MITIGATING SPAM USING NETWORK-LEVEL FEATURES

A Thesis
Presented to
The Academic Faculty

by

Anirudh V. Ramachandran

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Computer Science

School of Computer Science
Georgia Institute of Technology
August 2011

MITIGATING SPAM USING NETWORK-LEVEL FEATURES

To mom and dad for their love, support, and encouragement.

ACKNOWLEDGEMENTS

Most of the good work that my name appears on is directly due to my advisor, Prof. Nick Feamster. Nick is one of those few professors bold enough to offer funding to a green first-year PhD with no experience in networking or security, solely on the basis of a 7001 mini-project—all before he even started at Tech. I am glad he took that chance. Nick taught me almost everything I know about picking problems, crafting clever solutions, and, just as importantly, presenting it well. I'm also grateful for his suggestions, insights, and much-needed nudges in the right direction, as well as his informal advising style (hurrah for open-ended 3 AM meetings!). Despite his brilliance, Nick is also one of the hardest-working people I know, and I strive to emulate his work ethic.

I would also like to thank the rest of my committee. Prof. Wenke Lee's courses were enlightening, and he has offered precise and constructive suggestions whenever I have had the fortune to bounce my ideas off of him. Prof. Patrick Traynor gave excellent comments on my drafts, and his "Destructive Research" course was one of the most interesting courses I have taken at Tech—his choice of readings showed how and why many crucial real-world security mechanisms are fundamentally broken. My external committee members, Dr. Anirban Dasgupta and Prof. Kilian Weinberger, hosted me at Yahoo! Research during my most enjoyable summer internship. They tolerated my limited knowledge of theory and made that summer a great experience: Kilian provided great Machine Learning advice and first suggested that I try clustering on voting data from Yahoo! Mail (which became the key idea behind Chapter 6). Anirban helped me explore the various nooks and crannies of the behemoth that is Yahoo! Mail, and provided solutions—both theory and code—the many times that I got stuck.

Several other people have directly influenced my research. I'm grateful to the hosts of my previous internships and have learned a great deal from each: Prof. Scott Shenker of UC Berkeley for hosting me during summer 2006, and Dr. Balachander Krishnamurthy, Dr. Kobus van der Merwe, and Dr. Oliver Spatscheck of AT&T Research for summer 2007. I'm also lucky to have worked with Prof. Santosh Vempala; Santosh's keen intellect and unquenchable enthusiasm for applying theoretical ideas to practical problems were awe-inspiring. I also thank my other collaborators over the years for all the discussions and insights—Kaushik Bhandankar, Atish Das Sarma, David Dagon, Shuang Hao, Hitesh Khandelwal, Yogesh Mundada, Srini Seetharaman, and Mukarram Tariq; though not everything resulted in publications, their ideas and thoughts have shaped my research. The research in this dissertation would not have been interesting without the underlying data itself being interesting; I thank David Mazières for his spam trap data, Suresh Ramasubramanian for mail data from Outblaze, David Dagon for traces of botnet activity and DNSBL lookups, and Xiaopeng Xi and Jay Pujara of the Yahoo! anti-spam team for their mail logs.

My years at Tech would have been a lot less fun had it not been for the company of several people outside work. Thanks to my roommates over the years—Atish, Murtaza, Srikanth, and Yogesh—for the company and for putting up with me. I owe my sanity to labmates and local friends: Ahmed, Amogh, Atish, Bilal, Ilias, Manish, Millo, Mukarram, Partha, Ravi & Yarong, Ruomei, Saamer, Sai, and Srini. Amogh dutifully listened to my music recommendations though he probably hated most of them. Atish and I had some great conversations and I really enjoyed swapping crazy theories and startup ideas with him. Srini is generally awesome and has saved my skin so many times. Mukarram is one of the smartest people I have met and has been a constant source of inspiration. Of late, Ilias has provided excellent company, both for playing music and as a concert buddy. Thanks are also due to my mates from

Narmada Hostel and the 6th wing for all the good times and memories.

