When the window size is very small, the feature elements correspond to small, commonly occurring sequences of characters. For example, 'agr' can come from either 'vi**agr**a' or '**agr**ee'. Hence, the feature set of an individual email is likely to

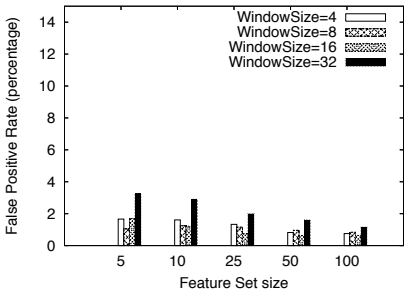


Figure 3.11: False Positive of ALPACAS for Various Parameter Setup

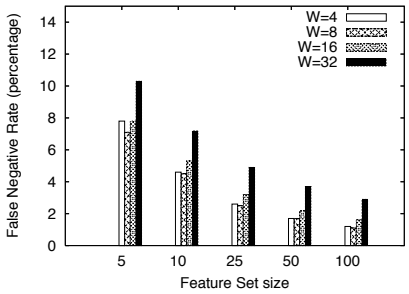


Figure 3.12: False Negative of ALPACAS for Various Parameter Setup

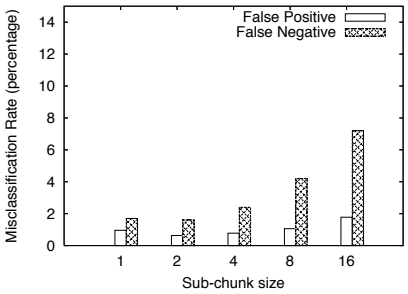


Figure 3.13: Effectiveness of Controlled Shuffling Strategy

exhibit high similarities to both ham and spam emails in the knowledgebases, which affects the classification accuracy. Conversely, when the window size is set to high values, even *similar* emails are likely to have very different feature sets. This is because, when the windows are bigger, each character of the email text appears in several windows. In this scenario, even a few differing characters between two emails can affect the similarity of their feature sets to a considerable extent. Thus, when window sizes are very large, the feature set of an individual email is likely to have very little similarity to either the spam or the ham emails in the knowledgebase. This again affects the classification accuracy.

To protect term-level privacy, we propose the shuffle method. We assume the entire email is a chunk divided into sub-chunks by a factor to increase the shuffling degree. Figure 3.13 shows the false positive and false negative rates for different sub-chunk sizes. The results show that when the shuffling degree increases, the accuracy drops. It is because increasing the shuffling degree would break the similarity among emails. However, we believe that with a small degree of shuffle, the ALPACAS approach can still achieve a high classification accuracy, and the attackers would spend much more effort to infer the content from a single shuffled feature element.

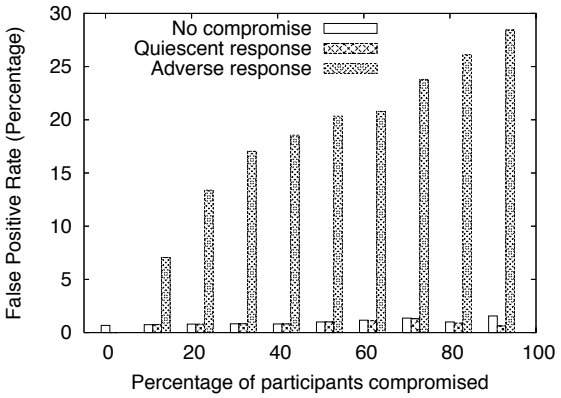


Figure 3.14: False Positive Percentages with compromises

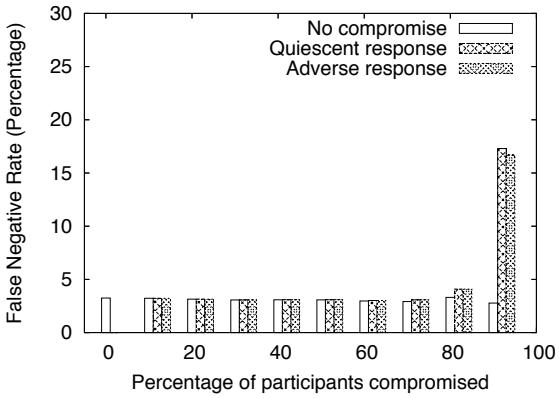


Figure 3.15: False Negative Percentage with compromises

3.3.8 Resilience to the Compromises

It is possible that an attacker can intrude into a system using the ALPACAS approach and disable the spam filtering capability by compromising the participants. It is necessary that ALPACAS resists resiliently to these attacks within a certain range even if some of the participants are compromised.

In this section, we study two scenarios of such attacks: 1) *Quiescent response*, where the intruder compromises participating entities and refuses to answer the queries from peers. 2) *Adverse response*, where the intruder compromises the participating entities and adversely sends the matched records back to the peers (i.e. sends ham records back to the query for spam records, or sends spam records to the query for ham records).

In general, ALPACAS resists the attack in the *quiescent response* scenario, but not in the *adverse response* scenario. Figure 3.14 and 3.15 shows the accuracy results for the ALPACAS approach when various percentages of the participating entities are compromised. Both false positive and false negative rates are measured upon the uncompromised entities and then compared to the scenario that no entity is compromised. This experiment is conducted when spam knowledge is shared across the collaborative network.

In the *quiescent response* scenario, our ALPACAS approach shows nearly the same false negative and false positive rates as the case when no compromise happens, even 80% of the participating entities are compromised. We believe it is because the spam resources are well distributed and duplicated according to our DHT-based architecture. Being quiescent doesn't prevent the ALPACAS approach from answering the query by retrieving spam knowledge from uncompromised entities. We also notice that the false positive is better under *quiescent response* scenario when over 80% of the participating entities are compromised. It is because fewer spam knowledge is returned to the querying entity, which favors the decision of ham if the max-similarity comparison is conducted on dissimilar messages.

In the *adverse response* scenario, the ALPACAS approach still has a nearly identical false negative rate until more than 80% of the participating entities are compromised. But the results show a worse false positive rate, which demonstrates the vulnerability of the ALPACAS approach to such attacks. We would like to employ a reputation system to identify such compromised participants in our future work.

3.4 Discussion

In the current design, we use a simple mechanism for the actual message classification. Approaches like statistical filtering can be utilized in conjunction with the feature preservation transformation scheme. One such strategy would be to apply Bayesian filtering on the feature elements. We believe that sophisticated classification techniques would further improve the filtering accuracy of the ALPACAS approach. Further, our design of the ALPACAS approach assumes that the email agents are stable (i.e., they have low failure rates). Techniques such as replication and finger-table based routing [38] can improve the resilience of the ALPACAS approach towards entries and exits of agents.

An interesting and important side-effect of our approach is that the collaborating agents can discover errors in their own training sets. The training sets for any anti-spam system are a set of emails that are manually classified as ham or spam. Due to the human involvement, it is

not unusual to find mislabeled emails in the training set. Errors in training sets can adversely impact the classification accuracy of any filter. However, in the ALPACAS approach, when an agent tries to upload the feature set of an email into the globally distributed databases, the respective rendezvous agents may notify it of any conflicts they may notice regarding the information being uploaded (such as similar emails that may have contradicting labels). This would enable the agent to re-verify the email's classification. In the process of our experimental evaluation, we found a small percentage of emails in the TREC email corpus that were mislabeled. Specifically, we found 204 pairs of mislabeled emails. Each pair contains two very similar messages, one labeled as ham and the other as spam.

The overhead issue is another challenge that cannot be ignored in the design of our ALPACAS system. The overhead is represented in the form of the number of feature sets transferred in the ALPACAS network. Usually, to gain the best performance, the less and the more precise information transferred in our system, the better the performance and the more accurate the system will be. However, precise information increases the possibility of privacy breach. We regard it as a trade-off between the two conflicting aspects: privacy protection and system overhead. In this work, we mainly discuss approaches to achieve high accuracy and strong privacy by trading off system overheads. As our future work, we plan to design methods to avoid transferring a large amount of feature sets in the system. For example, we can transfer matched feature sets obtained in the past few days instead of passing all the matched ones to the network. This is because the spam email appears more likely to be received by others in the close past rather than months ago. Selectively querying rendezvous agents can reduce the overhead as well.

The current design of the ALPACAS approach assumes that no participating email agent maliciously uploads erroneous information into the knowledgebases. Further, it is also assumed that no email agent in the ALPACAS approach mounts collaborative inference attacks. For example, if the rendezvous agents of an email exchange the feature elements they have received as a part of the query message, then they have a better chance of cor-

rectly guessing the contents of the email. Preventing these types of malicious behaviors by participating agents is a part of our ongoing work.

Chapter 4

A Comprehensive Study on Speeding up Statistical Spam Filter by Approximate Classification

In recent years, statistical-based Bayesian filters [73, 8, 7], which calculate the probability of a message being spam based on its contents, have found wide acceptance in tools used to block spam. These filters can be continually trained on updated corpora of spam and ham (good email), resulting in robust, adaptive, and highly accurate systems.

Bayesian filters usually perform a dictionary lookup on each individual token and summarize the result in order to arrive at a decision. It is not unusual to accumulate over 100,000 tokens in a dictionary, depending on how training is handled [7]. Unfortunately, the performance of these dictionary lookups is limited by the memory access rate, and is therefore relatively insensitive to increases in CPU speed. As a result of this lookup overhead, classification can be relatively slow. Bogofilter [7], a well-known, aggressively optimized Bayesian filter, processes email at a rate of 4Mb/sec on our reference machine. According to a previous survey on spam filter performance [9], most well-known spam filters, such as SpamAssassin[10], can only process at about 100Kb/Sec. This speed might work well for personal use, but it is clearly a bottleneck for enterprise-level message classification.

The goal of our work is to speed up spam filters while keeping high classification accuracy. Our overall acceleration comes from three improvements: 1) *Approximate pruning*, which reduces the latency of duplicate token search by approximating membership checking with Bloom filter. 2) *Approximate lookup*, which allows us to replace memory intensive dictionary lookup with extended Bloom filter based value retrieval. 3) *Approximate scoring*, which

replaces intensive floating point, logarithm operations with lookups on small cache-resident table.

In particular, the major gain of the speedup comes from *"Approximate lookup"*, which is enabled by two novel techniques. The first technique *approximates the dictionary lookup with hash-based Bloom filter* [23] *lookup*, which trades off memory accesses for increase in computation. Bloom filters have recently been used in computer network technologies ranging from web caching [24], and IP traceback [25], to high speed packet classification[26]. While Bloom filters are good for storing binary set membership information, statistical-based spam filters need to support value retrieval. To address this limitation, we extend Bloom filters to allow value retrieval and explore its impact on filter accuracy and throughput. Our second approximation method uses *lossy encoding*, which applies lossy compression to the statistical data by limiting the number of bits used to represent them. The goal is to increase the storage capacity of the Bloom filter and control its misclassification rate.

Approximations by both Bloom filter and lossy encoding introduce a risk of increasing the filter's classification error. We investigate the tradeoff between accuracy and speed, and present design choices that minimize message misclassification. Furthermore, we propose methods to ensure that misclassifications, if they do occur, are biased towards false negatives rather than false positives, as users tend to have much less tolerance to false positives.

Based on the overall approximation on three improvements we mentioned before, the work in this chapter presents analytical and experimental evaluations of the filters using these approximation techniques, collectively, known as Hash-based Approximate Inference (**HAI**). The HAI filter implementations can be applied to most Bayesian spam filters. In this work, the improved filters based on Bogofilter [7] or QSF(Quick Spam Filter) [89] have shown a factor of 6x speedup with similar false negative rates (7% more spam) and identical false positive rates compared to the original filters.

The scope of the work in this chapter is limited to optimizing the processing speed of a particular anti-spam filter and preserving its current classification accuracy. Difficulties and

limitations [46, 45] with the general statistic-based anti-spam approach are beyond the scope of this work.

4.1 Review of Bayesian Filters

Here we give a simple review to provide necessary background for the discussion of the filter acceleration.

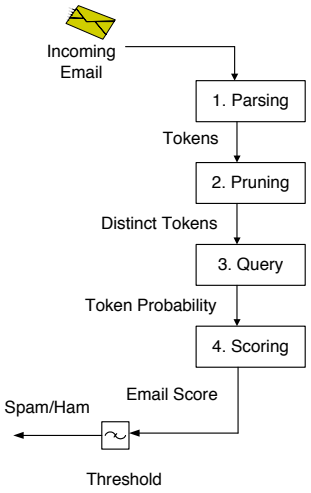
4.1.1 Anatomy of Bayesian Filters

Bayesian probability combination has been widely used in various message classifications. To classify a message, a traditional Bayesian filter typically processes the message in 4 stages as shown in Figure 4.1: 1) *Parsing stage*, where the message is parsed into a set of tokens (words or phrases). 2) *Pruning stage*, where the distinct tokens are extracted from the parsed result. Pruning is an optional stage of Bayesian filters depending on whether the score calculation considers duplicated tokens or not.[1] 3) *Query stage*, which looks up each token's occurrences in previously known types (spam or ham). The frequency statistics information is obtained from a set of training messages which are labeled explicitly as spam or ham and stored in a database for future lookup. 4) *Scoring stage*, where Bayesian filters combine all the token statistics of an incoming message to an overall score by a Bayesian probability calculation [48]. Finally, a filtering decision is made based on the score and a pre-defined threshold.

4.1.2 Score Calculation in Naive Bayesian filter

Most previous studies on statistical filters focus on various types of Bayesian probability calculation [48]. Usually, these filters first go through a training stage that gathers statistics of each token. The statistic in which we are mostly interested for a token T is its *spamminess*,

[1] We investigate several Bayesian filters implementations and find that some implementations have pruning stage for example, Bogofilter, spamAssassin while others such as QSF(Quick Spam Filter), spamBayes don't.



Stage	Bogofilter-style Implementation	Speedup
Pruning [Optional]	Traverse Token List	Membership Test with a traditional Bloom filter
Query	Database lookup	Approximate lookup with an extended Bloom filter
Scoring	Intensive arithmetic operation (both CPU and MEM)	Approximate table lookup

Figure 4.1: Bayesian Filter Stages: The stage with its output is on the left, the speedup techniques corresponding to the stages are on the right.

calculated as follows:

$$\mathbf{S}[T] = \frac{C_{spam}(T)}{C_{spam}(T) + C_{ham}(T)} \tag{4.1}$$

where $C_{spam}(T)$ and $C_{ham}(T)$ are the number of spam or ham messages containing token T, respectively.

