UNIVERSITY OF CALIFORNIA
RIVERSIDE

Fighting Spam, Phishing and Email Fraud

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Computer Science

by

Shalendra Chhabra

December 2005

Thesis Committee:

Professor Dimitrios Gunopulos, Chairperson
Professor Vana Kalogeraki
Professor Eamonn Keogh
Professor Mart Molle
Dr. William S. Yerazunis

Copyright by
Shalendra Chhabra
2005

The Thesis of Shalendra Chhabra is approved:

_____

_____

_____

_____

_____

Committee Chairperson

University of California, Riverside

Acknowledgements

Dreaming big with a drive to excel requires immense motivation and a positive attitude. It takes a lot of courage to follow one's heart and chase one's dreams. Great things are never done without hard work and sacrifice.

I am grateful to my *parents* for teaching me to dream big, for blessing me with the courage to follow my heart, for inspiring me to work hard, and for instilling the spirit of sportsmanship in me. They brought me up in a very optimistic environment, giving me a remarkable ability to judge good from bad. I am grateful to my *sisters* for being the best *sisters* in the world. Every success in life would not have been possible without their sacrifice, help, and encouragement. They taught me that *winners don't do different things but they do things differently*. I am grateful to my *grandparents*, my *uncle*, my *aunt*, my *cousins*, and other members of my *family* for their love, affection, and support during difficult times.

I am grateful to all my *friends, teachers*, and *roommates* for enriching my life.

I am highly indebted to members of the *Graduate Division* at the *University of California, Riverside (UCR)*, for providing me with an opportunity to pursue graduate studies at this temple of education.

This thesis is a compiled version of the cumulative knowledge and wisdom gained throughout my two years at the *Department of Computer Science and Engineering* at *UCR*; internships at *Mitsubishi Electric Research Laboratories (MERL)*; presentations at *MIT, Stanford University,* and *Cisco Systems*; interactions with people from around the world; and fruitful discussions from my interviews at *Microsoft, Cisco Systems, Intel*, and *Google*.

Many people have contributed to this thesis directly or indirectly. It's an honor to have come across so many wonderful people.

I am highly indebted to my advisor, *Professor Dimitrios Gunopulos*, for motivating, guiding, and helping me channel my creative energy at times when I was unable to focus. This thesis would not have been possible without his encouragement, wisdom, feedback, and support. His counselling kept me pointed in the right direction, and his periodic feedback protected me from pitfalls.

I am highly grateful to members of my thesis committee—*Professor Vana Kalogeraki* and *Professor Eamonn Keogh* for providing guidance, encouragement, and valuable insight. Communication with them has always brought cheerful spirits and inspiration in me.

I am highly indebted to *Dr. William "Bill" S. Yerazunis (Captain Crash of Junkyard Wars)* from *MERL* for being an amazing mentor and an incredible collaborator during 2004 and 2005. He is the smartest person I have ever come across. Working with him has added tremendous value to my life. Words are not enough to thank *Bill* for the role he has played in my successive successes.

I am highly indebted to *Professor Mart Molle* for his continued, unparalleled support as my graduate advisor. He is a very caring, understanding, excellent teacher who imparts novel ideas to his students. It was during one of his class projects in *Advance Computer Networks* class during Fall of 2003 that led me to the arena of spam-fighting. Words are not enough to express my gratitude for him.

I am grateful to *Dean Dallas L. Rabenstein* for approving continued support for my Master's Degree at *UCR*.

I am thankful to *Professor Stefano Lonardi* for cheering me with his wonderful smile. Fruitful discussions, and communication with him add more confidence in my life.

I am thankful to *Professor Harry C. Hsieh* for being my academic advisor in the early days at *UCR*; to *Professors Chinya V. Ravishankar, Satish Tripathi, Michalis Faloutsos, Christian R. Shelton*, and *Jun Yang* for very enlightening discussions, and to *Department Chair Professor Thomas Payne* for providing valuable teaching tips and resources. Many thanks to members of the Data Mining Lab—*Sharmila Subramaniam, Mirella Moro, Zografoula Vagena, Song Lin, Petko Bakalov, Marios Hadjieleftheriou, Dimitris Papadopoulos, Demetrios Z. Yazti*, and *Christina Charalambidou* for their help and a good time. Many thanks to the department's administrative and management staff—*Terri Phonharath, Kathy Vu, Lilian Kang, Mimi Trinh, Tiffany Arakai, Nicole Smidt, Dulce Shipley, Janice Leslie*, and *Emily Papavero*. Many thanks to *Kelly Hinosowa, Harrison Leong* and *Tatiana Rakic* for being the best international advisors. Many thanks to members of the systems group—*Victor Hill, Benjamin Arai, Conley Read, Craig Boucher, Jacob Lewallen, Benjamin Arai*, and *Chuck Boucher*. Many thanks to *Brian Linard, John Cortes, Ronald Feliciano,* and my *students* from *Computer Organization and Assembly Language* class for a good time.

I am highly grateful to *Dr. Bhiksha Raj* from *MERL* for being an incredible mentor and a great friend. His guidance and wisdom brought a smile to my face every day during the summers of 2004 and 2005. His inspirational notes crystallized my conviction to excel in constraints. This thesis would not have been in its present form without his constant support, motivation, and guidance.

I am grateful to *Paul Graham* for playing a major role in my success, for inspirational communication, for writing wonderful essays, for organizing the *MIT Spam Conference*, and for hosting dinners for anti-spam experts every year. It was because of the MIT Spam Conference of 2004 that I eventually met *Bill Yerazunis*.

I am thankful to *Joe Marks, Bent Schmidt-Nielsen,* and others from *MERL*; to *Rita Singh* from *Haikya Corporation* for great advice, and to *Christian Siefkes (Freie Universität, Berlin)* and *Fidelis Assis (Embratel)* for being great collaborators.

I am grateful to *Joshua Goodman* for introducing me to the *MSN Safety Team,* for providing me with an opportunity to volunteer at the *Second Conference on Email and Anti-Spam (CEAS 2005),* and for invaluable feedback and tips. I am grateful to *Manav Mishra* for stimulating discussions and feedback at the *Email Authentication Implementation Summit 2005, New York City,* and to *Robert Rounthwaite* for intellectual conversations during lunch at *CEAS 2005*. I am grateful to *Kumar Chellapilla* for invaluable feedback and tips. Many thanks to *Harry S. Katz, Kris Iverson, John Scarrow, Ning Zhang, Geoff Hulten, Collene Georgia, Jeff Aylesworth,* and others from *Microsoft Corporation* for brainstorming conversations, and for their time and help during relocation.

I am grateful to *Jim Fenton* for introducing me to the *Network and Spam Solutions Team* at *Cisco Systems* and for great discussions. Many thanks to *Shamim Pirzada, Sanjay Pol, Michael Thomas,* and others from *Cisco Systems* for their time and fruitful conversations.

Many thanks to *Michael Ripley, Gary Graunke, Michael Andre, Don Whiteside, Gary Mittelstaedt, Keith Shippy,* and *Janie C. Mason* from *Intel* and to *Charles Haynes, Kristin Kassaei,* and members of the Gmail team from *Google* for their time and very fruitful face-to-face discussions.

Many thanks to *lawyers Matthew Prince (Unspam Technologies), Jon Praed (Internet Law Group),* and *Aaron Kornblum (Microsoft)* for motivation, guidance, transcripts and relevant resources for legal actions against spam.

Many thanks to *Jonathan Zdziarski (DSPAM)* and *Richard Jowsey (Death2Spam)* for reviewing draft versions of this thesis. Many thanks to *Eric S. Johansson (CAMRAM)* for providing information and feedback on CAMRAM, and to *Matt Blumberg* and *Andy Sautins (Return Path)* for feedback on the *Bonded Sender* section. Many thanks to *Erik Brown* for designing the logo for the thesis. Many thanks to *Shawn R. Lesniak* and *Amir Friedländer* for making proofreading super-fun.

I am thankful to *Laird A. Breyer (University of Lancaster, UK); Regu Radhakrishnan, Ronald Johnson, Ajay Divakaran (MERL); David Silver (MarkMonitor); Katherine Bretz, Des Cahill, Doug Warren (Habeas); Ether Dyson (CNET); Miles Libbey (Yahoo); P. Oscar Boykin (University of Floria); John Graham-Cumming (Electric Cloud); Larry McGrath, Vickie Park, Dallas Johnson, Mitch Boretz, Monica Wicker, Kara Oswood, Karen Smith,*

Abraham Artuz (UCR); Larry Reeker (NIST); Jonathan Oliver, Chad West, Leon Hilton (MailFrontier); Daniel Quinlan (IronPort Systems); Gordon Cormack (University of Waterloo); Mike Linksvayer (Creative Commons); and *Rebecca Wetzel* for very fruitful discussions.

I am grateful to my previous mentors, members of the security group at the *University of Milan—- Professor Pierangela Samarati, Sabrina De Capitani, Stefano Paraboschi, Ernesto Damiani*; *Professor Jean Goubault Larrecq* at the *Laboratoire Spécification et Vérification, Ecole Normale Superieure De Cachan, France*; *Dr. N. Raja* at the *Tata Institute of Fundamental Research, India*, and *Professor S. C. Gupta* at the *Institute of Technology, Banaras Hindu University, India*, for exposing me to research during my undergraduate studies.

I am thankful to all my *friends* for standing by me during the worst times. To express my gratitude for my *friends* would require far more pages than those occupied by the contents of this thesis. My movie, *The BackBenchers*, may prove to be a better medium for this task, provided it is ever finished.

I am thankful to my former roommate *Casey Christine Hoover (UCR)* for being the best roommate in the world. Without the encouragement of this precious person, I would never have discussed and conducted anti-spam measures with the *Computing and Communications (C&C)* group at *UCR*. Her good words—**"uniqueness is a virtue, lead by example, actions speak louder than words, good leaders know how to ask for help, and you will be fine"**—created an optimistic air in the house and pumped me with confidence for presentations at the MIT Spam Conference (2005), for challenging interviews at big corporations, and during tough times, especially during my car accident.

Finally I am thankful to God for showering His love and affection, for bringing wonderful people into my life, and for everything I have achieved to this day.

With such great blessings and positive energy, a revolution to put an end to the spammers' business model is in the cards. It begins on the next page...


CRM114 - the Controllable Regex Mutilator
CRM114 - the Controllable Regex Mutilator
Fighting Spam, Phishing and Email Fraud
Revolution is in the Cards
SPAM
PHISHING
EMAIL FRAUD

To my parents and sisters

ABSTRACT OF THE THESIS

Fighting Spam, Phishing and Email Fraud

by

Shalendra Chhabra

Master of Science, Graduate Program in Computer Science
University of California, Riverside, December 2005
Professor Dimitrios Gunopulos, Chairperson

Spamming in the electronic communications medium is the action of sending unsolicited commercial messages in bulk without the explicit permission or desire of the recipients. The most common form of spam is email spam. Phishing is a particular type of spam characterized by attempts to masquerade as a reputed business. The objective is to trick recipients into divulging sensitive information such as bank account numbers, passwords, and credit card details. Spam and phishing cause billions of dollars' worth of losses to businesses.

Many initiatives on technical and legal levels are currently underway for fighting this challenge. In this thesis we will examine these issues from the aspects of network protocols, filtering, reputation, human psychology, scalability and corporate alliances.

We will present a comprehensive description of technical initiatives to fight spam which include server-side and client-side filtering (using statistical and collaborative techniques); lists (blacklist, whitelist, greylist and brownlist (in CAMRAM)); email authentication standards such as Identified Internet Mail (IIM) from Cisco Systems, Domain Keys (DK) from Yahoo!, Domain Keys Identified Mail (DKIM), Sender Policy Framework (SPF) from Pobox, Sender ID Framework (SIDF) from Microsoft Corporation; and emerging sender reputation and accreditation services (from Habeas, Return Path, IronPort Systems, and CipherTrust).

We will touch upon specific anti-spam techniques used in the popular spam-filtering appliances such as IronMail Connection Control from CiphterTrust, MailHurdle (RazorGate) from Mirapoint, Mail Security 8160 from Symantec, and MailGate Edge from Tumbleweed Communications.

We will investigate various tricks that spammers use to fool spam filters, and will also analyze a spammer's *to do* list by looking through Jeremy Jaynes's court transcripts. Jaynes was the world's eighth most prolific spammer until he was convicted and sentenced to nine years in prison under Virginia statute. We will illustrate malicious stages in a phishing attack lifecycle. We will also discuss the anatomy of a phishing email and various other tricks that fraudsters use in the spoofed web sites.

We will first explain *sender-pays* model for email and will then describe *The Penny Black Project*, a *challenge-response* system with bankable tokens developed by Microsoft Researchers.

We will focus on a special class of Human Interactive Proofs (HIPs) known as Completely Automatic Public Turing test to tell Computers and Humans Apart (CAPTCHA). CAPTCHAs are used by Hotmail, Yahoo!, Google, and other companies to prevent automatic registration of email accounts by bots and to prevent bulk mailing by the spammers.

We will explain a mechanism for throttling Internet resource abuse based on the cryptographic technique Proof of Work (PoW) known as Hashcash. We will then describe in detail a *hybrid sender-pays* email system based on Hashcash, known as the Campaign for Real Mail or CAMRAM.

We will also focus on the machine learning approach for spam filters. A large number of spam filters and other mail communication systems have been proposed and implemented in the past. We will describe a possible unification of these filters, allowing their technology to be described in a uniform way. In particular, describing these filters reveals a large commonality of designs and explains their similar performance. We will present results of our experiments with the Markov Random Field (MRF) model and Winnow-based approaches for spam-filtering in the open source spam filter CRM114 Discriminator Framework. Our results indicate that such models outperform the Naïve Bayesian approach for spam-filtering. We will illustrate CRM114 usage for small-, medium-, and large-scale enterprises (for filtering up to one million client email accounts). We will also investigate the significance of reputation-based protocols for the email communication flow.

We will highlight some problems with the current spam-fighting techniques. We conclude that with the combination of better spam-fighting techniques, legal actions, awareness among Internet users, and cooperation within the industry, the spammers' business model can be disrupted in the next few years.

In addition to work at the Department of Computer Science and Engineering, University of California Riverside, this thesis is a product of collaborative work at the Mitsubishi Electric Research Laboratories, Cambridge, MA (MERL), and enlightening interactions with the MSN Safety Team of Microsoft Corporation, Network and Spam Solutions Team of Cisco Systems, Gmail Team of Google, and Anti-Spam Team of Yahoo!. Preliminary results of this thesis have appeared on Slashdot and presentations at the *MIT Spam Conference (2005, 2004, 2003)*; Cisco Systems (2005); *Second Conference on Email and Anti-Spam (CEAS 2005), Stanford University*; *The Fourth IEEE International Conference on Data Mining, Brighton, UK (ICDM04)*; *8th European Conference on Principles and Practice of Knowledge Discovery in Databases, Pisa, Italy (PKDD 2004)* and *1st International Workshop on Peer2Peer Data Management, Security and Trust, Zaragoza, Spain, (PDMST04).*

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Contributions

The main contributions from the author of this thesis are:

- **The author presents** an in-depth study of the problem of spam and phishing.
- **The author presents** redacted court transcripts of Jeremy Jaynes, the world's eight most prolific spammer.
- **The author presents** an in-depth description of current initiatives for fighting spam.
- **The author proposes** a unified model of spam filtration for the spam filters currently available in the market, in collaboration with others.
- **The author proposes** a Markov Random Field (MRF) and a Winnow-based model in the CRM114 Discriminator, in collaboration with others.

- **The author presents** different configuration modes for implementing CRM114 at mailservers.
- **The author presents** the internals of a *hybrid sender-pays* system known as CAM-RAM.
- **The author presents** a reputation system built on the top of Gnutella known as SupRep.

1.2 Papers

The author, in collaboration with others, published the following papers during the course of his graduate studies:

1. *A Unified Model of Spam Filtration*. In MIT Spam Conference 2005, MIT, Cambridge.
2. *Spam Filtering using a Markov Random Field Model with Variable Weighting Schemas*. In Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM '04), Brighton, UK.
3. *Combining Winnow and Orthogonal Sparse Bigrams for Incremental Spam Filtering*. In Proceedings of the 15th European Conference on Machine Learning and 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD 2004), Lecture Notes in Computer Science, Springer-Verlag 2004.
4. *A Protocol for Reputation Management in Super-Peer Networks*. In Database and Expert Systems Applications, 15th International Workshop on (DEXA'04), Spain.

1.3 Presentations

The author gave the following presentations:

1. *Netizen, Authentication and Reputation,* Second Conference on Email and Anti-Spam (CEAS 2005), Stanford University.

2. *Its About You, Me and Every Netizen Because We've Got Spam and Phish*, Cisco Systems, April 18, 2005.

3. *A Unified Model of Spam Filtration*, MIT Spam Conference, MIT, Cambridge, January 21, 2005.

4. *Spam Filtering using a Markov Random Field Model with Variable Weighting Schemas*, ICDM04, Brighton, UK.

1.4 Postings on Slashdot

The author's post, titled *Microsoft Researchers on Stopping Spam* appeared on Slashdot on April 11, 2005.

1.5 Slashdot Book Reviews

The author's review for the book *Ending Spam* appeared on Slashdot on August 15, 2005.

1.6 Articles

The author's article, titled *A Quick Note on Yahoo! Mail SpamGuard*, was cited on Slashdot on April 11, 2005.

1.7 Volunteer Work

The author was a volunteer at the *Email Authenication Implementation Summit, New York City, 2005* and the *Second Conference on Email and Anti-Spam (CEAS 2005), Stanford University*.

1.8 Surveys

The author submitted a human subjects protocol, HS-05-037, entitled *Spam Survey at UCR: Collecting Useful Information about Spam (Unsolicited Commercial Email) Received by UCR Population for Providing Effective Guidelines for Design and Development of Effective Spam Filtering Solutions at UCR Mailservers to Curb Spam and Prevent Email Fraud* to the **Office of Research Integrity** at the **University of California, Riverside**. The protocol was approved, and an online survey about spam received by email users at the **University of California, Riverside** was conducted in May 2005. We received 1721 responses, around 10% of the university population.

1.9 Thesis Structure

The thesis titled *Fighting Spam, Phishing and Email Fraud*, is divided into ten chapters: *Introduction; Background; Related Work; A Unified Model of Spam Filtration; The CRM114 Discriminator Framework; The CAMRAM System; Spam Filtering Using a Markov Random Field Model; Combining Winnow and Orthogonal Sparse Bigrams for Incremental Spam Filtering; Reputation Systems; and Conclusion.*

Conclusion is followed by the *Bibliography*, which is followed by the **Vita** of the author.

In Chapter 2 we introduce the terminology of email, spam and phishing. We illustrate different types of spam plaguing the communication media. We describe Internet email flow and the Simple Mail Transfer Protocol (SMTP). We describe geographical origins of spam and trends in spam exploits derived by Microsoft Researchers from a corpus of hand-classified email messages obtained through the Hotmail Feedback Loop. We study Federal Trade Commission's (FTC's) analysis of False Claims and Categories of Spam (April 2003) in a random sample of spam messages drawn from a corpus of Unsolicited Commercial Email (UCE) database. We illustrate the decomposition of a phishing attack lifecycle in various malicious phases and discuss the anatomy of the phishing email. We then discuss tricks used in spoofed web sites. We discuss legal initiatives on spam and analyze a spammer's *to do* list by looking through Jeremy Jaynes's court transcripts. Jaynes was the world's eighth most prolific spammer until he was convicted and sentenced to nine years in prison under Virginia statute. This chapter concludes with a note on the effect of spam on people.

Chapter 3 is the related work section. We discuss blacklist, whitelist and greylist. We describe email authentication proposals such as *Sender Policy Framework (SPF)* from Pobox, *Sender ID Framework (SIDF)* from Microsoft Corporation, *DomainKeys (DK)* from Yahoo!, *Identified Internet Mail (IIM)* from Cisco Systems, *Domain Keys Identified Mail (DKIM)*, a combined proposal from Yahoo! and Cisco Systems. We present Email Authentication Score Card, a study conducted independently by MarkMonitor and VeriSign. We also discuss the methodology of statistical filters based on the machine learning approach.

We first explain *sender-pays* model for email and then describe *The Penny Black Project*, a *challenge-response* system with bankable tokens developed by Microsoft Researchers. We then discuss a special class of Human Interactive Proofs (HIPs) known as Completely Automatic Public Turing test to tell Computers and Humans Apart (CAPTCHA). CAPTCHAs are used by Hotmail, Yahoo!, Google, and other companies to prevent automatic registration of email accounts and to prevent automatic signing of email accounts by bots and bulk mailing by the spammers. We then explain a mechanism for throttling Internet resource abuse based on the cryptographic technique Proof of Work (PoW) known as Hashcash.

We then describe a proposal for controlling spam at the router level. We touch upon distributed collaborative filtering, use of social networks for filtering spam, and disposable email addresses. We then explain Project Honey Pot from Unspam Technologies. We also explain current reputation- and accreditation-based services from Habeas, Return Path, CipherTrust, and IronPort Systems. We then mention anti-spam techniques used in the popular spam-filtering appliances.

In Chapter 4, we discuss a unified model of spam filtration. A large number of spam filters and other mail communication systems have been proposed and implemented in the past. We describe a possible unification of these filters, allowing their technology and behaviour to be described in a uniform way. In particular, describing these filters reveals a large commonality of designs and explains their similar performance. A preliminary version of this chapter appeared at the *MIT Spam Conference 2005*.

In Chapter 5, we describe the open source spam filter Controllable Regex Mutilator, concept #114 (CRM114) Discriminator Framework. We illustrate its usage in a shared mode for more than one million client email accounts used by a large ISP company.

In Chapter 6, we describe the internals of a *sender-pays* system using Hashcash known as the Campaign for Real Mail or CAMRAM.

In Chapter 7, we describe a Markov Random Field (MRF) model based approach to filter spam. This model is integrated into the CRM114 Discriminator Framework. This approach examines the importance of neighborhood relationship (expressed as MRF cliques) among words in an email message for the purpose of spam classification. We propose and test several different theoretical bases for weighting schemes among corresponding neighborhood windows. Our results indicate that the MRF model outperforms the Naïve Bayesian model. Our results demonstrate that unexpected side effects depending on the neighborhood window size may have larger accuracy impact than the neighborhood relationship effects of the Markov Random Field.

A preliminary version of Chapter 7 appeared at *The Fourth IEEE International Conference on Data Mining (ICDM04), Brighton, UK*.

In Chapter 8, we describe a statistical but non-probabilistic classifier based on the Winnow algorithm. We introduce the concept of Orthogonal Sparse Bigrams (OSB). This model is also integrated into the CRM114 Discriminator Framework.

A preliminary version of this chapter appeared at the *8th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD) 2004, Pisa, Italy*.

In Chapter 9, we describe reputation models in other realms of computer science. These reputation models should be well studied in order to propose reputation systems for email communication and web services in the future.

In Chapter 10, we highlight some problems with the current spam-fighting techniques. We conclude that with the combination of better spam fighting techniques, legal actions, awareness among Internet users, and cooperation within the industry, the spammers' business model can be disrupted within the next few years.

Chapter 2

Background

2.1 Email, Spam and Phishing

In this section we explain the email system and the problem of spam and phishing.

2.1.1 Email

Email is a method of sending and receiving messages over electronic communication systems such as the Internet. The modern-day protocol for sending email is the Simple Mail Transfer Protocol (SMTP), proposed in 1982[106]. The most commonly-used protocols for email retrieval by client programs, Post Office Protocol (POP)[97] and Internet Message Access Protocol (IMAP)[63], were proposed in 1984 and 1996, respectively.

2.1.2 Spam

Spamming in the electronic communications medium is the action of sending unsolicited commercial messages in bulk without the explicit permission or desire of the recipients. Figure 2.1 depicts different types of spam such as email spam, instant messaging spam (spim), Usenet newsgroup spam, web search engines spam, weblogs spam and mobile phone messaging spam. A person engaged in spamming is called spammer.

In this thesis, spam refers to email “spam” and “ham” refers to legitimate email.

Figure 2.2 shows spam messages caught in my junk folder. Three typical examples of spam messages are shown in Figure 2.3, Figure 2.4 and Figure 2.5.

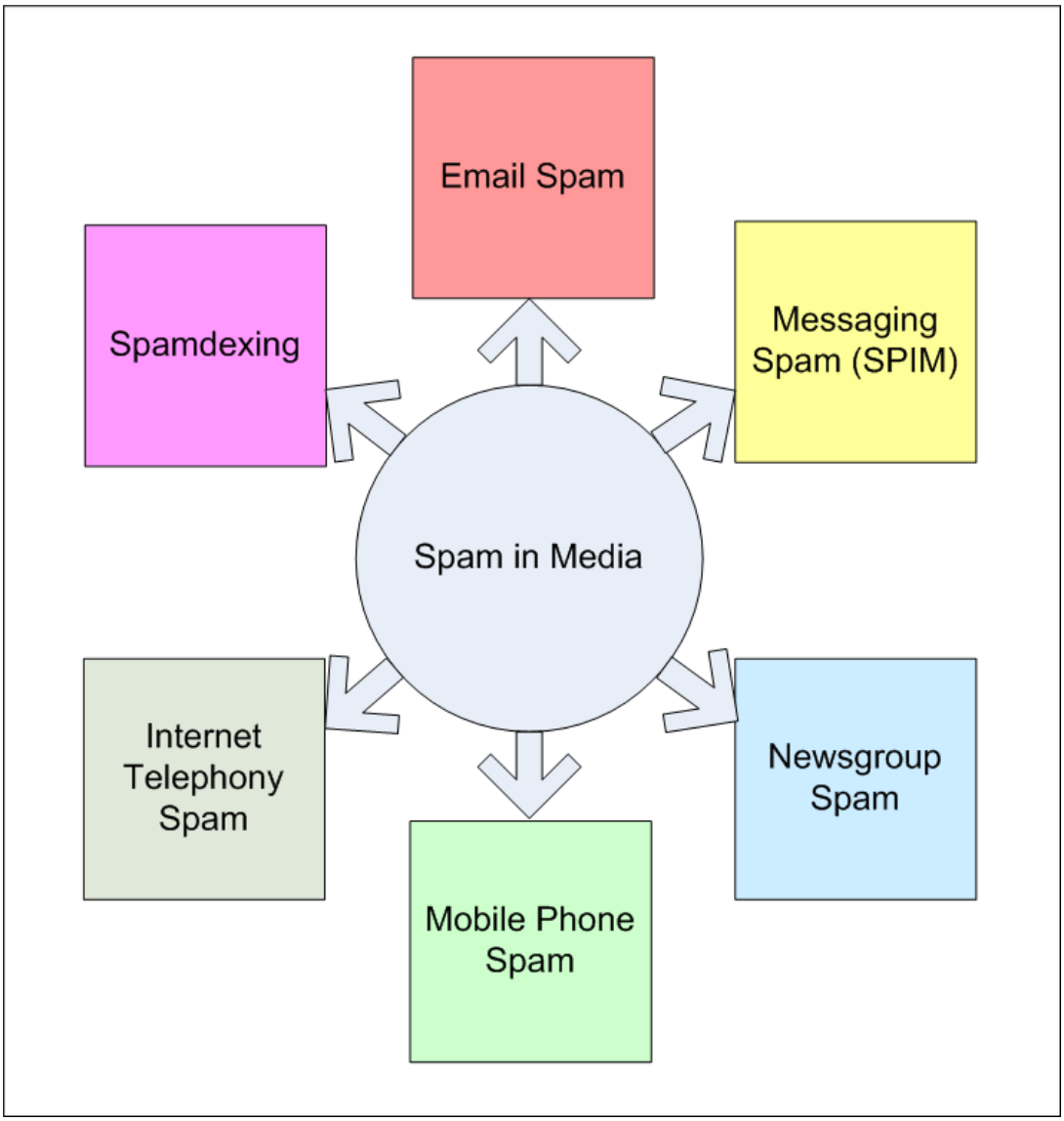


Figure 2.1: Different types of spam in the media.

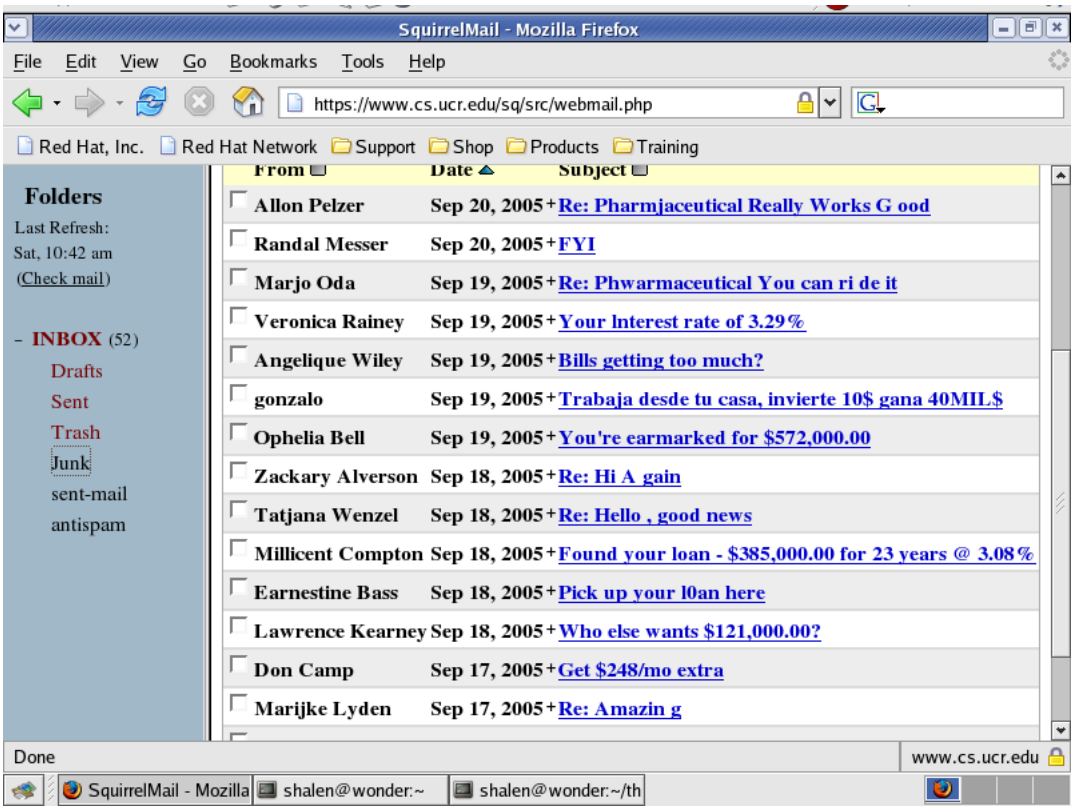


Figure 2.2: A snapshot of spam messages in my junk folder. Note the different subject lines. Most of them try to attract attention by including “Re:” in the subject line.

UNIVERSITY DIPLOMAS

OBTAIN A PROSPEROUS FUTURE, MONEY-EARNING POWER, AND THE PRESTIGE THAT COMES WITH HAVING THE CAREER POSITION YOU'VE ALWAYS DREAMED OF. DIPLOMAS FROM PRESTIGIOUS NON-ACCREDITED UNIVERSITIES BASED ON YOUR PRESENT KNOWLEDGE AND LIFE EXPERIENCE

If you qualify, no required tests, classes, books or examinations.

Bachelors', Masters', MBA's, Doctorate & Ph.D. degrees available in your field.

CONFIDENTIALITY ASSURED

CALL NOW TO RECEIVE YOUR DIPLOMA WITHIN 2 WEEKS

1-206-666-4393

CALL 24HRS, 7 DAYS A WEEK, INCLUDING SUNDAYS & HOLIDAYS

.....

to stop mailing please visit: http://remove-me-instantly.com

Figure 2.3: A spam message with a picture attachment offering university diplomas.

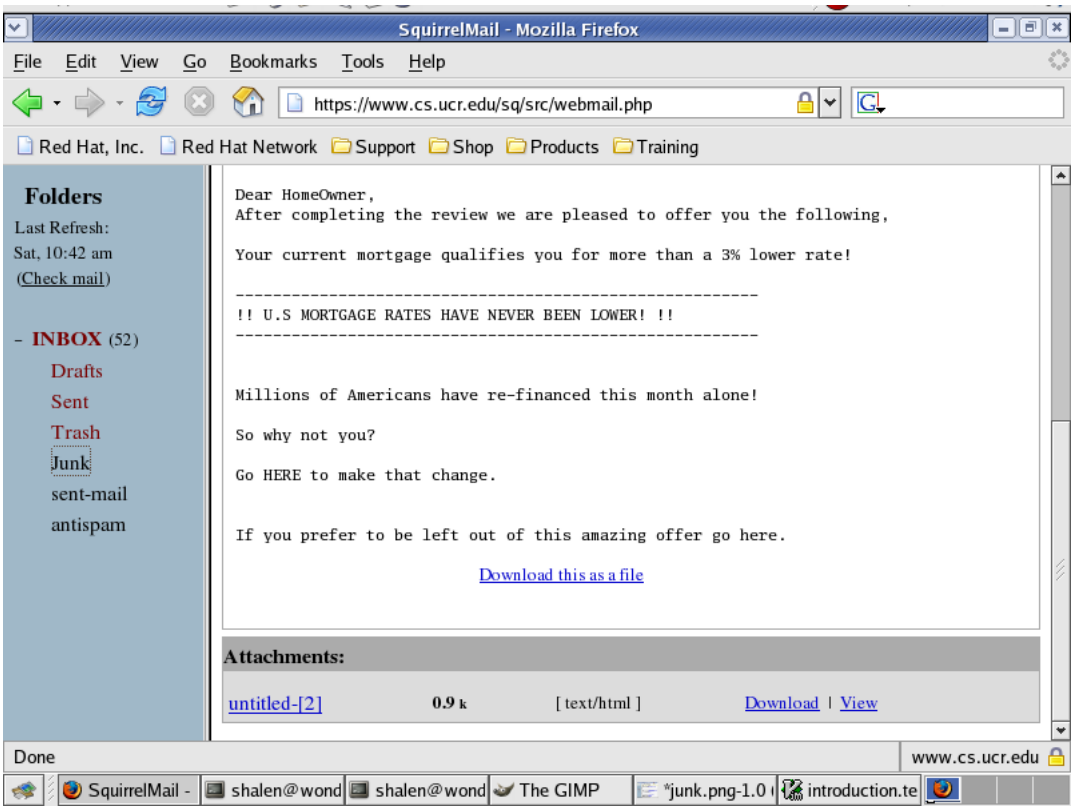


Figure 2.4: A spam message offering low mortgage rates.

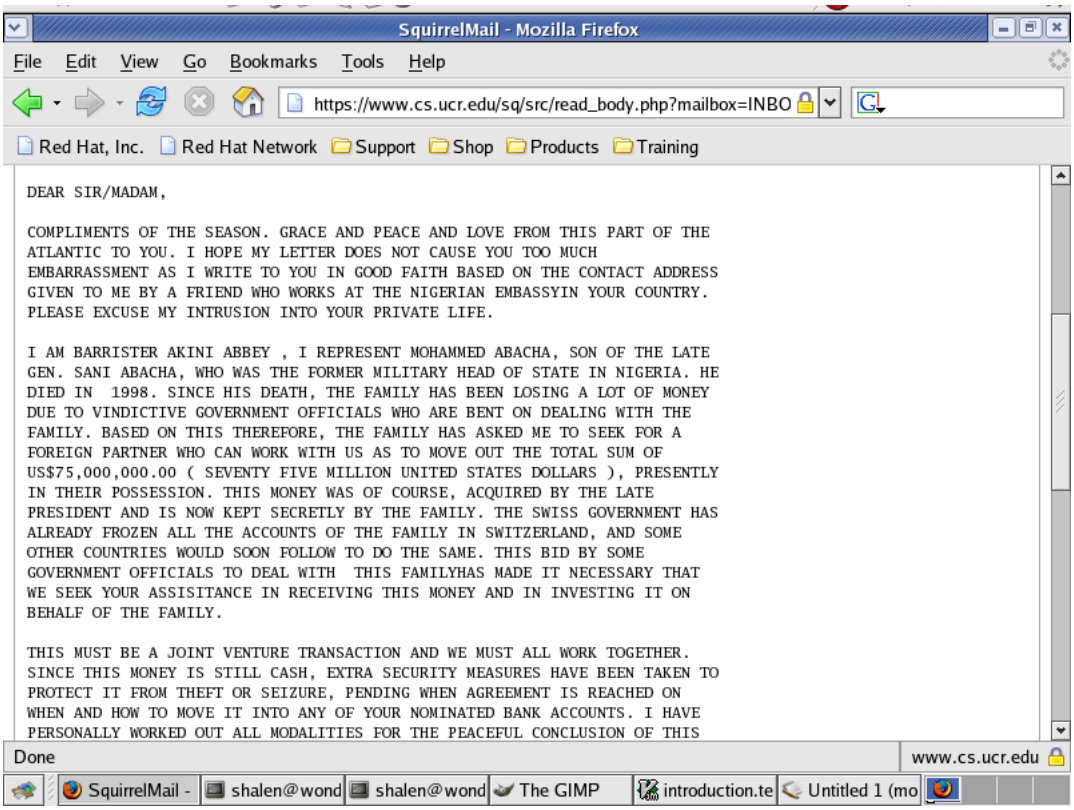


Figure 2.5: An example of Nigerian scam (419 Fraud).

2.1.3 Phishing

Phishing is a particular type of spam which reflects social engineering. Phishing frauds are characterized by attempts to masquerade as a trustworthy person or emulate an established, reputed business in an electronic communication such as email. The objective is to trick recipients into divulging sensitive information such as bank account numbers, passwords, and credit card details. A person engaged in phishing activities is called a phisher.

Two phishing attempts taken from MailFrontier's "Phishing IQ Test"[76] are shown in Figure 2.6 and Figure 2.7. The emails appearing to have originated from eBay and PayPal instead redirect recipients to fraudulent websites:

http://awcg1dln.com and *http://www.signupaccount.com*, respectively.

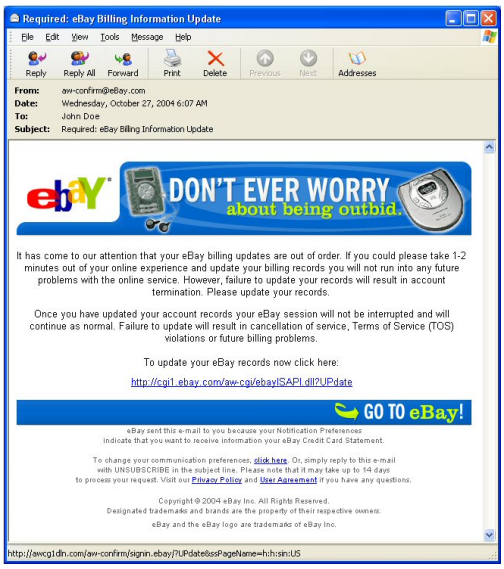


Figure 2.6: A phishing attempt to masquerade the sender as eBay. Note that the URL at the bottom redirects the recipient to the website *http://awcg1dln.com* and not *http://www.ebay.com*. Source: Phishing IQ Test, MailFrontier, Inc.

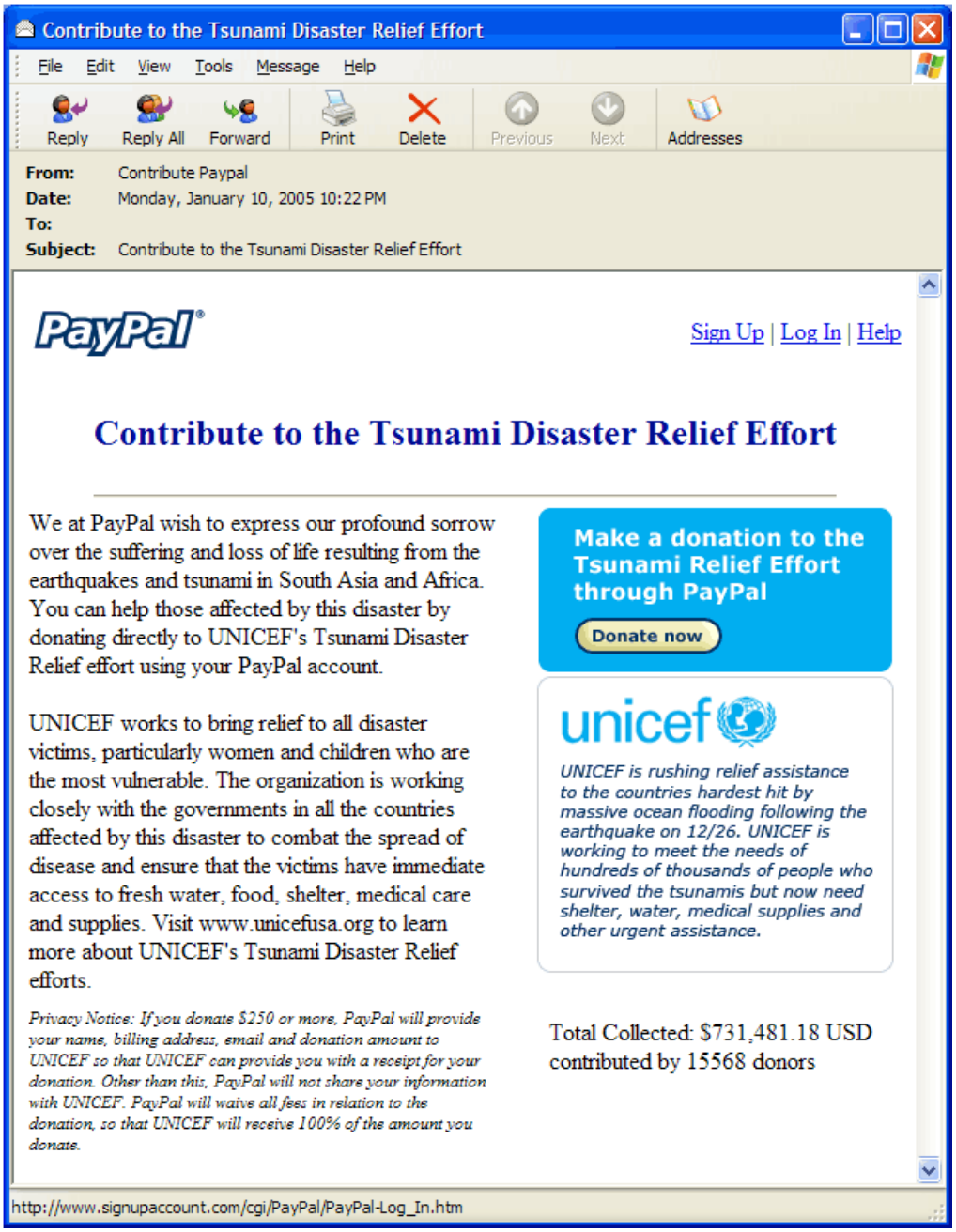


Figure 2.7: A phishing attempt to masquerade the sender as PayPal. Note that the URL at the bottom redirects the recipient to *http://www.signupaccount.com* and not *http://www.paypal.com*. Source: Phishing IQ Test, MailFrontier, Inc.