My parents have been pillars of support all through my PhD, providing constant encouragement, advice, news from home, and a clearer view of my own priorities. I dedicate this dissertation to them. My sister Inku and brother-in-law Pramu have provided encouragement and advice from not-too-far-away, and have helped me unwind after stressful periods so many times these past 6 years. Grandma has always been supportive and loving though I forget to call her for months on end. I am indebted to all of them.

BIBLIOGRAPHIC NOTES

Chapter 3 extends work from our paper "Understanding the Network-level Behavior of Spammers" [103] by adding newer longitudinal analyses. Chapter 4 contains material from the paper "Revealing Botnet Membership Using DNSBL Counter-Intelligence" [107]. Chapter 5 contains material from the paper "Filtering Spam with Behavioral Blacklisting" [108]. Chapter 6 contains material from the paper "Spam or Ham? Characterizing and Detecting Fraudulent *Not Spam* Reports in Webmail Systems" [102] that is under submission. Chapter 7 contains some material from the unpublished work "Spotting Spammers with a Dynamic Reputation System" [109] as well as new material.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Spam is an increasing menace for all forms of online messaging including email, instant messaging, social media, blogs, and Web forums. Many past and current approaches to tackling spam rely too heavily on content-based approaches, where filters use the content of spam messages to distinguish them from legitimate messages. This approach, however, aims at a moving target: spammers are free to evolve the content of their messages in a variety of ways in response to filtering rules, leaving content-based filters to play “catch-up”. Content-based filters also incur more overhead, because they need to accept, store, and process the content of an email before making a decision; with 90% of email—over 50 billion messages a day—being spam, content-based filters are expensive both to maintain and to scale.

In this dissertation, we introduce email spam filtering using *network-level features*. Network-level features are based on lightweight measurements that can be made in the network, often without processing or storing a message. Beyond just the IP address of a traffic source, network-level features also include the Autonomous System (AS) numbers of the source, flow sizes, packet header information, data that can be collected from structured application-level traffic streams such as DNS or HTTP information, and aggregates of these features (*e.g.*, the historical behavior of an IP address). Unlike content-based features Network-level features also affords the opportunity to observe the coordinated *behavior* of spammers. Network-level attributes of traffic stay relevant for longer periods and are harder for criminals to alter at will (*e.g.*, a bot cannot act independently of other bots in the botnet).

This dissertation has the following contributions.

1. We perform a detailed characterization of the network-level behavior of spam including its origins, volumetric and temporal behavior, and its relation to botnets and hijacked BGP routes. We further perform a longitudinal analysis of these features over a 6 year period to examine the robustness of network-level features for email classification. We find that IP-based reputation systems such as IP blacklists may not be able to keep up with the threat of spam from previously unseen IP addresses, and from new and stealthy attacks.

2. We present three unsupervised algorithms that detect correlated behavior of spammers using network-level features. First, we introduce the stealthy spammer behavior of reconnoitering IP Blacklists, and present techniques to detect such queries using temporal and spatial features. Second, we present *SpamTracker*, a system that distinguishes spammers from legitimate senders by applying clustering on the set of domains to which email is sent. Third, we introduce *vote-gaming attacks* in large Web-based email systems that pollutes user feedback on spam emails, and present an efficient clustering-based method to mitigate such attacks.

We have evaluated our algorithms on real-world datasets, and our work has also resulted in practical tools and applications: Our vote-gaming attack detection system has been put to use by Yahoo! Mail to detect compromised bot-controlled accounts. We have also designed a system to detect spam from potentially hijacked BGP prefixes and integrated it with our real-time dynamic blacklisting system, SpamSpotter.