To calculate the possibility for a message M with tokens $\{T_1, ..., T_N\}$, one needs to combine the individual token's spamminess to evaluate the overall message spamminess. A simple way to make classifications is to calculate the product of individual token's spamminess ($S[M] = \prod_{i=1}^{N} S[T_i]$) and compare it with the product of individual token's hamminess ($H[M] = \prod_{i=1}^{N} (1 - S[T_i])$). The message is considered spam if the overall spamminess product $S[M]$ is larger than the *hamminess* product $H[M]$.

$$S[M] = C^{-1}(-2\ln(\prod_{i=1}^{n} S[T_i]), 2n) \tag{4.2}$$

$$H[M] = C^{-1}(-2\ln(\prod_{i=1}^{n} (1 - S[T_i])), 2n) \tag{4.3}$$

Stage 1. Training
- Parse each email into its constituent tokens
- Generate a probability for each token W
  - $S[W] = C_{spam}(W)/(C_{ham}(W) + C_{spam}(W))$
- store spamminess values to a database

Stage 2. Filtering
- For each message M
- while (M not end) do
  - scan message for the next token T_i
  - optional token pruning
  - query the database for spamminess $S(T_i)$
  - calculate accumulated message probabilities
    - $S[M]$ and $H[M]$
- Calculate the overall message filtering indication by:
  - $I[M] = f(S[M], H[M])$
  - f is a filter dependent function,
  - such as $I[M] = \frac{1+S[M]-H[M]}{2}$
- if $I[M] > threshold$
  - msg is marked as *spam*
- else
  - msg is marked as *non-spam*

Figure 4.2: Outline for A Bayesian Filter Algorithm

The above description is used to illustrate the idea of statistic based filters using Bayesian classifications. In practice, various techniques are developed for combining token probabilities to enhance the filtering accuracy. For example, many Bayesian filters, including Bogofilter and QSF [89], use a method suggested by Robinson [49]: Chi-squared probability testing. The Chi-squared test calculates $S[M]$ and $H[M]$ based on the distribution of all the tokens' spamminess ($\{S[T_0], S[T_1], ...\}$) against a hypothesis, and scales $S[M]$ and $H[M]$ to a range of 0 to 1 by using an inversed chi-square function. Here we give the equations 4.2 and 4.3 to calculate $S[M]$ and $H[M]$, where $C^{-1}()$ is the inversed chi-square function, $2n$ is the degree

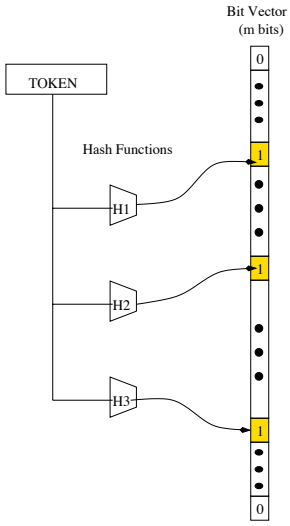


Figure 4.3: Training for a Bloom Filter

of freedom and n is the number of distinct tokens in the email. Details of this algorithm are described in [7, 49].

To avoid making filtering decisions when $H[M]$ and $S[M]$ are very close, several spam filters [8, 89, 7] calculate the following indicator instead of comparing H[M] and S[M] directly

$$\mathbf{I}[M] = \frac{1 + S[M] - H[M]}{2} \tag{4.4}$$

When $I > 0.5$, it indicates the corresponding message has a higher spam probability than ham probability, and should be classified accordingly. In practice, the final filter result is based on $I > thresh$, where $thresh$ is a user selected threshold. For conservative filtering, $thresh$ is a value closer to 1, which will filter fewer spam messages, but less likely to result in false positives. As $thresh$ gets smaller, the filter becomes more aggressive, blocking more spam messages but also at a higher risk of false positives. A general Bayesian filter algorithm is presented in Figure 4.2. It is first trained with known spam and ham to gather token statistics and then classifies messages by looking at its token's previously collected statistics.

A more detailed description of the Bayesian spam filter algorithm can be found in several recent publications [50, 73, 8, 7].

4.2 Our Approach

4.2.1 HAI Filter Architecture Overview

This section presents Hash-based Approximate Inference (HAI) filter. The HAI algorithm is presented in Figure 4.4, which applies a combination of 3 speedup techniques including *approximate pruning*, *approximate lookup* and *approximate scoring* corresponding to the typical Bayesian filters as shown in Figure 4.1.

In the *Pruning stage*, the conventional Bayesian filters such as Bogofilter conduct duplicate token search by traversing a token list. HAI filter replaces it with fast membership checking on a compact traditional Bloom filter (see section 4.2.2). The Bloom filter is initialized to be empty and each newly parsed token from the message is first checked against the Bloom filter. The token is discarded if it is already a member of the set; otherwise it becomes a member of the set and is passed onto the query stage. The effectiveness of this approximation is presented in section 4.3.7.

In the *Query stage*, Bayesian filters often rely on databases such as BerkeleyDB to store the token statistics. However, the multiple memory access latencies limit the database lookup speed. Accordingly, we approximate the token statistics lookup into an extended Bloom filter with value retrieval support (see section 4.2.3). Specifically, the approximation in this stage includes approximate quantization and approximate lookup (see section 4.2.4). The accuracy of the HAI filter relies on the setup of the extended Bloom filter. The details of controlling the accuracy of query stage are discussed in section 4.2.5.

In the *Scoring stage*, the overall email score is usually calculated by combining all the token probabilities via an inversed chi-square function (Fisher method [49]). It takes intensive floating point, logarithm operations if precise calculation is used. HAI reduces this overhead

by replacing it with a two dimensional cache-resident percentage points of chi-square distribution table. This table can be pre-calculated based on the inputs stated in equations 4.2 and 4.3. The effectiveness of this approximation is presented in section 4.3.7.

4.2.2 Bloom Filters

A Bloom filter is a compact data structure designed to store queryable set membership information [23]. Bloom filters were originally invented by Burton H. Bloom to store large amounts of static data such as English hyphenation rules.

A Bloom filter consists of a bit vector of length m that represents the set membership information about a dictionary of tokens. The filter is first populated with each member (token) of the set (Figure 4.3 shows a Bloom filter in this training phase). At the training phase, for each token in the membership set, h hash functions are computed on the token producing h hash values each ranging from 1 to m. Each of these hash values addresses a single bit in the m-bit vector, and sets that bit to 1. Hence for perfect hashes, each token causes h bits of the m-bit vector to be set to 1. In the case that a bit has already been set to 1 because of hash conflicts, that bit is not changed.

Querying a token's membership is similar to the training process. Figure 4.5 shows a Bloom filter in the query stage with a non-member token. For a given token, h hash results are produced and each addresses one bit. The token is guaranteed not in the set if any of these bits is not set to 1. If all the h bits are set to 1, the token is said to belong to the set. This claim is not always true because the fact of these h bits being 1 could be a result of the hashes of multiple other member tokens. This case is considered to be a false positive for membership testing. The likelihood of false positive occurrence can be made very small by carefully choosing the size of bit vector and number of hash functions. We illustrate this with a brief overview of the false positive probability derivation: Assuming perfect hash functions and an m-bit vector, the probability of setting a random bit to 1 by one hash is $1/m$, and thus the probability that a bit is not set by a single hash function is $(1 - 1/m)$. For h hash

Stage 1. Training
- Parse each email into its constituent tokens
- Generate a probability for each token W
  - $S[W] = C_{spam}(W)/(C_{ham}(W) + C_{spam}(W))$
- **quantizing the probability values**
- **store values to an extended bloom filter**

Stage 2. Filtering
- For each message M
- while (M not end) do
  - scan message for the next token T_i
  - optional token pruning
  - **query the extended bloom filter for** $S(T_i)$
  - calculate accumulated message probabilities
    - $S(M)$ and $H(M)$
- Calculate the overall message filtering indication by:
  - $I(M) = f(S(M), H(M))$
- if $I(M) > threshold$
  - msg is marked as *spam*
- else
  - msg is marked as *non-spam*

Figure 4.4: HAI Filter Algorithm (Highlights are changes made to Bayesian filters)

functions, the probability that a bit is not set by any of the hashes is $(1 - 1/m)^h$. For a member set with n tokens, the probability of a bit not set is

$$\mathbf{P0} = (1 - \frac{1}{\mathbf{m}})^{\mathbf{n*h}} \tag{4.5}$$

and the probability of a bit set to 1 is

$$\mathbf{P1} = 1 - (1 - \frac{1}{\mathbf{m}})^{\mathbf{n*h}} \tag{4.6}$$

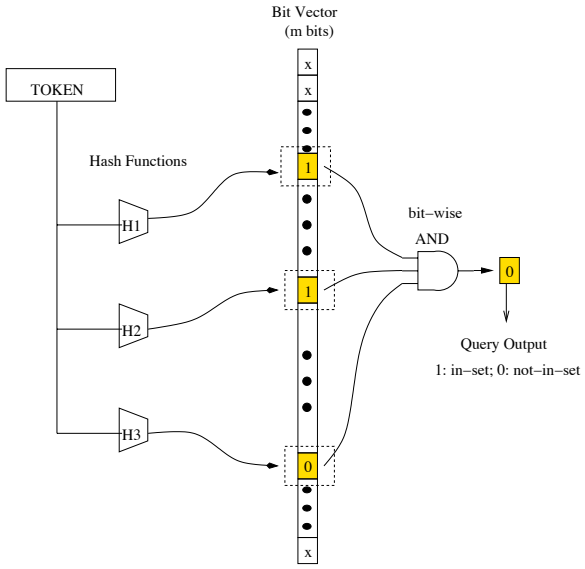


Figure 4.5: Query a Normal Bloom Filter

For a non-member token to be misclassified as a possible set member, all the h bits addressed by h hash functions must be 1. Thus the probability of a false positive is

$$\mathbf{P_{m,n,h}(fpos)} = (1 - (1 - \frac{1}{\mathbf{m}})^{\mathbf{n*h}})^{\mathbf{h}} \tag{4.7}$$

Note that the above probability is the false positive for token membership testing, which is *very different* from the false positive of email message classification.The latter usually combines multiple tokens' spamminess values in order to arrive at a probability result. The next section discusses how to control the effect of the Bloom filter misclassification to minimize the email message misclassifications.

4.2.3 Extending Bloom Filter

Traditional Bloom filters only make membership queries that verify whether a given token is in a set, but applications such as spam filters must retrieve each token's associated probability value. We extend the Bloom filter to serve for value queries while preserving the Bloom filter's

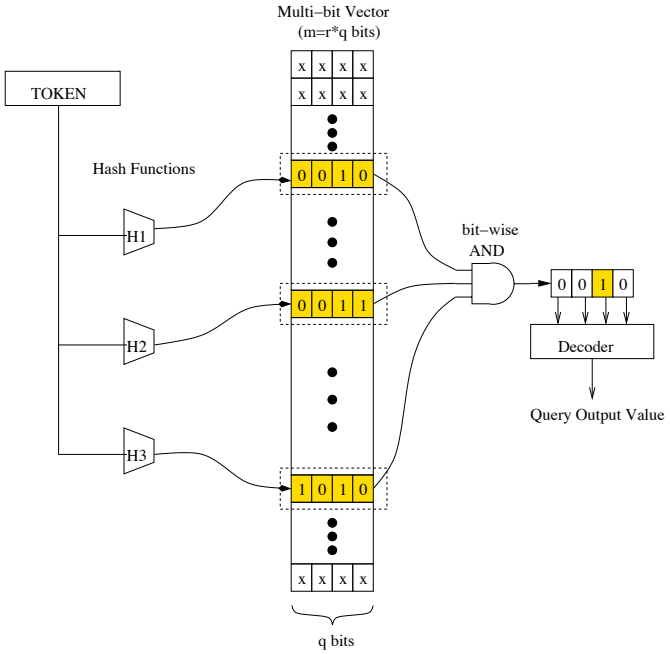


Figure 4.6: Bloom Filter Extension for Value Retrieving Query (The Bit-Vector has r entries and q bits per entry)

desired original operating characteristics. For a given token in the member set, the extension returns a value that corresponds to a given token.

The idea of this extension is to simply maintain a two-dimensional vector, which has a total *bit-vector size* of m bits, and every hash output points to one of the r entries, each of which has q bits (i.e m is the product of r and q). The traditional Bloom filter becomes a special case of this extension that uses one bit per entry (q=1).

Figure 4.6 shows the structure of this Bloom filter extension. It works in the following way to support value retrieval. During the Bloom filter training phase, each training token runs through the hash functions and addresses h entries (each entry contains q bits). Assume the token has an associated value (in integer) v in a range of 0 to $q-1$. The value v is then stored to the Bloom filter extension by setting the vth-bit to 1 on all these h entries. During the query phase, each incoming token also goes through the hashing and addresses h entries.

The query outcome for this token is based on the logical AND of all h entries. If none of the bits is set in the logical AND output, it indicates that the token in the query is not in the training set. If a bit is set, then based on the position of the bit, we retrieve the value associated with the token.

4.2.4 Extended Bloom Filter based Approximate Lookup

The query stage is improved by two means: approximation by quantization and approximate lookup.

Approximation by Quantization

To effectively use the Bloom filter extension for approximate value retrieval queries, we introduce lossy encoding (quantization) to represent the individual token's spamminess value. Consider the way that the Bloom filter extension represents a statistical value by marking one bit of a q-bit entry. It has to adopt some quantization technique if the amount of potential numbers to be represented is infinite.

During the training phase, each token T obtains a probability value p based on the relative frequency of occurrence in ham and spam. HAI differs from a traditional Bayesian filter in that it maps a token's probability value p to an integer value v between 0 and $q-1$, where q is a parameter of the Bloom filter extension called *quantization level*. The token is then considered to be associated with value v for storing and retrieving with the Bloom filter extension. When used at the end to calculate a message's spamminess, a token's probability value v is approximately mapped back to p based on the quantization mapping.

This work studies the effect of different quantization levels on Bloom filter's lookup performance. Two aspects of quantization effects need to be addressed. First, we would like to choose an optimal quantization level (q), which affects both the size (m) of the Bloom filter's bit-vector and the Bloom filter misclassification rate. The latter is discussed in the next subsection.

Second, for each given quantization level, we would like to pick the optimal mapping between the values to be quantized and the values after quantization for minimal errors. This work uses the Lloyd-Max algorithm to obtain the optimal quantizer for the token spamminess values for each given quantization level. The Lloyd-Max [52] algorithm is borrowed from previous studies of optimal quantization (such as those used in MPEG [53]) in the area of lossy encoding [54]. The Lloyd-Max algorithm is one of the popular algorithms that make a non-uniform optimal quantization that provides minimal "quality distortion" to videos.

Approximate Lookup

This extension allows the Bloom filter to support value retrieval queries at a cost of higher error rate compared to the original Bloom filter. Two types of misclassification could happen in this extended Bloom filter. This approximation happens in the query stage of the Bayesian filter in Figure 4.1.