2.2 Internet Email Agents

We will now describe various Internet email agents.

2.2.1 Email Address

An Internet email address is a string of the form username@host.domain. Ray Tomlinson[126] first initiated the use of the @ sign to separate the names of the user and the machine in 1971.

2.2.2 Mail User Agent (MUA)

A Mail User Agent (MUA), also known as email client, is a program that is used to send and read email. Example email clients include Eudora from Qualcomm, KMail from KDE, Mail from Apple Computer, Outlook Express from Microsoft and Thunderbird from Mozilla Foundation. In addition, there are web-based email programs and services (known as web-mail) such as SquirrelMail, AIM Mail from America Online, Yahoo!Mail from Yahoo!, Gmail from Google, and Hotmail from Microsoft.

2.2.3 Mail Transfer Agent (MTA)

A Mail Transfer Agent (MTA) is a program that is used to transfer email messages from one computer to another. Example MTAs include Sendmail from Sendmail, Inc.; Postfix from IBM; an Exim from the University of Cambridge.

2.2.4 Mail Delivery Agent (MDA)

A Mail Delivery Agent (MDA) is a program responsible for delivering incoming email messages to recipients' individual mailboxes. Example MDAs include Procmail and Maildrop.

2.3 Internet Email Flow

We will now describe email communication flow between Sender Alice and Recipient Bob over the Internet using various agents described in Section 2.2. The email communication flow between Sender Alice and Recipient Bob is illustrated in Figure 2.8.

1. *Sender Composes Message and MUA Transfers the Message to Local MTA*

   Sender Alice composes a message intended for Bob using her MUA such as Outlook Express. Alternatively, Sender Alice can use any of the webmail service such as Hotmail offered by Microsoft or Yahoo!Mail offered by Yahoo to compose a message. Alice then provides the email address of the recipient, which is *bob@b.org*. The MUA transfers the message in Internet message format[64] to local MTA using SMTP. The local MTA is *smtp.a.org*. This is shown in step 1 in Figure 2.8.

2. *Sending MTA Performs Lookup for the Mail Exchanger Records (MX) of the Recipient Domain through the Domain Name System (DNS)*

   The sending MTA *smtp.a.org* determines the recipient domain through the recipient email address. The recipient email address is *bob@b.org* and therefore the username is *bob* and the recipient domain is *b.org*. The sending MTA *smtp.a.org* makes a query requesting the Mail Exchanger records (MX) for the domain *b.org* through the Domain Name System (DNS)[117]. The MX records of the DNS entry of a particular domain have information about the mail exchange servers (MTAs) responsible for accepting messages for this particular domain[74]. This is shown in step 2 in Figure 2.8.

   As illustrated in step 3 in Figure 2.8, the DNS server for the the recipient domain *b.org* is *ns.b.org*. The DNS server *ns.b.org* responds to the DNS query from the sending MTA *smtp.a.org* with the MX records listing the mail exchange servers for the *b.org* domain.

   Domains usually have several mail exchange servers. The MX records are prioritized with a preference number which indicates the order in which these mail servers should be contacted for delivery. The MX record with the smallest preference number has the highest priority and is the first server to be tried.

   As illustrated in Figure 2.8, *mx.b.org* is the mail exchange server with the highest priority for the *b.org* domain.

3. *SMTP Connection between Sender/Forwarding MTA and the Recipient MTA*

   Sender/Forwarding MTA *smtp.a.org* uses SMTP to send the message to the recipient's MTA (i.e. to *mx.b.org*). This is shown in step 4 in Figure 2.8. The recipient MTA delivers the message to the recipient's (i.e Bob's) email box using its MDA

4. *Recipient Retrieves Email Using MUA*

   As illustrated in step 5 in Figure 2.8, Recipient Bob uses his MUA to retrieve the message (by using Post Office Protocol (POP3)[97] or Internet Message Access Protocol (IMAP)[63]). Alternatively, Bob can read his email by directly logging into *mx.b.org* or by using a webmail service such as Hotmail or Yahoo!Mail.

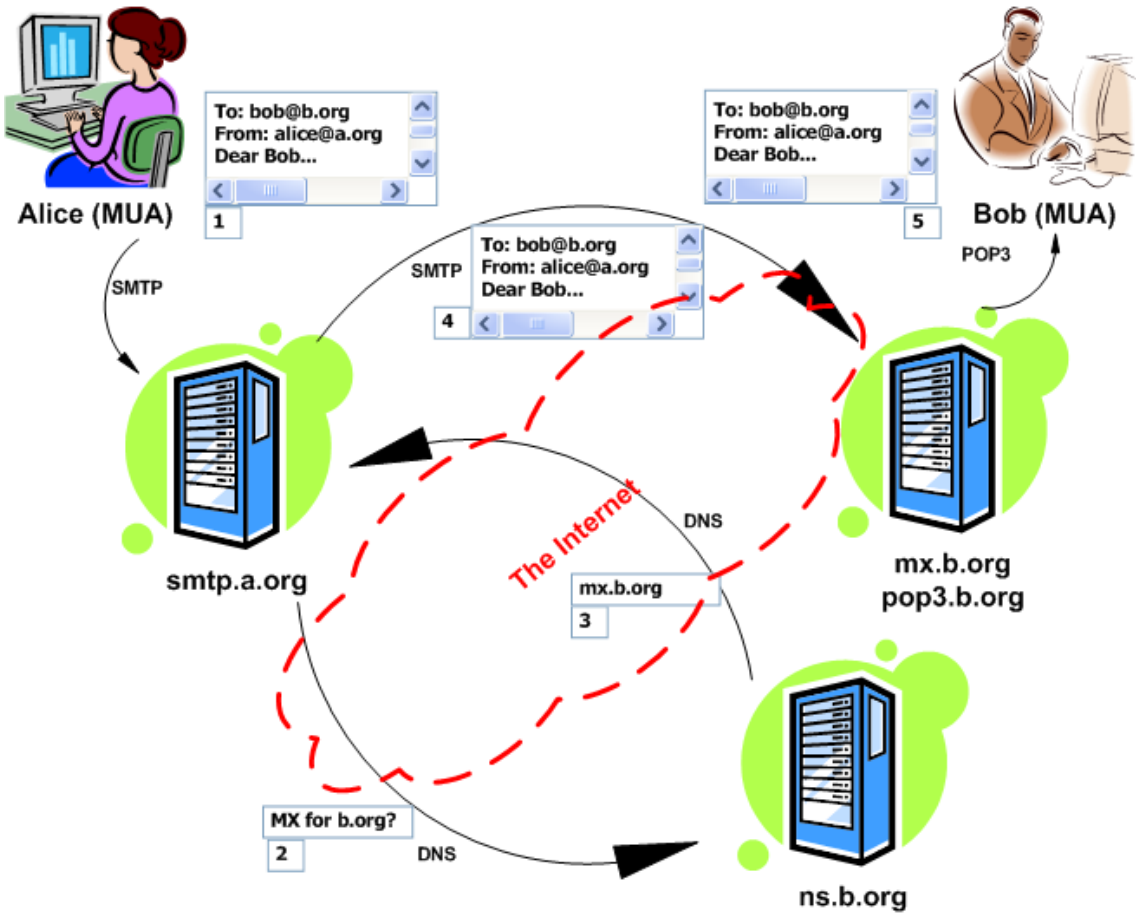


Figure 2.8: A schematic of the electronic mail communication flow over the Internet between Sender Alice and Recipient Bob.

2.4 Internet Email Format

The format of the Internet email messages is defined in RFC 822[114] and RFC 2822[64]. An Internet email message has two components: *Headers* and the *Body*. These components are described below.

1. *Headers*

   The headers form a collection of field/value pairs. Headers contain information about the sender, receiver, and the message. Each header has a name and a value and starts in the first character of a line, followed by a colon, followed by the value. Header names and values are restricted to 7-bit ASCII characters. Non-ASCII values are represented by using the Multipurpose Internet Mail Extensions (MIME) encoding[83]. *From*, *To*, *Subject* and *Date* are the mandatory headers in the email messages. Table 2.1 lists mandatory headers as well as some others commonly used in email messages. It is important to note that these headers have common uses but are completely up to the sender (i.e. values in these header fields can be easily spoofed).

2. *Body*

   The body of the email message is the message itself, with or without a signature at the end. The header section is separated from the body by a blank line. The body text in character sets other than US-ASCII and multi-part message bodies are represented by using the MIME encoding[82].

Header	Value Field
From	contains the email address, and optionally name, of the sender of the message
To	contains the email addresses, and optionally names, of the receivers of the message
Subject	contains a brief summary about the contents of the message
Date	contains the local time and date when the message was originally sent
CC	denotes Carbon Copy
BCC	denotes Blind Carbon Copy
Received	prepended by each mailserver which has just handled a particular message
Content-Type	contains information about the message format

Table 2.1: Headers and values used in the Internet email format. *From, To, Subject* and *Date* are the mandatory headers.

2.5 Details of the Simple Mail Transfer Protocol (SMTP)

The Simple Mail Transfer Protocol (SMTP) is the standard protocol used to exchange Internet mail. We will now describe the objective, design, commands, and a typical mail transaction in the Simple Mail Transfer Protocol (SMTP).

2.5.1 SMTP - Objective and Model

SMTP is designed to transfer mail reliably and efficiently (example: by using TCP). In order to transmit an email message, a SMTP client[107] (referred to as the sender SMTP in RFC 821[106]) establishes a two-way transmission channel with a SMTP server on port 25 (referred to as the receiver SMTP in RFC 821[106]). The SMTP client has the responsibility of transferring mail messages to one or more SMTP servers which may further act as SMTP clients. It is the responsibility of the SMTP server to report failure if it unable to transfer a mail message. The SMTP model[106] is shown in Figure 2.9.

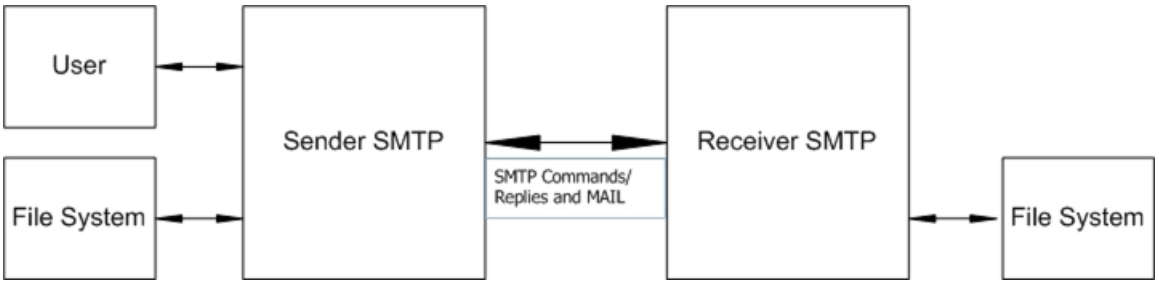


Figure 2.9: Model for SMTP usage as illustrated in RFC 821. Note that in RFC 821 terms *Sender-SMTP* and *Receiver-SMTP* are used while in RFC 2821 terms *Client-SMTP* and *Server-SMTP* are used.

2.5.2 Mail Object: *Envelope* and *Content*

SMTP transports a mail object. A mail object contains an *envelope* and *content* as described below.

- The *envelope* consists of an originator address (to which error reports *should be* directed), one or more recipient addresses, and optional protocol extension material.

- The SMTP *content* has two parts, *headers* and the *body*.

  The *headers* form a collection of field/value pairs as described in the RFC 2822[64] and the *body* is described according to MIME[82]. This is explained previously in Section 2.4.

2.5.3 Message Transfer

In SMTP, the message transfer can occur in a single connection between the original SMTP-sender and the final SMTP-recipient, or it can occur in a series of hops through intermediate systems. This implies that an SMTP server may be either:

1. *The Ultimate Destination*

2. *Intermediate Relay* (i.e. SMTP server adds trace information, acts as SMTP client, and transmits the message to another SMTP server for further relaying or for delivery)

3. *Gateway* (i.e. it may further transport the message using some protocol other than SMTP).

SMTP follows a *command response* protocol. SMTP commands are generated by the sending SMTP client and sent to the receiving SMTP server. The receiving SMTP server generates SMTP replies in response to these commands.

2.5.4 SMTP Commands

All SMTP commands begin with a command verb. SMTP commands are character strings terminated by the the space character (<SP>) if parameters follow or by the Carriage Return Line Feed (<CRLF>) character sequence.

A typical SMTP mail transaction involves several data objects. These data objects are communicated as arguments to different commands. SMTP commands are described below.

1. *Extended HELLO (EHLO) or HELLO (HELO)*

   These commands are used by the SMTP client to identify to the SMTP server. The argument field contains the fully-qualified domain name of the SMTP client if available.

2. *MAIL (MAIL)*

   The MAIL command is used to initiate a mail transaction between a SMTP client and a SMTP server. Upon successful delivery of the mail data, the SMTP server may, in turn, deliver it to one or more mailboxes or pass it on to another system (possibly using SMTP), thus acting as a relay. The argument field contains a reverse-path (sender mailbox).

3. *RECIPIENT (RCPT)*

   The RCPT command is used to identify an individual recipient of the mail data. SMTP supports specifying multiple recipients by multiple use of this command. The argument field contains a forward-path (destination mailbox).

4. *DATA (DATA)*

   The DATA command causes the mail data to be appended to the mail data buffer. The mail data is terminated by the character sequence <CRLF>.<CRLF>

5. *RESET (RSET)*

   The RSET command specifies that the current mail transaction will be aborted.

6. *VERIFY (VRFY)*

   The VRFY command asks the receiver to confirm that the argument identifies a user or a mailbox.

7. *EXPAND (EXPN)*

   The EXPN command asks the receiver to confirm that the argument identifies a mailing list, and, if true, returns the membership of the list.

   Spammers exploit VRFY and EXPN to learn about the accounts on the system. For this reason, these two commands are often disabled in the current SMTP servers.

8. *HELP (HELP)*

   The HELP command causes the server to send helpful information to the client.

9. *NOOP (NOOP)*

   The NOOP command has no effect on any parameters or previously entered commands. It specifies no action other than that the receiver sends an OK reply.

10. *QUIT (QUIT)*

    QUIT command specifies that the receiver MUST send an OK reply and then close the transmission channel.

There are restrictions on the order of usage of these commands.

2.5.5 SMTP Commands and Arguments

As mentioned previously in Section 2.5.4, a typical SMTP mail transaction involves several data objects and these data objects are communicated as arguments to different commands. For example: the reverse-path is the argument of the MAIL command, the forward-path is the argument of the RCPT command, and the mail data is the argument of the DATA command. These commands and arguments are shown in Table 2.2.

`MAIL FROM:<reverse-path>[SP <mail-parameters>] <CRLF>`
`RCPT TO :<forward-path>[SP <rcpt-parameters>] <CRLF>`
`DATA <CRLF>`

Table 2.2: SMTP commands and arguments.

2.5.6 SMTP Reply Codes

During the process of SMTP mail transaction, SMTP replies maintain synchronization of *requests-actions* (i.e. they convey state of the SMTP server to the SMTP client). Every SMTP command generates exactly one reply. All SMTP replies begin with a three-digit numeric code. Some SMTP reply codes with their meanings are described in Table 2.3.

Reply Code	**Meaning**
`220`	$< domain >$ `service ready`
`221`	$< domain >$ `service closing transmission channel`
`250`	`requested mail action okay, completed`
`354`	`start mail input; end with <CRLF>.<CRLF>`
`550`	`requested action not taken` `mailbox unavailable` (for ex: mailbox not found, no access, or command rejected for policy reasons)

Table 2.3: A partial list of SMTP reply codes with meanings.

2.5.7 SMTP Mail Transaction

In Table 2.4 we show a typical SMTP mail transaction demonstrating the usage of HELO, MAIL FROM, DATA, and QUIT commands with reply codes between a sending machine with the domain *sending-machine.net* and a receiving machine with the domain *receiving-machine.org*. Note that in the mail transaction, *Envelope Sender* is *snd@sending-machine.net*, while the email address in the claimed *From:* header is *thomas@sending-machine.net*. We will explain the difference between the two in the next section.

Sending Machine	**Receiving Machine**
	`220` receiving-machine.com `Simple Mail Transfer Service Ready`
`HELO` sending-machine.net	
	`250` receiving-machine.org
`MAIL FROM:` $< snd@sending - machine.net >$	
	`250 OK` (Sender OK)
`RCPT TO:` $< rpt@receiving - machine.org >$	
	`250 OK` (Recipient OK)
DATA	
	`354 Start mail input; end with <CRLF>.<CRLF>`
From:thomas@sending-machine.net To: shalen@receiving-machine.org Subject: Hi Hi, How are you? Thomas. .	
	`250 OK` (Message Accepted)
`QUIT`	
	`221` receiving-machine.org `Service closing Transmission Channel`

Table 2.4: SMTP mail transaction between *sending-machine.net* and *receiving-machine.org*

2.5.8 Difference between *Envelope Sender* and *From Address*

Envelope Sender is the email address provided by the sending email server in the MAIL FROM command. It is added as a *Return-Path:* header upon delivery of the email while the *From Address* is the email address listed in the *From:* line.

In the protocol flow described in Table 2.4, the *Envelope Sender* is

snd@sending-machine.org

while the email address listed in the *From:* header is

thomas@sending-machine.net.

The difference between the two as visible from the recipient's view upon delivery of the email is illustrated in Table 2.5.

Return-Path:$< snd@sending-machine.org >$ *other headers* From:thomas@sending-machine.net To: shalen@receiving-machine.com Subject: Hi Hi, How are you? Thomas. .

Table 2.5: Table showing difference between *Envelope Sender* and *From Address*. Upon delivery of the email *Envelope Sender* is added as *Return-Path:* header which is *snd@sending-machine.net* while the *From: Address* is *thomas@sending-machine.net*.

In mailing lists *From Address* is generally different from the *Envelope Sender* (the address specified in the *Return-Path:* header) in order to notify the list administrator about the bounces. The email address visible in the *From:* header is the email address of the person who posts to the list.

2.5.9 Open Mail Relays

An open mail relay is a SMTP mailserver configured in such a way that anyone (not just a local user) can send/relay/forward email through it. Spammers exploit this mechanism for sending spam. These days, a huge amount of spam is relayed, and therefore many MTAs are configured to reject email messages and connections from open mail relays.

2.6 Spam - Origin, Categories, False Claims and Exploits

We will now describe the results of a study on geographical origins of email and spam conducted by the Microsoft Corporation in 2004. We will then present the results of a study on false claims in spam conducted by the Federal Trade Commission's (FTC) Division of Marketing Practices in 2003. This is followed by the results of a study on trends in spam products and exploits analyzed through the Hotmail Feedback Loop conducted by the MSN Safety Team at Microsoft.

2.6.1 Geographical Origins of Email and Spam

Hulten etal.[60] analyzed a massive corpus of two million hand-classified email messages for the geographical origins of email and spam in 2004. This hand-classified corpus was collected through the Hotmail Feedback Loop involving Hotmail users receiving emails between April and June of 2003. In the Hotmail Feedback Loop, random Hotmail users are picked and asked to classify messages as email or spam in exchange for a financial incentive.

In their setup, each message collected through the feedback loop was tagged with the IP address of the computer that connected to Hotmail to deliver the message. The two million email messages in the corpus came from 214,000 distinct IP addresses, allocated to 157 different countries.

The geographical origins of email and spam obtained through this corpus are illustrated in Figure 2.10. Their results indicate that **30%** of the spam in their data set was **domestic**; **32% semi-domestic** (i.e. from nearby countries such as Canada, Mexico etc.) and **38% international** - implying that **70%** of the spam could be from **international locations** (and thus avoid U.S. legislation). **16%** of the spam in their dataset contained advertising for pornographic web sites.

Figure 2.10: Geographical origins of email and spam derived from a corpus of 2 million hand-classified email messages obtained through Hotmail Feedback Loop during April and June 2003. Western Europe, Japan, and New England sent more good mail than spam to Hotmail users. Asia, the Middle East, and Africa sent more spam than good mail to Hotmail users. Source - “Filtering Spam Email on a Global Scale”. Figure provided by Joshua Goodman, Microsoft Research.

2.6.2 Categories of Spam

Spam is Unsolicited Commercial Email (UCE) and, because of its very nature, keeps mutating. Spammers continue to evolve spam text and invent exploits in order to bypass spam filters.

The FTC's Division of Marketing Practices conducted an extensive study on spam in 2003. They present their results about categories and falsity in spam in their report "False Claims in Spam[49]". Their results are based on the review of a random sample of 1,000 pieces of UCE (spam) drawn from a pool of over 11,000,000 spam messages. The random sample of 1,000 spam messages was derived from three sources—the *UCE Database*, which consists of spam forwarded to the FTC by members of the public (450 random samples chosen from a total of 11,184,139 messages); the *Harvest Database*, which consists of messages received by undercover FTC email boxes seeded on Internet web pages and chat rooms (450 sample messages of 3651 messages); and spam received in official FTC inboxes (100 messages).

Their analysis of random spam text categorizes spam in eight general categories: *Investment/Business Opportunity, Adult, Finance, Products/Services, Health, Computers/Internet, Leisure/Travel, Education* and *Other*. This is described in Table 2.6 and their relative occurrence is illustrated in Figure 2.11.

Type of Offer	Description
Investment/Business Opportunity	work-at-home, franchise, chain letters, etc.
Adult	pornography, dating services, etc.
Finance	credit cards, refinancing, insurance, foreign money offers, etc.
Products/Services	products and services, other than those coded with greater specificity etc.
Health	dietary supplements, disease prevention, organ enlargement, etc.
Computers/Internet	web hosting, domain name registration, email marketing, etc.
Leisure/Travel	vacation properties, etc
Education	diplomas, job training, etc.
Other	catch-all for types of offers not captured by specific categories listed above.

Table 2.6: Types of offers made via spam in a random sample of 1000 spam messages. Source:"False Claims In Spam," FTC Division of Marketing Practices (April 2003).

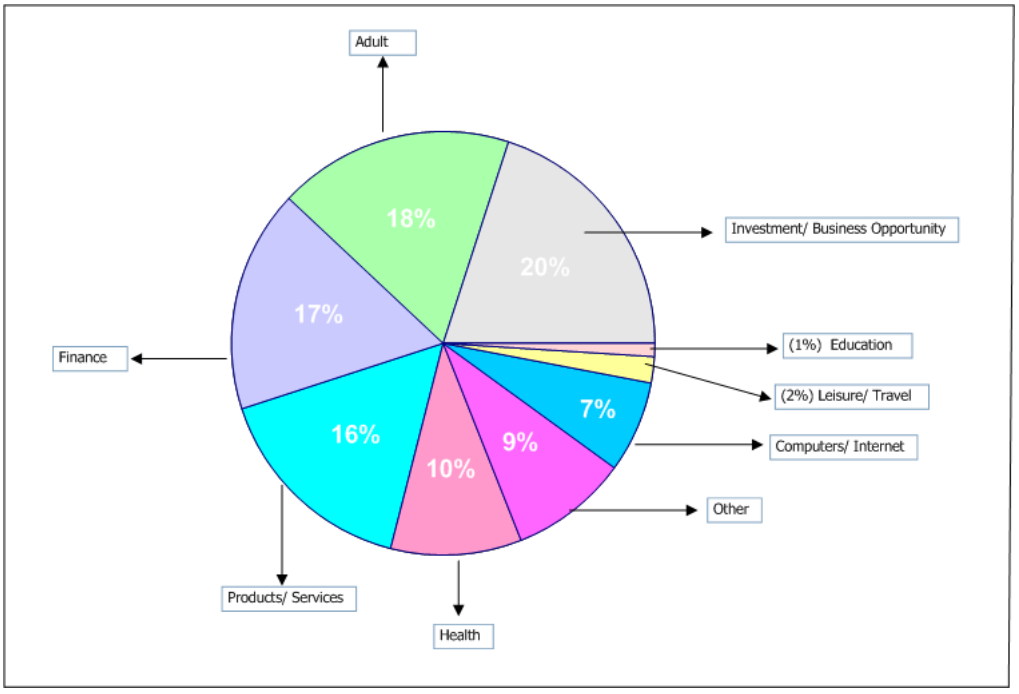


Figure 2.11: Types of offers made via spam in a random sample of 1000 spam messages. Source:"False Claims In Spam," FTC Division of Marketing Practices (April 2003).

2.6.3 False Claims in Spam

Spam is characterized by false information in the *header* and *body*. For example, "From" lines often obscure the true identity of the sender; "Subject" lines are often blank, connote a relationship (business or personal), or suggest that the message is in reply to a message previously sent by the recipient (for example, by including *Re:* in the Subject line). Out of 1000 random sample messages, FTC's analysis of *falsity* in *From* and *Subject* lines is presented in Table 2.7 and Table 2.8, respectively. Their relative occurrence is illustrated in Figure 2.12, Figure 2.13, Figure 2.14, and Figure 2.15, respectively. Relative occurrence of spam containing the *"ADV"* label required by several state laws is illustrated in Figure 2.16.

Type of Falsity in the "From" Line	**Description**
Blank	sender's identity has been stripped from the "From" line
Connotes Business Relationship	name of sender suggests a business relationship between sender and recipient (ex: "youraccount@vendorxyz.com")
Connotes Personal Relationship	name of sender suggests a personal relationship between sender and recipient (ex: use of a first name only, which may suggest that the message is from someone in the recipient's address book.)
Message from Recipient	sender's identifying information has been stripped from message and replaced with recipient's email address
Disguised in Other way	catch-all for other methods used to disguise the sender's true email address (ex: sender, as identified in the message text, uses another person or entity's name or email address in the "From" line)

Table 2.7: Falsity in "From" line in a random sample of 1000 spam messages. Source: "False Claims In Spam," FTC Division of Marketing Practices (April 2003).

Type of "Subject" Line Falsity	Description
Blank	contains no information about the subject of the message
Connotes Business Relationship	suggests existence of business relationship between sender and recipient (ex: "your order status")
Connotes Personal Relationship	suggests existence of personal relationship between sender and recipient (ex: "Bob says 'hi' ");
Unrelated to Content	content of message differs from description in Message "Subject" line
Re:	suggests that the message is in reply to a message previously sent by recipient
Other	catch-all for other methods used to disguise the true content of the message (ex: "Subject" line indicates that the message is "extremely urgent")

Table 2.8: Falsity in "Subject" line in a random sample of 1000 spam messages. Source: "False Claims In Spam," FTC Division of Marketing Practices (April 2003).

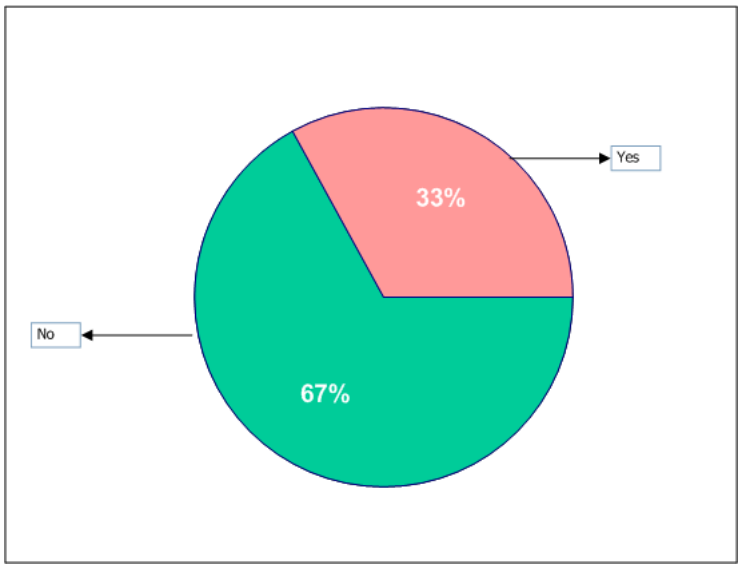


Figure 2.12: Percentage of spam with false "From" line in a random sample of 1000 spam messages. 33% percent of spam analyzed contained false information in the "From" line. Source: "False Claims In Spam," FTC Division of Marketing Practices April 2003.

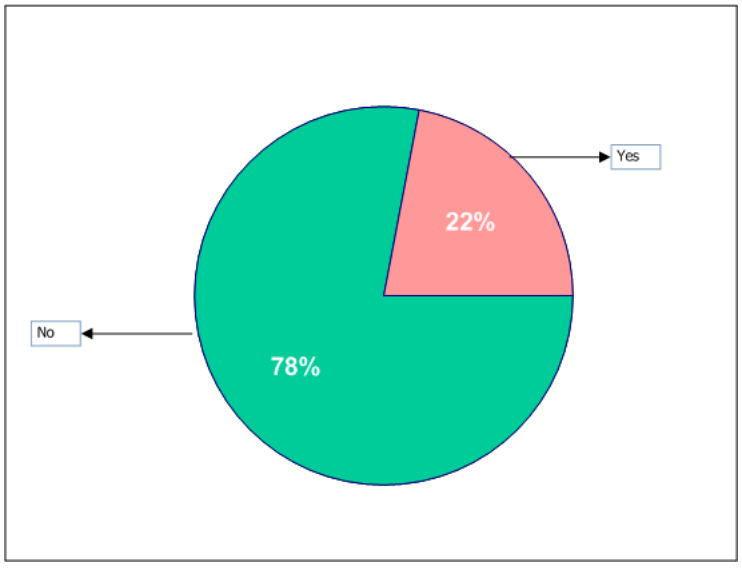


Figure 2.13: Percentage of spam with false "Subject" line in a random sample of 1000 spam messages. 22% percent of spam analyzed contained false information in the "Subject" line. Source: "False Claims In Spam," FTC Division of Marketing Practices April 2003.

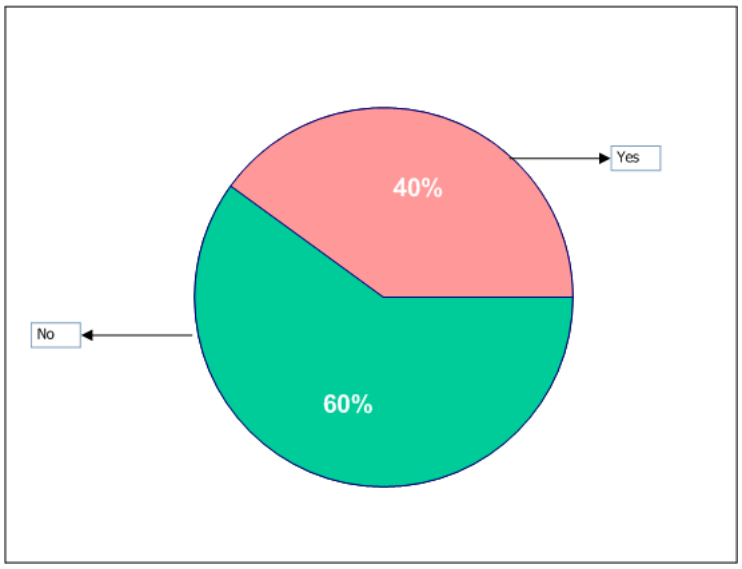


Figure 2.14: Percentage of spam with false “Text” line in a random sample of 1000 spam messages. 40% percent of spam analyzed contained false information in the body of the message. Source: “False Claims In Spam,” FTC Division of Marketing Practices (April 2003).

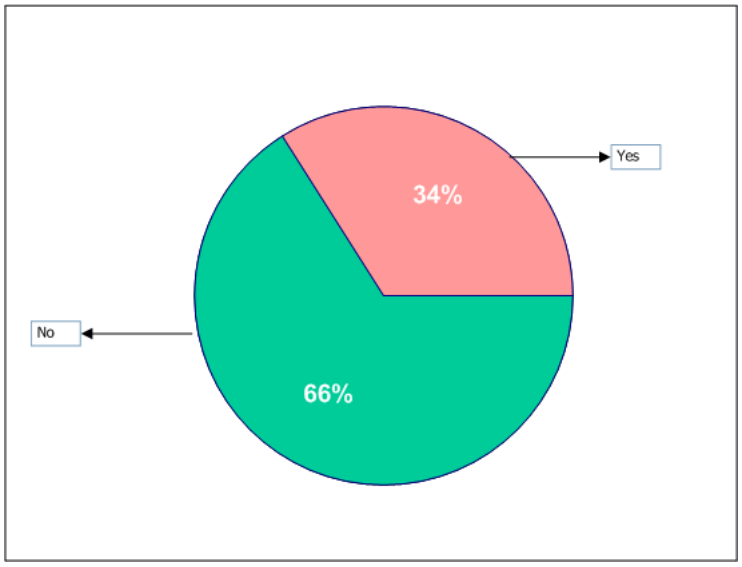


Figure 2.15: 66% of spam in a random sample of 1000 messages contained false information in “From” lines, “Subject” lines or “Message text”. Source: “False Claims In Spam,” FTC Division of Marketing Practices (April 2003).

+

2.6.4 Trends in Spam Products and Exploits

The MSN Safety Team at Microsoft Corporation conducted a study of trends in spam products and exploits from the corpus obtained through the Hotmail Feedback Loop in 2003 and 2004[61]. Their analysis confirms that spammers continue to evolve spam by tricks such as *word obscuring, url obscuring, domain spoofing, token breaking* and *character encoding*, etc.

In their setup they randomly sample 1,000 spam messages that arrived at Hotmail between 2/1/2004 and 3/1/2004 and call it *2004 spam*. They sample 200 messages that arrived at Hotmail between 3/15/2003 and 4/15/2003 and call it *2003 spam*. A comparison of trends in categories and exploits in spam messages between *2003 spam* sample and *2004 spam* sample from their study is reported in Table 2.9, Table 2.10, and Table 2.11, respectively.

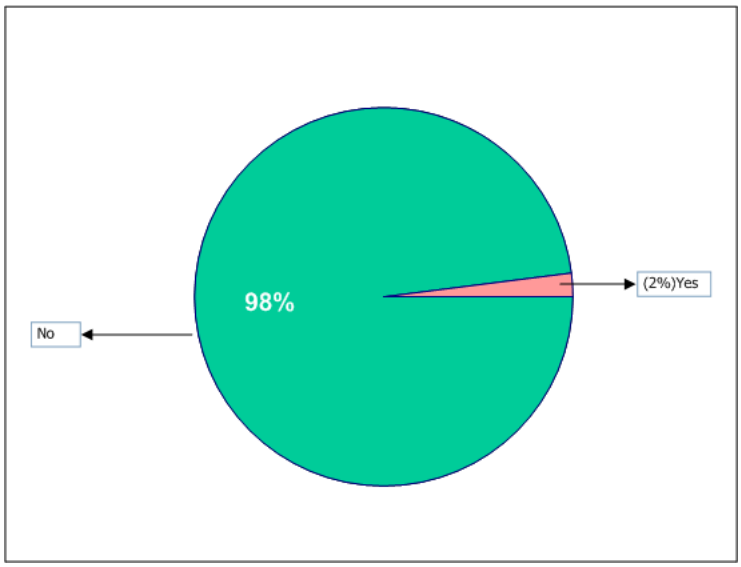


Figure 2.16: 2% of spam in a random sample of 1000 messages contained the “ADV” label in the subject line, which is required by several state laws. Source: “False Claims In Spam,” FTC Division of Marketing Practices (April 2003).

Product	**2003 Spam (%)**	**2004 Spam (%)**	**Description**
Porn/Sex Non-Graphic	17	34	enhancers with sexual connotation, links to porn
Insurance	1	4	health, dental, life, home auto insurance
Rx/Herbal	8	10	cheap drugs or herbal supplements
Financial	12	13	refinancing, get out - - of debt, financial advice
Travel/ Casino	2	3	selling airlines tickets, hotel reservations, rental car Internet casino and other gaming sites
Scams	8	6	get rich quick, Phisher scams etc.
Newsletters	9	6	any newsletter that is not selling something
Other Spam	13	8	everything else that appears to be spam
Porn/Sex Graphic	13	7	anything that contains pornographic images
Dubious Products	20	10	pirated software diplomas etc.

Table 2.9: Trends in categories of spam in random 2004 and 2003 spam samples. Source: “Trends in Spam Products and Methods,” Microsoft Corporation.

Exploit	2003 Spam (%)	2004 Spam (%)	Description
Word Obscuring	4	20	misspelling words, Inserting words in images etc.
URL Spamming	0	10	adding URLs to non-spam sites (ex: msn.com)
Domain Spoofing	41	50	using an invalid or fake domain in the from line
Token Breaking	7	15	breaking words with punctuation, space etc.
MIME Attacks	5	11	inserting non-spam content in one body part and spam content in another
Text Chaff	52	56	random strings of characters, random series or words, or unrelated sentences
URL Obscuring	22	17	encoding a URL in in hex, hiding the URL with an @ sign etc.
Character Encoding	5	0	pharmacy renders into pharmacy

Table 2.10: Trends in exploits in spam in random 2004 and 2003 spam samples. Source: "Trends in Spam Products and Methods," Microsoft Corporation.

Category	2003 Spam	2004 Spam
Scams	1.07	1.89
Financial	1.26	1.88
Porn/Sex Non-Graphic	1.85	2.44
Travel/ Casino	0.75	0.98
Other Spam	0.76	0.79
Insurance	1.50	1.52
Newsletters	0.00	0.00
Rx/Herbal	2.13	2.12
Dubious Products	1.20	1.15
Porn/Sex Graphic	2.16	1.33

Table 2.11: Trends in number of exploits per message in random 2004 and 2003 spam samples. Source: "Trends in Spam Products and Methods,"Microsoft Corporation.

2.7 Phishing - Attack Taxonomy, Lifecycle and Anatomy

The objective of phishing fraud is to acquire sensitive information such as passwords and credit card details. Phishing is a dangerous threat and causes billions of dollars' worth of losses to businesses. We now present phishing attack taxonomy, the lifecycle, and the anatomy of a phishing email.

2.7.1 Phishing Attack Taxonomy and Lifecycle

To tackle the phishing problem, the Financial Services Technology Consortium (FSTC) recently developed a taxonomy of phishing attacks[125]. The phishing attack lifecycle can be decomposed in *Planning, Setup, Attack, Collection, Fraud* and *Post-Attack Actions*. Each of these malicious stages spawn a series of malignant steps. The phishing attack cycle is illustrated in Figure 2.17.

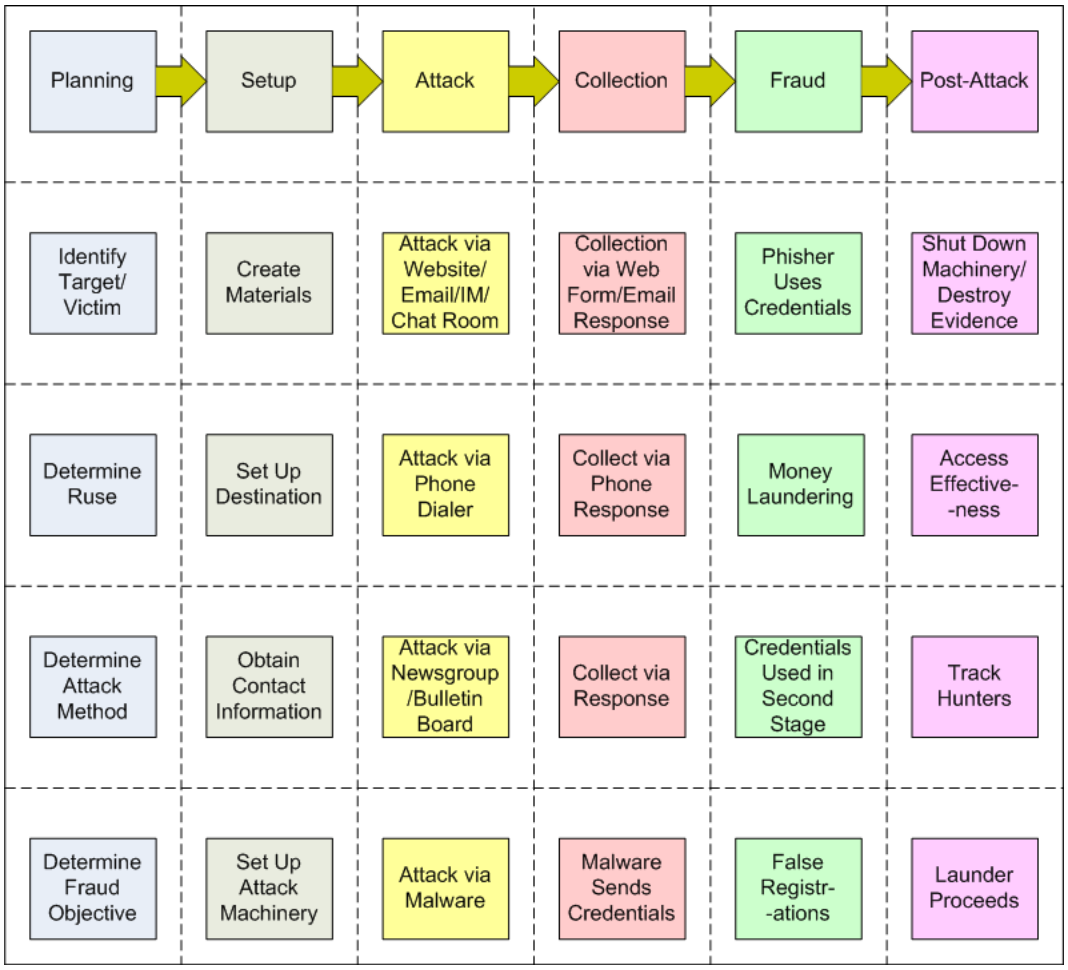


Figure 2.17: Phishing attack taxonomy and lifecycle. Source: "Tackling Phishing" by Rebecca Wetzel, Business Communications Review, Feb 2005. Figure redrawn with inputs from Rebecca Wetzel.