CHAPTER 1

INTRODUCTION

Email remains the major method for business and personal online communication for hundreds of millions of users and organizations. One of the key features that initially drove its widespread adoption—the ability for any Internet-connected host to send a message to a person using only their email address—has also become the Achilles heel for email, and the chief reason for the overwhelming amount of spam that must be processed by mail servers each day. Because a user might legitimately receive email from any Internet host, the protocol used to send and receive email—SMTP [68] (Simple Mail Transfer Protocol)—has no built-in measures for security or accountability. Indeed, the protocol designers are well aware of the security concerns of SMTP. To quote the SMTP RFC [68]:

> *SMTP mail is inherently insecure in that it is feasible for even fairly casual users to negotiate directly with receiving and relaying SMTP servers and create messages that will trick a naïve recipient into believing that they came from somewhere else.*
>
> – RFC 5321

Unfortunately, miscreants have been exploiting exactly this insecurity of the SMTP protocol for their sustenance. As the costs of sending a message went down with ubiquitous Internet adoption and increasing connection speeds, these spammers began to use automated programs to send large numbers of spam messages that link to scam Websites, phishing attacks, or sites that trick a user into installing malicious software on their computers. Over the past decade, spam volumes have grown so high that approximately 90% of all email—about 70 billion messages—on the Internet today is spam [5].

In the past, much spam was sent from dedicated "spam farms" and open mail relays [21, 90]; these services, however, were relatively easy for mail recipients to avoid using *IP address blacklists* (or "blocklists") of spam senders. Over the past few years, the chief sources of spam have been *botnets*—distributed networks of compromised computers under the control of a "botmaster". Botnets are used for coordinated attacks such as sending spam or mounting denial of service attacks. Botmasters assemble botnets by infecting the computers of unsuspecting users around the world, and then use the combined bandwidth and processing power of these infected machines to manufacture and send spam to millions of users. Symantec MessageLabs estimates that currently, 88.2% of all spam sent can be traced to a botnet [131].

1.1 Spam Filtering: Techniques and Challenges

This section overviews the chief approaches for filtering spam, and problems faced by each of these approaches due to new and emerging threats.

1.1.1 Types of Spam Filters

Figure 1.1 presents an overview of the chief classes of approaches for spam filtering available to a typical victim organization. The first line of defense includes lightweight *network-level* approaches, of which IP-based blacklisting is the only widely used approach. These methods are responsible in rejecting a majority of spam (70–90%) early [122], leaving messages from unknown spam senders to be filtered by the remaining two methods. The recipient organization then feeds the spam through a *content-based filter*, which uses features of the headers, body, or attachments of the message as well as URLs and other embedded artifacts to determine whether the message is potentially spam. Content-based filters are reputed to be highly accurate, but they also incur high overhead. The few spam messages that are not caught by