First, similar to the original Bloom filter, the extended one could misclassify a non-member token as a member and mistakenly provide a value. The chance of such false positive misclassification increases because if any bit of the multi-bit output entry is set to one by hash conflicts, a false positive will occur. To derive the probability of this false positive, let us first only consider one bit of the query outcome. From the single dimension Bloom filter (Equation 4.7), we can derive that the possibility for a single bit being zero as

$$\mathbf{P_{m,n,h}(0)} = 1 - \mathbf{P_{m,n,h}(fpos)} = 1 - (1 - (1 - \frac{1}{\mathbf{m}})^{\mathbf{n*h}})^{\mathbf{h}} \tag{4.8}$$

With the final logical AND output having q bits, the possibility of false positive becomes

$$\mathbf{P_{m,n,h,q}(fpos)} = 1 - (\mathbf{P_{m,n,h}(0)})^{\mathbf{q}} \tag{4.9}$$

Second, a new type of error occurs when more than one bit of the final Bloom filter outcome are set to 1. The probability of a multi-bit marking is equivalent to one minus the probability of all bits being set to zero and the probability of only one bit getting 1.

$$\mathbf{P_{m,n,h,q}(multi)} = 1 - (\mathbf{P_{m,n,h}(0)})^{\mathbf{q}} - \mathbf{q} * (1 - \mathbf{P_{m,n,h}(0)})^{(\mathbf{q-1})} \tag{4.10}$$

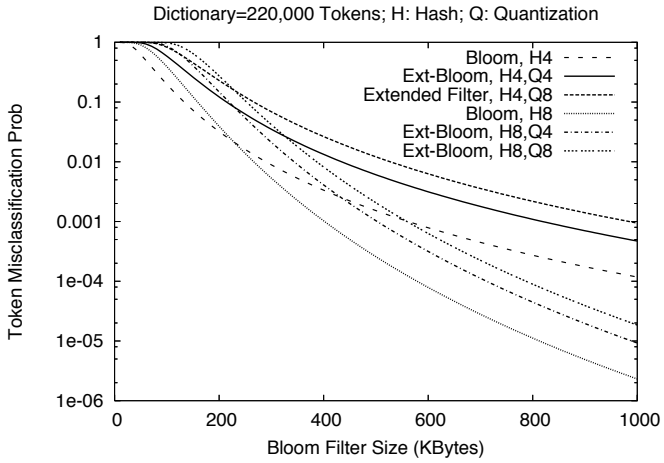


Figure 4.7: Lookup Error Rate vs. Bitmap Sizes

The probability rates for both types of errors depend on the number of tokens (n), the Bloom filter bit-vector size (m), the number of hash functions (h), and the quantization level (q). Figure 4.7 shows a theoretical error rate for a dictionary with 220,000 tokens versus various bit-vector sizes from 0 to 1 MB, 4 or 8 hash functions, and 4 or 8 bits quantization respectively. The dictionary size is selected based on the recommended token sizes by Bogofilter [7]. The results indicate that the selection of Bloom filter parameters (m,h,q) affects the misclassification rate significantly. For a small number of hash functions, the Bloom filter can reach less than a 0.1% token misclassification rate with less than 1MB memory under small quantization levels (4 or 8 bits).

4.2.5 Control the Lookup Accuracy

This section discusses how to reduce the total errors caused by the two approximations (quantization and approximate lookup) in order to limit their impact to the final message classification errors. We control the impact of these errors by choosing the appropriate Bloom filter size and quantization level that minimize the total lookup errors. In the case of multi-

bit marking, we control the query outcome in a way that is biased toward false negative classifications.

Selection of Bloom Filter Parameters

For a given dictionary size (n), to minimize the lookup errors and achieve high-speed lookups requires a careful selection of Bloom filter parameters: the size of the Bloom filter (m), the number of hash functions (h), and the quantization level (q). The parameter selection has to balance the error rate and the lookup speed. For example, large Bloom filter size is generally preferred for low misclassification rate, but Bloom filters with a size larger than the cache would degrade the query performance. The parameter selection also has to balance the approximation errors caused by quantization and hash-based lookups. For example, higher quantization levels (more bits used for quantizations) are preferred to store high precision values; but for a fixed Bloom filter size, higher quantization levels cause fewer rows in the Bloom filter and thus increase the misclassification rate (as indicated by Equation 4.9 and 4.10).

Previous Bloom filter applications [51, 26] have extensively studied the selection of Bloom filter size (m) and number of hash functions (h) involved in the tradeoff between size and error rate. The Bloom filter extension shares similar guidelines regarding the selection of these two parameters (m and h). This section focuses on the selection of quantization level (q) which is unique to this Bloom filter extension.

We define the problem of picking the appropriate quantization level as the following: *For a given Bloom filter size $m = r * q$, we would like to pick an appropriate quantization level q that minimizes the error between a token's lookup outcome value and the token's original statistical value.*

The expected error between a lookup outcome and its original value is a probability combination of the misclassification error (E_{lookup}) and the quantization error ($E_{quantiz}$). The

following equation represents the error as the sum of these errors:

$$\mathbf{E}_{overall} = \mathbf{P} * \mathbf{E}_{lookup} + (1 - \mathbf{P}) * \mathbf{E}_{quantiz} \tag{4.11}$$

in which P is the probability of token misclassifications.

To make a good choice of the Bloom filter parameters, we would need to know the distribution of the values to be quantized and stored in the Bloom filter. This is available for every training set, and its parameters may be determined experimentally. If the theoretical distribution of token statistics is known, the optimal parameter selection, in particular the appropriate quantization level for a given bit-vector size, can be done through a theoretical analysis.

For example, we assume the values to be stored follow a Gaussian distribution $G(\alpha = 0.5, \sigma)$, where $G(\alpha, \sigma)$ represents a Gaussian distribution with a mean of α and variance σ.

If no Bloom filter misclassification occurs, the value coming out from a Bloom filter lookup is assumed to be the same as the original value plus a quantization error. The distribution of this error, $E_{quantiz}$, follows a Gaussian distribution $G(\alpha = 0, \sigma/(2q))$, where q is the number of quantization levels.

If a Bloom filter misclassification occurs, e.g a token T that is not seen in the training set was mistakenly given a lookup outcome v, the lookup error is determined by v and the appropriate value for token T. We assume that the values of random tokens that are not in the training set should follow a Gaussian distribution $G(\alpha = 0.5, \sigma)$. This assumption reflects the idea that an unknown token should be considered to be neutral. We further assume that the classification outcome v is independent to the token when misclassifications occur, and thus v follows $G(\alpha = 0.5, \sigma/(2q))$. With these assumptions, the lookup error E_{lookup} follows a Gaussian distribution $G(\alpha = 0, \sigma + \sigma/(2q))$.

In addition, assuming the lookup misclassification occurs independently from the quantization errors, the linear combination of two Gaussian distributions is still a Gaussian distribution. The overall query outcome error thus has a mean α of 0, and variance is

$$(1 - \mathbf{P}_{m,n,h,q}) * (0.5/q) + \mathbf{P}_{m,n,h,q} * (\sigma + \sigma/(2 * q)) \tag{4.12}$$

in which, $P_{m,n,h,q}$ is the misclassification probability of a Bloom filter that has q levels of quantizations, h hash functions, total size m bits, and store values for n tokens. The best quantization level is q such that it minimizes the value of Equation 4.12.

Figure 4.8 shows the predicted error variances of the overall lookup error distribution assuming $\sigma = 1$. The result indicates that for a dictionary of 220,000 tokens, the quantization levels should be around 4 to achieve smaller error variances for small bit-vector size (e.g 256Kbytes). Quantization levels would have larger error on average with values smaller than 4 or larger than 6 would When larger bit-vector is used, quantization levels larger than 4 lead to smaller errors.

Figure 4.9 shows the predicted error variance for a Bloom filter with the same dictionary size, a fixed quantization level of 4, but with different hash functions. The result matches the intuition that, for a fixed quantization level, the selection of other Bloom filter parameters (h and m) agrees with early studies [51, 26]: for small bit-vector size, the number of hash functions needs to be small (around 6 for the 256Kbytes Bloom filter) in order to achieve a lower misclassification rate. As size of Bloom filter increases, more hash functions can be used to achieve a lower misclassification rate, but the increase in effectiveness is modest.

The above estimation is based on an unrealistic assumption of value distributions. In real messages, the occurrence of tokens is not independent and identically distributed (iid). This result is only for analytical purpose and is only shown as a guideline for selecting the Bloom filter parameters. The real error rate is determined by the specific distributions and messages. Furthermore, our overall concern is the final message classification performance (false positives, false negatives, and throughput). Therefore we used real messages to make a realistic study through experiments in order to evaluate the selection of the Bloom filter parameters. The results are presented in Section 4.3.

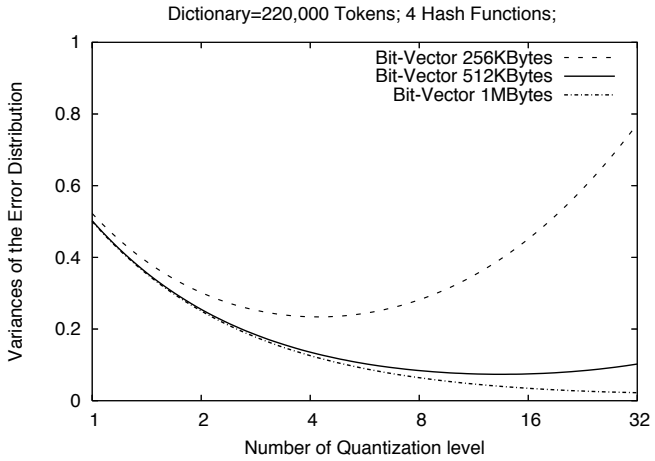


Figure 4.8: Error Distribution Variance vs. Quantization Levels

Policy for Multi-Bits Errors

The previous subsection discusses the selection of Bloom filter parameters to minimize the possibility of lookup errors. Although such cases rarely happen, errors could still occur. When a conflict caused by multiple bit-markings occurs, interpreting the outcome based on any bit could cause lookup error which later could potentially cause a message misclassification.

Although this can not be completely avoided, the impact of this error can be further minimized by making error biased toward a false negative classification rather than a false positive. When multiple possible values come out from one lookup query, we choose the smallest value as the Bloom filter outcome so that even if it is wrongly chosen, the error only makes the classification result less likely as spam. We evaluated the effectiveness of this policy and the result is presented in Section 4.3.6.

Selection of Hash Functions

Another fact that can affect Bloom filter lookup speed is the complexity of the hash functions. Popular hash functions, such as MD5 [55], have been designed for cryptographic purposes.

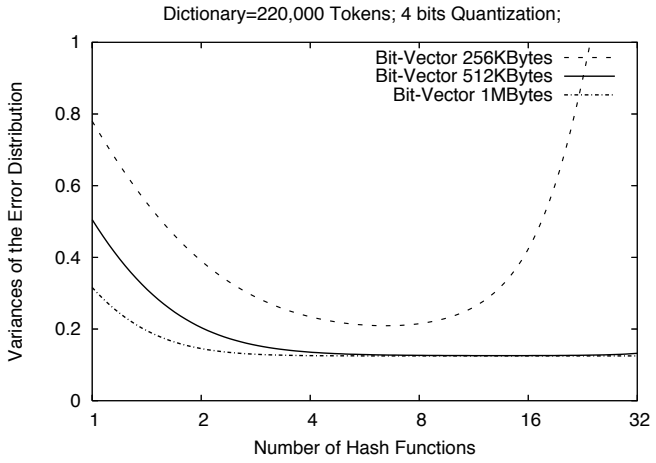


Figure 4.9: Error Distribution Variance vs. Number of Hash Functions

However, the effort to prevent information leaking is not the focus of hash-based lookup, whose main concern is the throughput. Therefore, simple but fast hash functions are preferred. This preference of choosing simple hash functions has also been used in [56, 58]. In this work, we adopt two strategies to design the hash functions. One strategy is to simplify the well-known MD5 hash function, we call it MD-. The core of MD5 is a combination of 4 "bitwise parallel" functions named F, G, H, and I by the specification [55]. MD- only uses the F function but with the same initial and end bit shuffles as in MD5. The other strategy we use is to build a fast hash function from scratch by mixing the token bytes with shift and xor bitwise operations. Details of the hash function selection are presented in the evaluation section.

4.3 Evaluation

4.3.1 Overall Performance

This section summarizes the overall HAI performance with well-selected Bloom filter parameters. The results presented in this section are based on a Bloom filter with a total size of

Table 4.1: Performance Comparison between Bogofilter, QSF and HAI Filters on a PC Server with AMD Athlon64 (m=512K, h=4, q=8, thresh=0.5, and CPU=1.8GHz)

	BogoFilter	HAI (bogo) filter	QSF	HAI (QSF) filter
Throughput (msg/sec)	418	2583	101	871
Dataset1 False Positive	0%	0%	0%	0%
Dataset1 False Negative	2.24%	9.36%	3.41%	9.21%
Dataset2 False Positive	0.20%	0.20%	0.23%	0.23%
Dataset2 False Negative	4.00%	4.80%	6.80%	9.83%

512 Kbytes, 4 hash functions, and an 8-bit quantization. We apply the filters on two test sets. Dataset1 is composed by 10,000 ham from mailinglists and 10,000 spam from SpamArchive [92]; dataset2 is composed by 6000 ham messages from SpamAssassin [10] and another 6000 spam from SpamArchive. The performance comparison results are in terms of both filter throughput (messages per second) and filter accuracy (both false positives and false negatives). The filter accuracy results presented in this subsection are based on a filter threshold of 0.5. Detailed studies for the selection of this threshold as well as other Bloom filter parameters are presented in the later part of the evaluation section.

Table 4.1 shows the overall performance of Bogofilter, QSF, and their HAI modifications. This result indicates that, for a Bloom filter size at 512KB, HAI filters can handle 2583 messages per second (i.e. 41Mbps for 2KByte size messages). HAI gets a 6 to 8 times throughput speedup compared to Bogofilter and QSF respectively, without introducing any additional false positives. The speedup comes with the penalty of higher false negative rates for HAI filters. Such penalty (e.g. about 7% of overall spam in dataset1) might look significant, but that still corresponds to more than 90% spam messages being blocked by the filter [2]. Whether this tradeoff is worthwhile or not completely depends on each particular site's needs. The goal of this work is not to advocate high throughput over accuracy but to provide a study

[2]We investigated the nature of those messages that causes additional false negatives to HAI. Most of them are non-English messages. It just happens to be the case that more of these spam are selected to Dataset1 than Dataset2.

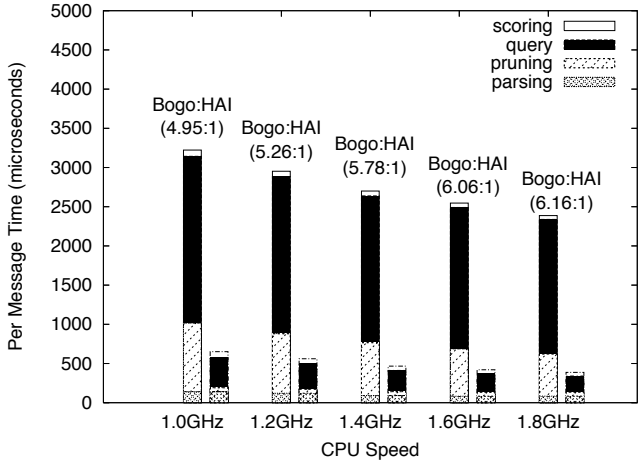


Figure 4.10: Filter Process Time Breakdown for Various CPU Speeds (HAI filter: 512KB bit-vector, h=4,q=8; Numbers inside the figure are speedup ratio)

of the trade-off between throughput increment and accuracy penalty. Furthermore, multiple levels of spam filtering can be used. For example, HAI filter quickly scanning all incoming messages, and later applying heavy filters only to "unsure" messages.