2.7.2 Anatomy of a Phishing Email

John Graham-Cumming has very elegantly presented tricks exploited by spammers in his paper “The Spammers Compendium”[55]. Drake etal. from MailFroniter[75] have extensively described key features present in phishing emails and tricks used in fraudulent web sites in their paper “The Anatomy of a Phishing Email”[43]. Below is their analysis of phishing emails and fraudulent web sites from the phishing samples forwarded to MailFrontier from its customers. We use phishing samples from their paper.

1. *An Attempt to Spoof Reputable Companies Using Convincing Strategies*

   Phishers try to spoof or emulate reputable companies like eBay, PayPal, Citibank etc. They often use the following tricks:

   - *Use of the legitimate company’s image, logo, icon and similar font schemes*
   - *Links to the legitimate websites/services*
   - *Spoofing headers to provide a look-a-like appearance of reputable companies such as @eBay.com, @paypal.com etc.*

2. *Claimed Sender has a Reputable Name but a Different Reply Address*

   In some cases, the email claims to be from a legitimate company but has a fraudulent value for the “Reply-To” header. This is illustrated in Table 2.12:

```
From :  Citibank
Reply To:  citibank3741@collegeclub.com
```

Table 2.12: Different reply address and claimed sender found in a phishing sample. Source: "Anatomy of a Phishing Email," MailFrontier, Inc.

3. *Offering Valuable Proposition*

   Such fraudulent emails try to convince recipients to reveal sensitive information by claiming that the recipient's account information is outdated, that a credit card has expired etc.

4. *Reflect Urgency Demanding Quick Response*

   Scammers try to convince recipients to respond quickly. An example of an urgent request is presented in Table 2.13:

```
If you don't respond within 24 hours after receiving
this email your account will be deactivated and
removed from the server
```

Table 2.13: A statement demanding quick response found in a phishing sample. Source: "Anatomy of a Phishing Email," MailFrontier, Inc.

5. *Assurance of Secure Transaction*

   Phishers try to gain the trust of recipients by assuring that the transaction is secure. An example is illustrated in Table 2.14. Phishers have also demonstrated the use of Secure Socket Layer (SSL) by using the `https` protocol. Unfortunately, fraudulent emails also use the `TRUSTe` symbol[121] frequently. `TRUSTe` is an independent non-profit organization that enables trust based on privacy for personal information on the Internet.

Your information is submitted via a secure server

Table 2.14: A statement assuring security found in a phishing sample. Source: "Anatomy of a Phishing Email," MailFrontier, Inc.

6. *Use of HTML Forms in the Email*

   Phishers have also used HTML forms in the body of the email message to collect information from the recipients. One such example from [43] in what appeared to be an email from eBay is presented in Table 2.15.

$<$ FORM action $=$ http : //www.christmas $-$ offer.com/ sendmail.php method $=$ get target $=$ _blank $>$

Table 2.15: Use of HTML forms in emails found in a phishing sample. Source: "Anatomy of a Phishing Email," MailFrontier, Inc.

7. *Look-A-Like Domain Names*

   Fraudsters register domain names that are very similar to the domain names of reputable companies. A fraudulent email from eBay used the link shown in the Table 2.16. Fraudsters are also early adopters of the email authentication standards. A recent study conducted by MarkMonitor revealed that more than 90% of the Fortune 100 companies have look-a-like domains with a published Sender Policy Framework (SPF) record for the email authentication. This study is presented in the next chapter.

http : //ebay $-$ securitycheck.easy.dk3.com

Table 2.16: A phishing email using a look-a-like domain name as eBay. Source: "Anatomy of a Phishing Email," MailFrontier, Inc.

8. *Mismatch between Link Text in the Email and the actual Link Destination*

   In phishing emails, the link text seen in the email is usually different from the actual link destination. In the example below, the email that appears to send the user to *http://account.earthlink.com* instead takes the user to *http://memberupdating.com.* The HTML code is presented in Table 2.17.

$<$ a class $=$ "m1" target $=$ "_blank"title $=$ "Update" href $=$ "http : //www.memberupdating.com" $>$ http : //account.earthlink.com $<$ /a $>$

Table 2.17: Different link text and link destination found in a phishing email. Source: "Anatomy of a Phishing Email," MailFrontier, Inc.

9. *Using JavaScript Event Handlers to Display False Information*

   Some fraudsters use the JavaScript event handler "OnMouseOver" to show a false URL in the status bar of the user's email application. Drake etal.[43] present an example of a fraudulent PayPal email in which, when the user puts the mouse over the link, the status bar will display:

   https://www.paypal.com/cgi-bin/webscr?cmd=_login-run

   However, the link actually takes the user to:

   http://leasurelandscapes.com/snow/webscr.dll

10. *Obscuring the URL Using an IP Address OR Hexadecimal Character Codes*

    Phishers try to hide the destination web site by obscuring the URL. They also use IP addresses in decimal, octal, hexadecimal format to obscure the URL. They often use hexadecimal character codes to represent the numbers in the IP address.

11. *Using a Reputable Entity or Word along with @ Symbol in a URL to Confuse Recipients*

    An Internet email address is of the form *username@hostaddress*. Whenever a *http* or *https* request with the above format is made, the part before the @ symbol is ignored and the browser is directed to the *hostaddress*. Spammers often use this format and reputable names before the @ character to trick recipients into thinking that the fetched destination web site belongs to the reputable business. Sometimes, fraudsters also substitute @ with %40 (the hexadecimal character representation of the @ character).

    In the example presented in Figure 2.18, the link appears to be going to eBay however the text before the @ symbol is ignored and the link sends the user to *"210.93.131.250"* which is the fraudulent web site's IP address.

http : //cgi1.ebay.com.aw − cgiebayISAPI.dll %00@210.93.131.250/my/index.htm

Table 2.18: URL obscuring with IP address and hexadecimal characters found in a phishing email. Source: "Anatomy of a Phishing Email," MailFrontier, Inc.

12. *Abusing Redirection Services*

    Phishers try to obscure URL by using redirection services *for example, cjb.net, tinyurl.com* to hide their URLs.

13. *Using Ports Other than 80*

    Fraudsters occasionally use ports other than 80 (http) to hide their location. A port can be specified with a colon (:) and a port number following the URL.

2.7.3 Tricks Used in Fraudulent Web Sites

Once the naïve users are on fraudulent web sites, fraudsters then use various other tricks to continue emulating the legitimate reputable business. Some of these tricks used in these sites are described below.

1. *Continuous Attempts to Emulate the Legitimate Reputable Company*

   Phishers continue to emulate the legitimate company by using their images, logos, fonts and color schemes.

2. *Use of Secure Socket Layer (SSL) Certificates*

   As aforementioned, phishers have demonstrated the use of Secure Socket Layer (SSL) protocol. These phishing sites often have URL beginning with “https://” to indicate that information is being transmitted over a secure channel and that the company has obtained a Secure Sockets Layer (SSL) certificate from a certifying authority

3. *Deceptive Information Processing Pages:*

   Phishing sites collect sensitive information from the users via forms and then display deceptive information processing pages such as thanking the recipient for providing the information, often redirecting them to the legitimate company’s web site in the end.

4. *Other Tricks:*

   Scammers and phishers often make use of various other tricks such as displaying a fake address bar, using pop-ups, and disabling the right-click button of a mouse to prevent the user from viewing and saving source code. They have also demonstrated the use of trojan horses and viruses for phishing.

2.8 Spam and the Law

Several legal initiatives are underway for tackling spam and phishing. In December 2003, President George W. Bush signed legislation to help fight spam. The bill is known as **Controlling the Assault of Non-Solicited Pornography and Marketing Act of 2003** (CAN-SPAM Act of 2003). Microsoft, Yahoo!, American Online, and Earthlink have filed lawsuits against individuals and companies who have spammed their customers. A Virginia court recently sentenced a prolific spammer, Jeremy Jaynes, to nine years in prison[15]; a Nigerian court sentenced a woman to two and a half years for phishing[16]. Michigan and Utah have both passed laws creating "do-not-contact" registries in July/August 2005, covering email addresses, instant messaging addresses and telephone numbers[86]. Microsoft Corporation recently won a seven-million dollar settlement from spammer Scott Richter[17].

The CAN-SPAM Act of 2003 is described in Section 2.8.1, and Jeremy Jaynes day-to-day action transcripts recovered at the time of his arrest (and then redacted by the Court) are presented in Section 2.8.2.

2.8.1 CAN-SPAM Act of 2003

CAN-SPAM defines spam as “any electronic mail message the primary purpose of which is the commercial advertisement or promotion of a commercial product or service (including content on an Internet website operated for a commercial purpose).”

The bill permits sending Unsolicited Commercial Email (UCE) provided it satisfies all of the requirements listed below.

1. It contains an opt-out mechanism.

2. It contains a valid subject line and header (routing) information.

3. It contains the legitimate physical address of the mailer.

4. It contains a label if the content is adult.

If a user opts out, the sender must remove his address within ten days.

So far, the CAN-SPAM Act has had a little or no effect on the presence of spam on the Internet.

2.8.2 Jeremy Jaynes Sentence

Jeremy Jaynes, one of world's most prolific spammer, was recently sentenced to nine years in prison for spamming by a Virginia Court. Jaynes was sending bulk emails with forged headers, which is a crime under Virginia statute[15]. Jaynes's day-to-day plan of action is presented in Figure 2.18, Figure2.19, and Figure2.20. These notes were recovered during his arrest and then redacted by the Court. A close look at Figure 2.18 confirms that spammers try to change their identities frequently and continue to evolve tricks in spam to fool spam filters. Note: "*get tons of ips, figure out filtering,*" etc.

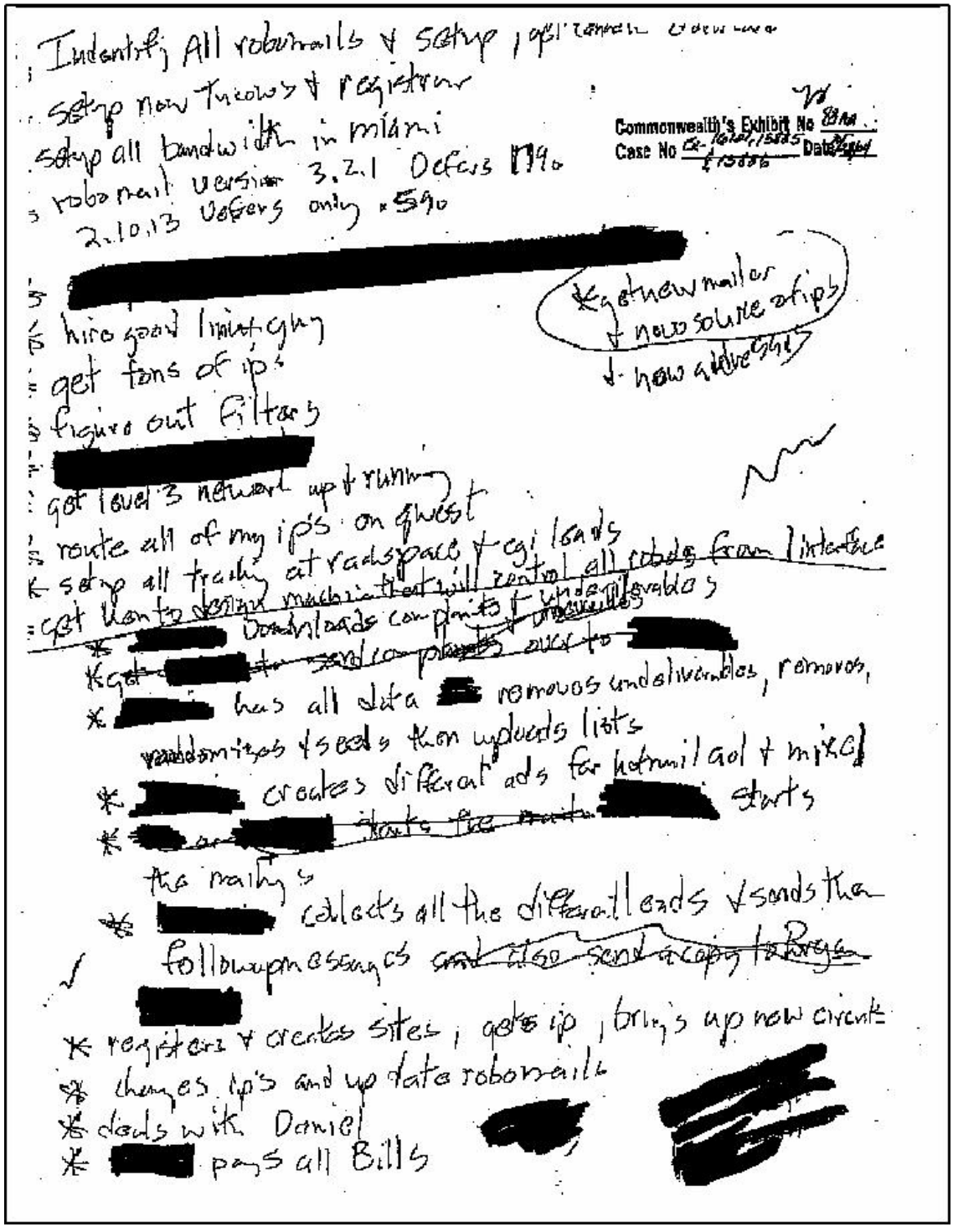


Figure 2.18: Jeremy Jaynes *to do* list 1 recovered during his arrest and then redacted by the court. Note: *get tons of ips, figure out filtering, change ips and update robomails*.

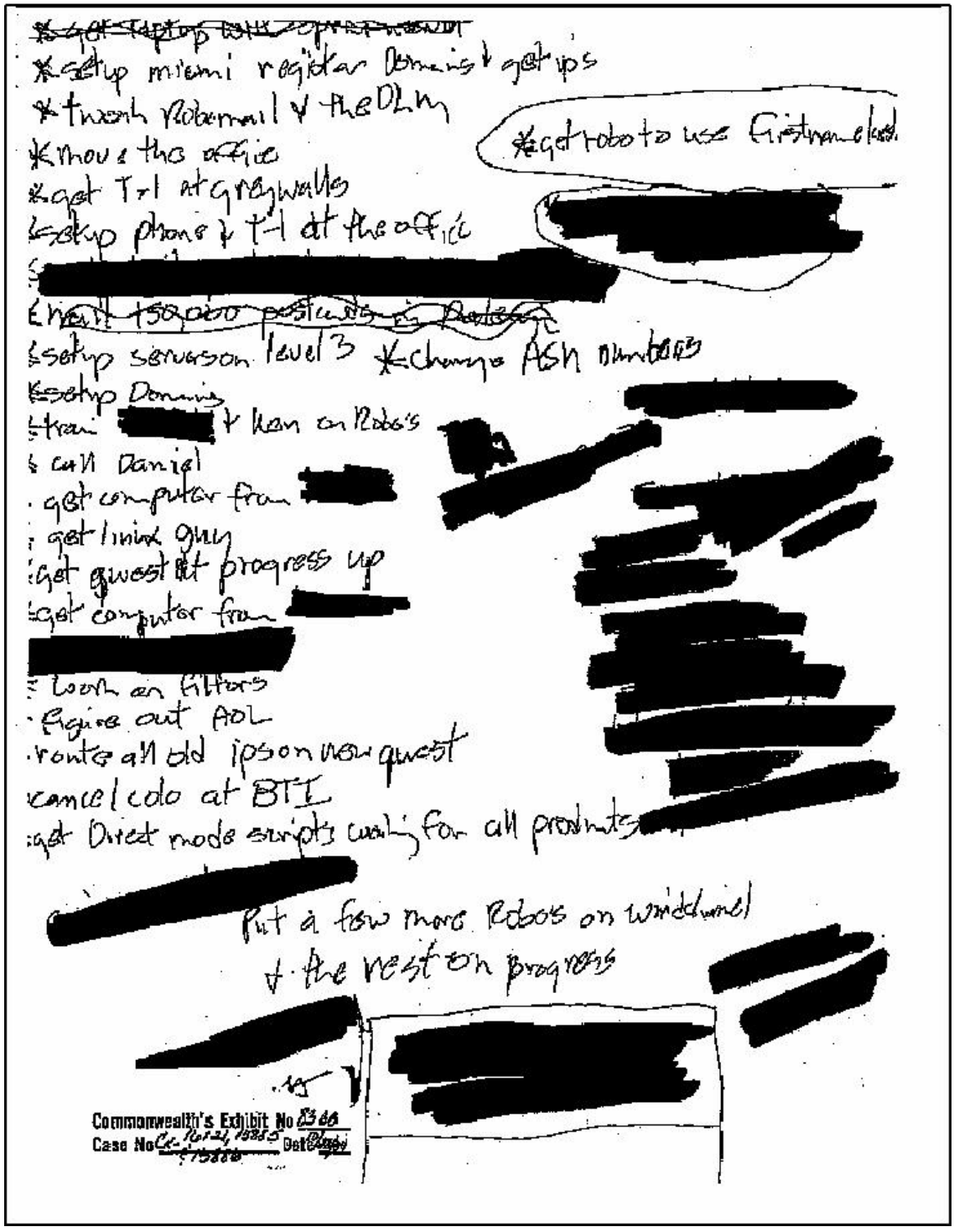


Figure 2.19: Jeremy Jaynes *to do* list 2 recovered during his arrest and then redacted by the court. Note: *figure out AOL*.

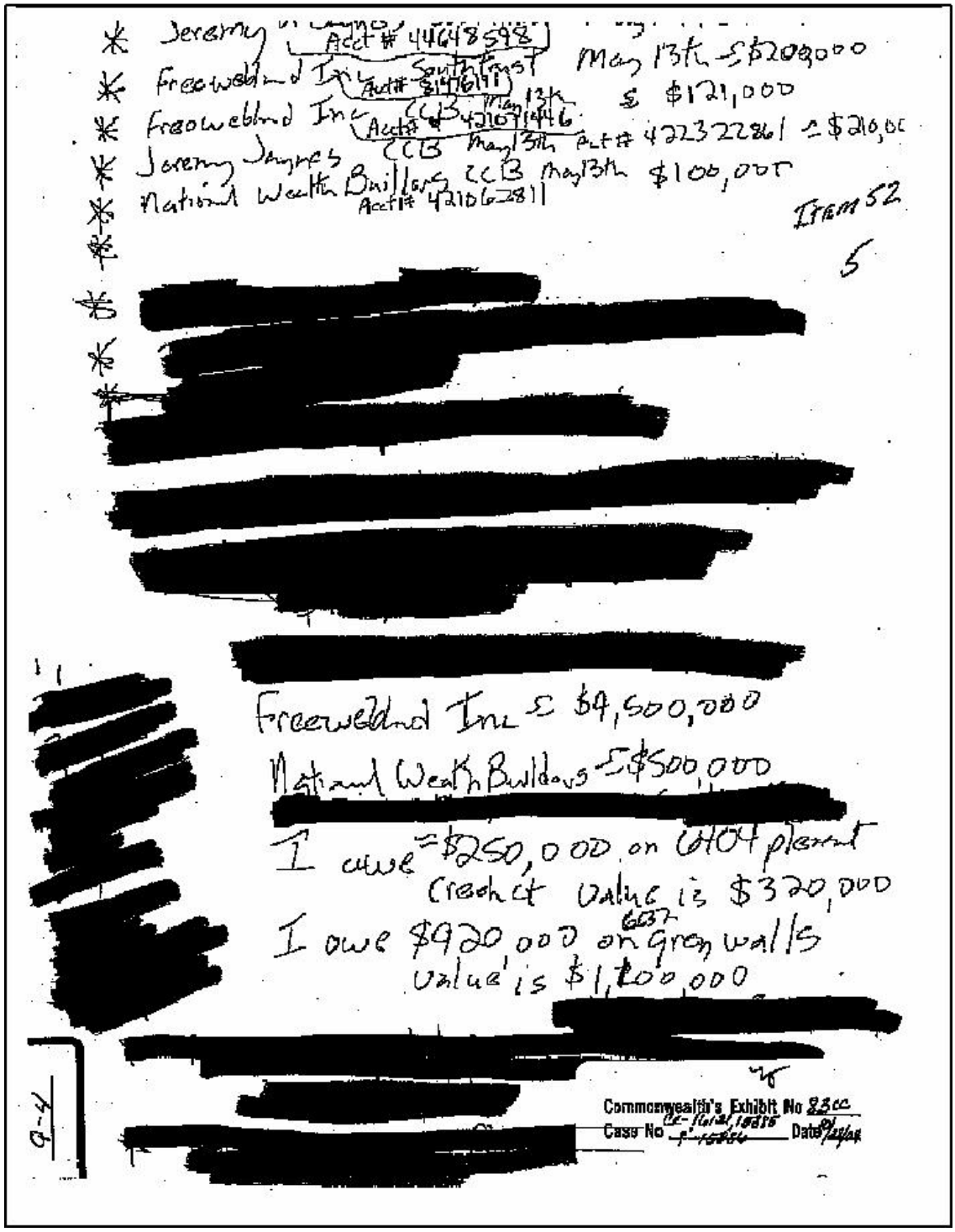


Figure 2.20: Jeremy Jaynes *to do* list 3 recovered during his arrest and then redacted by the court. Note: *$ involved*.

2.9 People and Spam

John Graham-Cumming conducted a spam survey on Slashdot in 2004[56] to understand the effect of spam on people. His survey had 4,691 participants, of which 94.5% were men, 4% were women and 1.5% were other. 50% of the participants were email users with more than five years' experience. Two respondents identified themselves as spammers. 80% of the respondents were between the ages of 18 and 45.

An analysis from the responses to the survey indicated that 76% participants believe that the spam problem will never go away, 9% believe that spam-filtering makes the spam problem worse, and, on an average 85% are using an anti-spam tool. 98.5% users receive spam. 1% of respondents claimed that they have bought from spam. The average user (both men and women) spends around 9 minutes a day dealing with spam. Figure 2.21 illustrates the effect of spam on the respondents of the Slashdot spam survey.

2.10 Spam Survey at University of California, Riverside

A spam survey was conducted at the University of California, Riverside (UCR), in May 2005 to understand the impact of spam on the university population. We had 1,721 mixed responses from students, staff, and professors of all age groups and genders. The results of the survey are unavailable at the time of publishing this thesis.

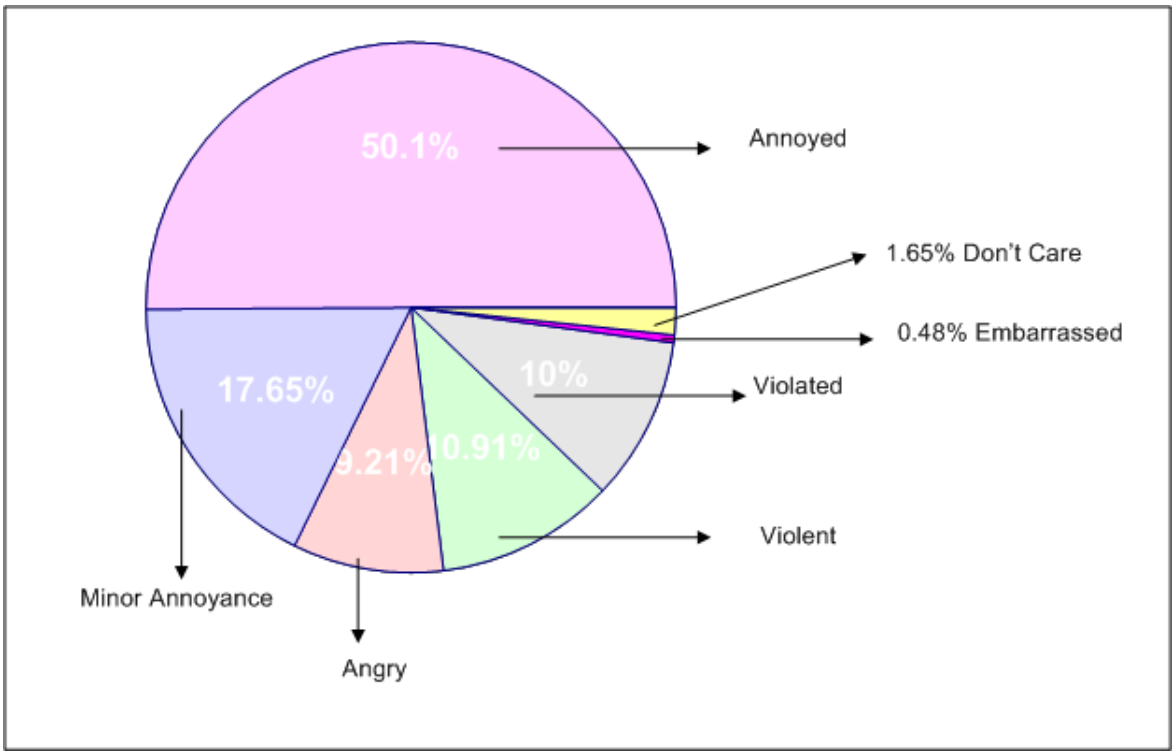


Figure 2.21: Effect of spam on people derived through responses of the spam survey conducted by John Graham-Cumming on Slashdot in 2004. Figure redrawn with permission from John Graham-Cumming.

Chapter 3

Related Work

Current technical initiatives to fight spam and phishing include server and client-side spam filtering, using lists (blacklist, whitelist, greylist), email authentication standards (Identified Internet Mail (IIM), DomainKeys (DK), Domain Keys Identified Mail (DKIM), Sender Policy Framework (SPF), Sender ID Framework), and emerging sender reputation and accreditation services. Figure 3.1 illustrates current technical initiatives for fighting spam and phishing. We will now describe each of these technical initiatives.

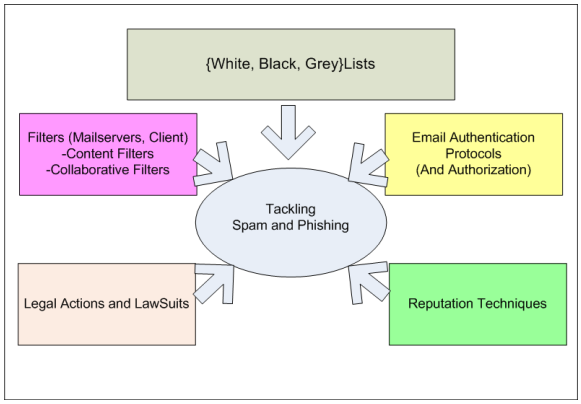


Figure 3.1: Current technical initiatives for tackling spam and phishing.

3.1 Whitelist, Blacklist and Greylist

3.1.1 Whitelist

A *whitelist* in the context of spam-filtering is a list of trusted senders (i.e. email received from the addresses in the *whitelist* are always delivered to the recipient's email inbox). The person maintaining the *whitelist* may have initiated communication with whitelisted senders in the past. All modern Mail Transfer Agents (MTAs) support whitelisting technique. It is a common practice to mirror one's address book as the *whitelist*.

3.1.2 Blacklist

A *blacklist* is the exact opposite of a whitelist. In the context of spam-filtering, emails received from the blacklisted addresses are always filtered out. Modern MTAs support Domain Name System Blackhole List (DNSBL). A DNSBL is a real-time database of IP addresses of spam sources and relays. The first DNSBL was the Real-Time Blackhole List (RBL) created by Paul Vixie for the Mail Abuse Prevention System (MAPS)[73]. The Spamhaus Block List (SBL) (a real time database of IP addresses of spam sources) and Spamhaus Exploits Block List (XBL) (a real-time database of IP addresses of worms/viruses/trojan exploits) are examples of RBLs from Spamhaus[119].

When a mailserver implementing DNS blacklist filtering receives a connection from a sender, it queries the DNSBL server for the client IP address. If the sender is listed in DNSBL, the mailserver can reject or flag messages from that client.

The DNSBL system is built on top of the Domain Name System (DNS). In the DNS, the address record *A* assigns an IP address to a domain name, *MX* record provides mail exchanger for that domain and *PTR* record (reverse record) associates an IP address with a canonical name. This is done by reversing the four IP octets followed by *IN-ADDR.ARPA*. For example for IP address *192.168.0.1*, the *PTR* record is *1.0.168.192.IN-ADDR.ARPA*.

The DNSBL lookup is similar to the reverse-DNS lookup, except that in the DNSBL system, *A* record is looked up and the forward domain is used, while in reverse-DNS, PTR record is looked up and the reverse domain *IN-ADDR.ARPA* is used.

The DNSBL query and result format is explained below.

A mailserver utilizing DNSBL service, say from *sbl.spamhaus.org*, receives a connection from a client with IP address *a.b.c.d*. The mailserver reverses the octets and appends the DNSBL's domain name, yielding *d.c.b.a.sbl.spamhaus.org*. This name is then looked up in the DNS as a domain name *A* record. If the client is listed, an IP address is returned; otherwise a code indicating no such domain is returned. DNSBL servers publish information about reasons for a client's listing in the DNS *TXT* record.

The DNSBL query to *sbl.spamhaus.org* zone returns *127.0.0.2* for sources of verified spam services, and queries to *xbl.spamhaus.org* zone return *127.0.0.4-6* for sources of worms and trojan exploits[119].

3.1.3 Greylist

A *greylist* is a cross between *whitelist* and *blacklist*[58]. A MTA implementing *greylist* is configured to *temporarily reject* an email if it has not encountered any of the items previously listed in a triplet (items in the triplet can be matched against the user's *whitelist*). The triplet contains:

1. The IP address of the connecting host.

2. The envelope sender address.

3. The envelope recipient address.

Greylisting is based on the assumption that the spammer's bulk mailing tools do not incorporate a mechanism for handling temporary bounces, while a legitimate mailserver will try to reconnect and make an attempt to deliver the email.

Greylisting risks delaying all unknown email and not just spam. Poorly configured legitimate mailservers can transform temporary rejects into permanent bounces, leading to rejection of legitimate mails.

3.2 Email Authentication

Authentication is a mechanism used to ensure that the person is the one he or she claims to be. Authorization is an access control mechanism. It is the process of granting privileges or permissions to entities to perform certain actions. Authentication and authorization are related in the sense that authenticated entities are authorized to execute actions in their scope.

In the context of Internet email, authentication refers to a mechanism by which the recipient of the email can authenticate the identity of the sender; Authorization refers to an access control mechanism granted to individuals by the domain authority on some criteria. For example, only authorized senders can be allowed to send email from that domain.

The biggest weakness with the SMTP is that the senders address can be easily spoofed. To prevent this, the email industry is currently devising protocols for email authentication.

Meng Weng Wong of Pobox[96] has very elegantly explained current email authentication proposals in his paper “Sender Authentication: What To Do”[101].

The current email authentication proposals can be broadly divided into two categories listed below.

1. *IP Based Authentication* such as SPF from Pobox[103] and Sender ID Framework (SIDF) from Microsoft Corporation[102].

2. *Cryptographic Based Authentication* such as Identified Internet Mail (IIM) from Cisco Systems[62] and DomainKeys (DK) from Yahoo![41].

We will now explain SPF followed by the Sender ID Framework (SIDF). This is followed by the description of the Identified Internet Mail (IIM) from Cisco Systems and the DomainKeys (DK) proposal from Yahoo!. IIM and DK have recently merged to form the Domain Keys Identified Mail Standard (DKIM)[42].

3.2.1 Sender Policy Framework (SPF)

In the SPF proposal, mailserver at the receiving side of the SMTP connection validates the IP address of the connecting client machine (specified in the MAIL FROM command) against the SPF record for that domain using the DNS. SPF records are published using TXT resource record of the the DNS.

For example, the SPF record for the domain *foobar.example.com* specified as *v=spf1 mx* implies that the mail exchanger servers (mx) for the *foobar.example.com* domain are explicitly permitted to send email originating from that domain.

The SPF record entry in the domain file for the domain *foobar.example.com* looks like[103]:

foo.example.com IN TXT "v=spf1 ..."

3.2.2 Sender ID Framework (SIDF) from Microsoft Corporation

Sender ID Framework is the based upon the Caller ID proposal[22] from Microsoft and SPF from Meng Weng Wong of Pobox. In SPF described above, the *Envelope Sender* address (i.e. the address mentioned in the MAIL FROM command) is verified. Sender ID proposal is based on the Sender Policy Framework (SPF) and uses Purported Responsible Address (PRA) instead of *Envelope Sender* address. The PRA as defined in the proposal "Purported Responsible Address in E-Mail Messages" by Lyon is the entity that, according to the headers, most recently caused the message to be delivered[99].

For example, a Sender ID-compliant MTA, upon delivery of an email to the recipient, would display in the headers field:

From PRA on behalf of From

This indicates that the PRA is the most recent sender in the current communication.

The Sender ID Framework (SIDF) is illustrated in Figure 3.2 and described below.

1. *Objective*

   The objective of the Sender ID Framework is to provide the recipient system with a mechanism to verify whether the email originated from the claimed sender domain. This is done by validating the IP address of the sender against the mailservers allowed to send email on behalf of that particular domain as published in its domain file.

2. *Publishing SPF Records in DNS*

   Domain owners/administrators publish SPF records as TXT records in the DNS to identify the email servers authorized to send email originating from that domain.

3. *Sender Side*

   The Sender dispatches email destined for the mailserver of the recipient system. This is shown in step 1 in Figure 3.2.

4. *Verification at the Receiving Side*

   Upon receiving the email, the Sender-ID enabled MTA at the recipient side checks the claimed sender domain and then looks up the SPF record for that domain using the DNS. This is shown in steps 3,4,5 and 6 in Figure 3.2.

   If the IP address of the sender matches any of the mailservers published in the SPF Record, the message is considered to be authenticated and delivered to the receiver. Authentication failure leads to non-delivery of the email. This is shown in step 7 in Figure 3.2.

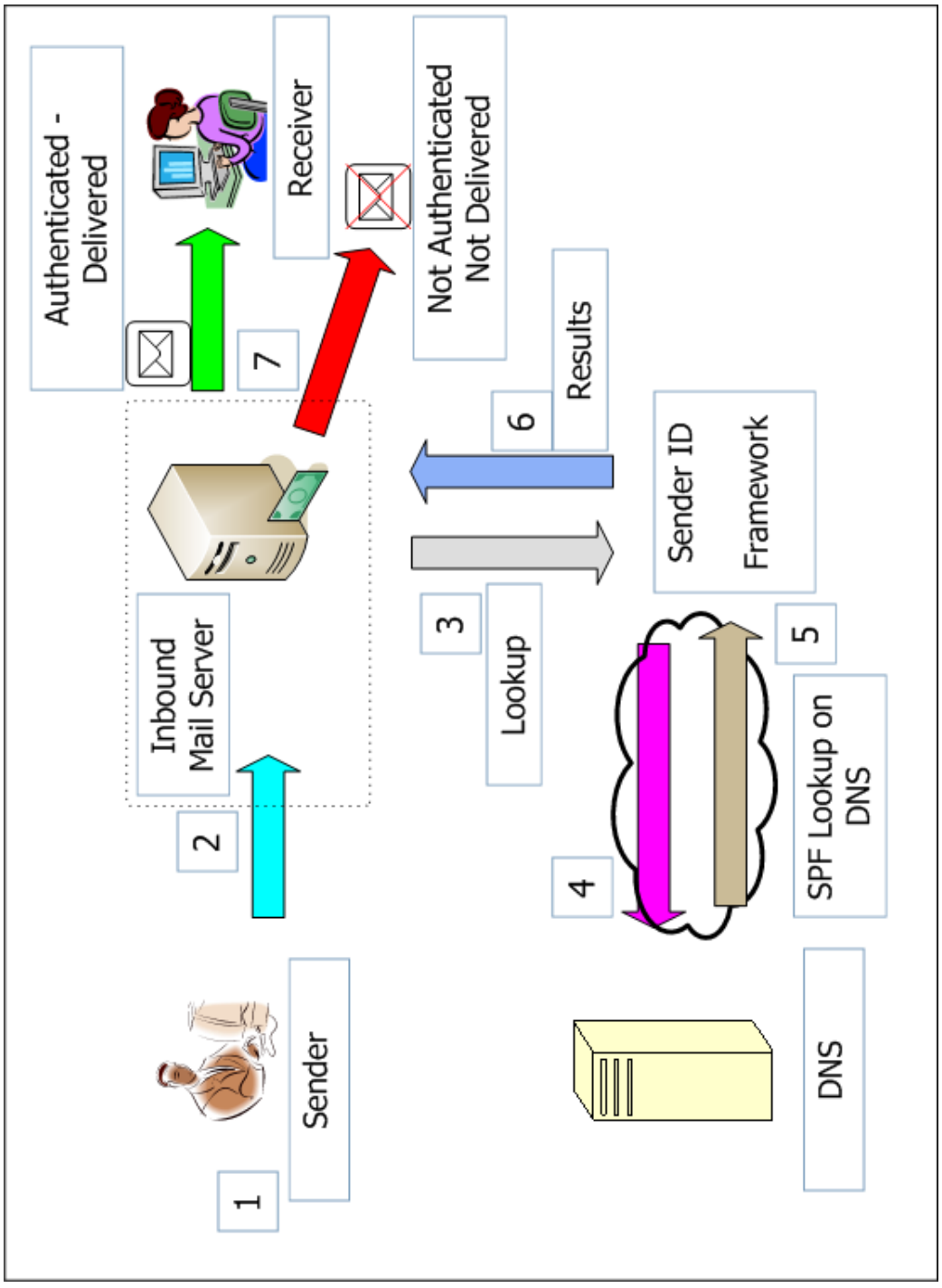


Figure 3.2: Sender ID Framework (SIDF) from Microsoft Corporation.

3.2.3 Email Authentication Score Card

Spammers, phishers, and fraudsters are smart people who have taken countermeasures to SPF. It has been observed that they are increasingly adopting SPF records for their domains in order to prevent spam rejection resulting from authentication failures.

In light of the above, MarkMonitor and VeriSign conducted independent studies on adoption of SPF by various domains on the Internet. Their studies are reported in Tables 3.1 and 3.2. They emphasize adoption of reputational information to protect corporate brands. Fraudsters are increasingly adopting email authentication standards for the look-a-like domains. They report that more than 90% of the Fortune 100 companies have look-a-like domains registered with SPF records. This is shown in Table 3.3.

Domain Sampling Authentication Study by MarkMonitor, Inc.
1.1 million .com/net domains with SPF1 and SPF2 records
38,664,506 .com domains 995,407 with SPF Records 2.57% Domains **4.41% domains with SPF Records**
5,926,755 .net domains 140,444 with SPF Records 2.4 % domains 62.7 % with MX Records **3.78 % domains with SPF Records** 44,51261 total domains

Table 3.1: Domain sampling authentication study by MarkMonitor. Source: Email Authentication Scorecard, David Silver, MarkMonitor. Email Authentication Implementation Summit, NYC, July 12, 2005. Table reproduced with permission from David Silver, MarkMonitor.

Domain Sampling Authentication Study by Verisign, Inc.
March 1.2 million .com/net domains with SPF1 and SPF2 records
July 1.6 million usable by SenderID
3.55% all .com/ .net domains 44,684 total domains Extrapolate to the population of 70 million domains, **2.48 million domains** with SPF 1/SPF2 records

Table 3.2: Domain sampling authentication study by VeriSign. Source: Email Authentication Score Card, David Silver, MarkMonitor. Email Authentication Implementation Summit, NYC, July 12, 2005. Table reproduced with permission from David Silver, MarkMonitor.

Trick	**Domain Name**
Transposed Letters	www.pfizre.com, www.krogre.com
Name Derivatives	www.wellsfargomastercard.com, www.fordmercury.com
Pluralized Versions	www.albertson.com, www.geicos.com
Alternative Extensions	www.dupont.net, www.merrill.net
Qwerty Mistypings	www.walmarr.com
Left Off Letters	www.dowcornin.com
Misspelled Words	www.bankofamarica.com

Table 3.3: Some look-a-like domains of reputed brands having SPF records. Source: Email Authentication Score Card, David Silver, MarkMonitor. Email Authentication Implementation Summit, NYC, July 12, 2005. Table reproduced with permission from David Silver, MarkMonitor.

3.2.4 Identified Internet Mail (IIM) from Cisco Systems, Inc.

Jim Fenton and Mike Thomas from Cisco Systems proposed Identified Internet Mail (IIM)[62] for email authentication in 2004. The IIM proposal provided message recipients with a mechanism to verify the integrity of the message using digital signatures and public key cryptography. It also provided a mechanism to authenticate the associated public key for determining the validity of the senders email addresses.

A digital signature is a hash of the message content signed with the private key. The sending side transmits the message along with its digital signature. At the recipient side, public key cryptography is used to recover the hash. This is done by decrypting the signature with the corresponding public key. The recipient system then computes the hash of the message separately using an already known *(agreed upon)* hash algorithm and decodes the digital signature. An exact match between the two hashes described above implies that the message integrity is preserved (i.e. the message has not been modified during transit).