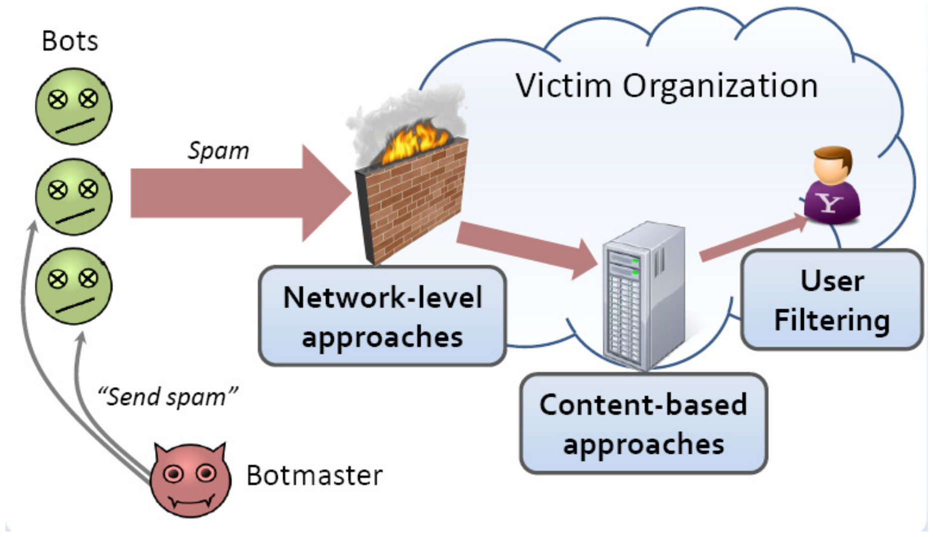


Figure 1.1: Overview of Spam Filtering Approaches

network-level or content-based filters end up in end-users' Inboxes. At this stage, organizations rely on end-users to report misclassified spam messages as such by, for example, clicking a "Report as Spam" button. User feedback is crucial: once a number of users report such messages, the organization's automatic classifiers can learn from the misclassified message to correctly filter similar spam messages in future—network-level filters can learn network-level features of the spam sender, and content-based filters can learn the content-based features that evaded it. Effective spam filtering thus depends on the correct performance of all three components.

In this dissertation, we find that the use of a practically unlimited supply of bots, emerging stealthy attacks, and constantly evolving spam content are posing massive threats to all steps of spam filtering. Spammers are sending spam from IP addresses that have not been observed spamming for many months, which makes them harder to be listed in blacklists. We also find that spammers are monitoring IP blacklists and sometimes sending spam using hijacked IP spaces, further complicating the IP blacklisting process. To pollute end-user reporting of spam messages, spammers use their own compromised user accounts to report *their own spam messages as "not* *spam"*.

Each stage of spam filtering faces a number of problems—some of which are organic to the technique, while others are due to new and stealthy attacks by spammers and other miscreants. The next section discusses the problems with each filtering approach.

1.1.2 Problems with Spam Filtering

Content-based Filters. *Content-based filters* such as SpamAssassin [120] are a widely-deployed defense against spam. These filters analyze the content of email to identify attributes that would distinguish large fractions of spam messages from legitimate ones. Because the majority of spam is auto-generated and designed to spread a small set of scams (*e.g.*, penny-stock scams, Rolex watches, pills from Canadian pharmacies, etc.), content-based filters are able to use statistical learning and classification to differentiate the majority of spam from legitimate email with few false positives.

However, content-based filters have several critical shortcomings. First, because they operate on the content of messages, the mail recipient must accept, process, and store *all* incoming messages—of which 90% is spam—before beginning classification. As spam volumes increase, recipients must continuously upgrade their infrastructure and software to be able to process email at scale. Second and perhaps more importantly, designing content-based filters is an arms race against spammers with no clear winner: as soon as a filter is fine-tuned to detect a particular spam campaign, the spammer will alter his spam template such that his spam evades the filter. For example, in response to a content-based filter that uses certain keywords to detect spam (*e.g.*, the phrase "Rolex Watches" and its variations), the spammer can embed the words in an image that is then sent in the spam message.

These shortcomings make content-based filters unsuitable for use as a first line of defense against spam. Indeed, most mail service administrators first apply lightweight network-level filters such as IP blacklists to summarily reject the vast majority of spam that strikes an organization's mail servers. While content-based filters remain highly accurate, especially with regards to false positive rates, they require constant maintenance to keep up with the mutations of spam templates. Furthermore, content-filters are dependent on network-level and end-user filters: because content-filters have high processing and storage overhead, they rely on network-level filters to limit the amount of spam they need to deal with; because they must constantly learn and adapt their classifiers for new campaigns, they rely on end-user filters to quickly and correctly report new spam templates that evade existing content filters. Thus, attacks that weaken other filters indirectly affects content-based filtering as well.

Network-level Filtering. Network-level features include easily-observable attributes of an email or email sender that are robust, lightweight, and applicable in detecting many attacks mounted by spammers. We define network-level features as observations and measurements that can be made cheaply and efficiently, usually at the network layer of the TCP/IP protocol stack, and from anywhere in the network (*i.e.*, not necessarily at the recipient's machine). Examples of network-level features include (1) attributes of a single email such as the sender's IP address, the sender's BGP prefix and route, the Autonomous System (AS) that owns the prefix, the hostname corresponding to the sender's IP address, the size of the email (in bytes), etc.; (2) attributes that are based on aggregates or historical observations, for example, the set of domains to which a particular sender sends email, the fraction of known spam senders in the BGP prefix of an email sender, the average and standard deviation of email size in all messages from a sender, etc.