4.3.2 Detailed Breakdown for Throughput

This section presents an in-depth study on the source of this speedup by looking at the detailed behaviors of Bogofilter and HAI filter. We decompose both Bogofilter and HAI to four steps (*parsing*, *pruning*, *query* and *scoring*) and measure the time spent on each step per message. These 4 steps have been described in the overview of Bayesian filter in section 4.1.1. To reiterate, the *parsing* step divides the message to tokens based on common delimiters; the *query* step takes each token and uses it as a key for a database lookup to retrieve the corresponding token statistics; and the *scoring* step combines all the query outcomes and calculates an overall message spamminess value.

Bogofilter adds a *pruning* step between *parsing* and *query*. Performance-wise, this pruning step is used to reduce unnecessary lookups. It removes duplicated tokens so that each unique token only triggers one query. It also eliminates tokens that are believed to be useless for anti-spam. Such tokens include email message id, MIME labels etc. They are discarded during the training phase and thus are not in the token database (DB). The pruning step of our HAI filter only preserves the function of removing duplicated tokens to keep the same scoring method (each token counts only once). It uses a standard (not extended) Bloom filter to check token duplications. This Bloom filter works independently to the extended Bloom filter for the query step. The duplicate token checking initializes a small Bloom filter (2KB in our implementation) to all zeros for each incoming message, and queries and trains the filter at the same time. When a token arrives, it is first tested against the Bloom filter. If the membership testing returns true, the token is considered a duplicate and discarded. Otherwise, the token is put in the Bloom filter as a new member token and will be used in the query step. The detailed algorithm for this duplication checking can be found in our early work on fast packet classification [26].

We measured the processing time for each step by testing Bogofilter and HAI with Dataset1. The average message size for Dataset1 is around 2KByte, and each message contains about 180 unique tokens. Figure 4.10 shows the Bogofilter versus HAI processing time comparison per message. It is obvious that query and pruning are the two bottleneck steps of Bogofilter, and the speedup of HAI comes from these two steps. HAI gains speedup for the query step by reducing the number of memory accesses for each token lookup. An HAI filter's lookup requires a small amount of memory accesses that only depends on the number of hash functions and the size of Bloom filter, not the total number of stored tokens as in the database lookup case. In addition, by making the bloom filter size small, most or all of it can fit in cache for lower memory access latency. This effect of cache size is presented in the next subsection. Bogofilter's query step, on the other hand, operates on Berkeley DB, which is implemented by a Btree. The query requires multiple comparisons between the input token

and tokens in the DB and the number of comparisons on average is proportional to the logarithm of the total number of tokens in the DB. Although a faster indexing mechanism is possible, a DB lookup essentially has to have a few token comparisons, whereas these comparisons are completely avoided in HAI by the use of the extended Bloom filter.

The HAI also gains significant speedup in its pruning step. We did not include this as part of the general HAI solution for two reasons. First, not all the Bayesian filters avoid searching duplicated tokens as Bogofilter does. Second, there is no fundamental reason that a DB-based Bayesian filter can not replace its pruning step with the one used by HAI. Even if Bogofilter adopts the same pruning step as HAI, without changing to approximate query as HAI does, Bogofilter would still be multiple times slower than HAI to process a message.

In addition to the throughput gain against Bogofilter, HAI's throughput scales better as CPU speed increases. The AMD Athlon64 based desktop supports CPU scaling. We adjusted the speed from 1.0 GHz to 1.8GHz and measured the throughput of both filters. Figure 4.10 shows a comparison of Bogofilter and HAI for various CPU speeds. Although both filters take less time to process a message as the CPU speed increases, the speedup ratio for HAI versus Bogofilter increases as CPU speed becomes higher.

We also inspect the effect of message size on the speedup ratio. As messages get large, the speedup ratios are about the same but with a slight decrease. The parsing step increases strictly proportional to the message sizes, but the number of unique tokens does not increase in a strict linear fashion. The bottleneck starts to shift from query and pruning toward parsing. However, even with a jumbo message size, token query continues consuming a significant amount of processing time for Bogofilter, and approximate classifications would still improve Bogofilter's performance.

Table 4.2: AMD L2 Cache Performance Counters for the Query Step Per Message (CPU=1.8GHz, and for HAI filter: h=4, q=8)

Configuration	L2 Miss	L2 Hit	Query Time
HAI (128KB)	8	835	202μs
HAI (256KB)	30	1120	208μs
HAI (512KB)	51	1264	216μs
HAI (1MB)	387	1066	257μs
BogoFilter	2157	605	1716μs

4.3.3 Effect of Bloom Filter Size

In this subsection we study the effect of Bloom filter size on the filter accuracy and processing throughput. Both the throughput and accuracy measurements were obtained from experiments using the Dataset1.

For the effect of Bloom filter size on throughput, Figure 4.11 shows that per message processing times are about the same for the HAI filters with a Bloom filter size less than 512KB. After the Bloom filter becomes close to or exceeds L2 cache size (512KB), the processing time in general increases as the Bloom filter size increases towards 16 Megabytes. The AMD Athlon64 processor provides performance counters for specific processor events, such as data cache hits and misses, to support memory system performance monitoring. To confirm the cache size effect, we capture the L2 cache performance counters before and after each query step.

Table 4.2 contains the average L2 memory access and miss counters of the query step for HAI or Bogofilter to process a message. As shown by Table 2, smaller Bloom filters can be put in higher level CPU caches and have fewer cache misses compared to larger Bloom filters. HAI filters clearly have a higher L2 hit rate than Bogofilter. L2 cache behavior is not the single factor that determines the processing time; the total computation of a program, memory access pattern, as well as L1 cache behaviors all affect the query time,

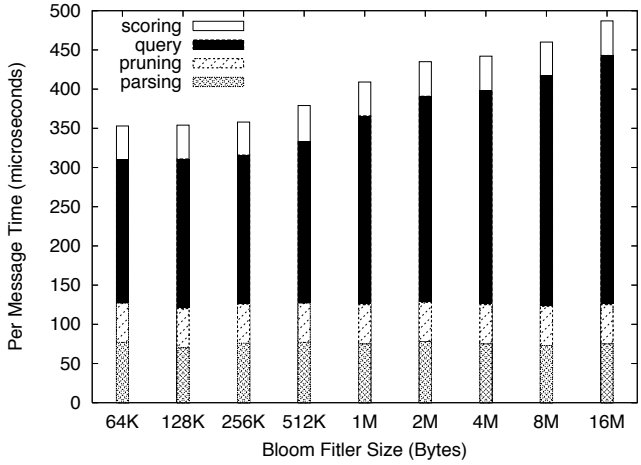


Figure 4.11: Processing Time VS. Bloom Sizes (AMD CPU=1.8GHz,h=4,q=8)

and a slight adjustment to the program (e.g different Bloom filter sizes) could change these memory accesses behaviors. This is also why the total L2 accesses (hits+misses) changes across different filter setups. Although we can not directly calculate the query time from L2 misses, it is clear that the cache access differences are a major source for the query time increment over filter size increment, which is shown in Figure 4.11.

The gain in throughput comes with a penalty on filter accuracy. To have an overall picture about how much change was brought to the filter accuracy by HAI, we present the false positive and false negative probabilities for the complete range of possible filtering thresholds from 0 to 1. Figure 4.12 shows a summary of the filter accuracy for HAI filters with various Bloom filter sizes ranging from 64KB to 16MB.

For the false positive result shown on the left half of Figure 4.12, HAI filters with smaller than 512KB show significant differences compared to those of Bogofilter. The smaller the Bloom filter size, the higher differences they can make.

However, for Bloom filter sizes larger or equal to 512KB, the email scores for ham messages are in fact very close to 0 and thus the false positive rates stay low. All filters have a

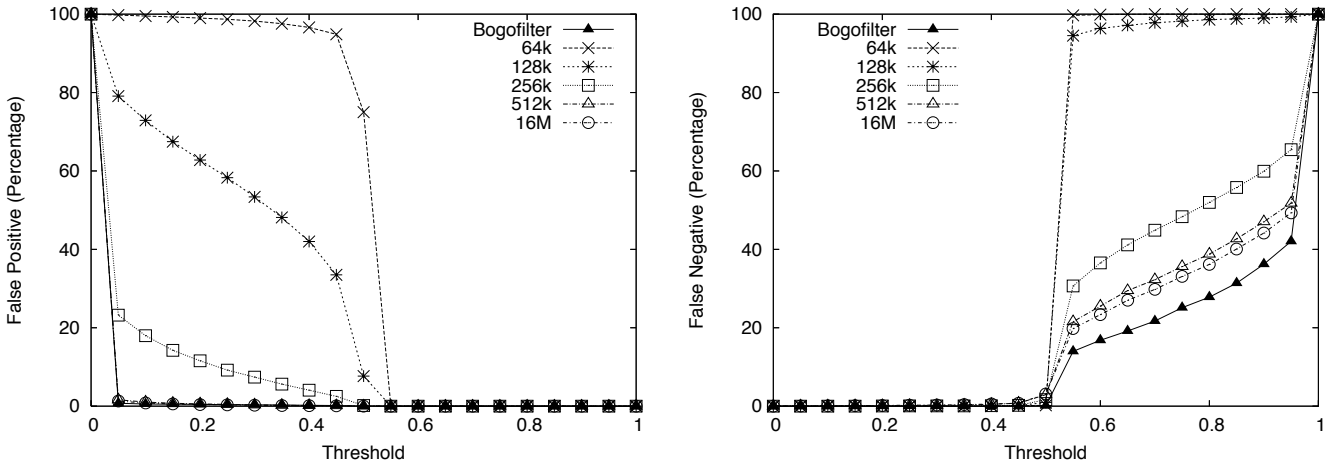


Figure 4.12: Filter Accuracy vs. Bloom Sizes (h=4, q=8) Results with 1MB to 8MB bit-vectors are all between the results of 512KB and 16MB and thus not shown in the Figure

close to zero false positive after the threshold gets larger than 0.5, even for those with a small Bloom filter size. This effect is due to the way that email score is calculated by Equation 4.4, which tends to produce a value around 0.5 to a randomly generated message. When token misclassification occurs, the lookup outcome is close to the outcome of a randomly generated message. Therefore, higher token misclassification rates tend to push an email (ham or spam) score toward 0.5, and both the false positives and false negatives results exhibit a significant change at threshold 0.5, no matter what Bloom filter sizes are used.

The false negative rate, which is measured over spam messages, is shown at the right half of Figure 4.12. The false negative result is similar to the false positive result in the sense that larger Bloom filter size gives closer results to the original Bogofilter, and filters smaller than 512KB differ from Bogofilter more significantly than those with a larger than 512KB bit-vector. Furthermore, compared to false positive, the results of false negatives show a relatively larger gap between the Bogofilter outcome and HAI filter, even with a large bit-vector at 16MB. We believe this is due to the quantization errors. A closer look at various quantization levels is presented later in this section. The accuracy results for HAI certainly

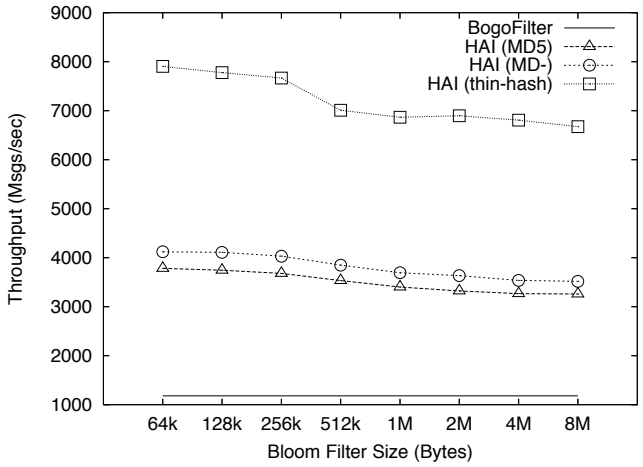


Figure 4.13: Filter Throughput vs. Bloom Filter Size (Intel P4 CPU=2.6GHz)

also depend on the test data set as well as the training data set. We discuss the effect of the training data set on the HAI filter accuracy in a later section.

4.3.4 Effect of Hash Functions

We consider two aspects of hash functions, the hash complexity and the number of hash functions, for the HAI filter performance. We compare three hash functions: MD5, MD- and the one we built from scratch. MD5 is picked to represent the well-known cryptographic hash functions that provide a well distributed hash output. MD- is our modification of MD5 to represent a simpler but faster hash. We build our own hash function by mixing the input token bytes with shift and xor bitwise operations. To generate a hash value, each byte of the token conducts 4 shift operations and is combined with previous bytes succesively by using xor operations and add operations. This hash function takes less than half the number of instructions as MD5. We call it "thin-hash". The throughput result is presented in Figure 4.13, which shows that MD- based HAI filter achieves about 10% higher throughput than the MD5 based filters. By using our own hash function, we achieve 95% throughput

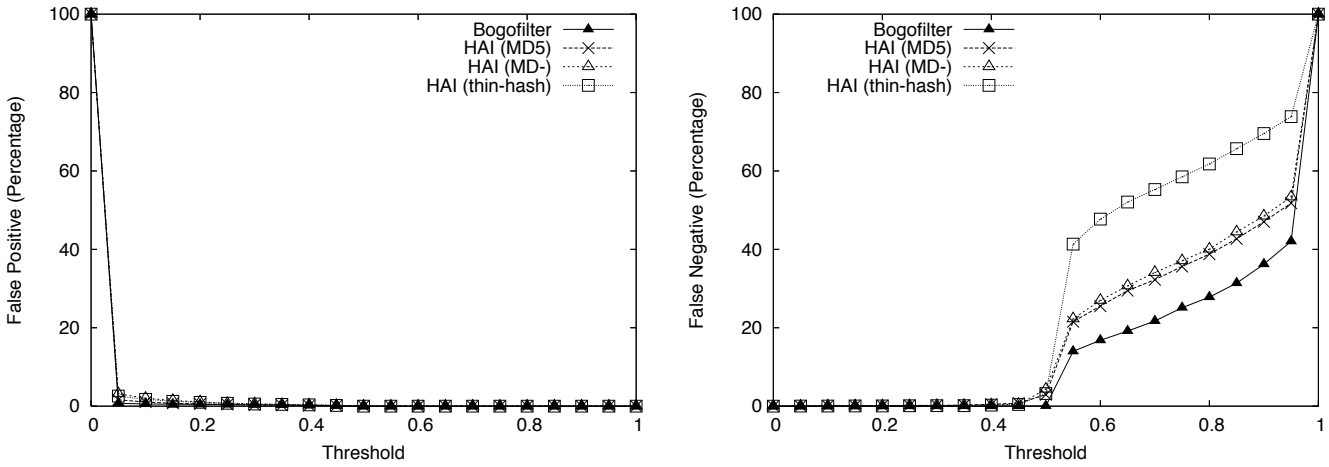


Figure 4.14: Filter Accuracy vs. Hash Algorithms (m=1MB, h=4, q=8)

improvement comparing to MD5 based filters. For accuracy measurement, the false positive rates are close to identical for all the three hash functions. The false negative results for MD5 and MD- are also very close to each other. Thin-hash function displays worse false negative results than the other two. We believe the reason that thin-hash and MD- have worse false negative rate than MD5 is due to the hash conflicts. MD5 has fewer hash conflicts than the other two. This result is demonstrated in Figure 4.14. The result also indicates the trade-off between throughput and accuracy. A much simpler hash function can reduce the cost of hash computation with sacrifices on the accuracy.