The IIM Protocol is illustrated in Figure 3.3.

1. *Objective*

   The objective of the IIM proposal was to identify fraudulent messages and to ensure that the sender of the message was authorized by the domain owner to use that address for sending email messages from that domain.

2. *Sending Domain Signs the Message*

   The MTA of the sending domain signs the message and inserts a new signature along with the public key (used for creating the signature) into the email headers field. This is step 1 in Figure 3.3. Alternatively, a signature can also be added by the user's MUA, MSA, or a subsequent MTA.

3. *Recipient Domain Verifies Integrity of the Message and Authorization of the Key using DNS or the Key Registration Server (KRS)*

   The MTA of the recipient domain verifies the integrity of the message by using the hash of the message content, signature and public key included in the signature header. The recipient MTA also verifies authorization of the public key (used for signing the message) with that particular email address by querying the DNS or the KRS of the sending domain. This is shown in steps 2 and 3 of Figure 3.3. Depending upon the outcome of the verification process, user defined policies can be applied to accept or reject the email.

4. *Use of Third Party Reputation Services*

   The IIM proposal allowed the use of third party reputation services for applying policy-based decisions on the signed messages. This is shown in step 4 in Figure 3.3. Policy-based decisions can be applied depending upon the reputation score of sending domains. However, traffic with the third party reputation services was specified in clear text in the proposal, which leaves the door open for sniffing attacks.

Overall, IIM is a great contribution leading the industry towards email authentication. IIM has now merged with the DomainKeys (DK) to form the Domain Keys Identified Mail (DKIM) standard. The concept of a KRS is eliminated in the DKIM proposal.

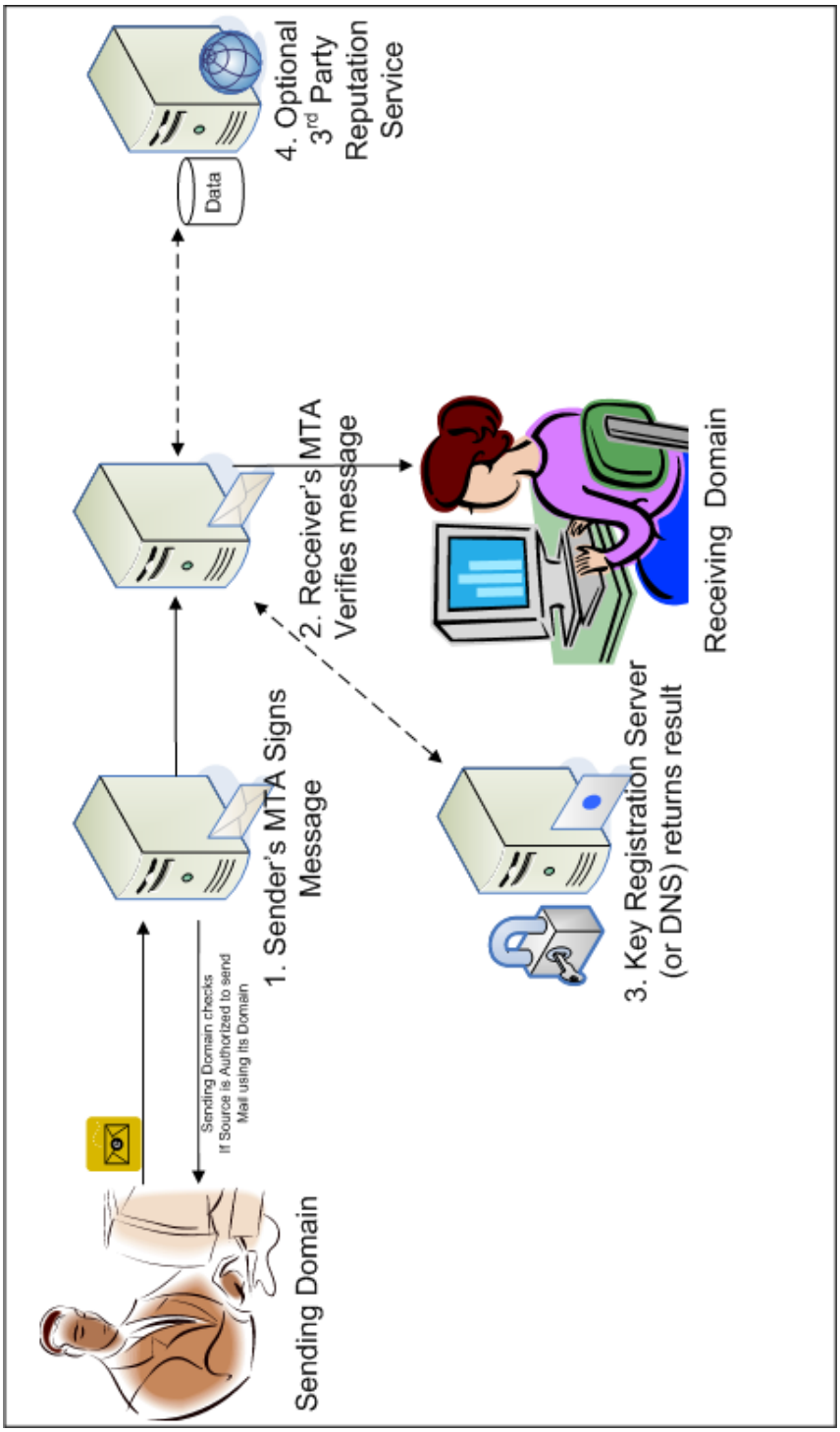


Figure 3.3: Identified Internet Mail (IIM) from Cisco Systems.

3.2.5 Domain Keys (DK) from Yahoo!, Inc.

Mark Delany from Yahoo! proposed "Domain-based Email Authentication Using Public-Keys Advertised in the DNS (DomainKeys)[41]" for email authentication. We now describe the DomainKeys protocol. The protocol is illustrated in Figure 3.4.

1. *Objective*

   The objective of DK is to provide recipient system with a mechanism to verify the integrity of the received message and also the authenticity of the domain of email senders using digital signatures, public key cryptography, and the DNS.

2. *Sending Side*

   (a) *Generation of the Public, Private Key Pair*

   The domain owners generate a pair of public and private keys. The private key is made available to DK outbound email servers and is used for signing all outgoing messages. The public key is published in the DNS entry of that particular domain.

   (b) *Signing of Messages*

   Upon receiving an email from an authorized user within the domain, a digital signature of the entire message using the private key is computed. This digital signature is prepended as a header to the email and sent for delivery destined at the recipient MTA.

3. *Receiving Side*

(a) *Extraction of Signature and Claimed From: Domain*

The recipient MTA which is DK-enabled extracts the signature and claimed *From:* domain from the email headers. It then looks up the public key for the claimed *From:* domain from the DNS.

(b) *Verification*

The public key retrieved from the DNS for that particular domain is then used to decrypt the signature to obtain the hash of the message as computed by the sender. The recipient system then computes the hash of the message. A match between the two hashes confirms that the email was sent with the permission of the claimed sending *From:* domain, and that the integrity of the email (headers and body) was preserved during transit.

(c) *Delivery*

Based on the verification results obtained above, the recipient email system applies local policies to deliver the email in the recipients' inbox or to trash it.

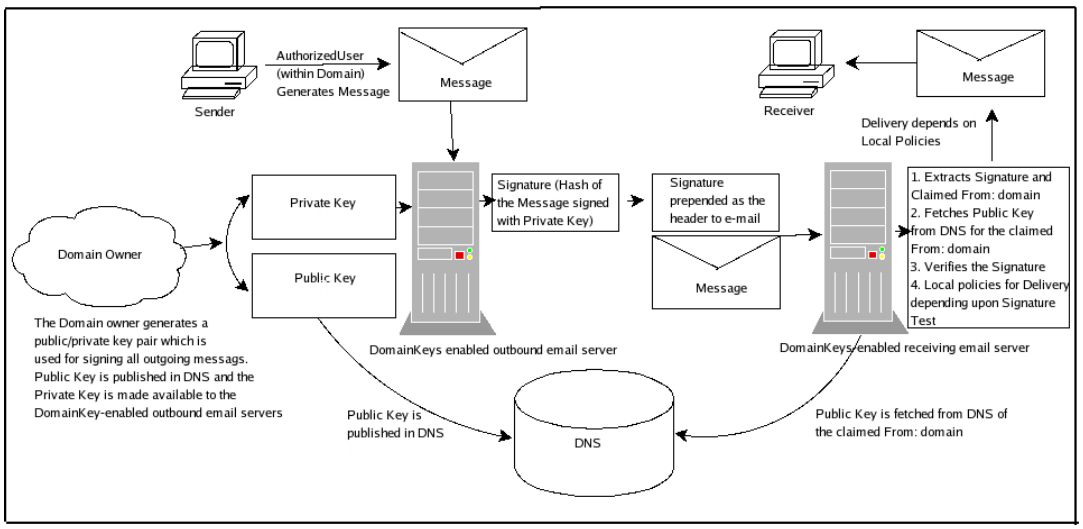


Figure 3.4: DomainKeys (DK) from Yahoo!.

3.3 Machine Learning Approach

Researchers and hackers have been actively interested in experimenting with machine learning techniques for filtering spam. A lot of spam filters available today, both open source and commercial, are based upon machine learning techniques. Most of these filters use the Naïve Bayesian model, Markov model, and other advanced techniques and heuristics. *Paul Graham* popularized the Naïve Bayesian model for spam-filtering through his essay "A Plan for Spam"[53] in 2002.

Some examples of open source filters based on the machine learning techniques are CRM114[35], DSPAM[130], SpamAssassin [110] and SpamBayes[111]. One of the best commercially-available spam filter is Death2Spam (D2S)[68].

A detailed unified model of spam filtration is explained in Chapter 4. The machine learning approach for filtering spam consists of the following steps.

1. *Corpus*: A corpus of spam and ham (legitimate messages) is obtained and validated.

2. *Feature Generation*: Features (attributes) representing each class (spam and ham) are generated. For example, individual words in messages can be used as features.

3. *Classifier Training*: A classifier based on a machine learning technique is trained by representing messages in terms of the features generated above.

4. *Threshold*: For each new message arriving at the mailbox, the classifier, operating on the extracted features, decides the target class the of the message (spam or ham).

Statistical approach for spam-filtering has been covered in [6] [8] [9] [10] [21] [25] [72] [77] [78] [79] [88] [92] [89] [90] [91] [94]. Jose Maria Gomez Hidalgo maintains an active bibliography on machine learning for spam detection[59].

3.4 Sender Pays/Sender Verification/Sender Compute

Recently, techniques based on the *sender-pays* model have been proposed to protect the Internet email system against spammers. In this model, some sort of action is required on the sender's part. The *sender-pays* model can be sub-categorized into *challenge-response* techniques and *micropayments* proposals. The *challenge-response* class of techniques can be further sub-categorized. The sub-categorization is shown below:

1. Challenge Response
    (a) Human Interactive Proof (HIP)
    (b) Proof of Work (PoW)
        i. Proof of Work (PoW) advocacy by Dwork and Moni Naor
        ii. Adam Back's Hashcash
2. Micropayments

3.4.1 Challenge Response

In *challenge-response* technique, the recipient system issues a challenge which the sending system has to respond to. The response is then verified by the recipient system. Upon success, the email is delivered to the recipient's inbox. It is best to use *challenge-response* techniques in combination with other techniques such as whitelisting and spam-filtering. For example, whitelisted senders should not be challenged, and legitimate senders who have responded

to the challenge already can be added to the whitelist. Emails classified as spam from the senders who are not whitelisted should be challenged[51].

The Penny Black Project

The Penny Black Project[118] at Microsoft Research is a *challenge-response* system with the notion of "bankable/refundable" tokens. Tokens can be generated by solving some sort of challenges or by making some monetary payment, and then can be deposited in an account. The challenge is a request for a token from the sender's account, which can then be refunded if the recipient informs the authority managing accounts that the email was a legitimate message and not spam[51].

The Penny Black Project has investigated the use of several currencies such as CPU cycles, memory cycles, and Turing tests to implement the *sender-pays* model. Senders can pre-compute the appropriate function tied to a particular message; senders can come up with the payment in response to a challenge after they have submitted their message, or senders can acquire a ticket pre-authorizing the message using the *Ticket Server*[118].

The *Ticket Server* is proposed in the paper "Bankable Postage for Network Services" by Abadi etal.[1]. They describe a scheme in which senders can send an email attached with a ticket obtained from the *Ticket Server*. The recipient can then determine the validity and freshness of the ticket from the *Ticket Server*. If the message is legitimate, recipients can inform the *Ticket Server* to refund the ticket to the sender.

3.4.2 Human Interactive Proofs (HIP) (CAPTCHA)

Human Interactive Proofs (HIPs) are challenges designed to distinguish humans from computers. In order to achieve this as originally proposed by Moni Naor[85], the computer presents a challenge that must be easily solvable for humans while hard for computers. HIPs are also known as Completely Automated Public Turing to tell Computers and Humans Apart (CAPTCHAs), or the reverse Turing Test. Specifically, CAPTCHAs are a special class of HIPs that require verification of the results by a computer and the public availability of the protocol/code[38]. Typically, these are image-based HIPs designed to distinguish a human from a computer program (bot). CAPTCHAs are commonly used for preventing automatic signing of the email accounts by bots and for challenging a user sending a large number of (possibly spam) emails. Since CAPTCHAs require a human to solve the challenge, they impose certain cost on spammers' actions.

Modern-day reading-based HIPs using clutter and distortion settings deployed by MSN, Hotmail, Register.com, Yahoo!, Ticketmaster and Google are illustrated in Figure 3.5.

The security of HIPs is dependent upon the *recognition* and *segmentation* tasks. *Recognition* and *segmentation* are well-known related problems in computer vision. In our context, *recognition* refers to the mechanism of identifying individual characters taken from the alphabet containing both lowercase and uppercase English characters, and the decimal number system. In practice, HIPs almost never use both upper and lowercase characters. Well designed HIPs usually exclude confusable pairs such as O and 0.

By *segmentation*, we mean *segmentation* at the lowest level, implying a mechanism of fragmenting the CAPTCHA image so that approximate locations of the constituent characters from the above alphabet are identified.

The most common reading-based visual HIPs are based on the character recognition tasks. Chellapilla etal.[27] from Microsoft Research suggest a combination of *recognition* and *segmentation* tasks for designing the next generation of CAPTCHAs, since recognition-based CAPTCHAs can easily be broken using machine learning techniques[28].

In light of the above, Chellapilla etal.[27] recently compared the single-character recognition ability of humans and computers by performing identical human user studies and computer experiments using convolutional neural networks as the core of the recognition engine. In their study, they assume that the segmentation problem is already solved (i.e. approximate locations of constituent characters are already identified). Their results indicate that computers are better than humans at single-character recognition under the commonly-used distortion and clutter scenarios in modern-day HIPs such as the ones illustrated in Figure 3.5.

They conducted a total of seven experiments with computers and humans. In each computer experiment, a convolution neural network was used as a recognizer. A total of 110,000 random characters were sampled using distortion and clutter settings. Of these, 90,000 characters were used for training, 10,000 for validation, and the remaining 10,000 for testing. The alphabet set consists of 31 characters from {A-Z,0-9}. Five characters {I, O, Q, 0, 1} were discarded because these can be easily confused (For example: 0 with O, I with 1). Thirty-point Times Roman font was used. An electronic human study closely matching the

computer experiments was conducted with 44 participants. The participants were assigned the task of recognizing characters under the same distortion conditions.

In all seven experiments, computers either outperformed or were on par with the human ability in recognizing the single characters. As previously mentioned, they used a total of 90,000 characters for training. A HIP has between 5 to 8 characters on average, and therefore 90,000 characters are equivalent to between 11,250 and 18,000 HIPs. Assuming the cost of solving a HIP is about $0.002[52], the total cost of labeling the training data amounts to $22.5 to $36.

Recent communication with Kumar Chellapilla and Joshua Goodman from Microsoft Research indicates that a customized HIP breaker can be built in a day for forty dollars, assuming that the tools for sampling, labeling, and neural network training have already been built.

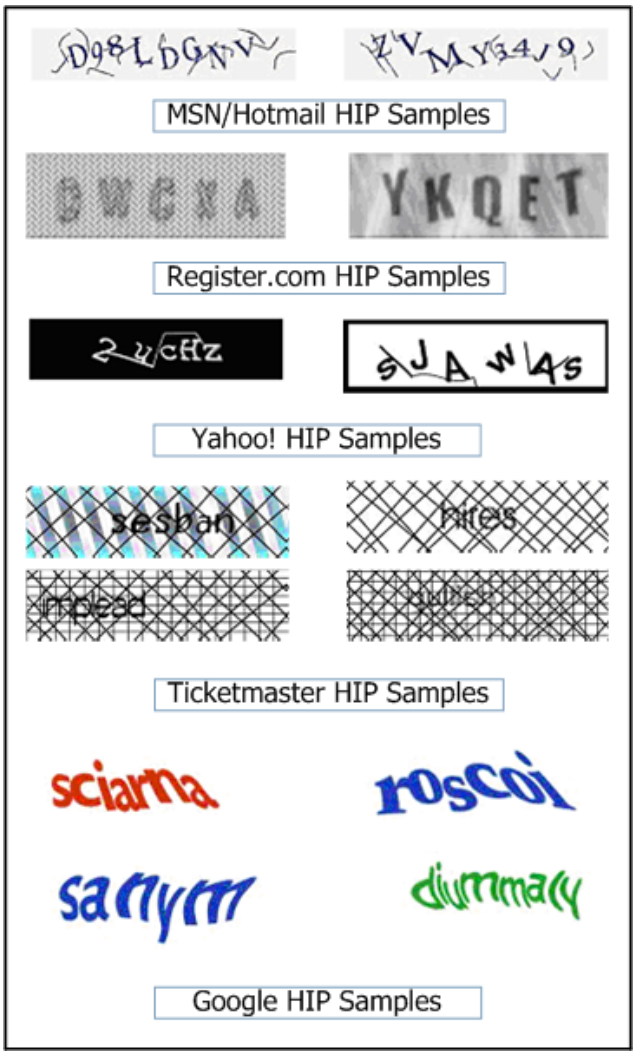


Figure 3.5: Modern-day reading based Human Interactive Proofs (HIPs) used by MSN/ Hotmail, Register.com, Yahoo!, Ticketmaster and Google. HIPs are also known as Completely Automated Public Turing tests to tell Computers and Humans Apart (CAPTCHA). Figure provided by Kumar Chellapilla, Microsoft Research.

3.4.3 Proof of Work (PoW)

Proof of Work (PoW) is a class of cryptographic protocols in which the prover demonstrates the verifier that he/she has performed a desired computation in a specified interval of time. Proof of Work puzzles have the following properties:

- PoW puzzles are hard to solve.
- PoW puzzles are easy to verify.
- There are no known shortcuts for solving such puzzles.

In their work, "Proofs of Work and Bread Pudding Protocols," Markus Jakobsson and Ari Juels have formally characterized the notion of Proof of Work (PoW) protocols and presented the idea of *bread pudding protocols*[66]. They define *bread pudding protocol* as a Proof of Work (PoW) such that the computational effort invested in the proof may be harvested to achieve a separately useful and verifiable correct computation.

1. *Proof of Work (PoW) Advocacy by Cynthia Dwork and Moni Naor (1992)*

   Dwork and Naor[45] first advocated the use of Proof of Work (PoW) in 1992 in their paper "Pricing via Processing or Combatting Junk Mail." The idea is that the sender of the email is required to spend some time performing complex computation in order to deliver email to the recipient. While this resource expense approach is a minimal burden on the legitimate sender, it is assumed that this will deter spammers, since spammers send bulk mail and hence require a significant amount of computational

time and resources. Dwork and Naor also introduced the notion of Proof of Work (PoW) with a *trap door*, a function that is easy to compute given the key *(trap door information)* but moderately hard to compute without the knowledge of this key.

2. *Adam Back's Hashcash (1997)*

   In 1997, Adam Back independently proposed a system based on the Proof of Work (PoW) known as the Hashcash[11][12]. The objective of Hashcash is to throttle abuse of Internet resources such as email and anonymous remailers. In Hashcash, the CPU cost function computes a token, which can be used as Proof of Work (PoW). Adam defines *cost function* as a function that is efficiently verifiable but parameterisably expensive to compute. In the Hashcash setting, a client computes a *token T* using a cost function MINT (). This token is then used to participate in a protocol with a server. The server verifies the value of the token using an evaluation function VALUE(). If the token has the required (correct) value, the server advances to the next step in the protocol. These functions are parameterized by the amount of work *w* the client has to spend in order to compute the token. The settings of *interactive* and *non-interactive cost functions* in Hashcash are described below.

   (a) *Interactive Cost-Functions*

   In this setting, the server uses the CHAL() function parameterized by the work factor *w* to compute a challenge *C*, which is then sent to the client. The client then mints a token *T* using the MINT() function and returns the token *T* to the

server which is then evaluated using a VALUE() function. If the value is correct, the server proceeds to the next step in the protocol.

$$\begin{cases} C \leftarrow CHAL(s, w) & \text{server challenge function} \\ T \leftarrow MINT(C) & \text{mint token based on challenge} \\ V \leftarrow VALUE(T) & \text{token evaluation function} \end{cases}$$

(b) *Non Interactive Cost-Functions*

In this setting, the client chooses its own challenge or random start value and generates a token *T* using the MINT() function. The token is then evaluated by the server using the VALUE() function. If the value is correct, the server proceeds to the next step in the protocol.

$$\begin{cases} T \leftarrow MINT(s, w) & \text{mint token} \\ V \leftarrow VALUE(T) & \text{token evaluation function} \end{cases}$$

Hashcash is based on *partial hash collisions*. A *partial hash collision* of *k* bits is defined as the match between *k* most significant bits between the hash of two numbers x and y (say SHA1(x) and SHA1(y)).

Campaign for Real Mail (CAMRAM)[23], a system based on the *sender pays* model uses, Hashcash. In such systems, the sender finds a number which, when prepended to the combined string of *From, To, Subject, Date*, has a hash whose first *k* bits are 0. *k* can be varied depending upon the time required to find the hash[52].

CAMRAM system uses CRM114 Filter for classifying emails into spam and good mail. We explain the architecture and working of the CAMRAM system in detail in chapter 6.

3.4.4 Micropayments

In Micropayments systems, small amounts of money (micropayments) are accumulated and collected as one regular payment. In the micropayment model for the Internet email, the recipient can charge a very small amount of money *($0.05)* from the sender through an online banking system. The recipient can keep the money if the email is a spam message. The money generated through these transactions can also be donated to charity. This proposal suffers from one major problem of establishing a worldwide bank or authority for keeping records of, tracking, and handling transactions[51].

3.5 Controlling Spam at the Router Level

Recently, Professor Mart Molle introduced the idea of fighting spam inside the network. The motivation for this early-detection strategy is saving network resources that would otherwise be wasted on the delivery of spam immediately discarded by the recipient's spam filter. Since no spam detection algorithm is perfect for everyone, however, network spam control would merely impose a limit on the number of copies of a particular bulk email message that can pass through the router per unit time.

In this way, legitimate but misclassified email would still reach its destination, but the spammer's cost of doing business would rise considerably because of the extra delivery time and/or personalization effort required for each message.

A detailed plan for implementing network spam control has been developed by Agrawal etal.[5]. After separating SMTP traffic from the fast path, the router tries to determine whether the new message is part of a bulk-delivery stream (i.e., it is similar to another message seen within the rate-control window) and if it contains spam. If both conditions hold, then the router blocks its delivery by closing the SMTP connection. The router could also set a TCP header flag to inform the recipient of the arrival of rate-compliant spam.

The Boyer Moore pattern-matching algorithm is used to identify bulk message streams by comparing each new email message with a cache of previously seen emails. This is achieved by using a two-level cache structure: *primary cache* and *secondary cache*. All emails received are kept in the *secondary cache*. As bulk mails are characterized by short span of time, all new emails received within a short span of time are compared with the signatures of the messages in the *secondary cache*. A match indicates that the message is bulk and the signature is moved to the *primary cache*. Thus, the *primary cache* contains signatures of bulk messages.

In the second phase, a Bayesian classification technique is applied to determine if the bulk stream is composed of spam messages.

3.6 Social Networks

Recently Boykin etal.[20] proposed use of *social networks* to fight spam. In their work, they create email graphs *(social networks)* from legitimate email (ham) and spam headers. Using *social networks* tools, they construct an anti-spam tool to identify a user's trusted network of friends and subnetworks generated by spam.

3.7 Distributed Collaborative Filtering

In this approach, a signature of every spam message received by the recipient is computed and added to a shared database. Upon arrival of a new email message, its signature is computed and then compared to the shared spam database. Depending upon the similarity measured between the signature of the message and the closest matching signature in the spam database, the message can be tagged as spam or ham. Vipul's Razor[123] and Distributed Checksum Clearinghouse (DCC)[40] are examples of successful modern-day collaborative spam filters.

3.8 Special Purpose One Time/Disposable Email Addresses

Special purpose/disposable email addresses are one approach to stay away from spam. A person can release the email address in a limited scope; for example, a person can give a different address to different correspondents. If any of those email addresses starts receiving spam, this implies that a particular email address has been abused and can be terminated, leading to the bouncing of all further emails.

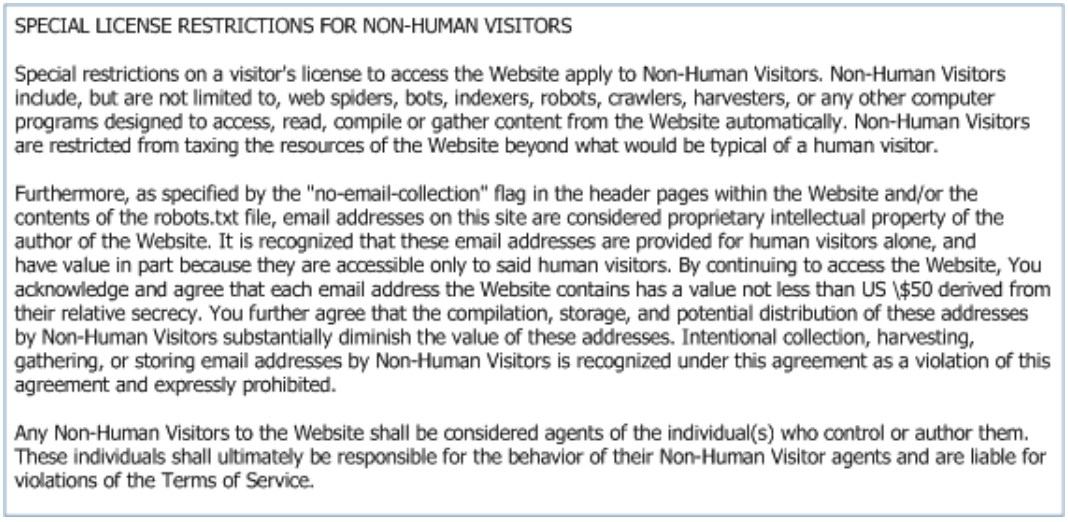
3.9 Tracking Harvesters through the Project Honey Pot

Email harvesting refers to the mechanism of collecting email addresses by web spiders or spambots which scan web pages, mailing lists or chat rooms looking for the @symbol. In order to avoid spambots, address-munging (i.e. inserting random text such that spambots cannot recognize email address while humans can) or the use of AT instead of @ is recommended.

Unspam Technologies came up with the idea of Project Honey Pot[98] for identifying spammers and their spambots used for harvesting email address from a website. In Project Honey Pot, the participating website installs software distributed by the Unspam Technologies. Email addresses specific to a particular website are generated and installed somewhere on the site. When those honeypot email addresses start receiving spam, the date, time, and IP address of the harvester are tracked.

The terms and conditions for the use of these email addresses listed on a particular web site is embedded in the comments of the HTML code of the page. Violators can be tracked and then subject to litigation.

Special license restrictions for non-human visitors from Project Honey Pot's terms and conditions section is illustrated in Figure 3.6.



SPECIAL LICENSE RESTRICTIONS FOR NON-HUMAN VISITORS

Special restrictions on a visitor's license to access the Website apply to Non-Human Visitors. Non-Human Visitors include, but are not limited to, web spiders, bots, indexers, robots, crawlers, harvesters, or any other computer programs designed to access, read, compile or gather content from the Website automatically. Non-Human Visitors are restricted from taxing the resources of the Website beyond what would be typical of a human visitor.

Furthermore, as specified by the "no-email-collection" flag in the header pages within the Website and/or the contents of the robots.txt file, email addresses on this site are considered proprietary intellectual property of the author of the Website. It is recognized that these email addresses are provided for human visitors alone, and have value in part because they are accessible only to said human visitors. By continuing to access the Website, You acknowledge and agree that each email address the Website contains has a value not less than US \$50 derived from their relative secrecy. You further agree that the compilation, storage, and potential distribution of these addresses by Non-Human Visitors substantially diminish the value of these addresses. Intentional collection, harvesting, gathering, or storing email addresses by Non-Human Visitors is recognized under this agreement as a violation of this agreement and expressly prohibited.

Any Non-Human Visitors to the Website shall be considered agents of the individual(s) who control or author them. These individuals shall ultimately be responsible for the behavior of their Non-Human Visitor agents and are liable for violations of the Terms of Service.

Figure 3.6: Project Honey Pot - special license restrictions for non human visitors such as bots. Figure provided by Matthew Prince, Unspam Technologies.

3.10 Accreditation and Reputation Services

3.10.1 AOL's Enhanced Whitelisting

America Online's Enhanced Whitelisting is a mechanism through which messages from the whitelisted bulk mailers to recipients are displayed with images and enabled links (i.e. images are not blocked and links are not disabled). Legitimate bulk mailers who are already on AOL's whitelist are automatically added to AOL's Enhanced Whitelisting after 30 days, if they adhere to the delivery standards according to AOL.

3.10.2 Habeas SafeList Program

Habeas offers a SafeList program through which legitimate email marketers are promised a guaranteed delivery of their email. More precisely, if the sender is certified by Habeas, it includes this information in the headers. Upon receiving emails claiming to be Habeas certified, the recipient MTA authenticates the IP address of the sender against Habeas SafeList. Upon authentication success, the email is delivered to the recipient's inbox. The working of the Habeas SafeList program is illustrated in Figure 3.7.

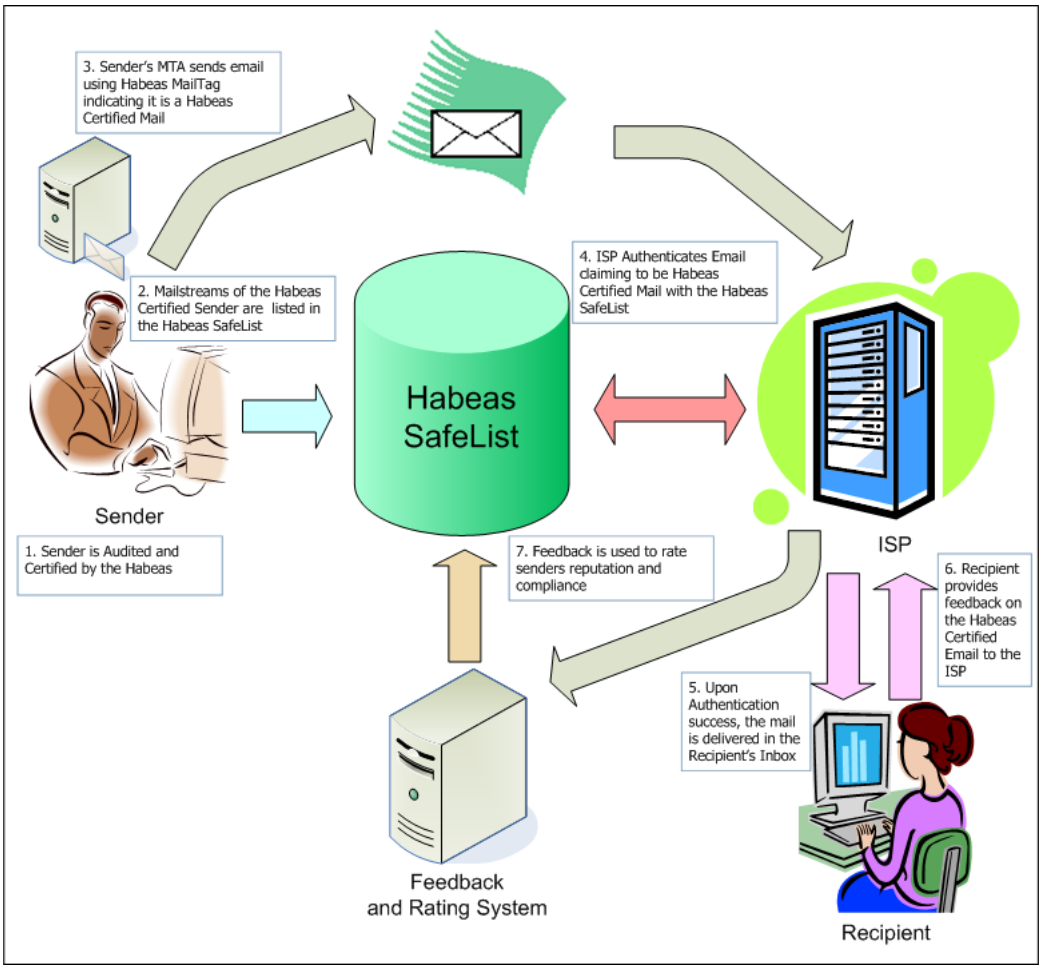


Figure 3.7: Habeas' SafeList program.

3.10.3 Return Path's Bonded Sender Program

The Bonded Sender program was introduced by the IronPort Systems in 2003 and was acquired by Return Path in early 2005. In the Bonded Sender program, qualified legitimate bulk mailers deposit a financial bond, and IP addresses of their outbound mail servers are then whitelisted. Upon receiving emails claiming to be affiliated with the Bonded Sender program, the recipient MTA authenticates the IP address of the sender against the Bonded

Sender whitelist using reverse DNS lookup. Upon authentication success, email is delivered in the recipient's inbox. The recipient provides feedback or complaints to its ISP which can then forward all complains to the Bonded Sender program authority. The financial bond of the bulk mailer is debited according to the number of complains received. The Bonded Sender program has a dispute resolution mechanism for senders to contest a debit of the bond. The Bonded Sender program is illustrated in Figure 3.8.

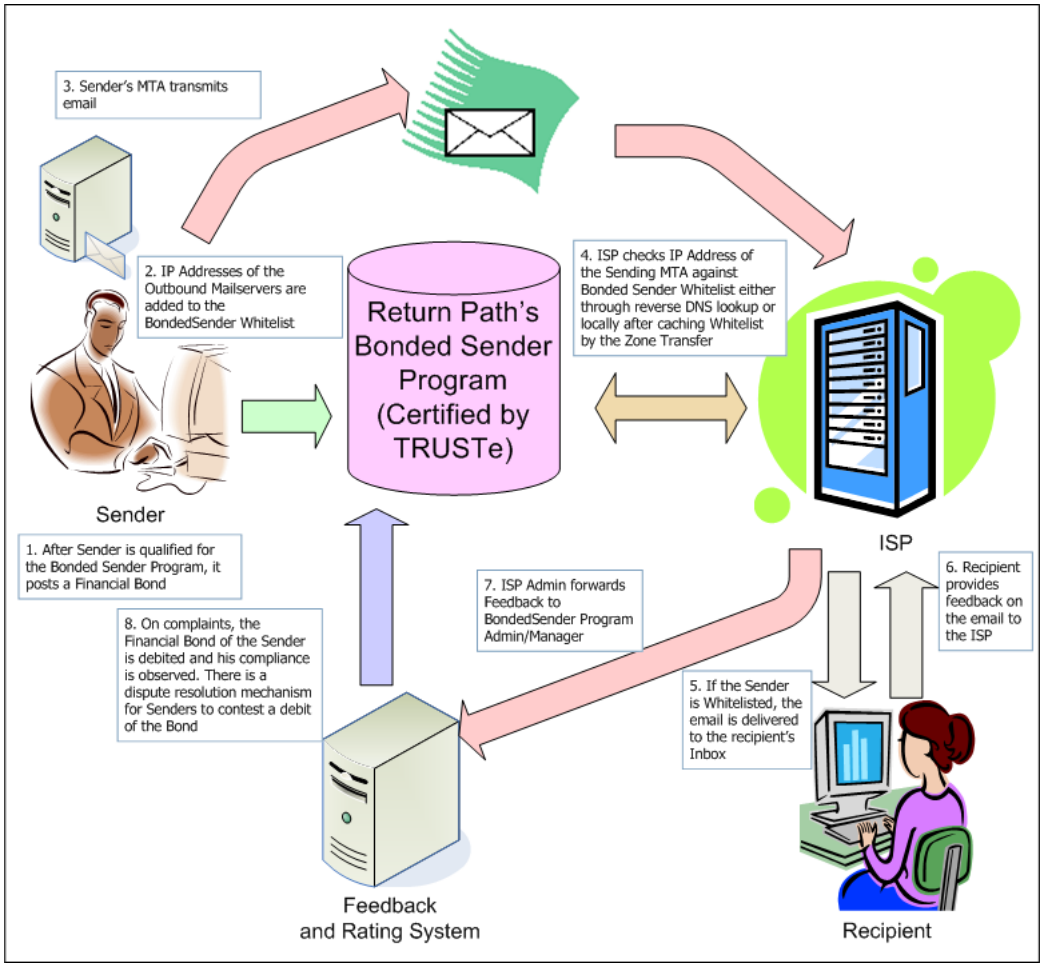


Figure 3.8: Return Path's Bonded Sender program.

3.10.4 CipherTrust's TrustedSource Reputation Service

TrustedSource portal from CipherTrust is a service which attempts to score the reputation of an IP address by combining attributes such as sending behaviour, whitelist and black-list. The data is gathered through already-installed IronMail Gateway appliances worldwide. TrustedSource service classifies reputation of an IP address in one of the four classes:

1. *Inoffensive:* Indicating a legitimate sender.

2. *Raised Concern:* Indicating a legitimate sender but suggesting further inspection of emails received from this address.

3. *Suspicious:* The IP address may belong to a spammer because it has shown many spam sender characteristics.

4. *Spam:* This IP address has been used for spamming or should not send any email messages.

3.10.5 IronPort's SenderBase Reputation Service

SenderBase is a similar reputation service from IronPort Systems and provides information about the email traffic sent by different IP addresses and domains.

3.11 Anti-Spam Appliances

Anti-spam appliances are the standalone appliances designed specifically for the purpose of filtering incoming spam. Some vendors selling such appliances are IronPort Systems [65], CipherTrust [31], Mirapoint [81], Barracuda Networks [13], Symantec [115] and Tumbleweed Communications[122]. Table 3.4 lists anti-spam techniques used in some of these appliances from the above vendors.

Appliance	**Technique**
CipherTrust IronMail Connection Control	uses TrustedSource sender reputation service and bandwidth throttling for rate limiting connections and traffic from known and suspected spammers
Mirapoint MailHurdle	uses greylisting
Symantec Mail Security 8160 Appliance	uses Brightmail sender reputation service to restrict connections and traffic from known and suspected spammers
Tumbleweed MailGate Edge	Uses sender authentication mechanisms to validate recipient and sender domains; protects against Denial of Service (DoS) and Directory Harvest Attacks (DHA)

Table 3.4: Anti-spam appliances from some vendors. Source: “Next-Gen Appliances Put SPAMMERS in the Crosshairs” by Logan G. Harbaugh, INFOWORLD, 08/29/2005.

Chapter 4

A Unified Model of Spam Filtration

4.1 Introduction

We describe three common spam filtration styles in the current state of the art. These are *Blacklisting, Human driven heuristic filtering, and Machine learning based filtering*. In this chapter, we consider a recursive description of these filters, compare them and consider higher level interactions of these filters in an Internet wide context.

1. *Blacklisting*

   Blacklisting deals with determining IP addresses and domains of spammers and blocking emails originating from these sources to mail servers. Some prime examples of blacklists are Spamhaus Black List (SBL) [19], SpamCop Blocking List (SCBL) [112], Composite Block List (CBL) [33].

2. *Heuristic Filtering*

   In heuristic filtering a human examines spam and nonspam texts for likely features, and writes specific code to trigger action on those features. These human created features are weighted (either manually or by an optimization algorithm), and thresholded to determine the spam or nonspam nature of a document. A prime example of such a heuristic filter is SpamAssassin[110]; other heuristic filters are used by major ISPs such as Earthlink[46] and Yahoo!.

3. *Statistical Filtering*

   In statistical filtering a human classifies a training set of texts; a machine-learning algorithm then creates and weights features according to an internal optimization algorithm. A number of these filters have been implemented, such as Death2Spam (D2S)[37], SpamBayes[111], SpamProbe[113], and the CRM114 Discriminator[35].

4.2 The Filtering Pipeline

In order to differentiate spam-filtering from the more generalized problems of information retrieval, we first define the set of spam filtering actions. Given a large set of email readers (who desire to reciprocally communicate without prearrangement with each other) and another set of email spammers (who wish to communicate with the readers who do not wish to communicate with the spammers), the set of filtering actions are the steps readers take *to maximize their desired communication and minimize their undesired communications*.

We now propose the following filtering pipeline as a generalized form for spam classification. Given an input text we perform the following sequential operations:

1. Initial (arbitrary) Transformation (a.k.a. MIME normalization)

2. Feature Extraction

3. Feature Weighting

4. Feature Weight Combination

5. Thresholding, yielding a go/no-go result (or perhaps go/unsure/no-go) result

Note that this classification model functions irrespective of the learning model used to update the databases involved; in the long term the learning model does not matter as long as the learning system produces (eventually) a satisfactory configuration. This includes both single-user, multi-user, and multi-host distributed filtering systems.

The filtering pipeline is graphically represented in Figure 4.1.