Network-level features offer a number of advantages over content-based features for spam filtering.

- *Network-level features are robust.* Unlike the content of an email that spammers can change at will, network-level features are far less malleable. Even if the botmaster can rotate the bot IP addresses from which he sends spam, he has little control over aggregate network-level features (BGP prefix, AS number, etc.) or the coordinated behavior of the spam bots in his botnet.

- *Network-level features are lightweight.* Many network level features are based on simple measurements that do not require an SMTP transaction to complete: attributes such as the IP address block and AS number of the sender can be detected anywhere in the network (*e.g.*, in the sender's own access network), and attributes such as message size can be estimated by measuring the flow size of the TCP connection.

- *Network-level features can detect correlated behavior.* Bots are usually centrally controlled and send spam in a coordinated fashion, and this coordinated behavior sets them apart from legitimate senders; indeed, this dissertation presents various coordinated attacks mounted by spam bots, and methods to mitigate such attacks. Content-based filters cannot detect such coordinated behavior because the content of spam sent by the bots in a botnet may be quite dissimilar to each other,

State-of-the-art network-level filtering is largely limited to IP-based blacklists. Although blacklist vendors claim 90% reduction in spam [122], our experiments indicate that the actual fraction of blacklisted spammers is much lower—about 60–70%. The prevalent use of large botnets is likely responsible for the inability of blacklists to keep their detection rates up. A large botnet implies that the potential *number of unique IP addresses* from which spam is sent is also large. Because many spam senders originate from IP address ranges that also contain legitimate email senders, IP blacklisting is also prone to false positives. An additional challenge in IP blacklisting involves the use of dynamic address assignment using protocols such as DHCP [30] by Internet

Service Providers; because DHCP can reassign a spam sender's IP address to a legitimate machine, blacklist vendors are hesitant to blacklist spammers from DHCP senders.[1] Indeed, we have discovered that even at a spam trap—a domain with no legitimate addresses that receives mail only from spammers—30% or more of spammers are not listed in any of six blacklists, likely because many of these spammers were never observed for months prior to when they sent out spam.

In this dissertation, we present new attacks that further reduce the effectiveness of using IP reputation to filter spam. We find that *spammers themselves may be stealthily monitoring IP blacklists to discover which of their bots are listed.* We also find evidence for spam from hijacked BGP prefixes, which implies that certain spammers may be hijacking a legitimate organization's IP space and sending spam from IP addresses within that space.

End-user Filtering. The final weapon in an organization's spam filtering arsenal are its end-users [39]. To quickly identify emerging spam campaigns that may have evaded both network-level and content-based filters, organizations rely on their users to quickly build consensus on the status of the spam message: if a few users who received the misclassified spam message in their Inboxes report the message as spam, the message can be retroactively removed from other users' Inboxes and placed in their spam folders. Furthermore, community clicks also aid IP reputation: if enough users report mails sent from a particular IP address as spam, the sender may also be placed in an IP blacklist.

End-user filtering has the potential to quickly neutralize spam campaigns and render "new" spamming IP addresses ineffective. To counter this defense mechanism, we discovered that spammers are mounting attacks of their own, especially within large Web-based email services such as Yahoo! Mail and Gmail, to pollute votes

[1]Except by policy, as in the Spamhaus Policy Blacklist (PBL) [123].

from legitimate users. *Spammer use bot-controlled compromised user accounts to dishonestly vote "not spam" on their own spam email* such that the true classification of spam email is delayed. This attack reduces the effectiveness of user filtering and delays the time before which the senders of spam can be placed on blacklists.