We also investigate the effect of using a different number of hash functions. Using a small number of hash functions reduces the number of marked bits, but has a higher probability of hash conflicts. Meanwhile using a large number of hash functions causes too many bits set in a Bloom filter with a limited bit-vector size and affects the accuracy. Figure 4.15 shows the filter accuracy results when using different numbers of hash functions. The Bogofilter false negative result is also shown in the figure as a reference. Only the false negative result is shown here because the false positive results are all zero. For the size of 512K byte Bloom

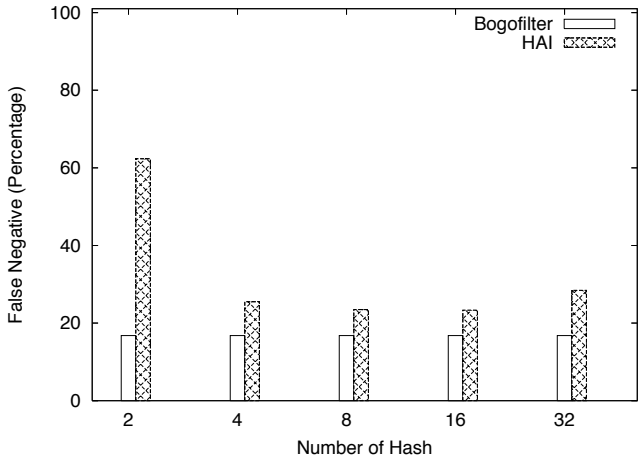


Figure 4.15: Filter Accuracy vs. Number of Hash Functions (m=512KB, q=8 bit, thresh=0.6)

filter, Figure 4.15 demonstrates that the best choice is to use 8 hashes, with both 4 and 16 hashes having very close results.

4.3.5 Effect of Quantization Levels

This section presents the experimental results on the effect of quantization level selection. The results were obtained in two steps. First, we isolated the effect of quantization from Bloom filter misclassifications. To study the quantization effect alone, we applied it directly to Bogofilter by quantizing all its statistical data in the database, and then measured its accuracy. Second, we did experiments with HAI filters with different quantization levels.

Figure 4.16 compares the filter accuracy among three cases: the original Bogofilter, Bogofilter with quantizations, and HAI filters, with the number of quantization bits set from 2 to 16. Quantization introduces errors to data representation, which in general reduces filter accuracy. The original Bogofilter's accuracy result is used as the best-case reference to compare the performance of filters with various quantization levels. As expected, for Bogofilters with quantization, 16 bit quantization performs best, but a quantization level around 4 to 8

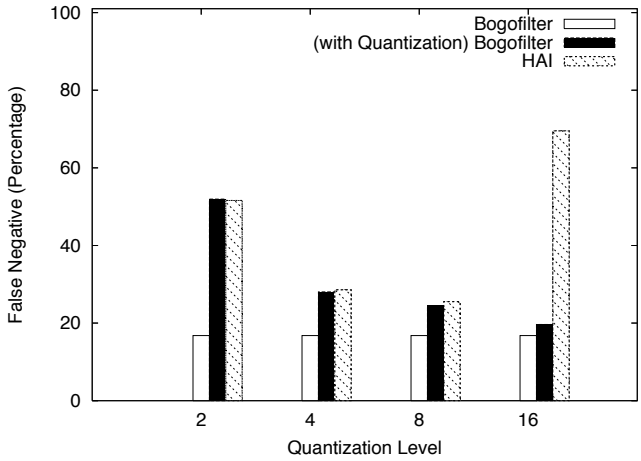


Figure 4.16: Filter Accuracy vs. Quantization Levels (m=512KB, h=4, thresh=0.6)

is also close to the original non-quantized case. However, for HAI filters, higher quantization levels no longer produce the closest accuracy results to Bogofilter. Instead, a quantization level at 8 shows the best accuracy results. By inspecting the token misclassification rate (by comparing each token outcome to Bogofilter outcome), we found that the token misclassification rate increases as more quantizing bits are used. The best quantization level has to be one that balances the quantization error and the misclassification rate. For the given experiment setup, the best quantization level is 8. Similar to the effect of Bloom filter size, the optimal selection depends on the number of tokens to be stored. Nevertheless, an important outcome from these experiments is that we have shown that a small quantization level can effectively produce a filter accuracy that is very close to the accuracy of the original Bogofilter.

4.3.6 Strategy for Handling Multi-bit Markings

In this section, we consider the strategies for handling the multi-bit marking error that is unique to the value retrieving extension of Bloom filter. When multi-bit marking occurs, the Bloom filter has to make a decision on the final lookup outcome. We consider three

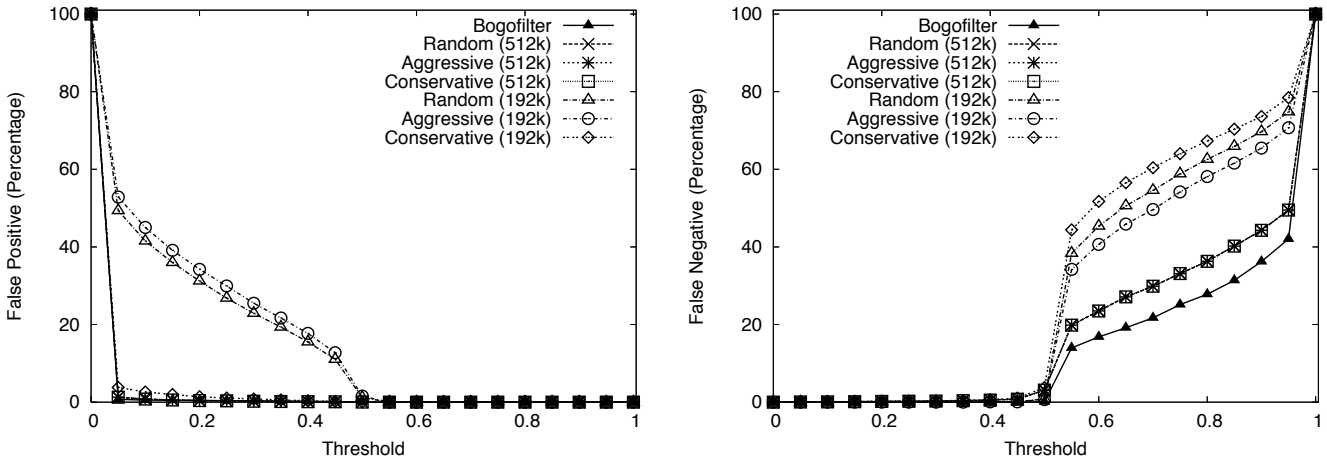


Figure 4.17: Filter Accuracy vs. Value Selection Strategies for Multi-bit Marking (m=512KB, h=4, q=8)

strategies for choosing the value. 1) *Aggressive Strategy*: every time multi-bit marking occurs, we always choose a value that indicates the highest spamminess; 2) *Randomly Selecting Strategy*: randomly pick one value; 3) *Conservative Strategy*: we always choose a value that indicates the lowest spamminess. Figure 4.17 shows the effect of different strategies on two Bloom filter sizes, 192KB and 512KB, respectively.

Figure 4.17 shows the effect of these three strategies for HAI under two different sizes: 192KB and 512KB. The two sizes are chosen to illustrate the impact of strategies under high and low multi-bit marking errors. Equation 4.10 indicates that the probability for multi-bit marking error is high when the Bloom filter size is small. When Bloom filter size is small (here 192KB) the aggressive strategy always has lower false negative rate than the other two. Meanwhile false positive should follow the reverse trend of the false negative. The aggressive strategy results show a very different false positive measurement compared to Bogofilter for any threshold less than 0.5. The conservative strategy gives the best result in the false positive measurement and has about the same false positive as Bogofilter. The random selection strategy is somewhere between but much closer to the aggressive approach.

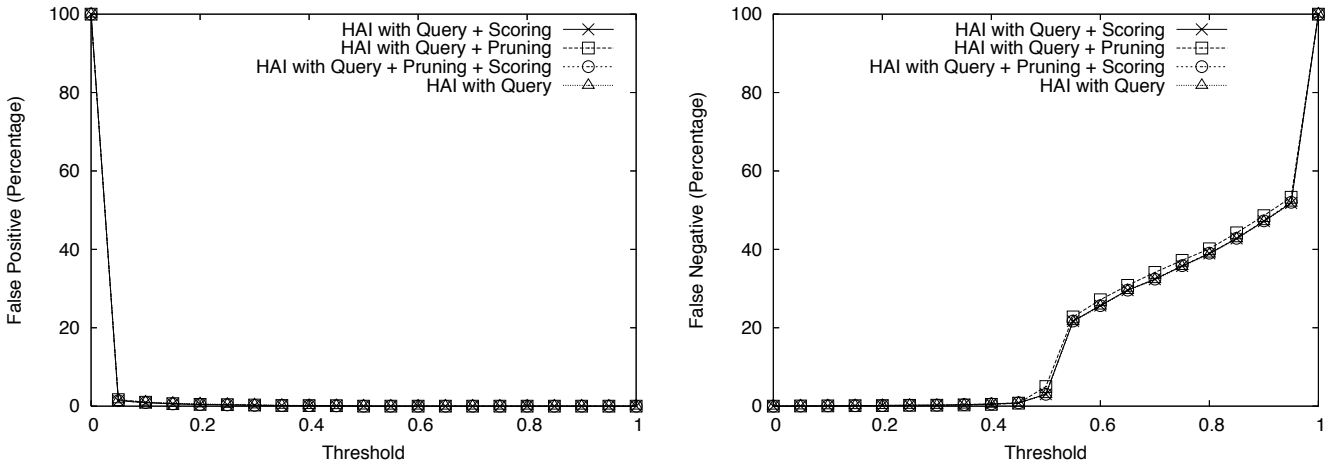


Figure 4.18: Filter Accuracy with Approximations in pruning and scoring stages(large bit-vector: 1MB, Quantization Level: 8, Hash Number: 4)

But for larger bit-vectors (here 512KB), the differences among strategies are small. This is because the overall multi-bit marking error is small. All three strategies lead to very low false positives. Users can retain the conservative strategy but still preserve a high filter accuracy. Overall this result indicates that the multi-bit handling strategies do not affect the filter accuracy in a significant way for large bit-vectors. However, to avoid false positives, we still recommend and use the third strategy in all other experiments.

4.3.7 Impacts of Approximations in the Pruning and the Scoring Stages

All previous HAI accuracy studies focus on the query stage only with fast pruning and fast scoring disabled. In practice, all three stages could introduce classification errors. In this section, we study the impacts of fast pruning and fast scoring on filter accuracy. The HAI filter accuracy with different setups is presented in Figure 4.18. The four setups are: 1) *HAI with Query*, where we setup the HAI with only approximate lookup enabled. 2) *HAI with Query + Pruning*, which is the HAI filter with approximate pruning and approximate lookup enabled. 3) *HAI with Query + Scoring*, which is the HAI filter with approximate

lookup and approximate scoring enabled. 4) *HAI with Query + Pruning + Scoring*, which is the HAI with all three approximations enabled. The result in Figure 4.18 shows that false positive rates remain identical. The false negative rates are very close and have no significant difference. This matches with our expectation that the main error introduced by the overall approximation comes from the approximate lookup.

4.4 Related Work

Message classification is a well-studied topic with applications in many domains. This section makes a brief description of the related work in two categories: classification techniques for anti-spam purpose, and fast classification techniques using Bloom filters.

4.4.1 Anti-SPAM Techniques

Anti-spam is a very active area of research, and various forms of filters, such as white-lists, black-lists [59, 60], and content-based lists [73] are widely used to defend against spam. White-list based filters only accept emails from known addresses. Black-list filters block emails from addresses known to send out spam. Content-based filters make estimations of spam likelihood based on the text of that email message and filter messages based on a pre-selected threshold. Most content-based filters use a Bayesian algorithm [73] to estimate message spamminess, and have been used in many spam filter products [9]. Recently, there have been several proposals about coordinated real-time spam blocking, such as the distributed checksum clearing house[74]. Most of these spam filters focus on improving the spam filtering accuracy. The work presented in this work differs from them by investigating the trade off between accuracy and throughput. We have shown that with a carefully chosen algorithm, Bayesian filters can gain throughput with only a small loss on false negative probabilities.

Many assumptions used by Bayesian filters to combine individual token probability for an overall score, such independent tokens, are not true for email messages, and more sophisticated classification techniques, such as k-nearest neighbors. In practice, naive Bayesian classifiers often perform well [8, 7, 10, 89], and the current state of spam filtering indicates that they work very well for email classifications. Nevertheless, the work presented in this work is to speedup the probability lookup stage for the probability calculation, and we expect the approach is applicable toward more sophisticated classification techniques.

4.4.2 Fast IP Processing by Bloom Filters

Hash-based compressed data structure has recently been applied in several network applications, including IP traceback [25], traffic measurements[57] and fast IP packet classifications [26]. For traffic measurement and traceback applications, Bloom filters are used to collect packet statistics and very often they are hardware-based Bloom filters [27]. The work presented in this work uses Bloom filters to improve software processing speed, and investigates the trade off between throughput and accuracy. Among all these previous Bloom filter applications, the closest related work is the high speed packet classifsication using Bloom filter, which first studied the tradeoff between accuracy and the processing speed [26]. This previous study uses Bloom filters for membership testing; the work in this work uses an extended Bloom filter that supports value retrieval. In addition, we considered an additional level of approximation by applying lossy encoding to data representations.

Chapter 5

Throttling Outgoing Spam for Webmail Services

Previous chapters discuss collaborative anti-spam solutions and a stand-alone Bayesian approach. These defenses to the spam attack are invoked after the message is completely arrived at the receiving mail server side. All these approaches consume internet bandwidth and receiving mail server storage space. This chapter explores a novel defense during the email conversation so that the email sender has to make efforts in terms of CPU consumption to deliver an email. However, the traditional SMTP protocol doesn't implement a cost mechanism, and the mail client and server have to be customized to incorporate the cost mechanism. Thus, deployment becomes a big issue. By studying the source of spam, sending spam from Email Service Providers (ESP) which are mostly in the form of webmail services(e.g. Hotmail, Yahoo, and 163.com) become very substantial. It is feasible to apply the cost mechanism and reduce outgoing spam from the ESP side.