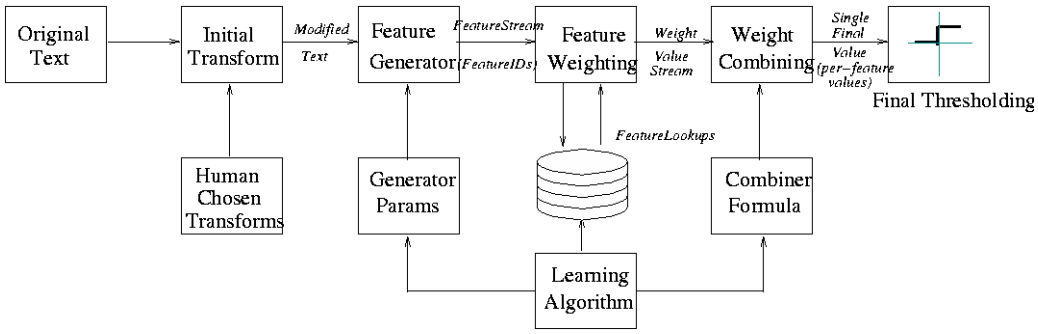


Figure 4.1: A generalized model for spam filtering pipelines.

4.2.1 Initial Transformation

The initial transformation (step 1) is often a null step - the output text is just the input text. Other common initial transformations are described below:

- *Character-Set Folding*

  Forcing the character set used in the message to the character set deemed "most meaningful" to the end user. For a typical US based user, this is base ASCII (also known as Latin-1) where accents are not significant.

- *Case-Folding*

  Removing extraneous case changes in the text.

- *MIME Normalization*

  Unpacking MIME encodings to a reasonable (and common) representation. In particular, the decoding of BASE64 texts is often useful, as some email systems encode a perfectly reasonable plain-ASCII text in BASE 64.

- *HTML Deboning*

  In some rare cases, HTML is an essential part of the message, but HTML also provides an incredibly rich environment to obscure content from a machine classifier while retaining the content for a human viewer. In particular, spammers often insert nonsense tags to break up otherwise recognizable words, while enthusiastic email authors often overuse tags such as $< bold >$ and $< color >$ to the detriment of the actual content.

- *Look-a-Like Transformations*

  Spammers often substitute characters for other characters that "look alike", in order to avoid known spammish keywords. Examples are using '@' instead of '**a**', '1' (the numeral) or ! (the punctuation) instead of '1' or 'i' (the letters), and $ instead of 'S'.

- *OCR/Machine Vision Operations*

  Using machine vision techniques to form a textural representation of an included image (such as a pornographic .jpg). The authors are unaware of any actual implementations doing OCR or machine vision classification at the time of writing this paper.

It should be realized that not all systems use an initial transformation; some systems work perfectly well with no preprocessing whatsoever.

It should also be noted that the initial *text-to-text transformation* is human-created, arbitrary, and rarely if ever 1-to-1 in terms of characters, words, or even lines of text. The arbitrary nature of this transformation is what we will eventually use to demonstrate that this generalized filtering form includes not only statistical methods, but also heuristic methods and black/white lists.

4.2.2 Feature Extraction

In the feature extraction step of the pipeline, the text is converted into a set of indexable features. We propose a two-step process in feature extraction:

1. Using a *regular expression (regex)* to segment the incoming text into interesting parts (tokens). This step is akin to *tokenization* in typical compiler.

2. Using a *tuple-based approach* to combine the tokens into features. This tuple-based approach allows arbitrarily nonadjacent tokens to be combined into features.

In prior work [105] it was found that the choice of tokenizing regexes between an intuitively good regex such as [[:graph:]]+ and a carefully crafted HTML-aware regex can affect overall accuracy by at least a factor of 25% on one standard corpus. It is likely that with multiple "typical" corpora at least another similar factor in accuracy by tokenization will occur.

The tuple-based approach to feature generation is a generalization of the techniques known to the authors [30][105][127][128]. Tuple-based feature generation often produces more features than the original text contained tokens; in some implementations there will be more features emitted from the feature generator than there were bytes of text in the original message.

Tuple-based feature generation uses a two-step process to convert textual tokens into features [120] [1]. The steps are:

[1] It is also computationally feasible to invert the order of text $\rightarrow$ tokenvalue and tuple $\rightarrow$ weight operation; For example the TIE system [105] forms the concatenated texts first, and then looks up the concatenated text to get a weight value. This is computationally equivalent; either method will generate identical results. Interestingly, TIE [120] is in some cases faster in this string-based operation than $CRM114$ is in the computationally equivalent hash-then-combine-then-look-up computation.

1. *Conversion of the textual tokens into unique (or nearly unique) numerical representations*

   For example, a lookup of textual tokens in a dictionary of tokens previously seen in learning corpus can guarantee unique numerical representations of tokens (textual tokens that haven't been seen in the learning corpus can be assigned a locally unique number but, by definition, no feature containing a previously unseen token exists in the learned corpora and so such tokens do not contribute to the useful classification feature set). Alternatively, in a faster implementation textual tokens could simply be hashed to a moderately long numerical representation; a 64-bit representation gives roughly 1.8E19 different possible hashes and the occasional hash collision is equivalent to a single typographical error in the human-created input text.

2. *The sequence of numerical representations of sequential features are then combined by stepwise dot products against every member of a tuple defining the feature generator*.

   This is stepping the sequential token values through a pipeline of length equal to or greater than the longest tuple and calculating the dot product of the pipeline contents against each member of a set of tuples. By describing the feature generation as tuples, we can obtain features from simple unigrams to arbitrary selection lattices. Of course, for reasonable tuple sets the pipeline length is usually quite small - there is evidence [105] that pipeline lengths in excess of five or six do not increase accuracy.

In a perfect world, this dot product would have tuple terms that are large prime numbers, and be carried out in infinite-precision (bignum) arithmetic, but a fast implementation can simply use small integer tuple values and register-based (fixnum) arithmetic with a trivially small decrease in accuracy (again, each collision caused by 64-bit finite arithmetic is equivalent to a single typographical error in the human-created input text).

For simplicity, we will represent all of our tuples in this paper in a normalized form - that is, tuple elements which correspond to pipeline tokens which are to be disregarded will always have the tuple element value 0, and all nonzero tuple elements will start at 1 and increase by 1 for each element that is to be regarded with unique position, and will reuse a previously used integer if the corresponding tuple element is to be considered interchangeable with a previous tuple element.

Many spam filters consider only single words to be features. Consider the very simple single-element tuple set $\{1\}$. In this case, each output feature is simply an input feature set, multiplied by 1. This gives the "typical" behavior of most word-at-a-time Bayesian spam filters.

The somewhat more interesting tuple set is:

$$\{ \quad 1, \quad 0 \quad \}$$

$$\{ \quad 1, \quad 2 \quad \}$$

It yields the classic "digraph" feature set as used by libbnr [14], where each word is taken as a feature, and each pair of words in sequence are also taken as a feature. The sequence

"foo bar" is not equivalent to the sequence "bar foo" (the zero term in the first tuple element multiplies the respective token's numerical value by $0-$ thereby disregarding the feature).

This tuple-based feature generation also allows for representation of bag-based (order does not matter) and queue-based (order matters, but interspersed ignored elements don't) feature generators.

For example, an order-ignoring ("bag-based") feature generator with a viewing pipeline length of 4, which when slid over the pipeline, generates all pairs taken two at a time is below:

$$\{ \quad 1, \quad 1, \quad 0, \quad 0 \quad \}$$

$$\{ \quad 1, \quad 0, \quad 1, \quad 0 \quad \}$$

$$\{ \quad 1, \quad 0, \quad 0, \quad 1 \quad \}$$

Note that the tuple coefficients for each token are each 1 and thus the same output numerical representation is generated without regard to the order of the incoming tokens.

A very similar but order-sensitive tuple set is below:

$$\{ \quad 1, \quad 2, \quad 0, \quad 0 \quad \}$$

$$\{ \quad 1, \quad 0, \quad 2, \quad 0 \quad \}$$

$$\{ \quad 1, \quad 0, \quad 0, \quad 2 \quad \}$$

Note that the coefficient of the first feature in the pipeline is 1 and the coefficient of the second feature in the pipeline is 2. This causes this particular tuple set to generate features where order is significant, but intervening tokens are not significant (for example, "foo bar",

"foo lion bar", and "foo lion tiger bar" all generate the "foo bar" feature value, but "bar foo", "bar lion foo", and "bar lion tiger foo" generate a different (but identical between themselves) value.

We can create "contains" feature generators. For example, the following tuple set creates features where the three tokens must occur in the order specified, but as long as the entire feature sequence fits into the pipeline length (here, 5), the precise positioning does not matter. Below is the tuple set:

$$\{ \quad 1, \quad 2, \quad 3, \quad 0, \quad 0 \quad \}$$

$$\{ \quad 1, \quad 2, \quad 0, \quad 3, \quad 0 \quad \}$$

$$\{ \quad 1, \quad 0, \quad 2, \quad 3, \quad 0 \quad \}$$

$$\{ \quad 1, \quad 2, \quad 0, \quad 0, \quad 3 \quad \}$$

$$\{ \quad 1, \quad 0, \quad 2, \quad 0, \quad 3 \quad \}$$

$$\{ \quad 1, \quad 0, \quad 0, \quad 2, \quad 3 \quad \}$$

Another interesting tuple set is below:

$$\{ \quad 1, \quad 2, \quad 0, \quad 0, \quad 0 \quad \}$$

$$\{ \quad 1, \quad 0, \quad 3, \quad 0, \quad 0 \quad \}$$

$$\{ \quad 1, \quad 0, \quad 0, \quad 4, \quad 0 \quad \}$$

$$\{ \quad 1, \quad 0, \quad 0, \quad 0, \quad 5 \quad \}$$

Note that this tuple set does *not* generate the unigram (that is, features representing single words, taken one at a time, do not appear in the output feature stream). This tuple set is

interesting because it has been shown experimentally to be particularly effective in spam filtering - in experiments, it was at least equal to the same tuple set including the unigram, and was often more accurate [105]. This is an interesting counterintuitive example where better accuracy is achieved with fewer features.

4.2.3 Feature Weighting

The third step in the filtration pipeline is feature weighting. This weighting has several parts:

- A part of the weighting is based on the *prior training* of the filter with respect to this particular feature; this is simply a *table (or database) lookup*. Often this part of the weighting is the number of times each feature has been seen in each of the respective training corpora [2]. There are experimental indications that for some tuple sets accuracies are higher if training and feature weighting are based on the number of training documents it appears in, rather than the number of times the feature appears (thus the same feature repeated several times in a document is used only once for the training or classification).

- A part of the weighting is based on the *tuple* itself - for example, it appears for some tuple sets and combiner rules to be advantageous to not weight all tuples evenly; tuples containing more nonzero terms appear to be more significant than mostly-zero tuples, and tuples with the nonzero terms adjacent or nearly adjacent are more significant than

[2]The reader should differentiate between "training texts offered" versus "training texts actually used". Many very effective learning algorithms do not use all of the training texts offered; the superfluous texts do not have their features added to the database.

those tuples with the nonzero terms widely separated. The Markovian weighting sets [127] using a full set of all possible combination tuples exhibit this behaviour. Other tuples sets and combiner rules show no such effect (OSB[105] tuples with Winnow-type combiner rules seem to work best with uniform tuple values)

- A part of the weighting may be based on *metafeatures* or *"database constants"*; for example, it appears advantageous to alter the weighting of features depending on the overall count of features learned, as well as the related count of example texts learned.

It is not necessarily the case that a feature's weight is a strict probability [or better: is a probability estimate]. Other weighting generators can be used as desired; it is perfectly reasonable to define weight of a feature by reference to a database produced by a learning algorithm. For example, the Winnow algorithm uses linear weights stored in a database. It is not reasonable to assume that every possible weighting generator will work with every possible combiner rule; in particular, Bayesian combiner rules need reasonable - valued probabilities as inputs (consider what happens to a Bayesian combiner rule if a local probability is greater than unity or less than zero?)

A very simple and obvious per-feature probabilistic weighing formula is given in Equation 4.1

$$Weight = \frac{TimesSeenInClass}{TimesSeenOverAllClasses} \tag{4.1}$$

Unfortunately, the weighting shown in Equation 4.1 yields absolute certainty when a feature has only been seen just once in a single class. A better-behaved per-feature weight as

used in the Markovian filter option in $CRM114$ Discriminator Framework [128] is shown in Equation 4.2.

$$Weight = \frac{TimesSeenInClass}{TimesSeenOverAllClasses + Constant} \tag{4.2}$$

Constant is some relatively small integer, such as 1 to 100. Note that, this is one common form of "smoothing".

Experimentally, we have found that a better local estimate of probabilistic weight takes into account the relative number of documents in each of the learned corpora; a simple renormalization of weights with respect to how many documents have been inserted into each database such as in Equation 4.3, gives a significant improvement in filtering accuracy.

$$Weight = \frac{TimesSeenInClass * DocumentsInClass}{(TimesSeenOverAllClasses + Constant) * TotalCorporaDocuments} \tag{4.3}$$

- *Other Weight Generators*

  It is not necessarily the case that a features weight is a strict probability. For example, the Winnow algorithm uses linear weights stored in a database; each feature's weight starts at 1.0000 [3]. Winnow Feature weights are promoted on correct learning by multiplying by a constant > 1 (typically in the range 1.1 to 1.23), and demoted on incorrect

[3] To be precise, in Winnow, all features have a value of 1.000 until learned otherwise; the Winnow type database handler is specifically programmed to give the default value of 1.000 when a feature is not found

learning by multiplying by a different constant < 1 (typically in the range 0.8 to 0.9) [4].

Other weighting generators can be used as desired; it is perfectly reasonable to define weight of a feature by reference to a database produced by a learning algorithm.

4.2.4 Weight Combination

At this point in the pipeline, we now have a series of weights corresponding to each feature in the unknown input text. We now combine these weights to obtain a single output result.

Some spam filters use completely linear weight combining rules - that is, their combining rules obey the laws of superposition, and the output value of a concatenated text is equal to the combining rule applied to the values of the unconcatenated text segments. Other filters have nonlinear features in their combining laws, such as "ignore any weight below a particular threshold". This type of nonlinearity is easily accommodated by simple modification of the combining rules below.

Other filters use a sorted-weight approach, typically "only use the most extreme N weights found in the document". This sorted-weight approach is less easily represented as a simple function of the stream of weights emitted from the weight generator software.

- *Bayesian Combining*

  A very common combining formula is the generalized Bayesian probability combiner formula - this relates a local probability due to a given feature, a prior probability (the

[4] It should be noted that Winnow learning is not "promote on correct", rather, it is a thickness-based learning algorithm that promotes/demotes if the final output is not outside a thick threshold; this means that for some ambiguous texts both the in-class and not-in-class databases will be updated.

probability of a document being in a particular class before this feature was encountered) and the posterior probability (the updated probability, given that the feature was found in the document)

$$P_{posterior} = \frac{P_{prior} * P_{local}}{\sum_{overallclasses}(P_{priorThatClass} * P_{localThatClass})} \tag{4.4}$$

- *Chi-Squared Combining*

  Another common combining rule is the *chi-squared rule*. In the chi-squared formulation, the observed occurrences of features are compared against the expected number of occurrences of those features. Typically this is done in a matrix format The actual chi-squared formula for the chi-squared value of one exclusive feature is given by the equation 4.5. All of the chi-square feature values are summed.

$$X^2 = ((Observed - Expected)^2/Expected) \tag{4.5}$$

Nature of Text	*Expected No. of Features*	*No. of Features Actually Observed*
Unknown text assumed to be Good	A	B
Unknown text assumed to be Spam	C	D

Table 4.1: Chi Square Formulation

- *Winnow Combining*

  If the weight calculation and value updating is performed using the Winnow algorithm, the combining rule is particularly simple and is given by Equation 4.6

$$WeightOut = WeightIn + LocalWeight \tag{4.6}$$

4.2.5 Final Thresholding

After the weights are combined, a final thresholding is performed. For filters that use probability, the final decision threshold value is typically 0.5 (ambivalent probability). As some filters authors and filter users consider it preferable to falsely accept some spam in order to decrease the amount of falsely rejected nonspam, not all filters use $p = 0.5$. (it is conjectured that, for filters using linear combining rules, that altering the decision threshold is completely equivalent to altering the training regimen).

4.3 Emulation of Other Filtering Methods

If this generalized model of spam filtering is truly general, we must show that it is possible to represent all possible filters within its framework. In one sense, this is trivial to prove, as the initial *text-to-text transform* can contain an arbitrary computation and all subsequent stages can operate in a pass-through mode. Despite this trivial proof, it's actually useful to consider

how to use an optimized implementation with parameterized code in the generalized filtering model to implement other types of filters such as heuristic filters and black/whitelists.

4.3.1 Emulating Whitelists and Blacklists in the Generalized Model

Emulation of whitelist/blacklist filtering in the generalized model is quite simple. All we need to do is to look for the whitelisted or blacklisted words in the input, and count them. If the whitelisted words outnumber the blacklisted words, the text is good, if the blacklisted words outnumber the whitelisted words, the text is spam, and it's indeterminate if there is a tie.

A set of parameters for the generalized model that produce a whitelist/blacklist filter is below:

1. The initial *text-to-text* transform is "none". That is, the output text is equal to the input text.

2. The token generator regex is [[:graph:]]+ (resulting in blank-delimited blacklisting and whitelisting.

3. The tuple set for feature generation is just $\{1\}$, giving feature Ids that correspond to single words.

4. The feature database is loaded only with those words that are blacklisted or whitelisted, with a value of $+1$ for whitelisted words, and -1 for blacklisted words. All other words return a 0.0 value.

5. The feature weight rule is

$$FeatureWeight = FeatureLookedUpValue$$

6. The feature combiner rule is

$$NewScore = OldScore + FeatureWeight$$

7. The final decision threshold is "> 0" implies good, "< 0" implies bad, otherwise unknown".

This particular implementation scores whitelist words and blacklist words equally; some people consider them equally valuable. For pure whitelisting or blacklisting, one could put only the respective whitelist or blacklist words into the feature database, or one could weight whitelist words with much higher weights than blacklist words (or vice versa).

4.3.2 Emulation of Heuristic Filters in the Generalized Model

We now consider the question of the emulation of heuristic filters in the generalized model. Heuristic filters are by definition created by expert humans, to trigger on specific features of the unknown text. The individual features may either accumulate a weight or score, or by themselves be sufficient to reject (or accept) an unknown text. It is possible to form

hybrid human+machine-created systems in this form. For example, SpamAssassin [110] has a feature library of several hundred human-written recognizers; the recognizers themselves have weights that are optimized by a genetic algorithm testing against a well-vetted base of spam and nonspam texts.

Emulation of these heuristic-feature-based filtering systems is easily performed by:

1. Generating a “local key”, a string with a vanishingly small probability of appearing in an incoming text; this local key can be constant for a user, or be randomly generated for each incoming text.

2. Executing a series of rewrite rules; each rewrite rule corresponds to one of the original heuristics. Whenever a rewrite rule matches the incoming text (corresponding to the original heuristic being triggered), the rewrite rule appends a new line at the end of the unknown text; this new line contains the local key followed by the heuristic rule’s unique identifier.

3. The second-from-last rewrite rule deletes every line in the unknown text that does not start with the local key.

4. the last rewrite rule deletes every copy of the textual representation of the local key from the text, leaving only the unique heuristic identifiers.

5. The text emitted from the preprocessor is now just the unique identifiers of the heuristic rules that were satisfied by the original text.

The resulting output text is now taken one unique identifier at a time (i.e., with the tuple set:

$$\{1\}$$

and the respective weightings of the unique identifiers are then looked up. For example, if we were emulating SpamAssassin [110], the respective weightings stored in the database would be the relative point values of each of the heuristic features found, the local weighting formula would be just:

$$LocalWeight = DatabaseValueReturned$$

and the combining rule would be the summation of the local weights is:

$$TotalWeight = TotalWeight + LocalWeight$$

The current version of SpamAssassin [110] at this writing uses a threshold of 4.5, so the final decision threshold is:

$$TotalWeight - 4.5 > 0$$

4.3.3 Examples of Popular Spam Filters in the Generalized Model

Here, we will show several popular spam filters as expressed in the generalized model.

1. *Classic Paul Graham's "A Plan For Spam" model (2002)[53]*

   (a) *Preprocessor:* lowercase, remove HTML comments, remove numeric-only constants

   (b) *Tokenizer:* [[-'$a-z]]+

   (c) *Feature Generator:* single tuple - { 1 }

   (d) *Lookups:* count of good occurrences "G", count of bad occurrences "B"

   (e) *Weight Generator:*

$$If(G + B < 5) : 0.5$$

$$Else(\lceil 0.99(\lfloor 0.01(\frac{Bad}{Good + Bad})\rfloor)\rceil)$$

   (f) *Weight Combiner:* Classic Bayesian, top 15 scorers only

   (g) *Final Decision Threshold:* 0.9

2. *Death2Spam (D2S) model[68]*

   (a) *Preprocessor:* lowercase, remove HTML comments, add specific markers for FROM and TO fields in header

   (b) *Tokenizer:* [[a-z]]+

   (c) *Feature Generator:* single tuple - { 1 }

(d) *Lookups:* count of good occurrences “G”, count of bad occurrences “B”

(e) *Weight Generator:*

$$(\lceil 0.99(\lfloor 0.01(\frac{Bad}{Good + Bad})\rfloor)\rceil)$$

(f) *Weight Combiner:* Classic Bayesian

(g) *Final Decision Threshold:* 0.5

3. *SpamAssassin model[110]*

(a) *Preprocessor:* all 300+ SpamAssassin heuristics (using the feature Ids as in current SA), deleting all non-featureID text as a final step before tokenizing

(b) *Tokenizer:* [[:graph:]]

(c) *Feature Generator:* single tuple - 1

(d) *Lookups:* precalculated per-feature weights of good occurrences “G”, weight of bad occurrences “B”

(e) *Weight Generator:* Simple lookups, no G/B combination

(f) *Weight Combiner:* Simple addition

$$NewScore = OldScore + B - G$$

(g) *Final Decision Threshold:* accept if $<$ 4.5

4. *CRM114 (2002 model)[35]*

   (a) *Preprocessor:* remove HTML comments, expand BASE64's

   (b) *Tokenizer:* [[:graph:]]

   (c) *Feature Generator:* Sparse Binary Polynomial Hash (SBPH) tuple set of window length 4 (note that CRM114 in 2002 did not use differing weightings depending on the tuple. CRM114 moved to a window-length 5 tuple set in early 2003)

$$\{\ 1,\ 0,\ 0,\ 0\ \}$$
$$\{\ 1,\ 2,\ 0,\ 0\ \}$$
$$\{\ 1,\ 0,\ 3,\ 0\ \}$$
$$\{\ 1,\ 2,\ 3,\ 0\ \}$$
$$\{\ 1,\ 0,\ 0,\ 4\ \}$$
$$\{\ 1,\ 2,\ 0,\ 4\ \}$$
$$\{\ 1,\ 0,\ 3,\ 4\ \}$$
$$\{\ 1,\ 2,\ 3,\ 4\ \}$$

   (d) *Lookups:* count of good occurrences "G", count of bad occurrences "B"

   (e) *Weight Generator:*

$$0.5 + \frac{Good}{(\frac{Good+Bad}{2})} + 16 \tag{4.7}$$

   (Note: 2002 CRM114 did not yet have superincreasing Markovian weightings that depended on the tuple being used)

(f) *Weight Combiner:* Classic Bayesian, score everything.

(g) *Final Decision Threshold:* accept if $P_{good} > 0.5$

4.3.4 Conclusion and Future Work

This chapter shows that a single unified pipeline, controlled by a relatively small set of input parameters (a set of rewrite rules, a regex, a set of tuples, a mapping/lookup table, and a chain combining rule) can describe nearly all of the spam filters of all variants typically available today. By showing the commonality of these filters, we hope to stimulate creative thought to advance the state of filtering art, with some hope of advancing the entire field of information retrieval.

Chapter 5

The CRM114 Discriminator Framework

5.1 Introduction

CRM114[35] is an acronym for the Controllable Regex Mutilator concept 114. It was created by Dr. William "Bill" S. Yerazunis from Mitsubishi Electric Research Laboratories (MERL) in 2002. It originally got its name from Stanley Kubrick's movie *Dr. Strangelove (or: How I Learned to Stop Worrying and Love the Bomb)*. The CRM114 Discriminator is analogous to the radio receiver in the movie which is designed to receive only authentic communications and reject any communications that might be false or misleading. In the context of spam filtering, CRM114's goal is to discriminate between authentic messages and to reject others.

Most spam filters are single-paradigm; to change the methodology of filtering requires a complete rewrite of the filter. CRM114 is different; It's not a filter per se; it is a language designed for writing filters and the most common filter usage is for spam.

Much of the work on spam-filtering models by the author of the thesis conducted in collaboration with William Yerazunis from MERL, Christian Siefkes from Freie Universität, Fidelis Assis from Embratel and Dimitrios Gunopulos from UCR[30][105][129], is implemented in different versions of the CRM114 Discriminator *(Please refer to Chapters 4, 7 and 8).*

At the time of this writing, CRM114 versions implemented the Naïve Bayes Model, the Markov Model, an OSB (Orthogonal Sparse Bigram) model, Littlestone's non-statistical Winnow algorithm[71], an experimental "voodoo" weighting model on an Orthogonal Sparse Bigrams (OSB) base, a bytewise correlator model, and a hyperspatial radiance model. A CRM114 version implementing Support Vector Machines (SVM) model is under development. Fidelis Assis from Embratel is the original inventor of the Orthogonal Sparse Bigrams (OSB).

All of the classifiers mentioned above are keyword selectable, so to switch to a different classifier model requires only changing a single flag and re-running the training corpus. Because the CRM114 system is a language, multiple classifiers can coexist peacefully in the same filter architecture.

The CRM114 language is explained in detail in the book *CRM114 Revealed,* which is freely available from the CRM114 homepage hosted at SourceForge[35].

5.2 CRM114 Discriminator and the Text Retrieval Conference (TREC) 2005

The Text Retrieval Conference (TREC)[116] is organized every year by the Information Technology Laboratory's (ITL), of National Institute of Standards and Technology (NIST), the Retrieval Group of the Information Access Division (IAD), and the Advanced Research and Development Activity (ARDA) of the U.S. Department of Defense. Professor Gordon Cormack is organizing the SPAM Track for TREC 2005[109]. The objective of the SPAM Track is to provide a standard evaluation of the current spam-filtering approaches. Results of SPAM Track will be available in November 2005.

The CRM114 Team for TREC (Fidelis Assis, Christian Siefkes, William S. Yerazunis and Shalendra Chhabra) have submitted the following four versions of CRM114 to the SPAM Track, TREC 2005. The algorithms implemented in these filters and the configuration set up are explained in detail in Chapter 7 and Chapter 8.

1. *crmSPAM1osf: CRM114 Orthogonal Sparse Bigram (OSB) Filter*

   The *OSB Filter* is a typical Bayesian classifier implementing Fidelis Assis's Orthogonal Sparse Bigrams (OSB) [105], a feature extraction technique derived from Sparse Binary Polynomial Hashing (SBPH)[127].

   This filter does not use any pre-trained information and the messages are not preprocessed in any way, not even mimedecoded.

2. *crmSPAM2win: CRM114 OSBF Filter*

   The *OSBF Filter* is a typical Bayesian classifier implementing the Orthogonal Sparse Bigrams (OSB) as the feature extraction technique at its front-end and an intuitively derived confidence factor, also known as *voodoo,* for noise reduction and greater accuracy.

   This configuration does not use any pre-trained information and the messages are not preprocessed in any way, not even mimedecoded.

3. *crmSPAM3osu: CRM114 OSB Unique Filter*

   The *OSB Unique* is a typical Bayesian classifier implementing the Orthogonal Sparse Bigrams (OSB) feature extraction technique with the restriction that features are considered only once irrespective of their occurrence in a document.

   This configuration does not use any pre-trained information and the messages are not preprocessed in any way, not even mimedecoded.

4. *crmSPAM4OSB: CRM114 Winnow Filter*

   This filter variation combines the Orthogonal Sparse Bigrams (OSB) feature combination technique with the Winnow algorithm[71] developed by Nick Littlestone. More details on this are available in Chapter 8. This classifier also does not use any pre-trained information and messages are not preprocessed in any way, not even mimedecoded.

5.3 Implementing CRM114 at Mailservers

We will now describe different configuration modes for implementing CRM114 at mailservers. In Section 5.4 we describe a generalized configuration mode for implementing CRM114, which is currently being tested at a medium-sized organization. This is illustrated in Figure 5.1. We are also aware of a large ISP company using CRM114 for filtering more than one million web based email accounts. We present the CRM114 configuration mode for such huge set of email users in Section 5.5.

5.4 A Generalized Configuration Mode for Implementing CRM114 at Mailservers

A generalized configuration mode for implementing CRM114 at mailservers is illustrated in Figure 5.1.

1. *Lookups by the Recipient MTA*

   Upon receiving SMTP connections from Sender/Forwarding MTA, the recipient MTA first performs DNS-based lookups. For example, it matches the IP address of the sender against a Real-time Blackhole List such as the one provided by the Spamhaus. It can also implement authentication mechanisms using SPF (i.e. it will validate the IP address of the connecting client machine obtained during the TCP/IP setup and also specified in the MAIL FROM command against the SPF record for that domain).

2. *MTA Calls MDA*

   Depending upon the local policies, the mail can then be rejected or allowed to pass through. The MTA then calls the MDA *(ex: Procmail, Maildrop)* which is the program responsible for delivering incoming mail to the recipients inbox.

3. *MDA Calls a Series of Filters: sanitizer, clamd, CRM114*

   A MDA like *Procmail* executes a series of filtering actions in a sequential fashion. As illustrated in Figure 5.1, the MDA *Procmail* first calls a filter *sanitizer* which is a tool for preventing attacks via trojans and worms in attachments. The *Procmail* then calls another filter, *clam daemon* (*clamd*). *Clamd* is a tool which scans files against viruses. Note that these filters are just described as an example setup. The administrators can choose any number of filters in any order.

   The next filter in the pipeline is the CRM114 Discriminator for filtering spam. The CRM114 classifies the email stream into three folders: *Good*, *Spam*, and *Unsure*. The MDA *Procmail* then delivers these classified emails into respective folders in the users inbox. As illustrated in Figure 5.1, a user can access these folders by any of the enabled standard mail retrieval program such as Internet Message Access Protocol (IMAP)[63], Post Office Protocol (POP)[97] or via webmail.

4. *Feedback Loop with Human Actions*

   After checking emails in the respective folders, the user can provide feedback by training the system about misclassified messages. This is shown in the oval displaying

Human Action in Figure 5.1. Thus, through human intervention, the system is trained about *good mail misclassified as spam, spam misclassified as good mail, spam in unsure folder* and *good mail in unsure folder*. A *Cron Daemon* can be programmed to train the learning acquired in the feedback loop to to the CRM statistics files of *spam, ham and whitelisted* senders. Ham emails which were misclassified as spam are delivered back in the GOOD folder through *Procmail*.

The CRM statistics files do not store the actual text but rather uses a hashed representation. The hashing references speeds up access per feature (typically under 1 microsecond per feature) and provides a modicum of actual security against snooping attacks since the text is not stored.

5. *Whitelisting through Outbound Mail*

   Also, all outbound email to recipients sent from the senders inside the domain are added to the *whitelist*, and the features in their content are trained in the CRM sparse spectra files for whitelisted senders.

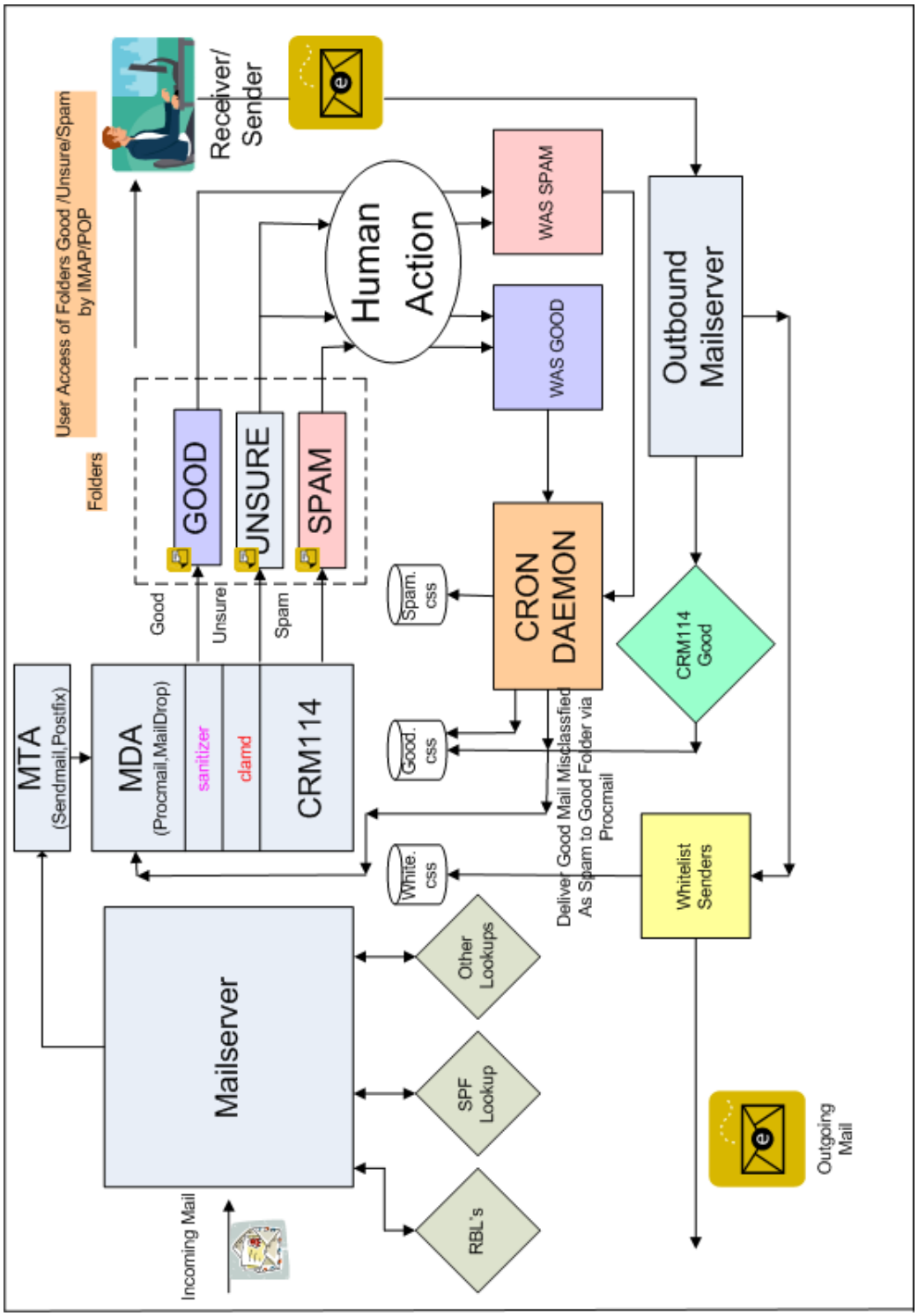


Figure 5.1: A generalized configuration mode for implementing CRM114 at mailservers. Figure drawn in collaboration with Ronald Johnson and William S. Yerazunis from Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA.

5.5 CRM114 Configuration Mode for Large Scale Enterprises

We are aware of a large webmail ISP company using CRM114 for more than one million email accounts of its users for filtering out spam. We now present CRM114 configuration mode for such large scale enterprises. This is illustrated in Figure 5.2.

1. *Look Ups by the Recipient MTA*

   Upon receiving SMTP connections from sender/forwarding MTA, the recipient MTA first performs DNS-based lookups. For example, it matches IP address of the sender against a Real-time Blackhole List (RBL) such as the one provided by the Spamhaus. It can also implement authentication mechanisms using SPF (i.e. it will validate the IP address of the connecting client machine obtained during the TCP/IP setup and also specified in the MAIL FROM command against the SPF record for that domain).

2. *MTA Calls MDA*

   Depending upon the local policies, the mail can then be rejected or allowed to pass through. The MTA then calls the MDA *(example: Procmail, Maildrop)*, which is the program responsible for delivering incoming mail to the recipient's inbox.

3. *MDA Calls the CRM114 Spam Filter*

   A MDA like *Maildrop* executes a series of filtering actions in a sequential fashion. The next filter in the pipeline is CRM114 Discriminator. A precise description of the working stages of CRM114 spam filter follows:

   (a) *CRM114 Matches Sender against Whitelist and Blacklist*

   CRM114 matches all incoming emails against a user's *whitelist* and *blacklist*. If the sender is *whitelisted*, CRM114 stops the evaluation and the email is delivered to the good mail spool. Usually the address book of the users is mirrored as the *whitelist*. Similarly, if the sender is *blacklisted*, the email is delivered to the spam mail spool.

   (b) *CRM114 Computes the Class*

   As illustrated in Figure 5.2, after testing the sender against the *whitelist* and *blacklist*, the CRM114 Discriminator *(set with a conservative decision point)* using its database, classifies the email stream into GOOD and SPAM. Emails are then delivered to the good mail spool and the spam mail spool, respectively.

4. *Training by the Systems Staff*

   A group of *trusted people* from the entire users set forms the training staff. The training staff is around 20 people from the systems staff of the company implementing CRM114 in this configuration mode. The systems staff is shown by the white oval in Figure 5.2. This set of trusted people use and train the global database of spam and good mail as if they were training their own local database. The result is a global database, shared by all users but trained for a broader audience. Care is taken while training, and all training sessions are logged.

   This particular configuration of CRM114 has shown remarkable accuracy in filtering spam in practice.

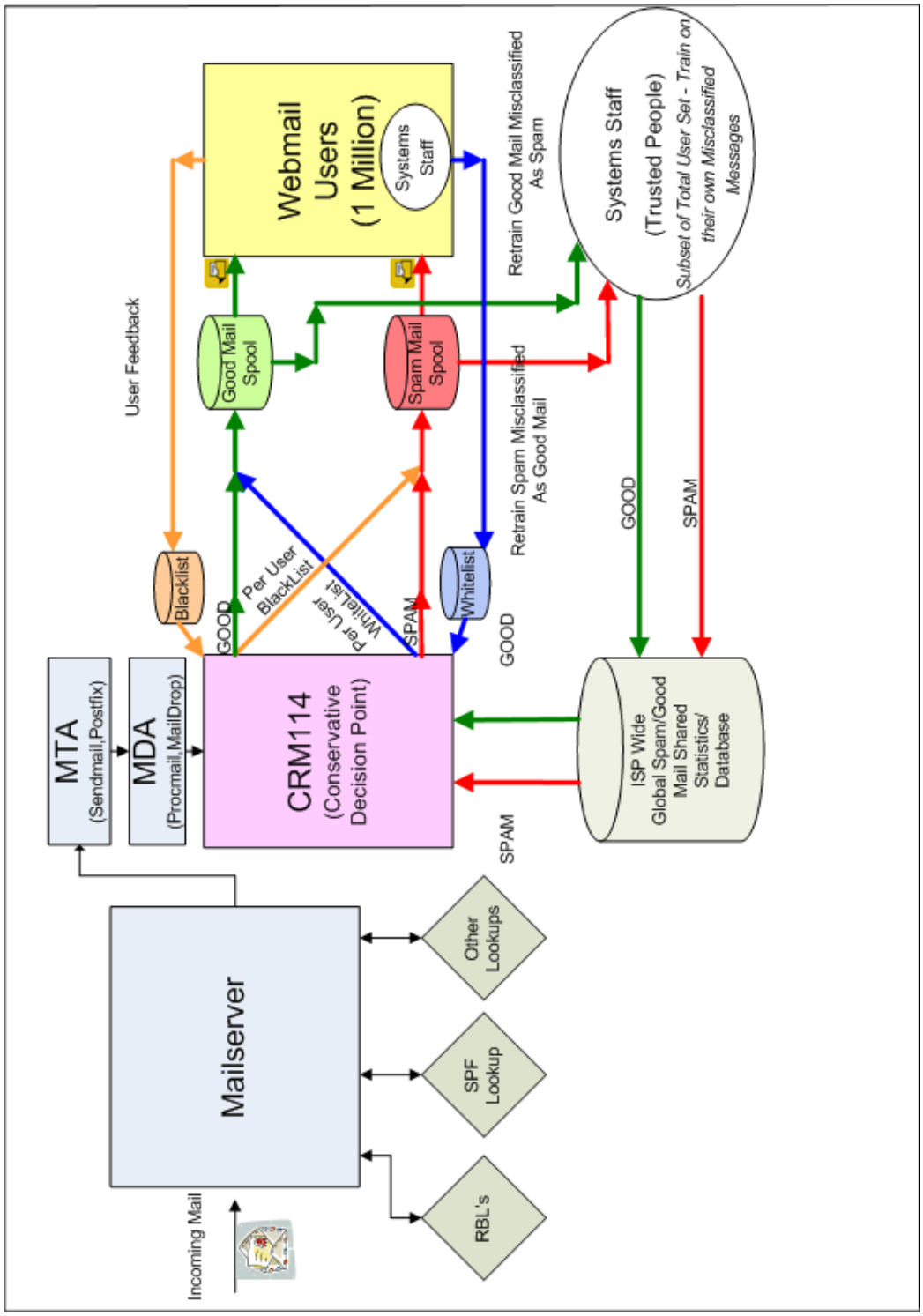


Figure 5.2: CRM114 configuration mode for filtering more than one million client email accounts used by a large ISP. Figure drawn in collaboration with Ronald Johnson, William S. Yerazunis from Mitsubishi Electric Research Laboratories (MERL), and Fidelis Assis from Embratel.