The contributions of this work are: 1) we push the spam filter to the early stage of the email delivery time, and 2) we combine the spam filter with the computational cost approach and dynamically assign costs to the senders.

But naively pushing the spam filter earlier at the ESP side is not applicable, because the ESP would not like to take the responsibility of blocking the suspicious messages. In addition, lack of recipient's preference makes the sender more vulnerable to the false positives.

In the following sections I will discuss our selective cost-based approach for ESP in detail.

5.1 Adaptively Throttling Approach

This section presents our approach of adaptively reducing outgoing spam on the ESP side. We first review the current email relaying practice of ESPs, and then we explain how to build the cost mechanism into the ESP message relaying process. Finally we present our adaptive cost assigning system that selectively adds cost to users.

5.1.1 ESP Email Delivery Protocol

To our knowledge, the current practice is limited to the following protocols: (1)SMTP, (2)HTTP, and (3)HTTP with WebDAV. The latter two are more popular for web based ESP service (used by Hotmail, yahoo, mail.sina, etc) because they provide identifications to the ESP. When HTTP is used, messages are delivered to the server with the HTTP Post command. WebDAV is an extension of HTTP that is designed to enable multiple users to manage and modify the files in a remote system. With WebDAV enabled clients, users can view, open, edit, and save files directly into the filesystem of the website as if it were a local system. Since email data are still delivered through the HTTP Post command, we present the mail client with WebDAV in the same way as the client purely using HTTP.

5.1.2 Cost Mechanism

The goal of this work is to integrate the cost approaches into these systems and show that putting spam filters at an early stage of email delivery can help reduce the spam from the sender. With the cost mechanism, the server would be able to assign a computational task to the client with a controllable difficulty. The server would then verify the computational results before accepting the messages for forwarding. The cost mechanism has to be robust, tamper resistant, and efficient. Many existing studies [68, 69] have addressed these issues and designed algorithms for this task. We are not going to repeat this task. Instead, we focus on how to combine them with spam filters.

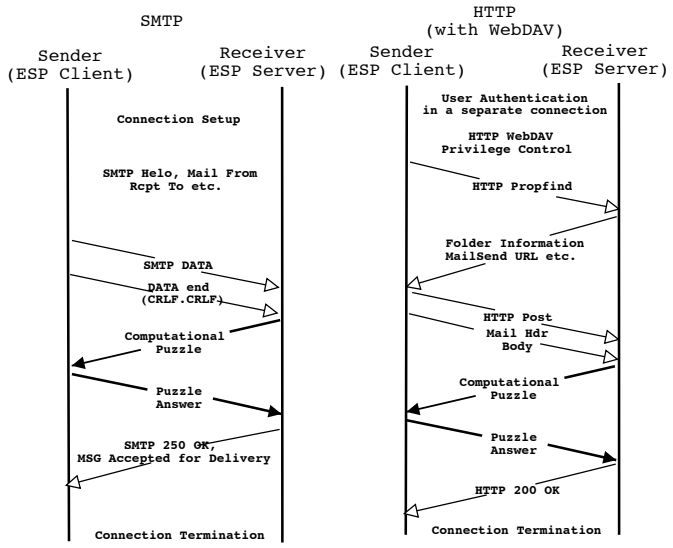


Figure 5.1: ESP Mail Protocols with Cost Mechanisms

We picked a simple computational puzzle algorithm for our system. In this algorithm, when a sender makes a connection and delivers a message to the ESP server, the server randomly generates a string for this connection, calculates and saves the MD5 hash output, and sends the hash output back to the sender (the ESP client). The email sender is asked to search for a string that has the same hash output and send back the string as the answer. The server controls the puzzle complexity by controlling the string length and the search space size.

Since we want to determine the computational complexity based on the message content, the server can only generate the computational puzzle after the message arrives and passes through our spam filters (to get a quality estimation).

Figure 5.1 illustrates where the computational mechanism is inserted into the original email delivery process for the two protocols, SMTP and HTTP, respectively. When the client uses SMTP to forward messages to the ESP server, the client's SMTP agent has to be modified. When SMTP is used, and ESP has no control over which SMTP client a

user adopts, then adding this mechanism would be considerably harder than the HTTP or HTTP with WebDAV cases. For the later case, the client side software is embedded in the web interface, which can be easily modified by the ESP server to add this cost mechanism, by using a client side script.

Notice that the ESP server has to enforce this mechanism in the sense that if a sender failed to supply the result of the computational puzzle associated with a message, the ESP would refuse to forward the message.

Because the ESP can also associate each email delivery attempt with a user account, the ESP can also apply more advanced cost control based on the account's overall behavior. For example, cost could be doubled if many email bounces happen, which is a good indication of sending unsolicited messages.

Penalties could also be used when a client refuses to send back answers but keeps making delivery attempts.

5.1.3 Selective Cost Assignment

With the knowledge of where in the delivery process we assign the cost, this subsection describes the algorithm of assigning puzzle difficulty.

We chose two guidelines for the difficulty assignment. First, we would like to assign no computational costs to every connection if the spam messages are very rare overall. Second, we would like to assign no or negligible computational cost to good email messages even when the overall spam volume is high.

To achieve this goal, we design a two level adaptation system in which an email connection's cost is assigned based on a product $C(m) = Q \times q(m)$, in which $C(m)$ is the cost level for a message m, and Q is the overall average message quality level measured over a recent history, and its value is between 0 (low spam ratio) and 1 (high spam ratio), and $q(m)$ is the quality measurement for this individual message with a value also ranging from 0 to 1.

Both the overall and the individual message quality measurements are made by a spam filter. Although spam filters can't judge the spamminess of a message with a 100 percent accuracy, the average score over many messages gives a good indication of the spam-and-non-spam ratio.

We choose a Bayesian based spam filter called QSF (quick spam filter) [89] for the message quality estimation at the ESP mail server side. QSF is a lightweight statistical spam filter written in C. In QSF, an overall score is calculated to find out whether the email should be considered spam or not. An evaluation of QSF by a third party shows its filtering precision is 99.1% with a 0.27% false negative and a 0.02% false positive rate.[71]

With this approach, there is still a large design space for choosing an adaptation algorithm. A few issues need to be addressed including over how long a period should the average quality measurements be made, how responsive should the system be towards message quality changes, and how much we adjust the cost each time we sense the quality changes.

We ended up choosing one proportional control algorithm, in which the cost level is assigned with the following equation:

- If $\overline{S} - S_m > 0, Q = P \times (\overline{S} - S_m)^i$
- If $\overline{S} - S_m <= 0, Q = 0$

Q is the cost we want to calculate, and $\overline{S}$ is the average email score over a recent period of time. We update $\overline{S}$ periodically, S_m is the mean score value of the good emails from the QSF training set. We calculate the distance between the $\overline{S}$ and S_m, and we raise this distance to the power of i, so that when the quality of emails are low, the score will be most likely high, and sender will get more punishment for those low quality emails. P is a multiplier used to map the value into the puzzle generator's input range.

Even with this algorithm, there are several challenges towards achieving this goal. We need to make the false positive impact as tiny as possible when the spam filter makes low quality estimation for a good mail. We also need to avoid high processing overhead, so that the ESP server can still support a large number of accounts.

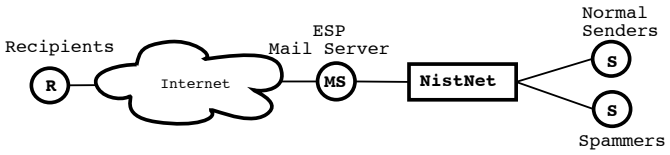


Figure 5.2: Emulation Topology

To address this limit, we set an upper bound to the cost level, so that even the maximum cost level would not cause a connection to delay more than 5 minutes. This number is estimated assuming the same level of computational power as our experiment machine at the ESP client side.

People have concerns that applying cost proportionally to the amount of messages might affect legitimate bulk email senders, such as Amazon and eBay. However, legitimate bulk email senders have motivations to identify themselves with the ESP so that they can be put on the white-list to avoid these computational costs. Furthermore, the cost is not only related to the message volume, but also the message quality. The aggregate cost for a large volume of good quality messages is still low.

5.2 Evaluation

This section presents an evaluation of our adaptive throttling system at the ESP. We first present the experiment methodology, including the experiment setup and the metrics we used to evaluate the system. Then we present the empirical results for both with and without the adaptive throttling system.

5.2.1 Evaluation Methodology

We evaluate the adaptive throttling approach through emulation. In the emulation, we set up a modified sendmail server as the ESP mail server, which accepts email from users through a webmail interface.

Figure 5.2 illustrates the topology used in our experiments. All the machines in the systems are 2.6GHz Dell PCs, running Linux 2.4.23, and they are connected through a 100Mbps switches.

The mail server is supposed to forward messages to the Internet. Since we only care about the quality of the outgoing email messages, we forward all the messages to /dev/null.

We use two machines to emulate normal senders and spammers. Both machines connect to the email server through a NIST Net router that emulates the network between the clients and the ESP mail server.

Because we are emulating many senders using a single machine, in our emulation, the cost of a computational puzzle given to the sender is reduced proportionally. A similar arrangement happens at the spammer side. The resource ratio is controlled with a parameter, and results of different ratios are presented in the next subsections. The cost mechanism is based on a MD-5 based client puzzle system described in the previous section.

We use real email messages (38591 non spam messages, and 18800 spam messages) obtained from the Internet for the experiments. The non-spam messages are obtained from several mailing-lists, including the well-known end-to-end [90] and perl monger [91]. The spam messages are obtained from the spamarchive [92] for the archived messages from Mar 19, 2004 to Sep 14, 2004. The distribution of the email scores used in our experiments is illustrated in Figure 5.3.

To play like a spammer, we mapped the mail queue to the ramdisk, and also turned off the mail log. To remove the overhead of the disk i/o, we also redirect unnecessary output to /dev/null.

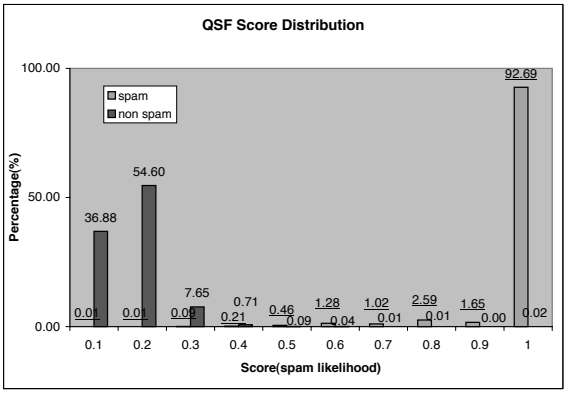


Figure 5.3: Email Score Distribution (The number with underline is the spam's percentage number.)

In our evaluation, we consider two possible spammer strategies. One is with a best effort approach, in which spammers keep sending as long as they have resources regardless of the cost the ESP assigned to them. The other strategy assumes the spammers adapt their behavior based on the cost and only try to send messages when the cost is low. If an attempt to send leads to a high cost, the spammer abandons it immediately without devoting any resources.

To evaluate the effectiveness of the throttling system, we chose to measure three metrics: the *spam ratio,* the *goodput* and the *normal email delay.*

The spam ratio is the percent of outgoing messages that are spam. We expect an ESP's outgoing messages have a low spam ratio, so that the ESP will not get onto black-lists or receive complaints. However, the spam ratio is not the only metric that concerns an ESP. The ESP should not drop all the messages. It has to keep a high throughput for legitimate messages. Therefore, we look at a second metric: the goodput. The goodput is the non-spam email throughput. We expect an effective control system can keep high goodput even with many spam attempts. We also measure the normal email delay in relaying the messages from the webmail interface to the mail server. Here the delay is measured by logging the

connection initiation time and the time when the server accepts the message for delivery. The delay is the time difference between the two, which includes the time spent in generating and solving the computational puzzle. We measure both average delay and the worst delay for non-spam messages. Ideally, an effective anti-spam system should introduce very low delay to non-spam messages.

5.2.2 Effectiveness

As a reference to measure the effectiveness of this outgoing spam control, we first measure the overall spam ratio, the goodput, and the delay behaviors of normal emails.

We run a server with both emulated normal users and spammers. We further assume that the server supports 100,000 users, and each user on average sends five emails a day. This number was obtained from a recent study on a British ESP [61]. Our own measurement over a nationwide ESP shows a similar rate. We control the normal email rate according to this average, and we emulate a spammer that sends emails in a *best-effort* way. In our measurements, we found the CPU is the bottleneck for email senders, rather than memory or bandwidth. With the best-effort strategy, the spammer automatically accepts whatever cost assigned from the ESP. We vary the spammer's CPU resources to show its impact on the spam ratio and the goodput and delay for normal messages. This result is presented in Figure 5.4. In this result, the spammer's CPU resources are represented by its ratio to the normal users' average computational power. Typically the ratio is around 1, meaning that the spammer uses a similar powerful machine as a normal user does. We consider the typical ratio range is between 0.1 and 10. The ratio is increased when spammer has a top-of-the-line system, or compromises a good number of zombie machines for sending emails. So we also consider some larger ratios to represent this scenario. The result indicates that when the spam volume increases proportionally with the spammer's resources (without requiring very powerful systems) the spam volume can bypass that of normal email messages (which is the goodput). The spam volume stops increasing once the email rate is high enough to hit

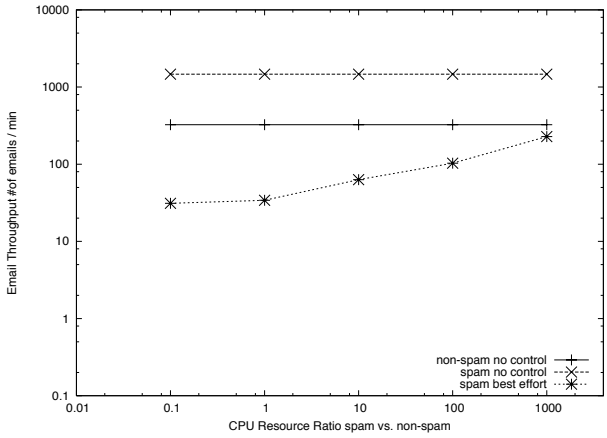


Figure 5.4: Throughput with best-effort spammers

Table 5.1: Delay for the normal messages with best-effort spammers (SD: Standard Deviation)

Resource Ratio	Average Delay(sec)	SD
0.1	0.56	1.46
1	0.85	3.51
10	2.43	6.66
100	6.9	10.63
1000	28.33	17.22

the server's maximum throughput. Under this situation, the majority of messages are spam. Arguably, the Internet email system is getting close to this situation, given reports that more than 50 percent of Internet messages are spam.