Chapter 6

The CAMRAM System

6.1 Introduction

CAMRAM[23] is an acronym for Campaign for Real Mail. Invented by Eric S. Johansson in 2002, it is a real-world system implementing the *hybrid sender-pays* model for email. A *naïve sender-pays* system sends a stamp to everyone all the time while a *hybrid sender-pays* system sends stamps to a sender the user has never emailed. This difference with *naïve sender-pays* system minimizes the work load on the ordinary user and increases the difference in workload between ordinary user and spammer.

CAMRAM is based on Adam Back's Hashcash[11], which we explained in detail in Chapter 3.

6.2 Architecture of the CAMRAM System

The CAMRAM system consists primarily of three major subsystems—two filter chains, one for outbound messages and one for inbound messages, and a user interface. The outbound filter chain is shown in Figure 6.2, and the inbound filter chain is shown in Figure 6.1. The CAMRAM user interface is shown in Figure 6.3. The CAMRAM system has a *dumpster* folder for all spam and a *spamtrap* folder for unsure mails.

Inbound messages go through a series of filters—*stamp filter, keyword filter, friends list filter* and *content filter*—to determine the class of the message (i.e. to determine if it is a good message or spam). Outbound messages are subjected to the *stamper filter*, and a stamp based on Hashcash[11] is generated for any message recipient not on the friends list; these addresses are fed into the friends database so that any replies are not subjected to the content filter.

The CAMRAM system is currently usable with Postfix MTA only because of the Postfix filter interfaces.

6.3 CAMRAM Inbound Filter

The CAMRAM inbound filtering chain consists of two filter stages, each with a series of filters. The first stage is designed to prevent the entry of spam into the mail system. The second stage is designed to eliminate spam from the message stream once it's in the mail system. These filter stages are described below.

1. *First Stage CAMRAM Filters*

   For the inbound filter chain one instance of the Postfix SMTP server *smtpd.1* is connected to the first stage CAMRAM filters. First stage CAMRAM filters consists of two filters, a *brown listing* filter and a *per address rate limiting* filter.

   (a) *Brown Listing Filter*

   The brown listing component of CAMRAM system provides an escape mechanism for inappropriately blacklisted senders. Brown listing is a slight modification of the blacklist. Any message coming from a blacklisted address is blocked unless the message contains a large stamp. A large stamp is defined as three bits larger than the baseline stamp (i.e. it takes eight times longer to compute). The computational load imposed by such a stamp is only practical if an individual is trying to get through to correct a blacklist problem.

   (b) *Per Address Rate Limiting Filter*

   The per address rate limiting filter is implemented for protecting mailservers and specific addresses on these servers against Denial of Service (DoS) attacks. If the arrival rate for a given address is above a threshold, only messages for that address are rejected with a 4xx SMTP error code. For example, a per address rate limiter message error message will look like this:

   ```
   421 mailbox temporarily disabled, not accepting messages
   ```

   The sending MTA in this case should re-queue the message and try again later.

2. *Message is Queued for Subsequent Filtering*

   After the message passes through these two filters, the source IP address is placed in the message and is then returned to the Postfix SMTP server for further processing, subsequent filtering, and delivery. Recording source IP with messages is useful for constructing a local blacklist if the message is confirmed as a spam message.

3. *Post Queuing Filtering*

   The CAMRAM post queue filtering consists of four filters—*Hashcash stamp filter*, *friends list filter*, *header keyword filter* and *CRM114 filter*. The filtering chain terminates as soon as the message is determined as good or is processed by the CRM114 filter. As soon as a message is determined good, it is passed to the Postfix SMTP server for further processing (i.e. local delivery, or relaying the message to another machine for local delivery).

   (a) *Hashcash Stamp Filter*

   This filter tests for the existence of the stamp. If the stamp is present, it then checks if the stamp is of "sufficient value." Any electronic currency needs protection against *parallel* and *serial* double spending. The Hashcash *double spending database* protects against serial double spending (i.e. detecting the same stamp arriving twice). Using an email address in the resources field protects against parallel double spending. If the email address in the resources field does not match the recipient's email address, the stamp is considered invalid.

(b) *Friends List Filter*

If the sender is in the friends list, the message is passed. The friends list is updated automatically from outbound traffic and the messages approved from the spamtrap. This feature can also be used to train the content filter. Every message that passes the friends list filter is considered good. If this message, when passed to the content filter, is classified as a bad message, the content filter can be trained with known good messages.

(c) *Header Keyword Filter*

This filter matches strings associated with a particular header. This is very useful for passing mailing lists through the inbox or for whitelisting senders.

(d) *CRM114 Filter*

The CRM114 filter is the last filter in the series. It scores the message and then separates it into three bands—green, yellow, and red. Green is automatically passed into the user's inbox, red is passed into a dumpster, and yellow is passed into a spamtrap.

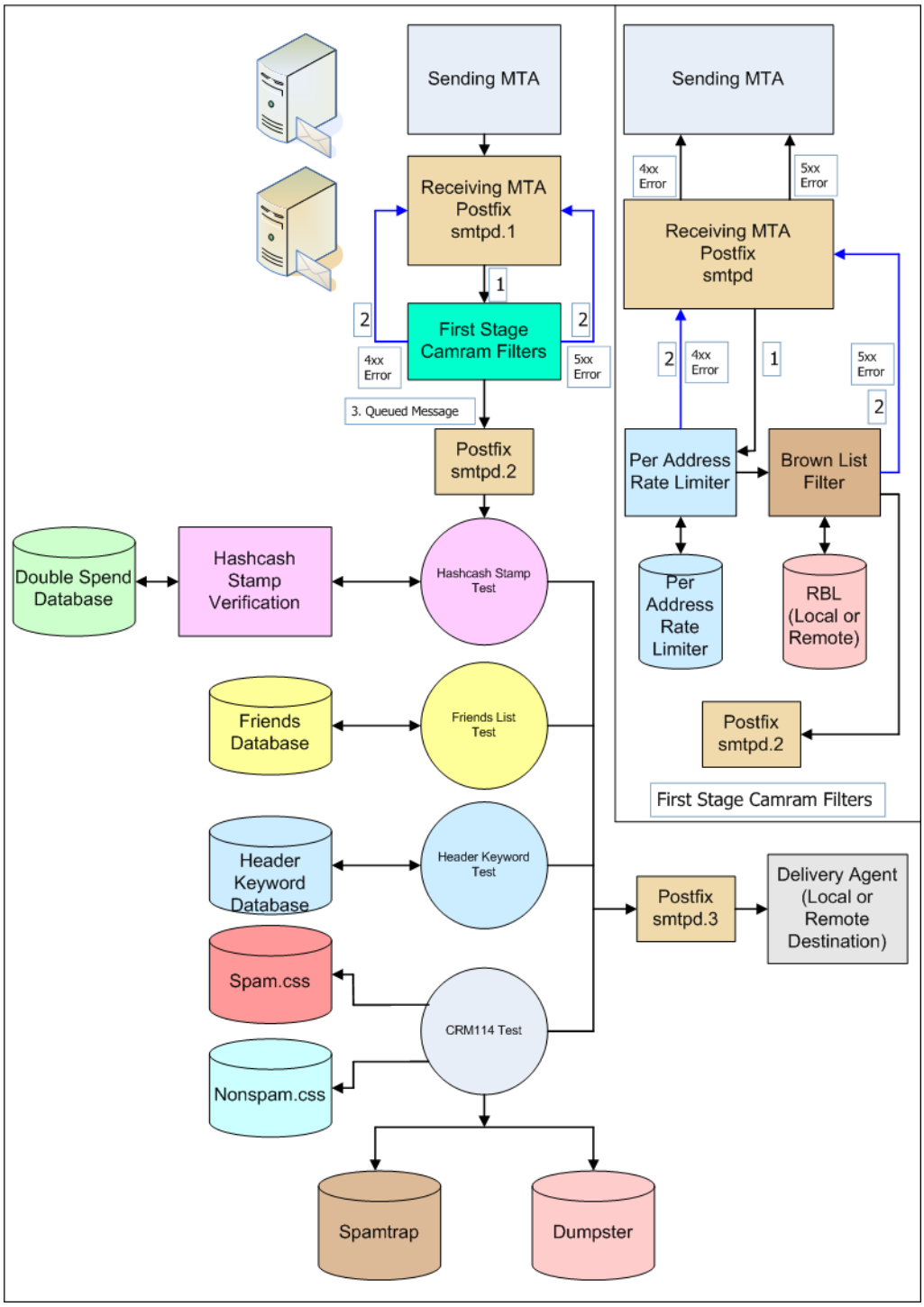


Figure 6.1: CAMRAM inbound filter. Note that the inbound filter chain is composed of first stage CAMRAM filters and four filters—*Hashcash stamp filter, friends list filter, header keyword filter and CRM114 filter*.

6.4 CAMRAM Outbound Filter

We will now explain the CAMRAM outbound filtering chain in detail.

1. *The CAMRAM Queue*

   At the second CAMRAM filter system interface, there is another instance of Postfix SMTP server, *smtpd* running with a filter connected to the post-queuing stage of the processing. As shown in Figure 6.2, messages are taken from the Postfix queue and placed in the CAMRAM queue. A second queue is needed because the stamp processing can take a significant amount of time and this might exceed the filter timeout value of the Postfix.

2. *Splitter Filter*

   After the message is removed from the queue, it is then split into individual messages which are then passed to the Stamper filter.

3. *Stamper Filter*

   Each message is then stamped and delivered. The stamper filter also records the recipient email address in the friends database so that all replies are directly delivered to the sender.

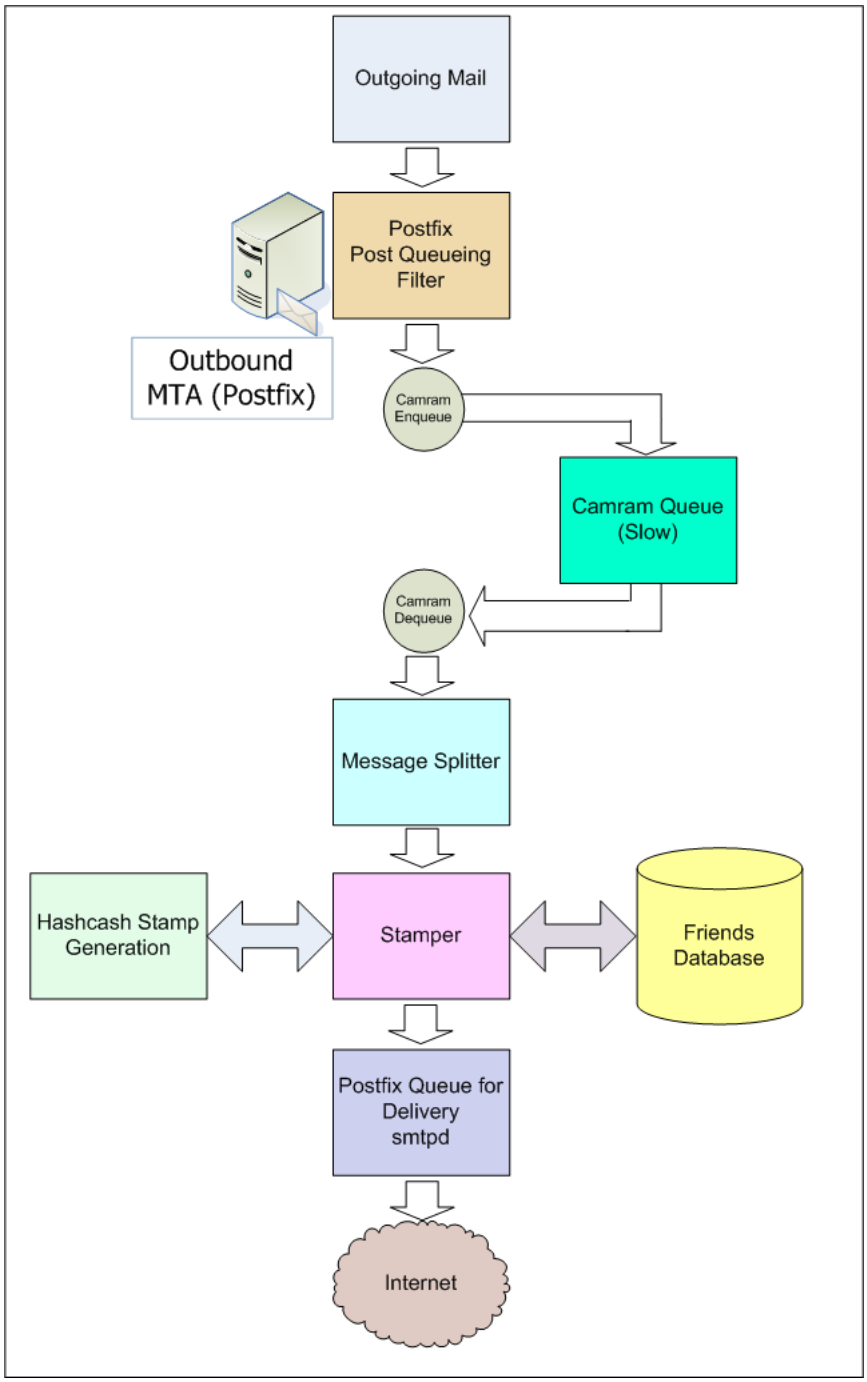


Figure 6.2: CAMRAM outbound filter.

6.5 CAMRAM User Interface

We will now describe the CAMRAM user interface. At the time of this writing, there are two primary user interface channels for CAMRAM, the web and the Mail User Agent (MUA).

1. *The Web Interface to CRM114*

   The web interface has six functionalities—*sort messages, recover, preferences, edit whitelists (keyword and friends)*, and *logout*.

   (a) *Correct.cgi*

   Sort messages functionality is shown as correct.cgi in Figure 6.3. This is the interface for designating messages in the spamtrap as good or bad. Each message in the user interface is color-coded (red/green/yellow), has a visible score, and has a checkbox so that the user can change the color state. If the message state is inconsistent with the score, CRM114 will retrain with the correct state.

   The correct.cgi interface also shows the thresholds between green, yellow, and red. During initial system set-up, the threshold limits for green and red are +/-350. During training, these thresholds are moved together and stop moving when they are +/-10 points apart.

   (b) *Recover.cgi*

   Recover.cgi is the interface for recovering messages from the dumpster. Normally a message placed in the dumpster is considered as spam, and the message expires after five days. However, if the content filter misclassifies a good message as

spam and moves the message to the dumpster (false positive), a recovery process can be used to recover the message.

2. *The MUA Interface to CAMRAM*

   At the time of this writing, the MUA interface to the CAMRAM remains primitive. A message can be dropped into a specified mailbox such as the junk folder. The contents of the mailbox are then harvested and trained as spam using the CRM114 filter, and then thrown into the dumpster. The MUA feedback mechanism works only with IMAP.

6.6 Snapshots of CAMRAM Interfaces

Snapshots of the CAMRAM interfaces are shown in Figure 6.4, Figure 6.5, Figure 6.6, Figure 6.7, Figure 6.8, Figure 6.9, Figure 6.10 and Figure 6.11.

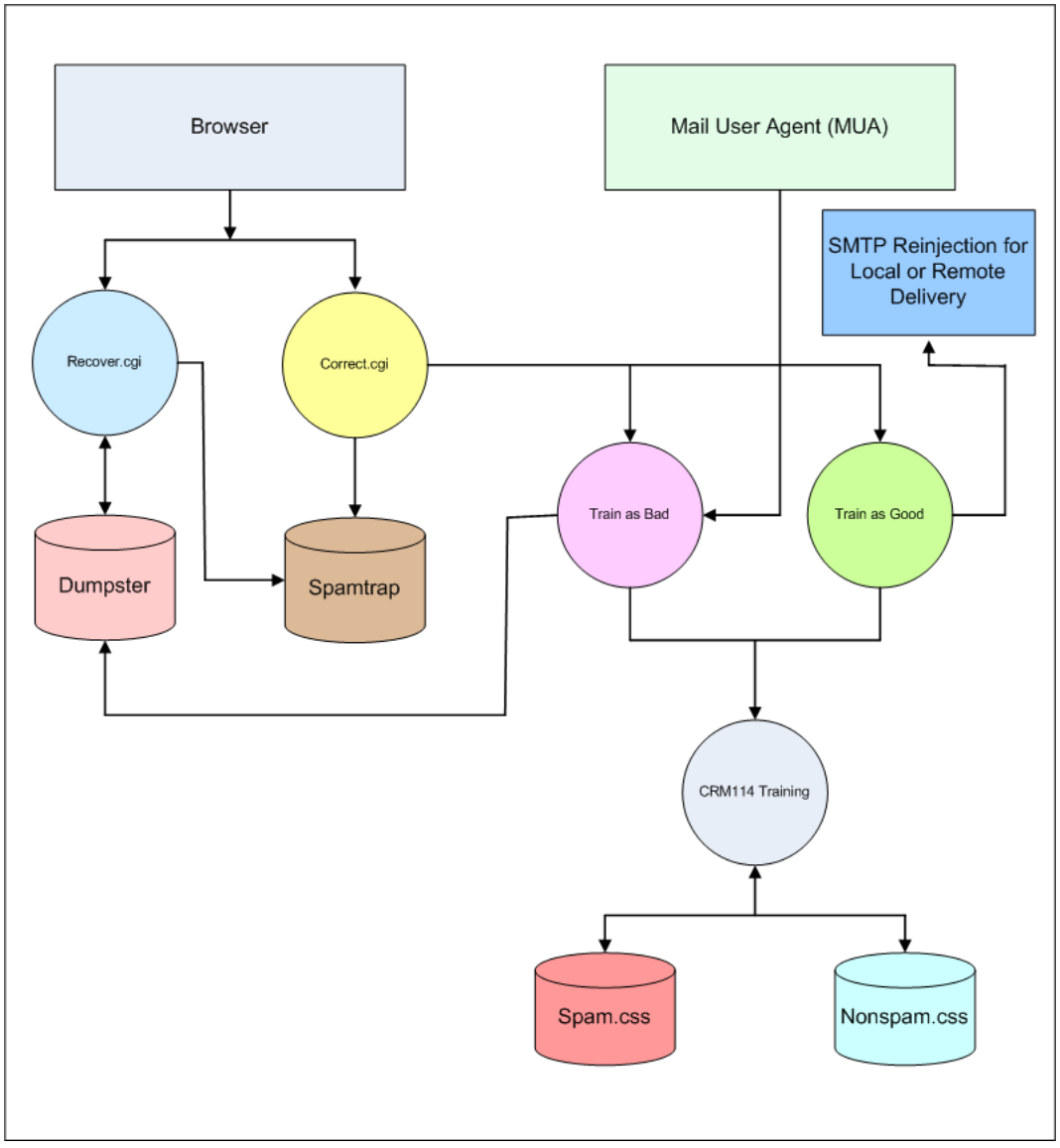


Figure 6.3: CAMRAM user interface.

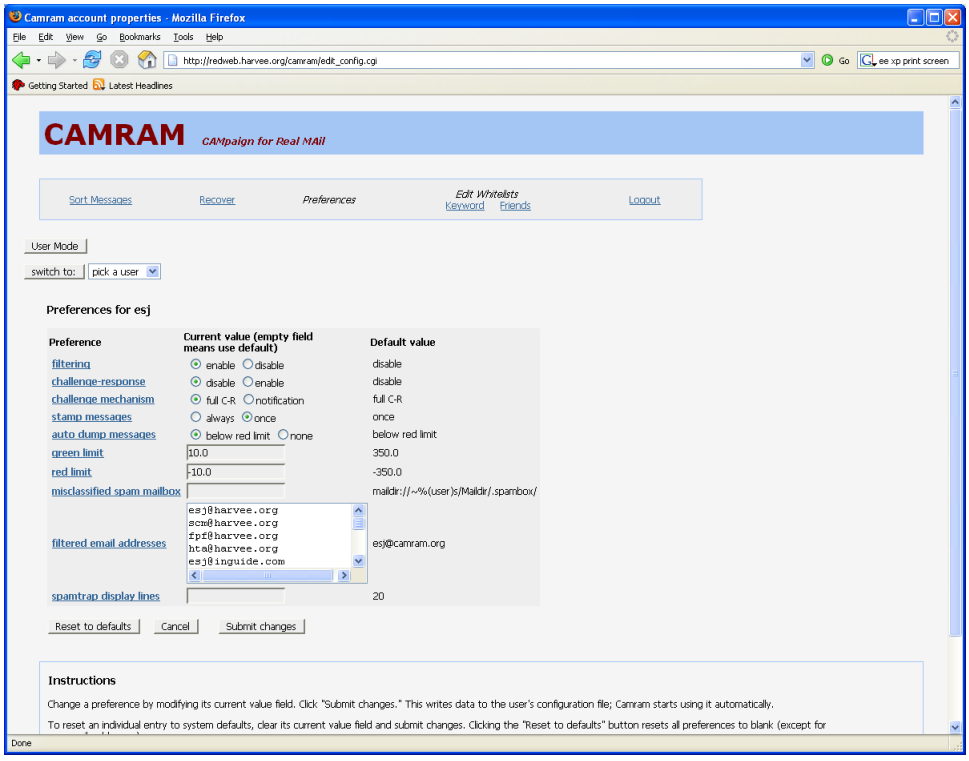


Figure 6.4: CAMRAM user interface displaying configuration settings. Source: Eric S. Johansson, CAMRAM.

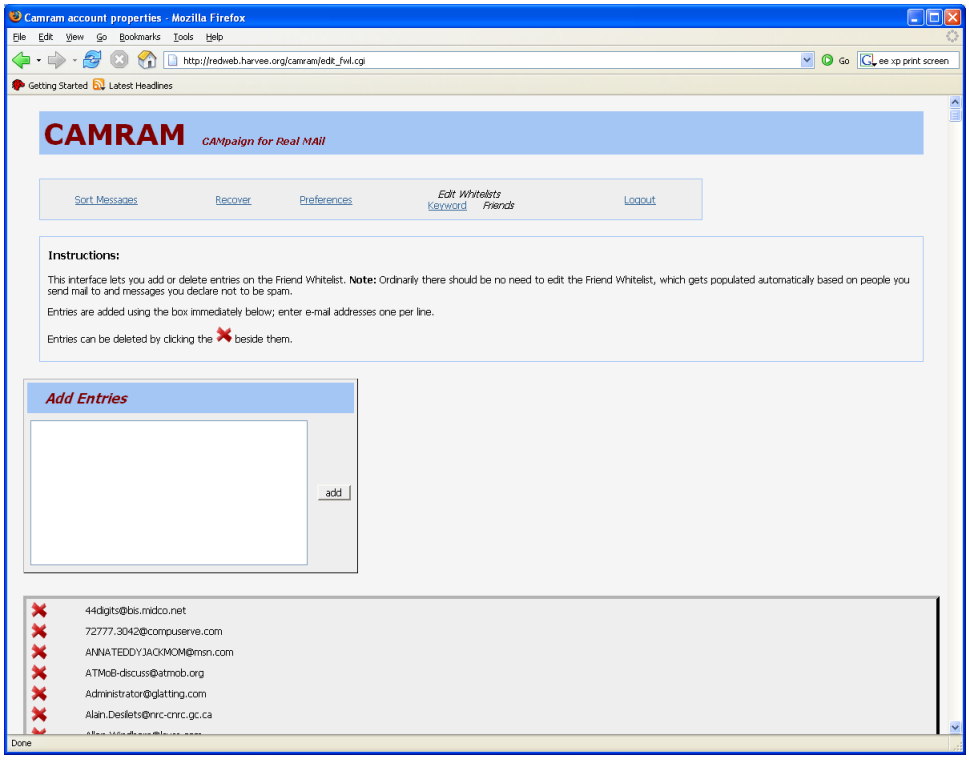


Figure 6.5: CAMRAM user interface displaying a mechanism for adding friends. Source: Eric S. Johansson, CAMRAM.

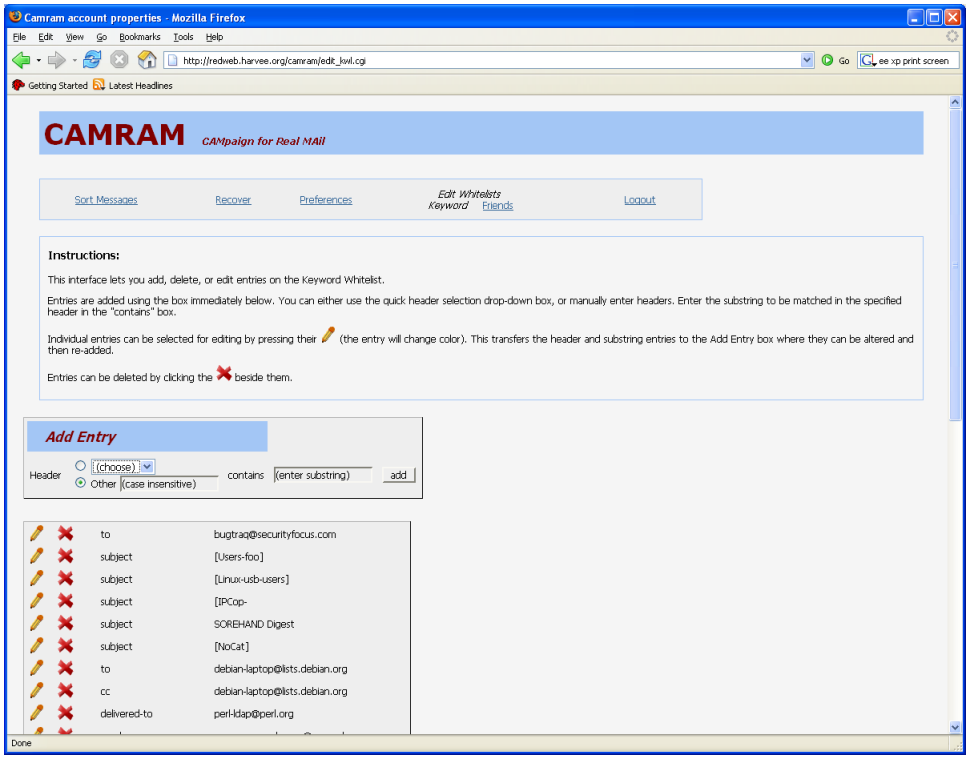


Figure 6.6: CAMRAM user interface displaying a mechanism for adding keywords present in the headers. Source: Eric S. Johansson, CAMRAM.

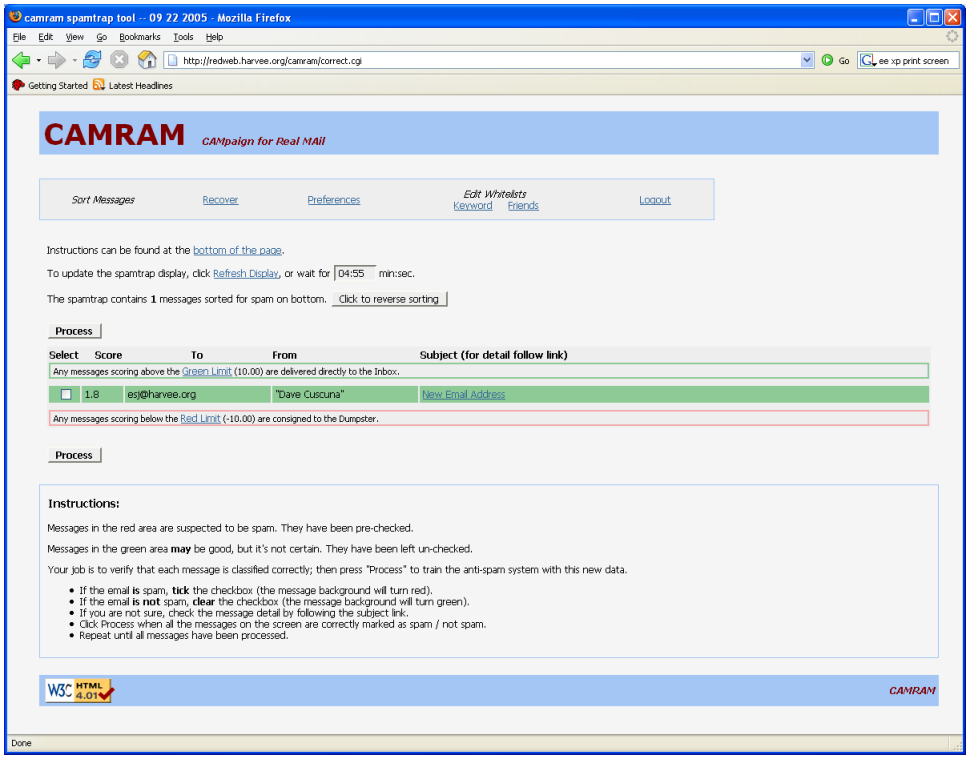


Figure 6.7: CAMRAM user interface displaying the sorting mechanism - 1. Source: Eric S. Johansson, CAMRAM.

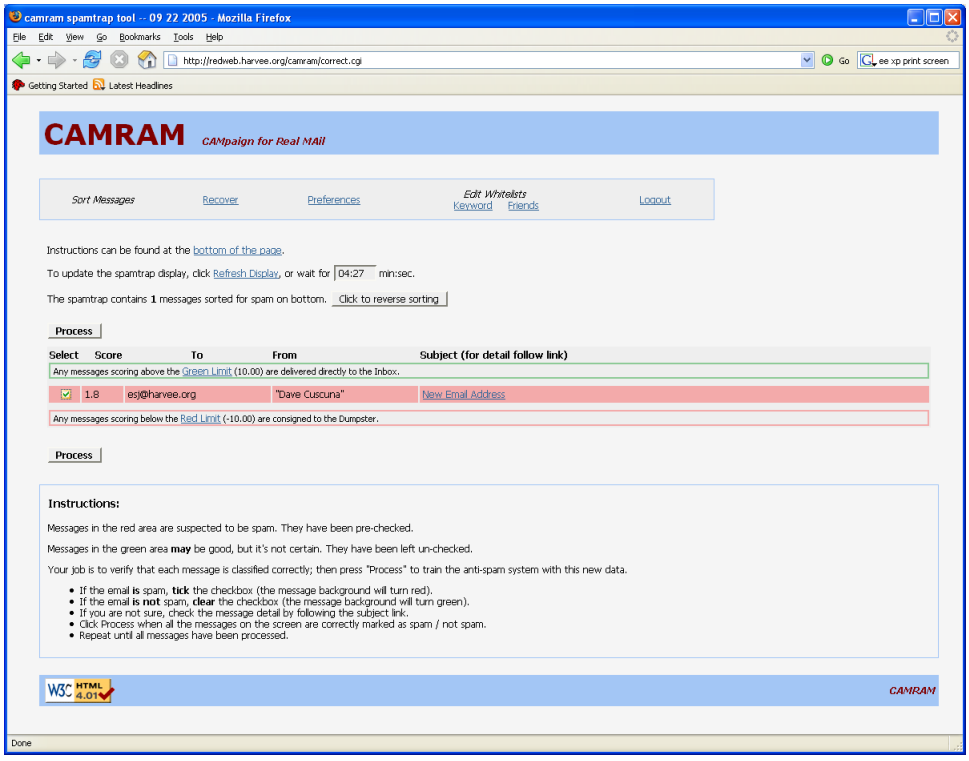


Figure 6.8: CAMRAM user interface displaying the sorting mechanism - 2. Source: Eric S. Johansson, CAMRAM.

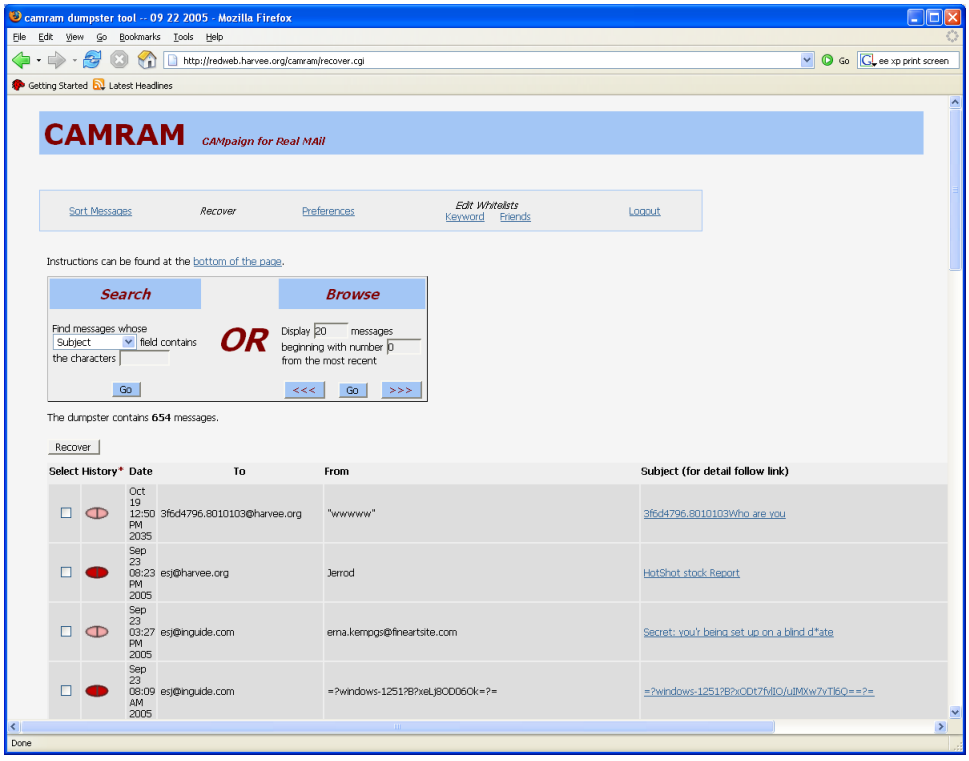


Figure 6.9: CAMRAM user interface displaying the recovery mechanism - 1. Source: Eric S. Johansson, CAMRAM.

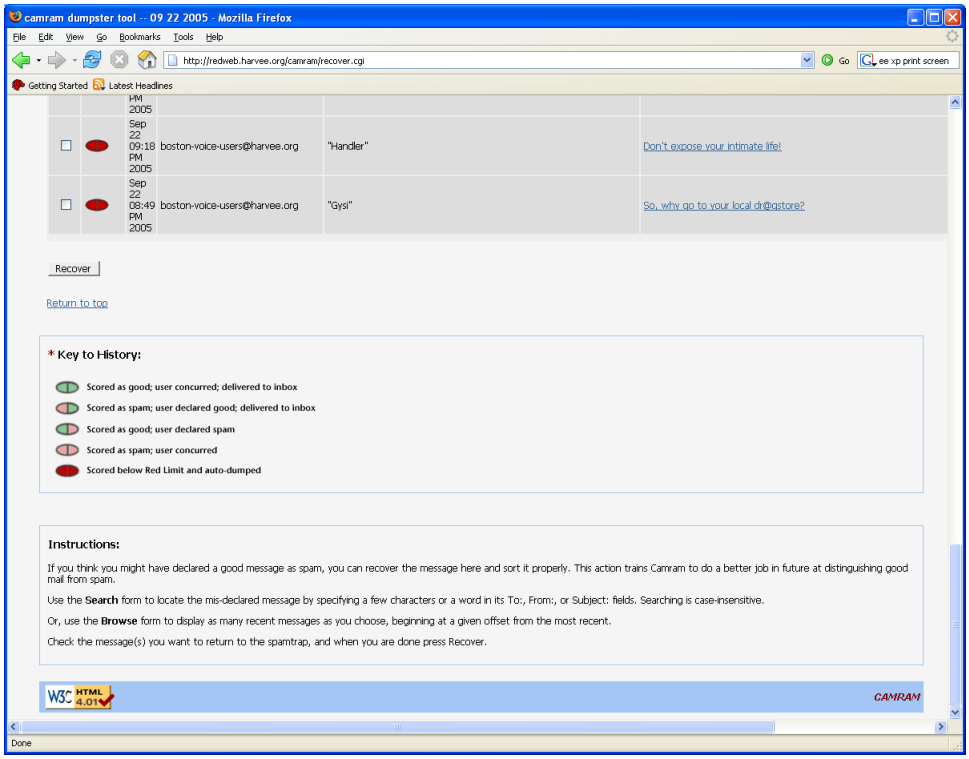


Figure 6.10: CAMRAM user interface displaying the recovery mechanism - 2. Source: Eric S. Johansson, CAMRAM.

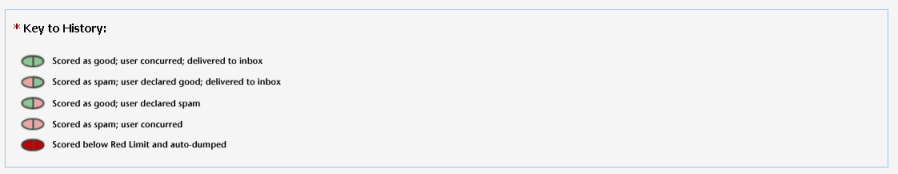


Figure 6.11: CAMRAM user interface - key to history. Source: Eric S. Johansson, CAMRAM.

Chapter 7

Spam Filtering Using a Markov Random Field Model

7.1 Introduction

Spam filtering problem can be seen as a particular instance of the Text Categorization problem (TC), in which only two classes are possible: *spam* and *legitimate email* or *ham*. Today, most of the spam filters implemented at the client side or at the ISP Level are Bayesian Style Spam Filters (with some heuristics) [53][54] violating the basic assumption of the Bayes rule (i.e. they treat words independent of each other). In this chapter, we present spam filtering based on the Markov Random Field Model with different weighting schemes of feature vectors for variable neighborhood of words. We present theoretical justification for our approach and conclude with results.

7.2 Related Work

Spam filtering has been treated as a particular instance of Text Categorization (TC). Much work has already been reported on spam filtering using the traditional classification and statistical techniques such as an application of learning algorithm RIPPER for classification based on TF-IDF weighting (Term Frequency $\times$ Inverse Document Frequency)[32]; Naïve Bayesian approach[100]; Support Vector Machines in comparison to boosting of C4.5 trees, RIPPER and Rocchio[44] etc. Recently Sparse Binary Polynomial Hash (SBPH) filtering technique, a generalization of the Bayesian method[127] and Markovian discrimination[128] have also been reported.

The classifier model in [128] uses empirically derived ad-hoc superincreasing weights. We develop more on [128], correlate it with Markovian Random Field Model, choose variable neighborhood windows for features using Hammersley-Clifford theorem [57] and present different weighting schemas for the corresponding neighborhood window. We implement our scheme in CRM114 filter [35]. Our results reflect the effect of neighborhood relationship among features and provide evidence that such a model is superior than existing Bayesian models used for spam filtering.

7.3 Markov Random Fields

Let $F = \{F_1, F_2 \ldots, F_m\}$ be a family of random variables defined on the discrete set of sites **S**, in which each random variable F_i takes a value f_i in the discrete label set **L**. The family **F**

is called a random field. The notation $F_i = f_i$ denotes the event that F_i takes the value f_i and the notation $(F_1 = f_1, \ldots, F_m = f_m)$ denotes the joint event. A joint event is abbreviated as **F** = *f* where $f = \{f_1, \ldots f_m\}$ is a *configuration* of F, corresponding to a realization of the field. For the label set **L**, the probability that random variable F_i takes the value f_i is denoted by $P(F_i = f_i)$, and abbreviated as $P(f_i)$ and the joint probability is denoted by $P(F = f) = P(F_1 = f_1, \ldots F_m = f_m)$ but abbreviated as *P(f)*. A site in the context of spam classification refers to the relative position of the word in a sequence and a label maps to word values. **F** is said to be a Markov random field on **S** with respect to a neighborhood system **N** if and only if the following two conditions hold:

1. $P(f) > 0, \forall f \in F \qquad (positivity)$

2. $P(f_i|f_{S-\{i\}}) = P(f_i|F_{N_i}) \qquad (Markovianity)$

where $\mathbf{S} - \{i\}$ is the set difference, $f_{S-\{i\}}$ denotes the set of labels at the sites in $\mathbf{S} - \{i\}$ and $f_{N_i} = \{f_i'|i' \in N_i\}$ stands for the set of labels at the sites neighboring i. When the positivity condition is satisfied, the joint probability of any random field is uniquely determined by its local conditional probabilities [18]. The Markovianity depicts the local characteristics of **F**. Only neighboring labels have direct interactions with each other. It is always possible to select sufficiently large $\mathbf{N_i}$ so that the Markovianity holds. The largest neighborhood consists of all other sites. Any **F** is a MRF with respect to such a neighborhood system.

7.4 Markov Random Field Model and CRM114

We have implemented our scheme in CRM114 Filter. Like other binary document classifiers, the CRM114 filter associates a binary class value $S \in \{spam, nonspam\}$ with any given document $\omega = (\omega_1, \ldots, \omega_n)$. As a word context sensitive classifier CRM114 does not treat the input document ω as a bag of independent words, but rather considers all relations between neighboring words to matter, for neighborhoods with variable window size (for example: up to 3, 4, 5, 6 words etc.).

We now derive a possible MRF model based on this neighborhood structure, thereby casting the classification problem as a partial Bayesian inference problem.

Our MRF model consists of a probability measure P defined on a set of configurations Ω. The elements $\omega \in \Omega$ represent all possible documents of interest, with the i-th component ω_i representing the i-th word or token. A random class function C is defined over Ω, $C : \Omega \rightarrow \{spam, nonspam\}$, such that C indicates the class of the document, and whose law is given by P.

In this framework, the document classification problem can be treated as the problem of computing the probability $P(C(w) = spam|\omega)$, or more precisely for MAP estimation (i.e. maximum a posteriori). The optimal class s^* is chosen as:

$$s^* = \arg\max_{s \in S} P(C(w) = s|\omega).$$

Let us define a k-neighborhood consisting of k consecutive token positions in a document. The cliques are defined to be all possible subsets of a neighborhood.