Now we look at email throughput as well as delay when the ESP uses our adaptive cost control algorithms. In this section, we assumed all the senders automatically accept the cost assigned from the ESP. We expected the spammers would suffer with the high cost assigned to them because most of their messages would have a high spam score. Certainly spammers

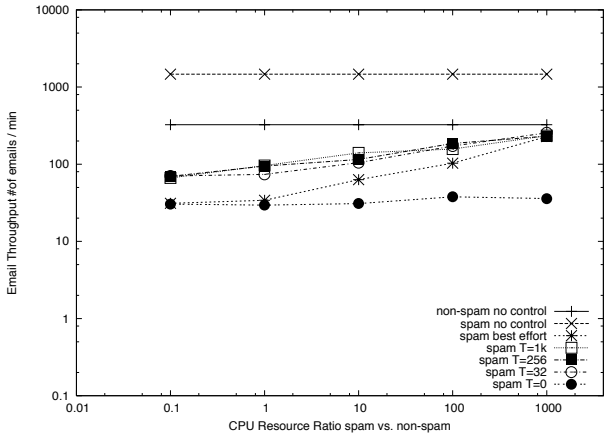


Figure 5.5: Throughput with "smart" spammers

could choose to abandon some connections to avoid high computational costs and optimize their throughput. We discuss this situation in the next subsection.

Figure 5.4 shows the resulting throughput for various spam resource amounts. Overall, the spam ratio is decreased after we apply our control system. For example, when the spammer has the same CPU resource as normal user, the spam ratio before control is 0.81, and it drops down to 0.09 after we use the control system. This benefit of a lower spam ratio and a higher goodput comes with a small impact to normal emails: per email delay could increase. To quantify this impact, we present the measured normal email delay result in Table 5.1, including both average delay and standard deviation of the average delay. The result is that most of the messages have very low delay when the spammer's resources are low or comparable to a normal user's resources. The delay increases when the spammer's resources get higher. However, the average delay is still within tens of seconds. The worst delay to legitimate emails is controlled below the maximum cost level which is 5 minutes. This time interval has been commonly used in the email delivery for timeout value, such as the SMTP commands.

Table 5.2: Delay for the normal emails with "smart" spammers (SD: Standard Deviation)

Resource Ratio	Average Delay(sec)	SD
0.1	2.97	6.01
1	4.35	8.05
10	9.15	12.89
100	13.69	14.03
1000	29.49	18.07

5.2.3 Smart Spammer

In this section, we consider a "smart" spammer, who selectively chooses to accept the ESP's cost assignment in order to send messages or chooses to abandon the messages when the cost is too high.

In this evaluation, we did not define the smartest strategy for spammers. Instead, we tried a range of spammer strategies, and studied the outcome of the spammer's throughput. A common aspect of these strategies was that the spammer chose to send only when the cost assigned from the ESP was below a threshold. In our evaluation, we tried static thresholds and also thresholds adjusted based on the sender side resources. The thresholds tried in our experiments include (1)$threshold = 0$, which means the spammer only sends message when the cost is zero. (2) $threshold = infinity$, which means the spammers always accepts and solves the computational puzzle regardless of the complexity level. Essentially it is the same as the best-effort. (3) $threshold = T$, the spammer only sends messages when the cost is below T, and abandons the connection for an assigned cost higher than T. We exponentially tried several T. As shown in Figure 5.5, the throughputs of different T didn't make much difference, but the per email average delay will be higher when the spammer chooses a higher throughput.

The result of the email throughput is presented in Figure 5.5, and the normal email delay whose T equals to *1k* is presented in Table 5.2. The resulting spammer throughputs for all the

thresholds we chose are all below the normal email throughput for various spammer resource setups. And to achieve a high spam throughput, no matter which threshold a spammer chooses, he inevitably has to increase his computation capacity. This will definitely limit the ability of the spammer to abuse the email system with limited computation resources. And in this measurement, the spam ratio also decreased overall after we apply our control system. For example, when the spammer has the same CPU resource as a normal user, the spam ratio decreases from 0.81 to below 0.22 under different thresholds we attempt.

In regard to the delay result, with smart spammer strategies the delay result for normal emails is only a few seconds longer than the case with best-effort spammers. Although this result does not necessarily show the highest possible throughput for a spammer, it does show that the spammer cannot achieve a very high throughput using only a simple strategy.

The above results only cover a special group of spammer's strategies. Another interesting strategy for spammers is to send non-spam along with spam messages. With the hope of bringing up the overall quality and keeping our control system assigning low computational cost, the spammer might get a good amount of spams out. One way to address this problem is to keep an absolute counter for emails that have a filter score high enough, and take this into account when calculating the overall quality of emails, rather than only using the average filter score. Certainly, investigating the best available strategy for spammers deserves more study, and we plan to pursue this in our future work.

5.2.4 Overhead

Although the system for outgoing spam control has been shown to be effective, it comes with some system overhead to the ESP servers and to the ESP users. The overhead comes from two components: the cost mechanism overhead, and the complexity estimation overhead.

When a computational puzzle is used, the cost mechanism overhead includes the server CPU overhead of (a) puzzle generation, (b) puzzle verification, and the client CPU overhead of (c) solving the puzzle at the legitimate client.

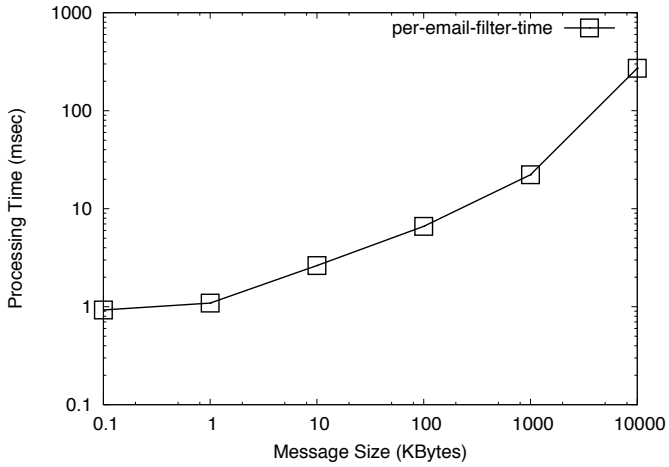


Figure 5.6: QSF Spam Filter Overhead

The computational puzzle overhead for the server is mostly 2 MD5 computations, which is negligible for modern systems. (The PC we used in our evaluations can compute more than 500,000 MD5 computations in one second.) The overhead for the sender is not negligible, but is considerably small. In fact the average cost for a normal sender can be estimated with the average normal email score. With our algorithms, a normal sender only needs to spend 1 second in average CPU computational time. Considering the average number of emails that a normal sender sends per day, this amount of time will not be a concern to the sender.

The complexity estimation is derived by a spam filter, which is not used for outgoing messages in current practice. This introduces the complexity estimation overhead. Since this overhead comes with the spam filter that scans the whole message, the overhead is associated with the message size. We measure the time spent on the spam filter for different message sizes. The result is presented in Figure 5.6. The result shows that the delay introduced by the filter clearly increases as the message size increases, and when the message is 10 MBytes, the delay is up to half a second. This delay is negligible because it is considerably small compared to the average normal email delay in Table 5.1 and Table 5.2.

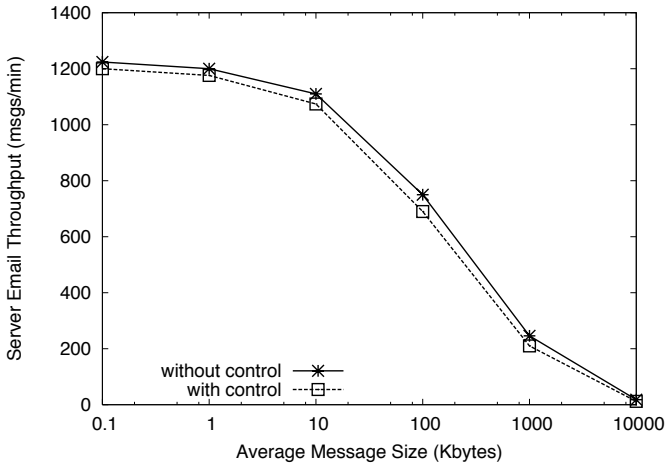


Figure 5.7: ESP Mail Server Overhead

To show the overall overhead, we measured the maximum throughput that an ESP server can achieve with and without the cost control for different sizes of messages. The result is presented in Figure 5.7. As we can see, the server throughput with control is lower than the one without it. However, the difference is not large. This is because the email servers are not CPU intensive, and most of the overhead introduced by the control system is on the CPU. We conclude that the overhead brought by applying spam filters to each message is acceptable and does not significantly degrade server performance. Hence, the system can be applied to ESP mail servers.

5.3 Related Work

Anti-spam is a very active area of research, and recently many anti-spam techniques have been produced. We classify them into two categories: spam filters and cost based approaches. This section first reviews these two categories and then discusses some existing practices for controlling outbound spam at the ESP side.

5.3.1 Spam Filters

Most of the current anti-spam research focuses on spam filters. Various forms of filters, such as white-lists, black-lists [59, 60], and content-based filters [73] are widely used to defend against spam. White-list based filters only accept emails from known addresses. Black-list filters block emails from addresses known to send out spam. Content-based filters make estimations of spam likelihood based on the text of that email message and filter messages based on a pre-selected likelihood threshold. For example, the famous filter from Paul Graham [73] assigns a likelihood value to each word or phrase based on its history of use in spam and takes the average as the overall spam-likelihood for the message.

Unfortunately all types of spam filters have false positives, with which legitimate messages are misclassified and get lost. Another problem with spam filters is that it can only filter a message after it has already been delivered and stored in the receiver's mail server.

The approach presented in this work also uses spam filters, but for a different purpose – not filtering messages, but estimating their "quality". The quality of the information is then used for selectively delaying messages. Thus a misclassification would only cause a small delay to a message, and the impact of a false positive would be much less severe than the method of dropping messages. This approach is applied at the sender side to reduce outgoing spam, thus it can be used as a complementary technique for the current filtering methods at the recipient side.

5.3.2 Cost-based Approach

A cost-based approach is the most promising general solution for resisting network abuse, such as spam [68, 69] and network DoS attacks [75, 76]. Cost takes many forms, such as monetary payments [77], "hashcash" [78], and computational puzzles [79]. By requiring the remote peer to consume some computational resources before granting the service, the protected side can reduce the risk of network abuse.

The most famous adoption of the cost approach is probably the challenge-response anti-spam scheme [80]. It has been used by Earthlink and a few other ESPs to filter incoming messages. With this scheme, a mail server automatically returns a challenge message which requires the client to perform a task, such as reading a picture, before it will deliver the message to the final recipient.

Dwork and Naor [79] proposed a general mechanism that requires a sender to compute a moderately hard pricing function or cryptographic puzzle for each message; the cost to compute the pricing function is negligible for normal users, but high for mass mailers. Recently, the use of cost-based approaches [68, 69, 81, 82] mostly address server resource exhaustion.

5.3.3 Previous Work on Outgoing Spam

The reverse Turing test is one well-known cost approach that has been widely adopted by many ESPs to reduce spam. In this approach, users are required to pass a simple test (e.g. reading text strings from a picture) before getting an account. Some ESPs (e.g mail.sina.com) even move a step further and require a reverse Turing test before sending any email messages.

A recent work by Goodman et. al. [67] shows that the sign up cost of the reverse Turing test is not large enough to deter spammers, and they propose an alternative that periodically imposes costs on senders only at the early signing up stage. Goodman et. al. shows that these costs at the initial stage is enough to deter spammers, thus reducing outbound spam messages for the ESP. Also Clayton[83] proposes to find spam senders by inspecting ISP logs. We can combine this log information with the throttling control proposed in this work by providing additional input information. Besides, rate limiting [84] is not a new idea, and it has been applied to emails. Our approach differs from previous works in that we provide an automatic control that drives the throttling effect rather than choosing a fixed throttling level.

Our approach in this work requires no human interaction for either tracking the senders or assigning and solving reverse turing tests. Instead, it automatically assigns computational

costs based on two factors, the individual email quality and the overall outgoing messages quality.

5.3.4 Dynamic Cost Control

While the possibility of adding delay and cost to abusers has been considered previously in works such as teergrubing [85] and tarpit [86], the works are limited to the recipient side. In our previous study [88], we studied the effects of introducing cost at the network transport layer on the email recipient side. The work presented in this chapter is similar to our previous work in the sense of selectively applying cost. The difference is that our previous work is purely recipient centered and requires a considerably large deployment to be effective. The work in this chapter is completely on the sender side, and because all the users have to be authenticated by ESPs and therefore use their proprietary process to forward messages, it is easier to deploy.

Chapter 6

Conclusion and Future Work

Spam is a common problem hard to eliminate in the world. To be a successful anti-spam solution, many criteria need to be considered. In this dissertation, we discuss security, high performance, and counterattack issues in designing the anti-spam systems.

This dissertation presents a prototype of collaborative spam filtering with considerations on both email privacy and classification accuracy. A trade-off between concealing email content and exposing enough information for spam filtering is studied. Our initial experiments show that ALPACAS system is very effective in filtering spam, has high resilience towards various attacks, and it provides strong privacy protection to the participating entities.

Anti-spam performance is another important issue concerned by researchers. Most of the existing anti-spam approaches introduce overheads which limit their scalability to the high speed Internet. This dissertation studies traditional Bayesian filters and proposes approximation techniques to speed up three different stages of Bayesian filters. We provide a mechanism to balance the two contradicting aspects which are accuracy and speed, and demonstrate a design space to approach both limits with a set of experiment results.

Lastly, the cost-based approach we have described show that the actual cost can be determined dynamically by estimating the email quality and spam behavior over a period of time. The system slows down spammers but assigns zero (or little cost) to the normal messages because they tend to have high quality. The same mechanism can be applied to different forms of cost. And a fine-grained cost estimation research can be further conducted.

6.1 Future Work

In addition to the privacy-preserving collaborative spam filtering work for proof of concept, I plan to adopt a sophisticated and robust P2P structure to address the challenges of dynamic entries and exits of participants. I also plan to emulate the collaboration on the real large scale test bed, such as PlanetLab. To study the degree of privacy protection, I plan to further design the threat model for term-level privacy breaches.

With the feature-preserving message transformation technique, a weapon introduced in chapter 3, I am planning to provide a technique to recycle the spam box, a file in which spam messges are stored. The purpose to recycle the spam box is to winnow the misclassified ham messages from the spam box. It is possible that a ham message is misclassified because of an aggressive threshold picked by a spam filter. And these misclassified messages can affect the accuracy of the successive incoming messages. I am planning to design a tool to compare the similarity between the emails in the ham box and emails in the spam box. We can obtain pairs of messages similar to each other but appear in different boxes. This implies that one of the message is misclassified. In addition to the similarity in the email body, the information of the email sender, the path that the email has gone through and the subject content of an email indicate the similarity. These information can be retrieved from email headers such as 'From:', 'Received:', 'Subject:', etc. Thus, I plan to investigate the similarity information in the email headers. By combining the similarity information both in headers and body, we could possibly fix the misclassification errors.

Performance is always a big issue for the content-based spam filter. We provide an approximation technique for Bayesian-based statistical filter in chapter 4. I would like to pursue the solutions for other anti-spam approaches such as rule-based, machine learning based approaches in our future work.