Thus if $\omega = (\omega_1, \ldots, \omega_n)$ is a document, then the first 3-neighborhood is $\{1, 2, 3\}$, and the associated cliques are $\{1\}$, $\{2\}$, $\{3\}$, $\{1, 2\}$, $\{1, 3\}$, $\{2, 3\}$.

We assume that the measure P is an exponential form Markov Random Field conditioned on $C(w)$. This postulate is natural in view of the characterization by Hammersley and Clifford [57], in terms of conditional distributions on cliques. The functional form of P is therefore fixed as follows:

$$P(\omega|C(\omega) = s) = Z_s^{-1} \exp\Big(\sum_i V_i^s(\omega_i) + \sum_{ij} V_{ij}^s(\omega_i, \omega_j) +$$

$$\cdots + \sum_{i_1,\ldots,i_k} V_{i_1,\ldots,i_k}^s(\omega_{i_1}, \ldots, \omega_{i_k})\Big),$$

where Z_s is the appropriate normalizing constant which guarantees that $\sum_\omega P(\omega|C(\omega)) = 1$ when summed over all possible documents ω. By the Hammersley and Clifford [57] characterization of MRFs, the functions V are nonzero if and only if the indices form a clique.

We can identify the conditional MRF with a specific CRM114 instance by assigning the required V functions from the local probability formulas. For example,

$$\log V_{ij}^s(\omega_i, \omega_j) = \Pi_{ij}(\omega, s),$$

where $\Pi_{ij}(\omega, s) = (\textit{local probability for } (\omega_i, \omega_j), \textit{given } C(\omega) = s)$. $\Pi_{ij}(\omega, s)$ cannot be

interpreted directly as conditional probabilities, however an easy product form solution is obtained i.e.

$$P(\omega|C(\omega) = s) = Z_s^{-1} \prod_{cliques\ c} \Pi_c(\omega, s)$$

which is quite different from a Naïve Bayesian model. In the special case of neighborhoods with $k = 1$, this reduces to a Naïve Bayesian model.

With this solution, Bayes' rule can be applied to obtain the class probability, given a document:

$$P(C(\omega) = s|\omega) = \frac{P(\omega|C(\omega) = s)P(s)}{P(\omega)}.$$

In the Bayesian framework, the two unknowns on the right are $P(s)$ (the prior) and $P(\omega)$. Normally, $P(\omega)$ is ignored, since it does not influence the MAP estimate, but it can also be expanded in the form:

$$P(\omega) = Z_{spam}^{-1} \prod_{cliques\ c} \Pi_c(\omega, spam)P(s)+$$

$$Z_{nonspam}^{-1} \prod_{cliques\ c} \Pi_c(\omega, nonspam)(1 - P(spam))$$

The Z^{-1} terms are the factors necessary to compensate for the inter-word dependence that a Naïve Bayesian Model normally ignores. While Z^{-1} terms cannot be computed exactly, a lower bound based on the neighborhood structure of the clique can be set and a possible value can be approximated. However, this approximate Markov Random Field Z^{-1}, bound merely guarantees the ability of the learning system to compensate for the inter-word dependence.

7.5 Features Vectors in the Chosen Neighborhood

We now present a few possible approximations and bounds on Z^{-1} such that the learning algorithms can compensate for inter-word dependence. This is done by re-defining the learnable features to be both single tokens and groupings of sequential tokens, and by varying the length of the grouping window. By forcing the shorter groupings of sequential tokens to have smaller Z^{-1} weightings, the inter-word dependence of natural language can be compensated. The larger groupings have greater clique potentials and the smaller groupings have smaller clique potentials. A secondary effect which is present is that if the Z^{-1} weights are superincreasing, then the classifier becomes a nonlinear classifier and is not bound by the limits of the Perceptron theorem. This superincreasing classifier can cut the feature hyperspace along a curved (and possibly disconnected) surface, in contrast to a linear Bayesian classifier that is limited to a flat hyperplane [80]. We now propose some Z^{-1} weighting schemes with superincreasing weights.

Let n-*sequence* denote a feature containing n sequential nonzero tokens, not separated by placeholders; n-*term* denotes a feature containing n nonzero tokens, ignoring placeholders; e.g. "A B C" would be a 3-sequence and a 3-term, "A B ? D" would be a 3-term, but not a sequence.

For each n-sequence, there are $Num(n) = 2^n - 2$ subterms ("-2" because the "empty feature"—only placeholders, no nonzero token—and the n-sequence are ignored).

The number of subterms with k tokens is given by the binomial coefficient: $Num(n,k) = \binom{n}{k}$, for $0 < k < n$. (For $k = 0$ and $k = n$ this also holds and yields the empty feature resp. the n-sequence itself: $\binom{n}{0} = \binom{n}{n} = 1$)

The weight $W(n)$ of a n-sequence should be larger then the weight of all subterms considered for this sequence. Thus $W(n)$ must be greater than the sum of the number of k-subterms $Num(n,k)$ times the weight of k-terms $W(k)$, for $0 < k < n$. This is guaranteed if:

$$W(n) > \sum_{k=1}^{n-1} \left(\binom{n}{k} \times W(k) \right) \tag{7.1}$$

Minimum Weighting Sequences: The minimum weighting scheme for a superincreasing set of Z^{-1} weights, can be evaluated as:

$$W(n) = \sum_{k=1}^{n-1} \left(\binom{n}{k} \times W(k) \right) + 1 \tag{7.2}$$

The resulting (considering that $\binom{n}{k} = \frac{n!}{k! \times (n-k)!}$) weighing sequences are shown in Table 7.1.

n	**Sequence**
1	1
2	1, 3
3	1, 3, 13
4	1, 3, 13, 75
5	1, 3, 13, 75, 541
6	1, 3, 13, 75, 541, 4683

Table 7.1: Minimum weighting sequences.

Exponential Weighting Sequences: Assuming a constant uncertainty and thus constant incremental information content per sequential token in the input text, an exponentially increasing Z^{-1} weighting model results. Unfortunately, for any particular fixed base, this exponential model is only superincreasing (in the sense described above) for a limited window length, and thus fails for longer window lengths. For any given window length, the exponential weighting $W(k) = base^{k-1}$ can be evaluated as:

$$base^{n-1} > \sum_{k=1}^{n-1} \left(\binom{n}{k} \times base^{k-1} \right) \tag{7.3}$$

Adding $base^{n-1}$ on both sides (considering that $\binom{x}{x} = 1$); multiplying both sides with *base* and adding 1 yields (considering that $\binom{x}{0} = 1$ and $x^0 = 1$):

$$2 \times base^n + 1 > \sum_{k=0}^{n} \left(\binom{n}{k} \times base^k \right) \tag{7.4}$$

Applying the Binomial Theorem : $(a+b)^n = \sum_{i=0}^{n} \left(\binom{n}{i} a^{n-i} b^i \right)$ and setting $a = 1$, $b = base$ we get:

$$2 \times base^n + 1 > (base + 1)^n \tag{7.5}$$

For $base = 2$ this postulate only holds for up to $n = 1$; for $base = 3$ and $base = 4$ up to $n = 2$; for $base = 5$ up to $n = 3$; for $base = 6$ up to $n = 4$; for $base = 7$ up to $n = 5$; for $base = 8$ up to $n = 6$.

The resulting exponential weighting schemas are shown in Table 7.2.

n	Sequence
1	1
2	1, 3
3	1, 5, 25
4	1, 6, 36, 216
5	1, 7, 49, 343, 2401
6	1, 8, 64, 512, 4096, 32768

Table 7.2: Exponential weighting sequences.

7.6 Training and Prediction using CRM114

7.6.1 Testing Procedure

In order to test our multiple hypothesis, a standardized spam/nonspam test set from SpamAssassin [110] was used. This test set is extraordinarily difficult to classify, even for humans. It consists of 1397 spam messages, 250 hard nonspams, and 2500 easy nonspams, for a total of 4147 messages. These 4147 messages were then shuffled into ten different standard sequences.

A full test set for TOE (Train Only Errors) was performed with all memory in the learning system initialized to zero. The learning system was then presented with each member of a standard sequence, in the order specified for that standard sequence, required to classify the message. If the classification was incorrect, the learning system under test was trained on the message in the correct category. The training system then moved on to the next message in the standard sequence. The first $4147 - 500 = 3647$ messages formed the "training set",

and the final 500 messages of each standard sequence formed the "testing set" used for final accuracy evaluation. At no time a system ever had the opportunity to learn on a "testing set" message before final accuracy evaluation; however systems were permitted to train on errors made in the final 500 "testing" messages. The final score was the total number of incorrect classifications made by each classifier in the $500 \times 10 = 5000$ testing messages.

This process of zeroing memory, training 3647 "training set" messages, then testing with the remaining 500 previously unseen "testing set" messages was repeated for each of the ten standard sequences. Each set of ten standard sequences (41470 messages) with TOE learning required approximately 2.5 hours of processor time on a TransMeta 933 MHz laptop.

7.6.2 Models Tested

Four different Z^{-1} weighting methodologies for differential evaluation of increasingly long matches were tested. These models correspond to increasingly accurate descriptions of known situations in the Markov Field Model.

The first model tested was Sparse Binary Polynomial Hashing (SBPH), which uses a constant weighting of 1.0 for all matches, irrespective of the length. With a window length of 1, SBPH is identical to the common Naïve Bayesian model without discarding any features as "too uncommon" or "too ambivalent". Testing showed that best results occurred when the maximum window length was five tokens. The second model tested was the Exponential

Superincreasing Markovian model (ESM), which uses an empirically-derived formula that yields weights of 1, 4, 16, 64, 256, and 1024 for matches of one, two, three, four, five, and six words, respectively.

The third model tested was the Minimum Weighting System (MWS) model. This model uses the minimum weight increase necessary to assure that a single occurrence of a feature of length N words can override a single occurrence of all of its internal features (that is, all features of lengths $1, 2, \ldots, N-1$). This is a different notion than the superincreasing ESM model, and produces weights of 1, 3, 13, 75, 541 and 4683 as deduced above. This model is also the minimum set of weights necessary to produce a weighting capable of exceeding the Perceptron limitation and the minimum weighting capable of computing an XOR, so any model exceeding the MWS model is also capable of computing XOR and is not bound by the Perceptron limit.

The fourth model tested uses a variable base to form an exponential series (ES), with a base chosen to assure that the values are always above the MWS threshold for any value of window length used. For our tests, we used a base of N = 8, yielding weights of 1, 8, 64, 512, 4096, and 32768.

A summary of the term weighting length is shown in Table 7.3.

Model	**Weighing Sequence**
SBPH	1, 1, 1, 1, 1, 1
ESM	1, 4, 16, 64, 256, 1024
MWS	1, 3, 13, 75, 541, 4683
ES	1, 8, 64, 512, 4096, 32768

Table 7.3: A summary of tested models with their weighting sequences.

Note that ESM is not above the minimum weight of MWS for features of four words or longer, implying that ESM cannot compute XORs of more than three terms. An example of the weighting model used for these tests is presented in the Table 7.4. Here, the phrase "Do you feel lucky today?" is broken into a series of subfeatures, and the respective weightings given to those subfeatures in each of SBPH, ESM, MWS, and ES are shown. These weights are used as multiplicative factors when calculating the local probability of each subfeature as it is evaluated in an otherwise-conventional Bayesian Chain Rule evaluation. The local probability of each class in CRM114 is given by

$$P = 0.5 + (((f_c * w) - (f'_c * w))/(m * ((f_{totalhits} * w) + n))) \tag{7.6}$$

where f_c is the number of feature hits in this class, f'_c is the number of feature hits in the other class, $f_{totalhits}$ is the number of the total hits and w is the weight. In our implementation, $m = 16$ and $n = 1$. These experimentally determined constants generate probabilities close enough to 0.5 so as to avoid numerical underflow errors even over thousands of repeated applications of the Bayesian Chain Rule.

Sub Feature	Text	SBPH	ESM	MWS	ES
0	Do	1	1	1	1
1	Do you	1	4	3	8
2	Do$< skip >$feel	1	4	3	8
3	Do you feel	1	16	13	64
4	Do$< skip >< skip >$lucky	1	4	3	8
5	Do you $< skip >$ lucky	1	16	13	64
6	Do $< skip >$ feel lucky	1	16	13	64
7	Do you feel lucky	1	64	75	512
8	Do $< skip >< skip >< skip >$ today?	1	4	3	8
9	Do you $< skip >< skip >$today?	1	16	13	64
10	Do $< skip > feel < skip >$today?	1	16	13	64
11	Do you feel $< skip >$ today?	1	64	75	512
12	Do $< skip >< skip >$ lucky today?	1	4	3	8
13	Do you $< skip >$ lucky today?	1	64	75	512
14	Do $< skip >$ feel lucky today?	1	64	75	512
15	Do you feel lucky today?	1	256	541	4096

Table 7.4: Example subphrases and relative weights with the models tested.

7.6.3 Test Results

As a matter of convenience, the feature slots available to each of the implementations were limited to one million features. If feature memory was nearing exhaustion a randomly chosen low-count feature was deleted. All of the variations ran with the same feature elimination algorithm. The only elements of the software that were varied were the window length parameter and the term weighting length table. All four models with a window length of 1 are exactly equivalent to each other and to a pure Bayesian model as postulated by Graham [53]. Each of these advanced models is also more accurate than a pure Bayesian model in every window length > 1.

The results are shown in the Table 7.5 (in errors per 500x$10 = 5000$ final test messages, smaller numbers means better performance).

Win	**1**	**2**	**3**	**4**	**5**	**6**
SBPH	101	72	70	69	66	76
% A	97.98	98.56	98.6	98.62	98.68	98.48
ESM	101	77	67	62	56	86
% A	97.98	98.46	98.66	98.76	98.88	98.28
MWS	101	78	64	62	60	87
% A	97.98	98.44	98.72	98.76	98.80	98.26
ES	101	76	71	84	60	92
% A	97.98	98.48	98.58	98.32	98.80	98.16

Table 7.5: Errors and accuracy (% A) per 5000 test messages with varying window sizes (Win).

7.6.4 Discussion

From Figure 7.1 it is evident that even though ESM has a theoretical weakness due to coefficients less than the MWS for window lengths of four or greater, it has the best accuracy for all tested configurations. The increased error rates for all systems as the window length increased from 5 to 6 is not surprising — as at this point the systems exceeds the one-million-feature slot limit causing seldom-used features to be deleted. Since these seldom-used features were actually the high-weighted features, memory reclamation causes considerable increase in the error rate for long window length Markovian Random Field classifiers.

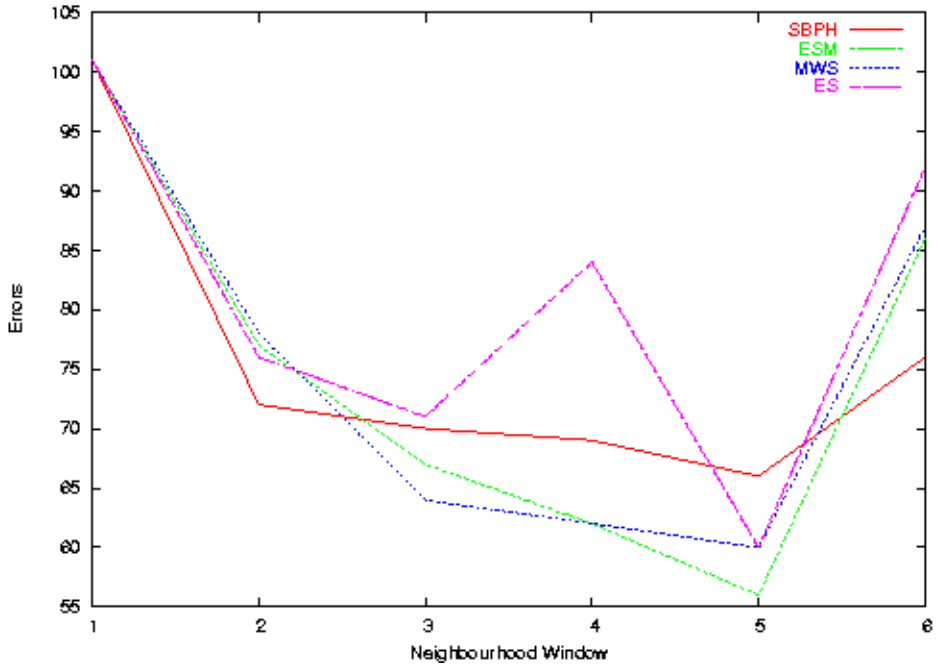


Figure 7.1: Comparison of errors in the tested models with variable neighborhood windows.

7.7 Conclusion and Future Work

We have derived a generalized form of weighting schemas for the classifiers with super-increasing weights. The weighting sequences define a set of clique potentials, where the neighborhood of a single word is given by the words surrounding it. For a neighborhood window of size 2, “pairwise only dependence” by [18] is reflected.

Determining a generalized optimal window size may be the subject of future work. An interesting direction of future research is the combination of Sparse Binary Polynomial Hashing (SBPH) feature combination technique with other learning algorithms. Recently we have obtained significant improvements by combining a closely related feature combination technique with a variant of the Winnow algorithm [71] using TIES [120] [105].

Chapter 8

Combining Winnow and Orthogonal Sparse Bigrams for Incremental Spam Filtering

8.1 Introduction

Spam filtering can be viewed as a classic example of a text categorization task with a strong practical application. While keyword, fingerprint, whitelist/blacklist, and heuristic–based filters such as SpamAssassin[110] have been successfully deployed, these filters have experienced a decrease in accuracy as spammers introduce specific countermeasures. The current best-of-breed anti-spam filters are all probabilistic systems. Most of them are based on Naïve Bayes as described by Graham[54] and implemented in SpamBayes[111]; others such as the

CRM114 Discriminator can be modeled by a Markov Random Field[30, 128]. Other approaches such as Maximum Entropy Modeling[131] lack a property that is important for spam filtering—they are not *incremental*, they cannot adapt their classification model in a single pass over the data.

As a statistical, but non-probabilistic alternative we examine the incremental *Winnow* algorithm. Our experiments show that Winnow reduces the error rate by more than 75% compared to Naïve Bayes and by more than 50% compared to CRM114.

The feature space considered by most current methods is limited to individual tokens (unigrams) or bigrams. The *Sparse Binary Polynomial Hashing (SBPH)* technique (cf. Sec. 8.4.1) introduced by CRM114 is more expressive but imposes a large runtime and memory overhead. We propose *orthogonal sparse bigrams (OSB)* as an alternative that retains the expressivity of SBPH, but avoids most of the cost. Experimentally OSB leads to equal or slightly better filtering than SBPH. We also analyze the preprocessing and tokenization steps and find that further improvements are possible here.

In the next section we present the Winnow algorithm. The following two sections are dedicated to feature generation and combination. In Section 8.5 we detail our experimental results. Finally we discuss related methods and future work.

8.2 The Winnow Classification Algorithm

The Winnow algorithm introduced by [71] is a statistical, but not a probabilistic algorithm, i.e. it does not directly calculate probabilities for classes. Instead it calculates a *score* for each class.[1]

Winnow keeps an n-dimensional weight vector $w^c = (w_1^c, w_2^c, \ldots w_n^c)$ for each class c, where w_i^c is the weight of the ith feature. The algorithm predicts 1 for a class iff the summed weights (called the score Ω) surpass a predefined threshold θ:

$$\Omega = \sum_{j=1}^{m} w_j^c > \theta.$$

Otherwise ($\Omega \leq \theta$) the algorithm predicts 0. $m \leq n$ is the number of active (present) feature in the instance to classify.

The goal of the algorithm is to learn a linear separator over the feature space that predicts 1 for the true class of each instance and 0 for all other classes on this instance. The initial weight of each feature is 1.0. Weights are updated whenever the prediction for a class is wrong.

If 0 is predicted instead of 1, the weights of all active features are increased by multiplying them with a *promotion factor* α, $\alpha > 1$: $w_j^c \leftarrow \alpha \times w_j^c$. If 1 is predicted instead of 0, the active weights are multiplied with a *demotion factor* β, $0 < \beta < 1$: $w_j^c \leftarrow \beta \times w_j^c$.

[1] There are ways to convert the scores calculated by Winnow into confidence estimates, but these are not discussed here since they are not of direct relevance for the purpose of this chapter.

In text classification, the number of features depends on the length of the text, so it varies enormously from instance to instance. Thus instead of using a fixed threshold we set the threshold to the number m of features that are active in the given instance: $\theta = m$. Thus initial scores are equal to θ since the initial weight of each feature is 1.0.

In multi-classification, where an instance can belong to several classes at once, the algorithm would predict all classes whose result is higher than the threshold. But for the task at hand, there is exactly one correct class for each instance, thus we employ a *winner-takes-all* approach where the class with the highest score is predicted.

This means that there are situations where the algorithm will be trained even though it did not make a mistake. This happens whenever the scores of both classes[2] are at the same side of the threshold and the score of the true class is higher than the other one—in this case the prediction of Winnow will be correct but it will still promote/demote the weights of the class that was at the wrong side of the threshold.

The complexity of processing an instance depends only on the number of active features n_a, not on the number of all features n_t. Similar to *SNoW* [24], a sparse architecture is used where features are allocated whenever the need to promote/demote them arises for the first time. In sparse Winnow, the number of instances required to learn a linear separator (if exists) depends linearly on the number of relevant features n_r and only logarithmically on the number of active features, i.e. it scales with $O(n_r \log n_a)$ (cf. [84, Sec. 2]).

[2] resp. two or more classes in other tasks involving more than two classes

Winnow is a non-parametric approach; it does not assume a particular probabilistic model underlying the training data. Winnow is a linear separator in the Perceptron sense, but by providing a feature space that itself allows conjunction and disjunction, complex non-linear features may be recognized by the composite feature-extractor + Winnow system.

8.2.1 Thick Threshold

In our implementation of Winnow, we use a *thick threshold* for learning (cf. [36, Sec. 4.2]). A *thick threshold* causes a training instance to be re-trained even if the classification was correct if the classifier result was near the threshold. Two additional thresholds θ^+ and θ^- with $\theta^- < \theta < \theta^+$ are defined and each instance whose score falls in the range $[\theta^-, \theta^+]$ is considered a mistake. In this way, a large margin classifier will be trained.

8.2.2 Feature Pruning

The feature combination methods discussed in Section 8.4 generate enormous numbers of features. To keep the feature space tractable, features are stored in an LRU (Least Recently Used) cache. The feature store is limited to a configurable number of elements; whenever it is full, the least recently seen feature is deleted. When a deleted feature is encountered again, it will be considered as a new feature whose weights are still at their default values.

8.3 Feature Generation

8.3.1 Preprocessing

In our tests, we did not perform language-specific preprocessing techniques such as word stemming, stop word removal, or case folding. We did compare three types of email-specific preprocessing.

- Preprocessing via *mimedecode*, a utility for decoding typical mail encodings (Base64, Quoted-Printable etc.)
- Preprocessing via Jaakko Hyvätti's *normalizemime* [87]. This program converts the character set to UTF-8, decoding Base64, Quoted-Printable and URL encoding and adding warn tokens in case of encoding errors. It also appends a copy of HTML/XML message bodies with most tags removed, decodes HTML entities and limits the size of attached binary files.
- No preprocessing. Use the raw mail including large blocks of Base64 data in the encoded form.

Expect for the comparison of these alternatives, all experiments were performed on *normalizemime*-preprocessed mails.

8.3.2 Tokenization

Tokenization is the first stage in the classification pipeline; it involves breaking the text stream into tokens ("words"), usually by means of a regular expression. We tested four different tokenization schemas:

P (Plain): Tokens contain any sequences of printable characters; they are separated by non-printable characters (whitespace and control characters).

C (CRM114): The current default pattern of CRM114—tokens start with a printable character; followed by any number of alphanumeric characters + dashes, dots, commas and colons; optionally concluded by any printable character.

S (Simplified): A modification of the CRM114 pattern that excludes dots, commas and colons from the middle of the pattern. With this pattern, domain names and mail addresses will be split at dots, so the classifier can recognize a domain even if subdomains vary.

X (XML/HTML+header-aware): A modification of the **S** schema that allows matching typical XML/HTML markup[3], mail headers (terminated by ":"), and protocols such as "http://" in a token. Punctuation marks such as "." and "," are not allowed at the end of tokens, so normal words will be recognized no matter where in a sentence they occur without being "contaminated" by trailing punctuation.

[3] Start/end/empty tags: `<tag> </tag> <br/>`; Doctype declarations: `<!DOCTYPE`; processing instructions: `<?xml-stylesheet`; entity + character references: `&mdash;`; attributes terminated by "="; attribute values surrounded by quotes.

The **X** schema was used for all tests unless explicitly stated otherwise. The actual tokenization schemas are defined as the regular expressions given in Table 8.1. These patterns use Unicode categories—`[^\p{Z}\p{C}]` means everything except whitespace and control chars; `\p{L}`, `\p{M}`, `\p{N}` represent letters, marks, and digits, respectively.

Name	Regular Expression
P	`[^\p{Z}\p{C}]+`
C	`[^\p{Z}\p{C}][-.,:\p{L}\p{M}\p{N}]*[^\p{Z}\p{C}]?`
S	`[^\p{Z}\p{C}][-\p{L}\p{M}\p{N}]*[^\p{Z}\p{C}]?`
X	`[^\p{Z}\p{C}][/!?#]?[-\p{L}\p{M}\p{N}]*(?:["'=;]|/?>|:/*)?`

Table 8.1: Tokenization patterns.

8.4 Feature Combination

8.4.1 Sparse Binary Polynomial Hashing

Sparse Binary Polynomial Hashing (SBPH) is a feature combination technique introduced by the CRM114 Discriminator[35][127]. SBPH slides a window of length N over the tokenized text. For each window position, all of the possible in-order combinations of the N tokens are generated; those combinations that contain at least the newest element of the window are retained. For a window of length N, this generates 2^{N-1} features. Each of these joint features can be mapped to one of the odd binary numbers from 1 to $2^N - 1$ where original features at "1" positions are visible while original features at "0" positions are hidden and marked as skipped.

It should be noted that the features generated by SBPH are not linearly independent and that even a compact representation of the feature stream generated by SBPH may be significantly longer than the original text.

8.4.2 Orthogonal Sparse Bigrams

Since the expressivity of SBPH is sufficient for many applications, we now consider if it is possible to use a smaller feature set and thereby increase speed and decrease memory requirements. For this, we consider only word pairs containing a common word inside the window, and requiring the newest member of the window to be one of the two words in the pair. The idea behind this approach is to gain speed by working only with an *orthogonal* feature set inside the window, rather that the prolific and probably redundant features generated by SBPH.

Instead of all odd numbers, only those with two bits "1" in their binary representations are used: $2^n + 1$, for $n = 1$ to $N - 1$. With this restriction, only $N - 1$ combinations with exactly two words are produced. We call them *Orthogonal Sparse Bigrams (OSB)*—"sparse" because most combinations have skipped words; only the first one is a conventional bigram.

With a sequence of five words, $w_1, \ldots, w_5$, OSB produces four combined features:

			$w4$	$w5$
		$w3$	<*skip*>	$w5$
	$w2$	<*skip*>	<*skip*>	$w5$
$w1$	<*skip*>	<*skip*>	<*skip*>	$w5$

Because of the reduced number of combined features, $N - 1$ in OSB versus 2^{N-1} in SBPH, text classification with OSB can be considerably faster than with SBPH. Table 8.2 and Table 8.3 show features generated by SBPH and OSB respectively.

Number		SBPH				
1	(1)					today?
3	(11)				lucky	today?
5	(101)			feel	<*skip*>	today?
7	(111)			feel	lucky	today?
9	(1001)		you	<*skip*>	<*skip*>	today?
11	(1011)		you	<*skip*>	lucky	today?
13	(1101)		you	feel	<*skip*>	today?
15	(1111)		you	feel	lucky	today?
17	(10001)	Do	<*skip*>	<*skip*>	<*skip*>	today?
19	(10011)	Do	<*skip*>	<*skip*>	lucky	today?
21	(10101)	Do	<*skip*>	feel	<*skip*>	today?
23	(10111)	Do	<*skip*>	feel	lucky	today?
25	(11001)	Do	you	<*skip*>	<*skip*>	today?
27	(11011)	Do	you	<*skip*>	lucky	today?
29	(11101)	Do	you	feel	<*skip*>	today?
31	(11111)	Do	you	feel	lucky	today?

Table 8.2: Features generated by SBPH.

Number		OSB				
1	(1)					
3	(11)				lucky	today?
5	(101)			feel	<*skip*>	today?
7	(111)					
9	(1001)		you	<*skip*>	<*skip*>	today?
11	(1011)					
13	(1101)					
15	(1111)					
17	(10001)	Do	<*skip*>	<*skip*>	<*skip*>	today?
19	(10011)					
21	(10101)					
23	(10111)					
25	(11001)					
27	(11011)					
29	(11101)					
31	(11111)					

Table 8.3: Features generated by OSB.

Note that the *orthogonal sparse bigrams* form an almost complete basis set—by "ORing" features in the OSB set, any feature in the SBPH feature set can be obtained, except for the unigram (the single-word feature). However, there is no such redundancy in the OSB feature set; it is not possible to obtain any OSB feature by adding, ORing, or subtracting any other pairs of other OSB features; all of the OSB features are unique and not redundant.

Since the first term, unigram w_5, cannot be obtained by ORing OSB features it seems reasonable to add it as an extra feature. However the experiments reported in Section 8.5.4 show that adding unigrams does *not* increase accuracy; in fact, it sometimes decreased accuracy.

8.5 Experimental Results

8.5.1 Testing Procedure

In order to test our multiple hypotheses, we used a standardized spam/nonspam test corpus from SpamAssassin [110]. This test corpus is extraordinarily difficult to classify, even for humans. It consists of 1397 spam messages, 250 hard nonspams, and 2500 easy nonspams, for a total of 4147 messages. These 4147 messages were "shuffled" into ten different standard sequences; results were averages over these ten runs. We re-used the corpus and the standard sequences from [30, 128].

Each test run begins with initializing all memory in the learning system to zero. Then the learning system was presented with each member of a standard sequence, in the order specified for that standard sequence, and required to classify the message. After each classification

the true class of the message was revealed and the classifier had the possibility to update its prediction model accordingly prior to classifying the next message.[4] The training system then moved on to the next message in the standard sequence. The final 500 messages of each standard sequence were the *test set* used for final accuracy evaluation; we also report results on an extended test set containing the last 1000 messages of each run and on all (4147) messages. At no time a system ever had the opportunity to learn on a message before predicting the class of this message. For evaluation we calculated the *error rate* $E = \frac{\textit{number of misclassifications}}{\textit{number of all classifications}}$; occasionally we mention the *accuracy* $A = 1 - E$.

This process was repeated for each of the ten standard sequences. Each complete set of ten standard sequences (41470 messages) required approximately 25–30 minutes of processor time on a 1266 MHz Pentium III for OSB-5.[5] The average number of errors per test run is given in parenthesis.

8.5.2 Parameter Tuning

We used a slightly different setup for tuning the Winnow parameters since it would have been unfair to tune the parameters on the test set. The last 500 messages of each run were reserved as test set for evaluation, while the preceding 1000 messages were used as *development set* for determining the best parameter values. The **S** tokenization was used for the tests in the section.

[4] In actual usage training will not be quite as incremental since mail is read in batches.

[5] For SBPH-5 it was about two hours which it not surprising since SBPH-5 generates four times as many features as OSB-5.

Best performance was found with Winnow using 1.23 as promotion factor, 0.83 as demotion factor, and a threshold thickness of 5%.[6] These parameter values turned out to be best for both OSB and SBPH—the results reported in Tables 8.4 and 8.5 are for OSB.

Promotion	1.35	1.25	1.25	1.23	1.2	1.1
Demotion	0.8	0.8	0.83	0.83	0.83	0.9
Test Set	0.44% (2.2)	0.36% (1.8)	0.44% (2.2)	**0.32% (1.6)**	0.44% (2.2)	0.48% (2.4)
Devel. Set	0.52% (5.2)	0.51% (5.1)	0.52% (5.2)	**0.49% (4.9)**	0.51% (5.1)	0.62% (6.2)
All	**1.26% (52.4)**	1.31% (54.3)	1.33% (55.1)	1.32% (54.7)	1.34% (55.4)	1.50% (62.2)

Table 8.4: Promotion and demotion factors.

Threshold thickness.	0%	5%	10%
Test Set	0.68% (3.4)	**0.32% (1.6)**	0.44% (2.2)
Development Set	0.88% (8.8)	**0.49% (4.9)**	0.56% (5.6)
All	1.77% (73.5)	**1.32% (54.7)**	1.38% (57.1)

Table 8.5: Threshold Thickness

8.5.3 Feature Store Size and Comparison With SBPH

Table 8.6 compares orthogonal sparse bigrams and SBPH for different sizes of the feature store. OSB reached best results with 600,000 features (with an error rate of 0.32%), while SBPH peaked at 1,600,000 features (with a slightly higher error rate of 0.36%). Further increasing the number of features permitted in the store negatively affects accuracy. This indicates that the LRU pruning mechanism is efficient at discarding irrelevant features that are mostly noise.

[6] In either direction, i.e. $\theta^- = 0.95\,\theta$, $\theta^+ = 1.05\,\theta$.

	OSB				
Store Size	400000	500000	600000	700000	800000
Last 500	0.36% (1.8)	0.38% (1.9)	**0.32% (1.6)**	0.44% (2.2)	0.44% (2.2)
Last 1000	0.37% (3.7)	0.37% (3.7)	**0.33% (3.3)**	0.37% (3.7)	0.37% (3.7)
All	1.26% (52.3)	1.29% (53.4)	**1.24% (51.4)**	1.26% (52.2)	1.27% (52.5)
	SBPH				
Store Size	1400000	1600000	1800000	2097152 (2^{21})	2400000
Last 500	0.38% (1.9)	**0.36% (1.8)**	0.42% (2.1)	0.44% (2.2)	0.42% (2.1)
Last 1000	0.37% (3.7)	**0.34% (3.4)**	0.38% (3.8)	0.39% (3.9)	0.38% (3.8)
All	1.35% (55.8)	**1.28% (53.1)**	1.30% (54)	1.30% (54)	1.31% (54.2)

Table 8.6: Comparison of SBPH and OSB with different feature storage sizes.

8.5.4 Unigram Inclusion

The inclusion of individual tokens (unigrams) in addition to orthogonal sparse bigrams does not generally increase accuracy, as can be seen in Table 8.7, showing OSB without unigrams peaking at 0.32% error rate, while adding unigrams pushes the error rate up to 0.38%.

	OSB only	**OSB + Unigrams**	
Store Size	600000	600000	750000
Last 500	**0.32% (1.6)**	0.38% (1.9)	0.42% (2.1)
Last 1000	**0.33% (3.3)**	**0.33% (3.3)**	0.36% (3.6)
All	1.24% (51.4)	**1.22% (50.6)**	1.24% (51.4)

Table 8.7: Utility of single tokens (Unigrams).

8.5.5 Window Sizes

The results of varying window size as a system parameter are shown in Table 8.8. Again, we note that the optimal combination for the test set uses a window size of five tokens (our default setting, yielding a 0.32% error rate), with both shorter and longer windows producing worse error rates.

Window Size	Unigrams	2 (Bigrams)	3	4	5	6	7
Store Size	all (ca.55000)	150000	300000	450000	600000	750000	900000
Last 500	0.46% (2.3)	0.48% (2.4)	0.42% (2.1)	0.44% (2.2)	**0.32% (1.6)**	0.38% (1.9)	0.42% (2.1)
Last 1000	0.50% (5)	0.43% (4.3)	0.39% (3.9)	0.40% (4)	**0.33% (3.3)**	0.38% (3.8)	0.37% (3.7)
All	1.43% (59.2)	1.23% (51.2)	1.24% (51.4)	1.26% (52.2)	1.24% (51.4)	1.28% (53)	**1.22% (50.8)**
Store Size		all (ca.220000)	all (ca.500000)	600000		900000	1050000
Last 500		0.48% (2.4)	0.42% (2.1)	0.42% (2.1)		0.40% (2)	0.46% (2.3)
Last 1000		0.43% (4.3)	0.38% (3.8)	0.38% (3.8)		0.38% (3.8)	0.40% (4)
All		1.24% (51.3)	**1.22% (50.6)**	1.25% (51.8)		1.27% (52.5)	1.25% (51.7)

Table 8.8: Sliding window size.

This “U” curve is not unexpected on an information-theoretic basis. English text has a typical entropy of around 1–1.5 bits per character and around five characters per word. If we assume that a text contains mainly letters, digits, and some punctuation symbols, most characters can be represented in six bits, yielding a word content of 30 bits. Therefore, at one bit per character, English text becomes uncorrelated at a window length of six words or longer, and features obtained at these window lengths are not significant.

These results also show that using OSB-5 is significantly better then using only single tokens (error rate of 0.46%) or conventional bigrams (0.48%).

8.5.6 Preprocessing and Tokenization

Results with *normalizemime* were generally better than the other two options, reducing the error rate by up to 25% (Table 8.9). Accuracy on raw and *mimedecoded* mails was roughly comparable.

Preprocessing	none	mimedecode	normalizemime
Last 500	0.42% (2.1)	0.46% (2.3)	**0.32% (1.6)**
Last 1000	0.37% (3.7)	0.35% (3.5)	**0.33% (3.3)**
All	1.27% (52.5)	1.26% (52.1)	**1.24% (51.4)**

Table 8.9: Preprocessing.

The **S** tokenization schema initially learns more slowly (the overall error rate is somewhat higher) but is finally just as good as the **X** schema (Table 8.10). **P** and **C** both result in lower accuracy, even though they initially learn quick.

Schema	**X**	**S**	**C**	**P**
Last 500	**0.32% (1.6)**	**0.32% (1.6)**	0.44% (2.2)	0.42% (2.1)
Last 1000	**0.33% (3.3)**	**0.33% (3.3)**	0.39% (3.9)	0.38% (3.8)
All	1.24% (51.4)	1.32% (54.7)	1.28% (52.9)	**1.23% (51.1)**

Table 8.10: Tokenization schemas.

8.5.7 Comparison with CRM114 and Naïve Bayes

The results for CRM114 and Naïve Bayes on the last 500 mails are the best results reported in [30]. For a fair comparison, these tests were all run using the **C** tokenization schema on raw mails without preprocessing. The best reported *CRM114* weighting model is based on empirically derived weightings and is a rough approximation of a Markov Random Field. This model reduces to a Naïve Bayes model when the window size is set to 1—the results for this case are shown in the first column of Table 8.11.

	Naïve Bayes	**CRM114**	**Winnow+OSB**
Last 500	2.02% (10.1)	1.12% (5.6)	**0.48% (2.4)**
All	3.44% (142.8)	2.71% (112.5)	**1.35% (55.8)**

Table 8.11: Comparison with Naïve Bayes and CRM114.

8.5.8 Speed of Learning

The learning rate for the Winnow classifier combined with the OSB feature generator is shown in Figure 8.1. Note that the rightmost column shows the incremental error rate on new messages. After having classified 1000 messages, Winnow+OSB achieves error rates below 1% on new mails.

8.6 Related Work

Winnow has been used for text classification before (e.g. [36]), but not (as far as we know) for spam filtering and not together with expressive feature combination techniques such as SBPH or OSB.

Bigrams and n-grams are a classical technique; SBPH has been introduced in [127]. We propose orthogonal sparse bigrams as a minimalistic alternative to SBPH that is new, to the best of our knowledge.

An LRU mechanism for feature set pruning has been employed by the first author in [104]. We suppose that others have done the same since the idea seems to suggest itself; but currently we are not aware of such usage.

Mails	Error Rate (Avg. Errors)		New Error Rate (Avg. New Errors)	
25	30.80%	(7.7)	30.80%	(7.7)
50	21.40%	(10.7)	12.00%	(3)
100	14.00%	(14)	6.60%	(3.3)
200	9.75%	(19.5)	5.50%	(5.5)
400	6.38%	(25.5)	3.00%	(6)
600	4.97%	(29.8)	2.15%	(4.3)
800	4.09%	(32.7)	1.45%	(2.9)
1000	3.50%	(35)	1.15%	(2.3)
1200	3.04%	(36.5)	0.75%	(1.5)
1600	2.48%	(39.7)	0.80%	(3.2)
2000	2.12%	(42.3)	0.65%	(2.6)
2400	1.85%	(44.4)	0.53%	(2.1)
2800	1.65%	(46.2)	0.45%	(1.8)
3200	1.51%	(48.2)	0.50%	(2)
3600	1.38%	(49.7)	0.38%	(1.5)
4000	1.28%	(51.1)	0.35%	(1.4)
4147	1.24%	(51.4)	0.20%	(0.3)

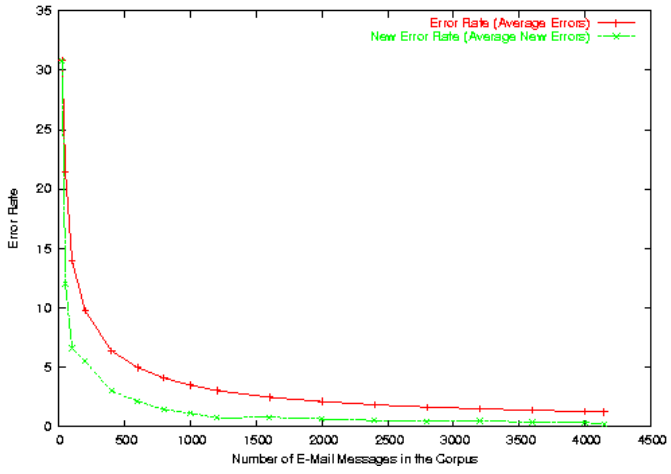


Figure 8.1: Learning curve for the best setting ($\text{Winnow}_{1.23,0.83,5\%}$ with 1,600,000 features, OSB-5, **X** tokenization).