As zombies and botnets become the major threat to the Internet, as many as 100,000 computers can be compromised and distribute spam emails. To efficiently detect such attack from zombies and botnets, I plan to work on a reputation system to evaluate each sending

IP address. And based on the reputation, several approaches can be effectively adopted to prevent spam. For example, the SMTP connections can be slowed down if the reputation of the sender is bad. Or the connection can be dropped if the sender is a known spammer.

Bibliography

[1] Phishing, available at http://en.wikipedia.org/wiki/Phishing/

[2] Sober worm, available at http://en.wikipedia.org/wiki/Sober_worm/

[3] Jonsson, J. and B. Kaliski, Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1, RFC 3447, February 2003

[4] A. Broder, Some Applications of Rabins Fingerprinting Method, *Sequences II: Methods in Communications, Security, and Computer Science, Springer-Verlag*, pages:143–152, 1993

[5] Lakshmish Ramaswamy, Arun Iyenagar, Ling Liu and Fred Douglis, Automatic Detection of Fragments in Dynamically Generated Web Pages, In *Proceedings of 13th International Conference on the World Wide Web (WWW-2004)*, pages:443–454, May 2004

[6] Joshua Goodman, Tutorial on Junk E-mail Filtering, available at http://www.research.microsoft.com/ joshuago/tutorialOnJunkMailFilteringjune4.ppt

[7] Eric S. Raymond, Bogofilter: A Fast Open Source Bayesian Spam Filters, http://bogofilter.sourceforge.net/

[8] T.A Meyer and B. Whateley, SpamBayes: Effective open-source, Bayesian based, email classifications, In *Proceedings of the First Email and SPAM conference*, July, 2004

[9] Sam Holden, Spam Filter Evaluations, http://sam.holden.id.au/writings/spam2/

[10] Matt Sergeant, Internet Level Spam Detection and SpamAssassin, In *Proceedings of the 2003 Spam Conference*, Cambridge, MA, 2003

[11] Cole, William K. Blacklists, Blocklists, DNSBL's and survival. http://www.scconsult.com/bill/dnsblhelp.html

[12] John R. Levine, Experiences with Greylisting, In the Proceedings of Second Conference on Email and Anti-Spam, CEAS 2005.

[13] David Lary, University of Georgia Mail Server Statistics

[14] Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1, RFC 4408

[15] DomainKeys Identified Mail (DKIM) Signatures draft-ietf-dkim-base-10, Internet Draft.

[16] RFC 3174 - US Secure Hash Algorithm 1 (SHA1), http://www.faqs.org/rfcs/rfc3174.html

[17] Matthew Prince, Benjamin Dahl, Lee Holloway, Arthur Keller, Eric Langheinrich, Understanding How Spammers Steal Your E-Mail Address: An Analysis of the First Six Months of Data from Project Honey Pot, In *Proceedings of Second Conference on Email and Anti-Spam CEAS 2005*, CA, 2005

[18] Cloudmark Corp., Spamnet Anti-spam System, available at http://www.cloudmark.com/desktop

[19] Cyphertrust Corp., Iron Mail System, available at http://www.cyphertrust.com

[20] Alan Gray and Mads Haahr, Personalised, Collaborative Spam Filtering, In *Proceedings of the Second Email and SPAM conference (CEAS)*, July, 2005

[21] Ernesto Damiani and Sabrina De Capitani di Vimercati and Stefano Paraboschi and Pierangela Samarati, P2P-Based Collaborative Spam Detection and Filtering, In *Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, August, 2004

[22] Feng Zhou and Li Zhuang and Ben Y. Zhao and Ling Huang and Anthony D. Joseph and John Kubiatowicz, Approximate Object Location and Spam Filtering on Peer-to-peer Systems, In *Proceedings of ACM/IFIP/USENIX Middleware Conference 2003*

[23] B. Bloom, Space/time Trade-offs in Hash Coding with Allowable Errors, *CACM*, vol.13, no.7, Jul 1970

[24] L. Fan and P. Cao and J. Almeida and A.Z.Broder, Summary cache: a scalable wide-area web cache sharing protocol, In *Journal of IEEE and ACM Transaction on Networking*, vol.8, no.3, pages:281–293, 2000

[25] A. Snoeren and C. Partridge and L. Sanchez and C. Jones and F. Tchakountio and S. Kent and W. Strayer, Hash-based IP Traceback, In *Proceedings of ACM SIGCOMM'01*, AUG, 2001

[26] Francis Chang and Kang Li and Wuchang Feng, Approximate Packet Classification Caching, In *Proceedings of IEEE Infocom 2004*, Mar, 2004

[27] Sarang Dharmapurikar and Praveen Krishnamurthy and David E. Taylor, Longest Prefix Matching Using Bloom Filters, In *Proceedings of ACM SIGCOMM 2003*, page:201-212, August 2003

[28] Don Evett, Spam Statistics 2006, available at http://spam-filter-review.toptenreviews.com/spam-statistics.html

[29] Vipul Ved Prakash, Vipul's Razor Anti Spam System, available at http://razor.sourceforge.net

[30] Feng Zhou, Li Zhuang, SpamWatch A Peer-to-peer Spam Filtering System, available at http://www.cs.berkeley.edu/ zf/spamwatch

[31] A. Broder and S. C. Glassman and M. S. Manasse and G. Zweig, Syntactic Clustering of the Web, In *Proceedings of the* 6^{th} *International World Wide Web Conference*, April, 1997

[32] A. Broder, On Resemblance and Containment of Documents, In *Proceedings of SEQUENCES-97*, 1997

[33] Z. Bar-Yossef and S Rajagopalan, Template Detection via Data Mining and its Applications, In *Proceedings of the* 11^{th} *International World Wide Web Conference*, May, 2002

[34] P. Kulkarni and F. Douglis and J. LaVoie and J. Tracey, Redundancy Elimination Within Large Collections of Files, In *Proceedings of the USENIX Annual Technical Conference*, June, 2004

[35] Lakshmish Ramaswamy and Arun Iyengar and Ling Liu and Fred Douglis, Automatic Detection of Fragments in Dynamic Web Pages and its Impact on Caching, In Journal of *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 17, no. 6, June, 2005

[36] Micheal. O. Rabin, Fingerprinting by Random Polynomials, Center for Research in Computing Technology, Harvard University, 1981

[37] R. Tewari and M. Dahlin and H. Vin and J. Kay, Beyond Hierarchies: Design Considerations for Distributed Caching on the Internet, In *Proceedings of International Conference on Distributed Computing Systems*, May, 1999

[38] Ion Stoica and Robert Morris and David Karger and M. Frans Kaashoek and Hari Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, In *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, August, 2001

[39] Sylvia Ratnasamy and Paul Francis and Mark Handley and Richard M. Karp and Scott Shenker, A Scalable Content-Addressable Network, In *Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, August, 2001

[40] Lakshmish Ramaswamy and Ling Liu and Arun Iyengar, Cache Clouds: Cooperative Caching of Dynamic Documents in Edge Networks, In *Proceedings of the* 25^{th} *International Conference on Distributed Computing Systems(ICDCS-2005)*, June, 2005

[41] Gordon V. Cormark and Thomas Lynam, Spam Corpus Creation for TREC, In *Proceedings of the Second Email and SPAM conference (CEAS)*, July, 2005

[42] National Institute of Standards and Technology, The Spam Track in Text Retrieval Conference, available at http://trec.nist.gov/, 2006

[43] Ashwin Machanavajjhala and Johannes Gehrke and Daniel Kifer and Muthuramakrishnan Venkitasubramaniam, l-Diversity: Privacy Beyond k -Anonymity, In *Proceedings of the* 22^{nd} *IEEE International Conference on Data Engineering (ICDE 2006)*, April, 2006

[44] Roberto J. Bayardo and Rakesh Agrawal, Data Privacy Through Optimal K-Anonymization, In *Proceedings of the* 21^{st} *IEEE International Conference on Data Engineering (ICDE 2005)*, April, 2005

[45] Gregory L. Wittel and S. Felix Wu, On Attacking Statistical Spam Filters, In *Proceedings of the First Email and SPAM conference*, July, 2004

[46] Daniel Lowd and Christopher Meek, Good Word Attacks on Statistical Spam Filters, In *Proceedings of the Second Email and SPAM conference*, July, 2005

[47] Steve Webb and S. Chitti and C. Pu, An Experimental Evaluation of Spam Filter Performance and Robustness Against Attack, In *Proceedings of the 1st International*

Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2005), December 2005, San Jose, CA

[48] D. Heckerman and M. P. Wellman, Bayesian Networks, In *Journal of Communications of the ACM*, vol.38, no.3, pages:27–30, March, 1995

[49] Gary Robinson, A Statistical Approach to the Spam Problem, In *Linux Journal 107, March 2003*

[50] I. Androutsopoulos and J. Koutsias and K. Chandrinos and G. Paliouras and C. Spyropoulos, An Evaluation of Naive Bayesian Anti-Spam Filtering, In *Proceedings of the workshop on Machine Learning in the New Information Age, 2000*, Barcelona, Spain, May, 2000

[51] Andrei Broder and Michael Mitzenmacher, A Survey of Network Applications of Bloom Filters, *Internet Mathmatics*, vol.1, no.4, pages:485–509, 2004

[52] Quantizing for minimum distortion, In *IEEE Transactions on Information Theory*, vol.28, no.2, pages:7–12, 1960

[53] MPEG, available at http://www.mpegif.org

[54] T. Berger and J. D. Gibson, Lossy Source Coding, In *Journal of IEEE Transactions on Information Theory*, vol.44, no.6, pages:2693—2723, 1998

[55] R. Rivest, The MD5 Message-Digest Algorithm, *RFC 1321*, April, 1992

[56] Ozgun Erdogan and Pei Cao, Hash-AV: Fast Virus Signature Scanning by Cache-Resident Filters, In *Proceedings of Globecom'05*, November, 2005

[57] C. Estan and G. Varghese, New Directions in Traffic Measurement and Accounting, In Proceedings of Internet Measurement Workshop, page: 75-80, Nov 2001

[58] Kang Li and Francis Chang and Wu-chang Feng and Damien Burger, Architecture for Packet Classification Caching, In *Proceedings of IEEE ICON'2003*

[59] Philip Jacob, The Sapm Problem: Moving Beyond RBSs. *http://theory.whirlycott.com/rbl*, 2003.

[60] Michelle Delio, Not All Asian E-Mail is Sapm. *Wired News*, Feb 19, 2002

[61] Ben Laurie and Richard Clayton, "Proof-of-Work" Proves Not to Work, In *Proceedings of the Third Annual Workshop on Economics and Information Security(WEIS04).* Minneapolis, USA. May, 2004

[62] Kang Li, Calton Pu, and Mustaque Ahamad, Resisting SPAM Delivery by TCP Damping, In *Proceedings of the first Conference on Email and Anti-Spam (CEAS 2004)*, Mountain View, CA.

[63] Before Spam Brings the Web to Its Knees. Available at http://www.businessweek.com/technology/content/jun2003/tc20030610_1670_tc104.htm, June 10, 2003

[64] Sharon Gaudin, False Positives: Spam's Casualty of War Costing Billions, Available at http://www.enterpriseitplanet.com/security/news/article.php/2246371

[65] Nucleus Research Spam: The Silent ROI Killer, Available at http://www.nucleusresearch.com/research/d59.html, July 2003

[66] S. Radicati and M. Khmartseva, The IT Cost Of Spam. The Messaging Technology Report, Vol.12, No.8, Aug, 2003

[67] Joshua T. Goodman and Robert Rounthwaite, Stopping Outgoing Spam, In *Proceedings of the 5th ACM conference on Electronic Commerce*, ISBN:1-58113-711-0, pages 30–39, New York, NY, USA. 2004

[68] A. Juels and J. Brainard, Client Puzzles: A Cryptographic Defense Against Connection Depletion, *NDSS* pages:151–165, 1999

[69] D. Dean and A. Stubblefield, Using Client Puzzles to Protect TLS, In *Proceedings of 10th Annual USENIX Security Symposium*, 2001

[70] Paul Festa, Microsoft anti-spam campaign hypocritical, available at http://news.zdnet.co.uk/business/

[71] Sam Holden, Spam Filter Evaluations, available at http://sam.holden.id.au/writings/spam2/

[72] Sam Vaknin, The Economics of Spam, *United Press International*, July 23, 2002

[73] Paul Graham, A Plan for Spam, *The First SPAM Conference*, 2003

[74] Vernon Schryver, Distribute Checksum Clearinghouse, available at http://www.rhyolite.com/anti-spam/dcc/

[75] F. Kargl and J. Maier and M. Weber, Protecting Web Servers from Distributed Denial of Service Attack, In *Proceedings of World Wide Web*, pages:514–524, 2001

[76] J. Jung and B. Krishnamurthy and M. Rabinovich, Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites, *International World Wide Web Conference*, pages:252–262, MAY, 2002

[77] D. Mankins and R. Krishnan and C. Boyd and J. Zaho and M. Frentz, Mitigating Distributed Denial of Service Attacks with Dynamic Resource Pricing, In *Proceedings of Annual Computer Security Applications Conference (ACSAC 2001)*

[78] A. Back, Hashcash: A Denial of Service Counter-Measure, *Cypherspace*, http://cypherspace.org/hashcash/hashcash.pdf, AUG, 2002

[79] C. Dwork and M. Naor, Pricing via Processing or Combatting Junk Mail, *Crypto*, AUG, 1992

[80] Jonathan Krim, Challenge-Response' Technology Rejects Messages Unless Senders Are Cleared by Recipients, Washington Post News article, May 7, 2003

[81] T. Aura and P. Nikander and J. Leiwo, DoS-Resistant Authentication with Client Puzzles, *Lecture Notes in Computer Science*, vol 2133, 2001

[82] M. Castro and P. Druschel and A. Ganesh and A. Rowstron and D. Wallach, Security for Peer-to-Peer Routing Overlays, In *Proceedings of OSDI*, DEC, 2002

[83] Richard Clayton, Stopping Spam by Extrusion Detection, In *Proceedings of the First Email and SPAM conference*, July, 2004

[84] Hotmail sets email limits, http://news.zdnet.co.uk/business/0,39020645,2132358,00.htm

[85] Axel Zinser, Teergrubing, http://iksjena.de/mitarb/lutz/usenet/teergrube

[86] Marty Lamb, Using Statistics to Cause Spammers Pain, http://www.martiansoftware.com

[87] Kumar Chellapilla, Kevin Larson, Patrice Simard, Mary Czerwinski, Computers beat Humans at Single Character Recognition in Reading based Human Interaction Proofs (HIPs), In *Proceedings of Second Conference On Email and Anti-Spam, CEAS 2005*

[88] Kang Li and Calton Pu and Mustaque Ahamad, Resisting SPAM Delivery by TCP Damping, In *Proceedings of the First Email and SPAM conference*, 2004

[89] Andrew Wood, Quick Spam Filter, available at http://freshmeat.net/projects/qsf/

[90] The end2end-interest Archives, available at http://www.postel.org/pipermail/end2end-interest/

[91] Perl Monger Mailinglist, available at http://mail.pm.org/archives/classiccity-pm/

[92] Paul Judge, The SPAM Archive, available at http://www.spamarchive.org