8.7 Conclusion and Future Work

We have introduced *Orthogonal Sparse Bigrams (OSB)* as a new feature combination technique for text classification that combines a high expressivity with relatively low computational load. By combining OSB with the *Winnow* algorithm we halved the error rate compared to a state-of-the-art spam filter, while still retaining the property of *incrementality*. By refining the preprocessing and tokenization steps we were able to further reduce the error rate by 33%.[7]

One obvious direction for future work is to apply the combination of Winnow + OSB to other classification tasks, some of which will involve more than two classes. For such tasks we are working on an "ultraconservative" [34] variation of the Winnow algorithm that according to preliminary results promises to yield better results on problems with three or more classes.

Currently our Winnow implementation supports only binary features; how often a feature (sparse bigram) appears in a text is not taken into account. We plan to address this by introducing a *strength* for each feature (cf. [36, Sec. 4.3]).

Also of interest is the difference in performance between the LRU (least-recently-used) pruning algorithm used here and the random-discard algorithm used in CRM114 [30]. When the random-discard algorithm in CRM114 triggered, it almost always resulted in a decrease in accuracy; here we found that an LRU algorithm could act to provide an *increase* in accuracy. Analysis and determination of the magnitude of this effect will be a concern in future work.

[7] Our algorithm is freely available as part of the *TIES* system [120].

Chapter 9

Reputation Systems

9.1 Introduction

The problem of spam is being tackled with a variety of techniques. Recently, the industry has started focusing on reputation strategies to complement other spam-fighting techniques. Blacklist and whitelist are common examples of reputation systems in the present-day Internet email system. We expect use of advanced reputation models in this system soon. We will therefore describe trust and reputation systems that have already been proposed for realms such as ecommerce, recommendation systems, and peer-to-peer networks[2][3][26].

9.2 Trust and Reputation

In their paper "A Survey of Trust and Reputation Systems for Online Service Provision"[67], Jϕsang etal. very elegantly present state-of-the-art trust and reputation systems. Definitions for *trust* and *reputation* are different for each of these works. For simplicity, we define *trust* as a party's belief in another party based on its own direct experiences; *reputation* is a party's belief in another party based on the recommendations received from other parties and a summary of its behavior from past transactions.

9.2.1 Common Online Reputation Systems

We now describe some common online reputation systems.

1. *Web of Trust*

   Web of trust is a method to establish authenticity of the association between a public key and a user. This concept is used in Pretty Good Privacy (PGP) [95]. Key-signing parties are generally arranged to endorse the association between the public-key and a user. Note that web of trust is in contrast with the Certification Authorities (CA) used in the Public Key Infrastructure (PKI), suffering from the problem of scalability.

2. *eBay's Feedback Forum*

   eBay[47] has a centralized reputation system where feedback for sellers are collected from buyers after every transaction in the form of ratings (also comments) and a reputation score is returned. The ratings can be positive (1), negative (-1) or neutral (0).

The reputation score for a particular seller is computed by deducting unique buyers' total negative ratings from their total positive ratings. These ratings and comments are visible to any buyer. Based upon such reputation representation, a buyer can decide beforehand whether to conduct a transaction with any seller or not.

3. *Slashdot Reputation System*

   Slashdot[108] is a forum for posting articles and sharing comments. The comments on Slashdot are moderated. Slashdot's reputation system consists of two moderation layers, M1 for moderating postings and M2 for moderating M1 moderators. Each *registered* user of Slashdot maintains a *Karma* which can take any of the discrete values *Terrible, Bad, Neutral, Positive, Good* and *Excellent*. An integer score between -1 to 5 is maintained for each comment. The initial score is 1 but can also be influenced from the comment provider's Karma. The purpose of the comment score is to be able to filter the good comments from the bad[67].

4. *Certifications in Advogato[4]*

   Advogato is a community of open-source developers. Members of the Advogato community certify each other's skill levels. The reputation system in Advogato reflects the extent of reliability of member X's piece of code. A trust metric evaluates the peer certificates and decides on a trust level for each member. Advogato is attack-resistant against fake certifications[70].

9.2.2 Reputation Scoring System

A reputation system is very much dependent upon the *scores* used to rate the participants in the system. A good reputation-scoring system, therefore, should have the following properties[39]:

1. The scoring system should be quite accurate for a long-term performance.
2. The scoring system should have a weight towards current user behavior and should reflect the opinions of its users.
3. The scoring system should be efficient and convenient for recalculating a score quickly.
4. The scoring system should be robust against attacks.
5. The scoring system should be quite amenable to statistical evaluations; for example, it should be easy to find outliers.
6. Any scoring system should be easy to verify.
7. The scores generated should realistically imply an attribute that the common users can understand.

9.3 Reputation Network Architectures

The network architecture in a reputation system determines the communication flow of the reputation scores and ratings among its participants. A reputation system can have either a centralized or a distributed architecture. These network architectures are explained below.

9.3.1 Centralized Architecture

In centralized reputation systems, a central authority (reputation center) is responsible for collecting ratings from members who conducted transactions with other members of the community in the past. This reputation center computes a reputation score for every participant and makes all scores publicly available. The participants can then use these scores so as to decide whether to conduct transactions with another party or not.

Note that in these kinds of protocols, parties have bidirectional flow of information with the reputation center to provide ratings about transactions conducted with other members of the community, and also to inquire about the ratings of other parties. The reputation systems of eBay and Slashdot are examples of this type.

9.3.2 Distributed Reputation Systems

In distributed reputation systems, there is no centralized authority responsible for maintaining reputation scores for the members of the community. Instead, each participant simply saves its experience (i.e. vote/opinion) about transactions conducted with other parties and provides this information on request from the inquiring parties.

In our paper "A Protocol for Reputation Management in Super-Peer Networks"[29], Chhabra etal. we describe one such robust protocol *SupRep* built on the top of Gnutella v0.6[50]. The SupRep protocol is illustrated in Figure 9.1.

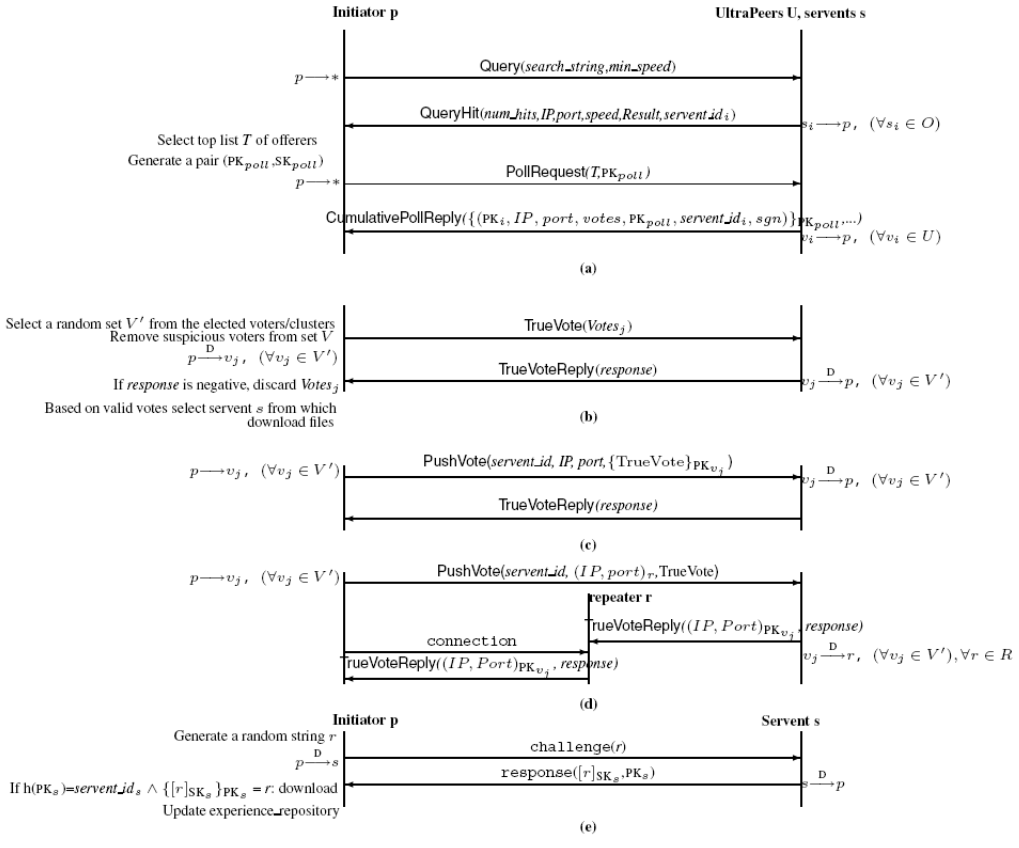


Figure 9.1: SupRep protocol built on the top of Gnutella v0.6. (a) query and poll; (b)-(d) vote verification; (e) resource download.

9.4 Reputation Computation Engines

This section describes algorithms that are used in various reputation systems to compute the final reputation score after ratings and feedback from the participants are obtained.

9.4.1 Summation/Average of Votes

In the eBay reputation system, the final reputation score for a seller is computed by subtracting its total negative ratings from the total positive ratings. The advantage of this model lies in its simplicity, but one disadvantage is that this model reflects a poor picture of participants' reputation score. Amazon[7] and Epinions[48] compute reputation score as the average of all ratings. Some models in this category compute a weighted average of all the ratings where the rating weight can be determined by factors such as the reputation of the rater and the age of the rating, etc.

9.4.2 Bayesian Systems

Bayesian systems take binary votes as input (i.e. positive or negative), and reputation scores are computed by statistical updating of beta probability density functions. *A posteriori* (i.e. the updated) reputation score is computed by combining the *a priori* (i.e. previous) reputation score with the new rating[67].

9.4.3 Discrete Trust Models

In their paper "Supporting Trust in Virtual Communities," Rahman etal.[2] describe an example of such a class. Trustworthiness of an agent can be assessed in discrete verbal statements such as *Very Trustworthy, Trustworthy, Untrustworthy* and *Very Untrustworthy*. The requesting party can consider the weight of the referring agent before taking its referral into account.

9.4.4 Flow Models

In this kind of system, the reputation score is computed by transitive iteration through loops or through arbitrarily long chains[67]. Google's PageRank[93] and EigenTrust[69] are examples of this category.

Chapter 10

Conclusion

Spam, phishing and email fraud are very serious problems polluting the Internet at the moment. Spammers are continuously inventing new tricks and hacks to fool technological solutions. The presence of compromised machines *(zombies)* over the Internet is aggravating this problem further. The current spam-fighting solutions are far from perfect; for example, spam filters will always have the problem of false positives; most of the greylisting solutions will malfunction if spammers start incorporating mechanisms for SMTP retrying in their bulk mailing tools; whitelisting will be rendered useless if spammers steal or forge origination as a whitelisted sender. Computers have been shown to be better than humans at single-character recognition used in modern-day CAPTCHAs, and a customized CAPTCHA breaker can be built in a day at a cost of forty dollars. Spammers have already started deploying email authentication mechanisms such as the Sender Policy Framework (SPF) in order to prevent email rejection resulting from authentication failures. Authentication protocols that

incorporate cryptography using SHA1 as the hashing algorithm should pay attention to the recent collision attacks on SHA1 algorithm[124]. Reputation solutions are currently being considered, but differences in framework and rating assessment in such solutions will lead to inconsistency. Poor rating of entire blocks of IP addresses can lead to the rejection of certain legitimate international mail, which conflicts with the original intention behind Internet email.

A collaborative effort among industry leaders, governments, and Internet users is required to destroy the spammers' business model. We have already explained various tricks spammers use to exploit technological solutions. We presented transcripts recovered during the arrest of the world's eighth most prolific spammer Jeremy Jaynes. As evident from his *to do* list, spamming is a multimillion-dollar business with no cost to the sender, and hence it becomes a very lucrative business for people willing to spam.

We presented various technological and legal initiatives to solve this problem, including our work on the CRM114 filter. We illustrated and explained CRM114 usage for small-, medium-, and large-scale enterprises (for filtering up to one million client email accounts). We presented the internals of a system using CRM114, implementing the concept of Internet postage known as the CAMRAM. We described a unified model of spam filtration followed by all the spam filters currently available in the market. We also presented the Markov Random Field model and Nick Littlestone's Winnow-based machine learning techniques for spam-filtering, which have shown significant improvements in accuracy over the Naïve Bayesian filtering technique. We hightlighted some reputation solutions proposed in

other realms of computer science for the industry considering a robust and consistent reputation solution for future Internet email. We suggest seeking cooperation from domain registrars, as they have all information for tracking fraudsters who buy and throw away thousands of domain names in their phishing activities.

Spammers and phishers are tainting the merits of email communication by continuously abusing the Internet, polluting traffic, cheating naïve Internet users, and destroying productivity. Winning the war against such entities is a responsibility that rests on this current generation's leaders.

Overall, we are very confident that, with an aggressive collaborative effort from the technical industry, cooperation from governments in legal spheres, and awareness among Internet users, we can destroy the spammers' business model in the next couple of years, reducing the problem of spam to an unfortunate event in the history of the Internet.

Bibliography

[1] Martin Abadi, Andrew Birrell, Mike Burrows, Frank Dabek, and Ted Wobber. Bankable Postage for Network Services. In *8th Asian Computing Science Conference, Mumbai, India*, 2003. http://research.microsoft.com/research/sv/sv-pubs/TicketServer.pdf.

[2] Alfarez Abdul-Rahman and Stephen Hailes. Supporting Trust in Virtual Communities. In *HICSS*, 2000. citeseer.ist.psu.edu/article/abdul-rahman00supporting.html.

[3] Karl Aberer and Zoran Despotovic. Managing Trust in a Peer-2-Peer Information System. In Henrique Paques, Ling Liu, and David Grossman, editors, *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM01)*, pages 310–317. ACM Press, 2001. http://citeseer.nj.nec.com/aberer01managing.html.

[4] Advogato. http://www.advogato.com/.

[5] Banit Agrawal, Nitin Kumar, and Mart Molle. Controlling Spam E-mail at the Routers. In *IEEE International Conference on Communcations (ICC 05), Seoul Korea*, 2005. http://www.cs.ucr.edu/~mart/preprints/icc_spam.pdf.

[6] Shabbir Ahmed and Farzana Mithun. Word Stemming to Enhance Spam Filtering. In *Proceedings of the First Conference on Email and Anti-Spam (CEAS)*, 2004. http://www.ceas.cc/papers-2004/167.pdf.

[7] Amazon. http://www.amazon.com.

[8] I. Androutsopoulos, J. Koutsias, K.V. Chandrinos, G. Paliouras, and C.D. Spyropoulos. An Evaluation of Naive Bayesian Anti-Spam Filtering. In *G. Potamias, V. Moustakis, and M.n van Someren, editors, Proceedings of the Workshop on Machine Learning in the New Information Age, 11th European Conference on Machine Learning (ECML 2000), pages 9–17, Barcelona, Spain*, 2000. http://arXiv.org/abs/cs.CL/0006013.

[9] I. Androutsopoulos, G. Paliouras, V. Karkaletsis, G. Sakkis, C.D. Spyropoulos, and P. Stamatopoulos. Learning to Filter Spam E-mail: A Comparison of a Naive Bayesian and a Memory-Based Approach. In *H. Zaragoza, P. Gallinari, , and M. Rajman,*

editors, Proceedings of the Workshop on Machine Learning and Textual Information Access, 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2000), pages 1–13, Lyon, France, 2000. http://arXiv.org/abs/cs/0009009.

[10] Ion Androutsopoulos, John Koutsias, and Konstandinos V. Chandrinos andConstantine D. Spyropoulos. An Experimental Comparison of Naive Bayesian and Keyword-Based Anti-Spam Filtering with Personal E-mail Messages. In *Nicholas J. Belkin, Peter Ingwersen, and Mun-Kew Leong, editors, Proceedings of SIGIR-00, 23rd ACM International Conference on Research and Development in Information Retrieval, pages 160–167, Athens, GR. ACM Press, New York, US.*, 2000. http://www.acm.org/pubs/articles/proceedings/ir/345508/p160-androutsopo%ulos/p160-androutsopoulos.pdf.

[11] Adam Back. Hashcash, 1997. http://www.cypherspace.org/hashcash/.

[12] Adam Back. Hashcash - A Denial of Service Counter-Measure. http://www.hashcash.org/papers/hashcash.pdf, 2002.

[13] Barracuda Networks. http://www.barracudanetworks.com/.

[14] Bayesian Noise Reduction (BNR) Processing Library. http://www.nuclearelephant.com/projects/dspam/bnr.html.

[15] BBC News, 04/09/2005, Man Gets Nine Years for Spamming. http://news.bbc.co.uk/2/hi/americas/4426949.stm.

[16] BBC News, 07/16/2005, Jail for Nigerian Bank Fraudster. http://news.bbc.co.uk/2/hi/africa/4690031.stm.

[17] BBC News, 08/10/2005, Microsoft in 7m Dollar Spam Settlement. http://news.bbc.co.uk/2/hi/business/4137352.stm.

[18] J. Besag. Spatial Interaction and the Statistical Analysis of Lattice Systems. In *Journal of the Royal Statistical Society, Series B*, volume 36, pages 192–236, 1974.

[19] BlackLists. http://www.email-policy.com/Spam-black-lists.htm.

[20] P. O. Boykin and V. Roychowdhury. Leveraging Social Networks to Fight Spam. In *IEEE Computer, Vol. 38, No. 4, pages 61-68*, 2005. http://boykin.acis.ufl.edu/~boykin/papers/spamgraph.ps.

[21] Brian Burton. Bayesian Spam Filtering Tweaks. In *Proceedings of the MIT Spam Conference*, 2003. http://spamprobe.sourceforge.net/paper.html.

[22] Caller ID for E-mail. http://xml.coverpages.org/draft-atkinson-callerid-00.txt.

[23] Camram. http://www.camram.org.

[24] Andrew J. Carlson, Chad M. Cumby, Nicholas D. Rizzolo, Jeff L. Rosen, and Dan Roth. SNoW User Manual. Version: January, 2004. Technical report, UIUC, 2004. http://l2r.cs.uiuc.edu/~cogcomp/software/snow-userguide.ps.gz.

[25] Xavier Carreras and Llus Mrquez. Boosting Trees for Anti-Spam Email Filtering. In *Proceedings of RANLP-2001, 4th International Conference on Recent Advances in Natural Language Processing*, 2001. http://www.lsi.upc.es/~carreras/pub/boospam.ps.

[26] Sonja Buchegger Ch. A Robust Reputation System for P2P and Mobile Ad-hoc Networks. In *Buchegger, S., Boudec, J.Y.L.: In: Proceedings of the Second Workshop on the Economics of Peer-to-Peer Systems. (2004)*, 2004. http://citeseer.ist.psu.edu/article/robust04robust.html.

[27] Kumar Chellapilla, Kevin Larson, Patrice Simard, and Mary Czerwinski. Computers beat Humans at Single Character Recognition in Reading Based Human Interaction Proofs (HIPs). In *Second Conference on Email and Anti-Spam (CEAS)*, 2005. http://www.ceas.cc/papers-2005/160.pdf.

[28] Kumar Chellapilla and Patrice Y. Simard. Using Machine Learning to Break Visual Human Interaction Proofs (HIPs). In *Advances in Neural Information Processing Systems 17. Neural Information ProcessingSystems (NIPS 2004)*. MIT Press, 2004. http://www.books.nips.cc/papers/files/nips17/NIPS2004_0843.pdf.

[29] Shalendra Chhabra, Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. A Protocol for Reputation Management in Super-Peer Networks. In *15th International Workshop on Database and Expert Systems Applications, Zaragoza, Spain*, pages 979–983, 2004. http://www.cs.ucr.edu/~schhabra/pdmst04.pdf.

[30] Shalendra Chhabra, William S. Yerazunis, and Christian Siefkes. Spam Filtering Using a Markov Random Field Model with Variable Weighting Schemas. In *Proceedings of the Fourth IEEE International Conference on Data Mining, (ICDM)*, 2004. http://www.cs.ucr.edu/~schhabra/icdm04.pdf.

[31] CipherTrust, Inc. http://www.ciphertrust.com/.

[32] W. W. Cohen. Learning Rules that Classify E-mail. In *AAAI Spring Symposium on Machine Learning in Information Access*, California, 1996. www.cs.cmu.edu/~wcohen/postscript/aaai-ss-96.ps.

[33] Composite Block List (CBL). http://cbl.abuseat.org/.

[34] Koby Crammer and Yoram Singer. Ultraconservative Online Algorithms for Multiclass Problems. In *14th Annual Conference on Computational Learning Theory (COLT)*, pages 99–115, Berlin, 2001. Springer. http://citeseer.ist.psu.edu/crammer01ultraconservative.html.

[35] CRM114 - the Controllable Regex Mutilator. http://crm114.sourceforge.net/.

[36] Ido Dagan, Yael Karov, and Dan Roth. Mistake-Driven Learning in Text Categorization. In Claire Cardie and Ralph Weischedel, editors, *Proceedings of EMNLP-97, 2nd Conference on Empirical Methods in Natural Language Processing*, pages 55–63, Providence, US, 1997. Association for Computational Linguistics. http://citeseer.ist.psu.edu/552405.html.

[37] Death2Spam. http://www.death2spam.com.

[38] Rachna Dhamija and J. D. Tygar. Phish and HIPs: Human Interactive Proofs to Detect Phishing Attacks. In *In Human Interactive Proofs: Second International Workshop (HIP 2005)*, pages 127–141, 2005. http://www.cs.berkeley.edu/~tygar/papers/Phish_and_HIPs.pdf.

[39] Roger Dingledine. Accountability. In Andy Oram, editor, *Peer to Peer, Harnessing the Power of Disruptive Technologies*. OReilly, 2001.

[40] Distributed Checksum Clearinghouse. http://freshmeat.net/projects/dcc-source/.

[41] Domain-Based Email Authentication Using Public-Keys Advertised in the DNS (DomainKeys). http://www.ietf.org/internet-drafts/draft-delany-domainkeys-base-02.txt.

[42] DomainKeys Identified Mail (DKIM). http://mipassoc.org/dkim/specs/draft-allman-dkim-base-00-10dc.txt.

[43] Christine E. Drake, Jonathan J. Oliver, and Eugene J. Koontz. Anatomy of a Phishing Email. In *Conference on Email and Anti Spam (CEAS)*, 2004. http://www.ceas.cc/papers-2004/114.pdf.

[44] H. Drucker, D. Wu, and V. N. Vapnik. Support Vector Machines for Spam Categorization. In *IEEE Trans. on Neural Networks*, pages 1048–1054, 1999. http://www.site.uottawa.ca/~nat/Courses/NLP-Course/itnn_1999_09_1048.pdf.

[45] C. Dwork and M. Naor. Pricing via Processing or Combatting Junk Mail. In *Proceedings of CRYPTO'92, Lecture Notes in Computer Science 740, pages 137–147*, 1992. http://research.microsoft.com/research/sv/PennyBlack/junk1.pdf.

[46] EarthLink. http://http://www.earthlink.net/.

[47] eBay, Inc. http://www.ebay.com/.

[48] Epinions. http://www.epinions.com/.

[49] False Claims in Spam. http://www.ftc.gov/reports/spam/030429spamreport.pdf.

[50] Gnutella - A Protocol for a Revolution. http://rfc-gnutella.sourceforge.net/.

[51] Joshua Goodman. Spam: Technologies and Policies, Whitepaper, November 2003, Microsoft Corporation. http://research.microsoft.com/~joshuago/spamtech.pdf.

[52] Joshua Goodman and Robert Rounthwaite. Stopping Outgoing Spam. In *ACM Conference on E-Commerce*, 2004. http://research.microsoft.com/~joshuago/outgoingspam-final-submit.pdf.

[53] Paul Graham. A Plan for Spam. http://www.paulgraham.com/spam.html.

[54] Paul Graham. Better Bayesian Filtering. In *MIT Spam Conference*, 2003. http://www.paulgraham.com/better.html.

[55] John Graham-Cumming. The Spammers Compendium. In *MIT Spam Conference*, 2003. http://www.jgc.org/tsc.

[56] John Graham-Cumming. People and Spam. In *MIT Spam Conference*, 2005. http://www.jgc.org/pdf/spamconf2005.pdf.

[57] J. M. Hammersley and P Clifford. Markov Field on Finite Graphs and Lattices. In *Unpublished*, 1971.

[58] Evan Harris. The Next Step in the Spam Control War: Greylisting. http://projects.puremagic.com/greylisting/whitepaper.html.

[59] Jose Maria Gomez Hidalgo. Bibliography on Machine Learning for Spam Detection. http://liinwww.ira.uka.de/bibliography/Ai/MLSpamBibliography.html.

[60] Geoff Hulten, Joshua T. Goodman, and Robert Rounthwaite. Filtering Spam E-mail On a Global Scale. In *WWW (Alternate Track Papers & Posters), pages 366–367*, 2004. http://research.microsoft.com/~joshuago/www2004-submission.pdf.

[61] Geoff Hulten, Anthony Penta, Gopalakrishnan Seshadrinathan, and Manav Mishra. Trends in Spam Products and Methods. In *First Conference on Email and Anti-Spam (CEAS)*, 2004. http://www.ceas.cc/papers-2004/index.html.

[62] Identified Internet Mail. http://www.identifiedmail.com/draft-fenton-identified-mail.txt.

[63] Internet Message Access Protocol - Version 4rev1. http://www.faqs.org/rfcs/rfc3501.html.

[64] Internet Message Format. http://www.faqs.org/rfcs/rfc2822.html.

[65] IronPort Systems, Inc. http://www.ironport.com/.

[66] Markus Jakobsson and Ari Juels. Proofs of Work and Bread Pudding Protocols. In *In B. Preneel, Editor, Communications and Multimedia Security, pages 258-272, Kluwer Academic Publishers*, 1999. http://www.rsasecurity.com/rsalabs/node.asp?id=2049.

[67] Audun Josang, Roslan Ismail, and Colin Boyd. A Survey of Trust and Reputation Systems for Online Service Provision. In *Decision Support Systems*, 2005. http://security.dstc.edu.au/papers/JIB2005-DSS.pdf.

[68] Richard Jowsey. Death2spam. http://www.death2spam.net/.

[69] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Twelfth International World Wide Web Conference*, 2003. http://citeseer.ist.psu.edu/551866.html.

[70] Raph Levien. Attack Resistant Trust Metrics. In *PhD Thesis, UC Berkeley*, 2004. http://www.levien.com/thesis/compact.pdf.

[71] Nick Littlestone. Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm. *Machine Learning*, 2:285–318, 1988. http://ai.stanford.edu/~pabbeel/depth_qual/littlestone1988.pdf.

[72] David Madigan. Statistics and the War on Spam. In *In Statistics, A Guide to the Unknown*, 2004. http://www.stat.rutgers.edu/~madigan/PAPERS/sagtu.pdf.

[73] Mail Abuse Prevention System. http://www.mail-abuse.com.

[74] Mail Routing and the Domain System. ftp://ftp.is.co.za/rfc/rfc974.txt.

[75] MailFrontier, Inc. http://www.mailfrontier.com/.

[76] MailFrontier Phishing IQ Test. http://survey.mailfrontier.com/survey/quiztest.html.

[77] Bart Massey, Mick Thomure, Raya Budrevich, and Scott Long. Learning Spam: Simple Techniques for Freely-available Software. In *In Proceeding of the 2003 Usenix Annual Technical Conference, Freenix Track*, 2003. http://nexp.cs.pdx.edu/twiki-psam/pub/PSAM/PsamDocumentation/spam.pdf.

[78] T.A Meyer and B Whateley. SpamBayes: Effective Open-source, Bayesian Based, Email Classification System. In *Proceedings of the First Conference on Email and Anti-Spam (CEAS)*, 2004. http://www.ceas.cc/papers-2004/136.pdf.

[79] Eirinaios Michelakis, Ion Androutsopoulos, Georgios Paliouras, George Sakkis, and Panagiotis Stamatopoulos. Filtron: A Learning-Based Anti-Spam Filter. In *Proceedings of the First Conference on Email and Anti-Spam (CEAS)*, 2004. http://www.ceas.cc/papers-2004/142.pdf.

[80] Minsky and Papert. Perceptrons. Cambridge, MA, 1969. MIT Press.

[81] Mirapoint, Inc. http://www.mirapoint.com/.

[82] Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. http://www.ietf.org/rfc/rfc2045.txt.

[83] Multipurpose Internet Mail Extensions Part Three: Message Header Extensions for Non-ASCII Text. http://www.ietf.org/rfc/rfc2047.txt.

[84] Marcia Munoz, Visin Punyakanok, Dan Roth, and Dav Zimak. A Learning Approach to Shallow Parsing. Technical Report UIUCDCS-R-99-2087, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1999. http://citeseer.ist.psu.edu/333381.html.

[85] Moni Naor. Verification of a Human in the Loop or Identification via the Turing Test, 1996. http://citeseer.ist.psu.edu/naor96verification.html.

[86] New Michigan and Utah Child Protection Registry Laws. http://www.spamfo.co.uk/index.php?option=com_content\&task=view\&id=000344.

[87] normalizemime v2004-02-04. http://hyvatti.iki.fi/~jaakko/spam/.

[88] Henrik Nottelmann and Norbert Fuhr. Learning Probabilistic Datalog Rules for Information Classification and Transformation. In *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM)*, 2001. http://ls6-www.informatik.uni-dortmund.de/ir/publications/2001/Nottelma%nn_Fuhr:01.html.

[89] Cormac O'Brien and Carl Vogel. Comparing SpamAssassin with CBDF Email Filtering. In *Proceedings of the 7th Annual CLUK Research Colloquium*, 2004. http://www.cs.tcd.ie/Cormac.OBrien/spamAss.pdf.

[90] T. Oda and T. White. Developing an Immunity to Spam. In *Genetic and Evolutionary Computation - GECCO, Chicago, IL, USA. Lecture Notes in Computer Science, Vol. 2723, Springer, pages 231–242*, 2003. http://terri.zone12.com/doc/academic/spam_gecco2003.pdf.

[91] T. Oda and T. White. Increasing the Accuracy of a Spam-detecting Artificial Immune System. In *Proceedings of the Congress on Evolutionary Computation (CEC 2003)*, 2003. http://terri.zone12.com/doc/academic/spam_cec2003.pdf.

[92] Cormac OBrien and Carl Vogel. Spam Filters: Bayes vs. Chi-squared; Letters vs. Words. In *Proceedings of the International Symposium on Information and Communication Technologies*, 2003. http://www.cs.tcd.ie/publications/tech-reports/reports.03/TCD-CS-2003-1%3.pdf.

[93] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998. http://citeseer.ist.psu.edu/page98pagerank.html.

[94] Patrick Pantel and Dekang Lin. Spamcop: A Spam Classification and Organization Program. In *Learning for Text Categorization: Papers from the 1998 Workshop, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05*, 1998. http://www.cs.ualberta.ca/~ppantel/Download/Papers/aaai98.pdf.

[95] PGP: Pretty Good Privacy. http://www.pgpi.org/.

[96] Pobox. http://pobox.com.

[97] Post Office Protocol- Version 3. http://www.ietf.org/rfc/rfc1939.txt.

[98] Project Honey Pot. http://www.projecthoneypot.org/.

[99] Purported Responsible Address in E-Mail Messages. http://www.ietf.org/internet-drafts/draft-lyon-senderid-pra-01.txt.

[100] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A Bayesian Approach to Filtering Junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05. http://research.microsoft.com/~horvitz/junkfilter.htm.

[101] Sender Authentication. http://www.spf.pobox.com/whitepaper.pdf.

[102] Sender ID: Authenticating E-Mail. http://xml.coverpages.org/draft-ietf-marid-core-03.txt.

[103] Sender Policy Framework (SPF) for Authorizing Use of Domains in E-MAIL, version 1. http://www.ietf.org/internet-drafts/draft-schlitt-spf-classic-02.txt.

[104] Christian Siefkes. A Toolkit for Caching and Prefetching in the Context of Web Application Platforms. Diplomarbeit, TU Berlin, 2002. http://www.siefkes.net/diplom/.

[105] Christian Siefkes, Fidelis Assis, Shalendra Chhabra, and William S. Yerazunis. Combining Winnow and Orthogonal Sparse Bigrams for Incremental Spam Filtering. In *Proceedings of the 15th European Conference on Machine Learning and 8th European Conference on Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD 2004), Lecture Notes in Computer Science. Springer, 2004. Copyright Springer-Verlag*, 2004. http://www.cs.ucr.edu/~schhabra/winnow-spam.pdf.

[106] Simple Mail Transfer Protocol. http://www.faqs.org/rfcs/rfc821.html.

[107] Simple Mail Transfer Protocol. http://www.faqs.org/rfcs/rfc2821.html.

[108] Slashdot. http://www.slashdot.org/.

[109] SPAM Track, TREC 2005. http://plg.uwaterloo.ca/~gvcormac/spam/.

[110] SpamAssassin. http://spamassassin.apache.org/.

[111] SpamBayes. http://spambayes.sourceforge.net/.

[112] SpamCop Blocking List (SCBL). http://www.spamcop.net/bl.shtml.

[113] SpamProbe. http://spamprobe.sourceforge.net/.

[114] Standard for the Format of ARPA Internet Text Messages. http://www.faqs.org/rfcs/rfc822.html.

[115] Symantec Corporation. http://www.symantec.com/.

[116] Text Retrieval Conference (TREC). http://trec.nist.gov/.

[117] The Domain Naming Convention for Internet User Applications. ftp://ftp.is.co.za/rfc/rfc819.txt.

[118] The Penny Black Project. http://research.microsoft.com/research/sv/PennyBlack/.

[119] The Spamhaus Project. http://www.spamhaus.org.

[120] Trainable Incremental Extraction System. http://www.inf.fu-berlin.de/inst/ag-db/software/ties/.

[121] TRUSTe. http://www.truste.org/.

[122] Tumbleweed Communications Corporation. /http://www.tumbleweed.com/.

[123] Vipul's Razor. http://razor.sourceforge.net/.

[124] Xiaoyun. Wang, Yiqun Lisa Yin, and Hongbo. Yu. Finding Collisions in the Full SHA-1. In *Crypto05. Santa Barbara,CA*, 2005. http://theory.csail.mit.edu/~yiqun/pub.html.

[125] Rebecca Wetzel. Tackling Phishing. http://www.netforecast.com/Articles/RW-Phishing-BCR05,02.pdf.

[126] Wikipedia. The Free Encyclopedia. http://en.wikipedia.org/wiki/Main_Page.

[127] William S. Yerazunis. Sparse Binary Polynomial Hashing and the CRM114 discriminator. In *2003 Spam Conference*, Cambridge, MA, 2003. MIT. http://crm114.sourceforge.net/CRM114_paper.html.

[128] William S. Yerazunis. The Spam-Filtering Accuracy Plateau at 99.9% Accuracy and How to Get Past It. In *2004 Spam Conference*, Cambridge, MA, 2004. MIT. http://crm114.sourceforge.net/Plateau_Paper.pdf.

[129] William S. Yerazunis, Shalendra Chhabra, Christian Siefkes, Fidelis Assis, and Dimitrios Gunopulos. A Unified Model of Spam Filtration. In *MIT Spam Conference*, 2005. http://www.cs.ucr.edu/~schhabra/UnifiedFilters.pdf.

[130] Jonathan Zdziarski. The DSPAM Project. http://www.nuclearelephant.com/projects/dspam/.

[131] Le Zhang and Tian Yao. Filtering Junk Mail with A Maximum Entropy Model. In *Proceeding of 20th International Conference on Computer Processing of Oriental Languages (ICCPOL03)*, 2003. http://www.nlplab.cn/zhangle/paper/junk.pdf.

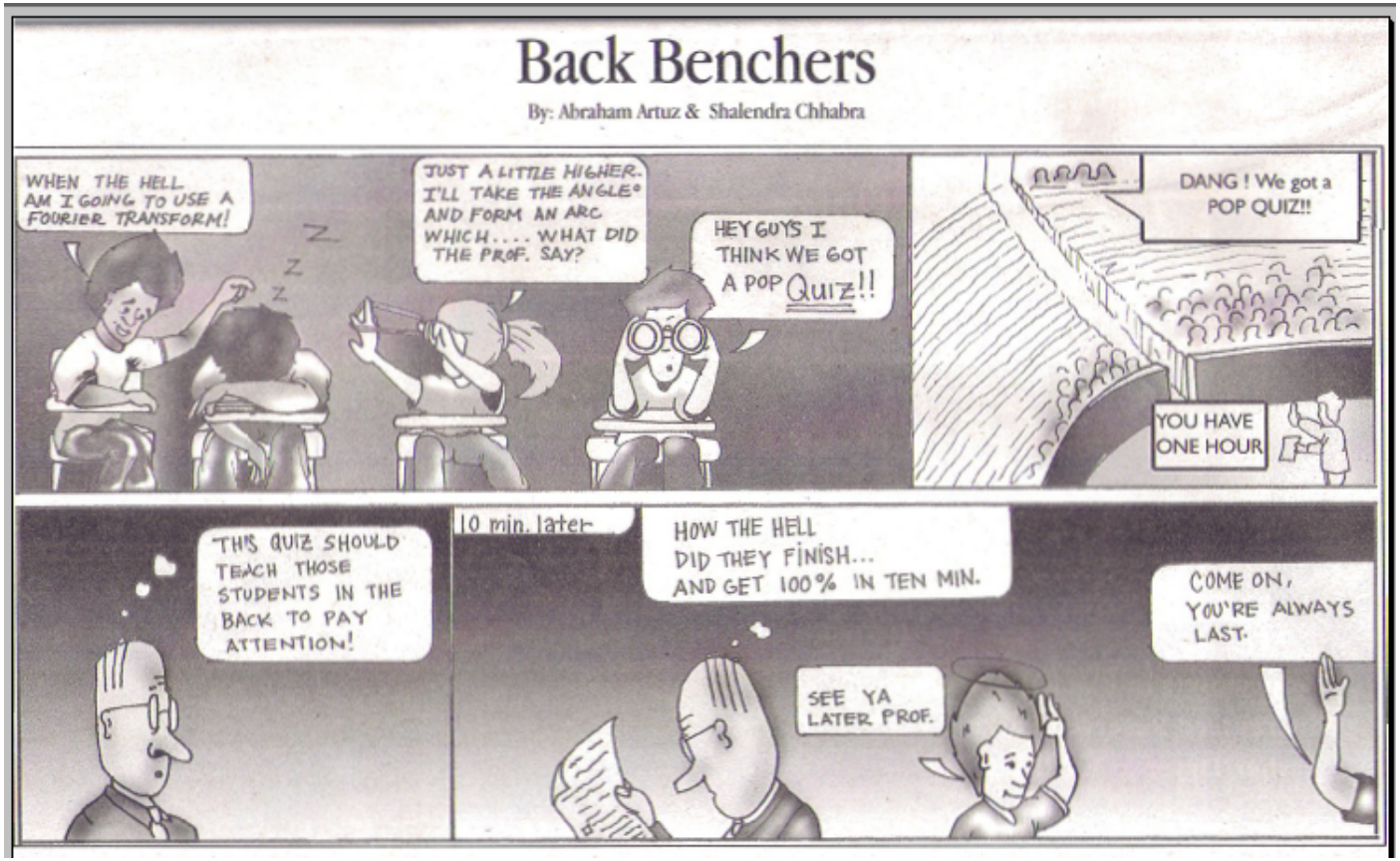
Vita

Shalendra Chhabra was born in India on November 29, 1980. He was awarded the Gold Medal by the Lions Club and the Merit Certificate by the Government of India for securing first place in the district in the 1996 High School Examination. After clearing the Indian Institute of Technology Joint Entrance Examination (IIT-JEE) in 1999, he joined the Institute of Technology, Banaras Hindu University (ITBHU) in Varanasi, India, where he received a Bachelor of Technology (BTech) with Honors in Electrical Engineering in 2003. During his undergraduate studies he won numerous awards and competitions, including a Certificate of Achievement and Honorable Mention at the ACM International Collegiate Software Programming Contest (Asia Zone) in 2001 and a medal from IEE UK for placing first in an All-India Engineering Contest in 2002. He was a Student-Fellow of the Indian Academcy of Sciences (IAS) (2001) and has worked at the Tata Institute of Fundamental Research (TIFR) (National Center of Government of India for Nuclear Science and Mathematics) (2001); Laboratoire Spécification et Vérification, Ecole Normale Superieure De Cachan (LSV, ENS De Cachan) (France) (2002) and University of Milan, Crema (Italy) (2003).

He joined the Department of Computer Science and Engineering, University of California, Riverside (UCR), as a graduate student in September 2003. He is a recipient of the Dean's Fellowship and is completing a Master's Degree in Computer Science (MS) in October 2005. He has been with Mitsubishi Electric Research Laboratories, MA (MERL), for the summers of 2004 and 2005. During his academic studies he has published 4 papers and has presented his work at the Massachusetts Institute of Technology (MIT), Stanford University and Cisco Systems. He is on the development team of an open source spam filter, the CRM114 Discriminator, and he has been cited on Slashdot. His Erdős Number is 3.

He is a dancer by hobby and is working on his movie *The BackBenchers* at the University of California, Riverside.

He is joining Microsoft Corporation as a Program Manager in the MSN Safety Team.


Back Benchers
By: Abraham Artuz & Shalendra Chhabra
WHEN THE HELL AM I GOING TO USE A FOURIER TRANSFORM!
Z Z Z
JUST A LITTLE HIGHER. I'LL TAKE THE ANGLE° AND FORM AN ARC WHICH.... WHAT DID THE PROF. SAY?
HEY GUYS I THINK WE GOT A POP Quiz!!
DANG ! We got a POP QUIZ!!
YOU HAVE ONE HOUR
THIS QUIZ SHOULD TEACH THOSE STUDENTS IN THE BACK TO PAY ATTENTION!
10 min. later
HOW THE HELL DID THEY FINISH... AND GET 100% IN TEN MIN.
SEE YA LATER PROF.
COME ON, YOU'RE ALWAYS LAST.