1.2 Thesis Statement

> In this dissertation, we will show that although network-level features offer unique advantages over content-based features for spam filtering, IP-based reputation methods for filtering spammers are becoming less effective. Our evidence includes spammer techniques of using previously unseen IP addresses to send spam, as well as stealthy attacks such as IP blacklist reconnaissance, spam from hijacked prefixes, and dishonest voting in Webmail systems. We posit that, although IP-based reputation has become less reliable, network-level features also expose the coordinated behavior of malicious senders—a consequence of using centrally-controlled botnets to send spam. We show that network operators can construct classifiers to identify such coordinated behavior, and use these behavioral classifiers to mitigate various spam-related attacks in real time. As future work, we argue that behavioral classifiers can be applied to many botnet-orchestrated attacks beyond merely spamming, for example, denial of service, click fraud, scam hosting, etc.

1.3 Contributions

In support of the thesis statement, this dissertation presents the following contributions.

- *Characterization of the Network-level Behavior of Spammers.* We present the first detailed analysis of spam sending behavior using data collected from a high-volume spam "trap" or "sinkhole"—a domain that receives only spam—over a 17 month period from August 2004–December 2005. Further, we present a unique 6.5-year longitudinal analysis of spam from spam sinkholes by analyzing the evolution of the network-level properties of spam senders from the perspective of a network operator of a specific domain; we show that network-level classifiers can be a powerful tool especially in the face of continually changing spam campaigns and new spammer IP addresses which reduces the effectiveness of IP-based blacklists.

- *New Spammer Attacks.* We introduce several new attacks mounted by spammers such as (1) sending spam by hijacking large BGP prefixes using short-lived announcements; (2) conducting stealthy reconnaissance on a DNS-based IP blacklist to discover when a spam bot is blacklisted; (3) casting dishonest "not spam" votes on spam email in Web-based email services to delay the true classification of spam.

- *Defenses Against Spammer Attacks.* We present algorithms to detect coordinated spammer behavior and to mitigate spammer attacks. Specifically, (1) to detect bots that perform reconnaissance on DNS-based blacklists, we present an algorithm that uses temporal and spatial relationships of bots that query the blacklist; (2) to detect spam senders using their sending behavior, we present *SpamTracker*, a clustering-based algorithm that uses the pattern of recipient domains targeted by spam bots; (3) to detect bots that attempt to "game" the true classification of spam in Web-based email services, we present an efficient clustering-based algorithm based on the number of Webmail user identities shared by each bot.

- *Supporting Tools and Techniques.* In support of our algorithms, we present the following tools: (1) to make the deployment of algorithms such as SpamTracker and SNARE [51] easier, we designed a generic real-time dynamic spam filtering service based on the DNS-based query scheme used in IP blacklists that aims to be both fast and scalable; (2) to detect spam from potentially hijacked routes, we design a tool, SpamLoJack, that joins BGP announcements with the IP addresses of spam senders in *real-time* to discover spam sent using potentially hijacked or short-lived BGP routes.

1.4 Roadmap

Figure 1.2 presents the overview of the topics addressed by various chapters of this dissertation.

Chapter 2.. This chapter presents background into spam and efforts in spam filtering, with emphasis on network-level approaches to filtering spam. This chapter sets the context for the work presented in dissertation within the wider body of spam-filtering and botnet detection research.

Chapter 3. This chapter presents background on spammers, the prevalence of bots in spamming, and introduces network-level features of spammers. This chapter presents a detailed study using 17 months of spam data that analyzes various network-level features of spam messages, including the top IP address ranges, AS numbers, and country codes of spammers; the types of operating systems used to send spam; the average sizes of spam messages, etc.; the chapter also investigates the continuing relevance of these features over time using a 6-year longitudinal spam dataset. We find that identifying and filtering spam based on individual IP addresses may be prone both to false positives and false negatives, and filtering based on larger IP blocks, or based on the sending *behavior* of spammers might overcome these shortcomings. This chapter also presents a detailed analysis of a stealthy spamming method, which we

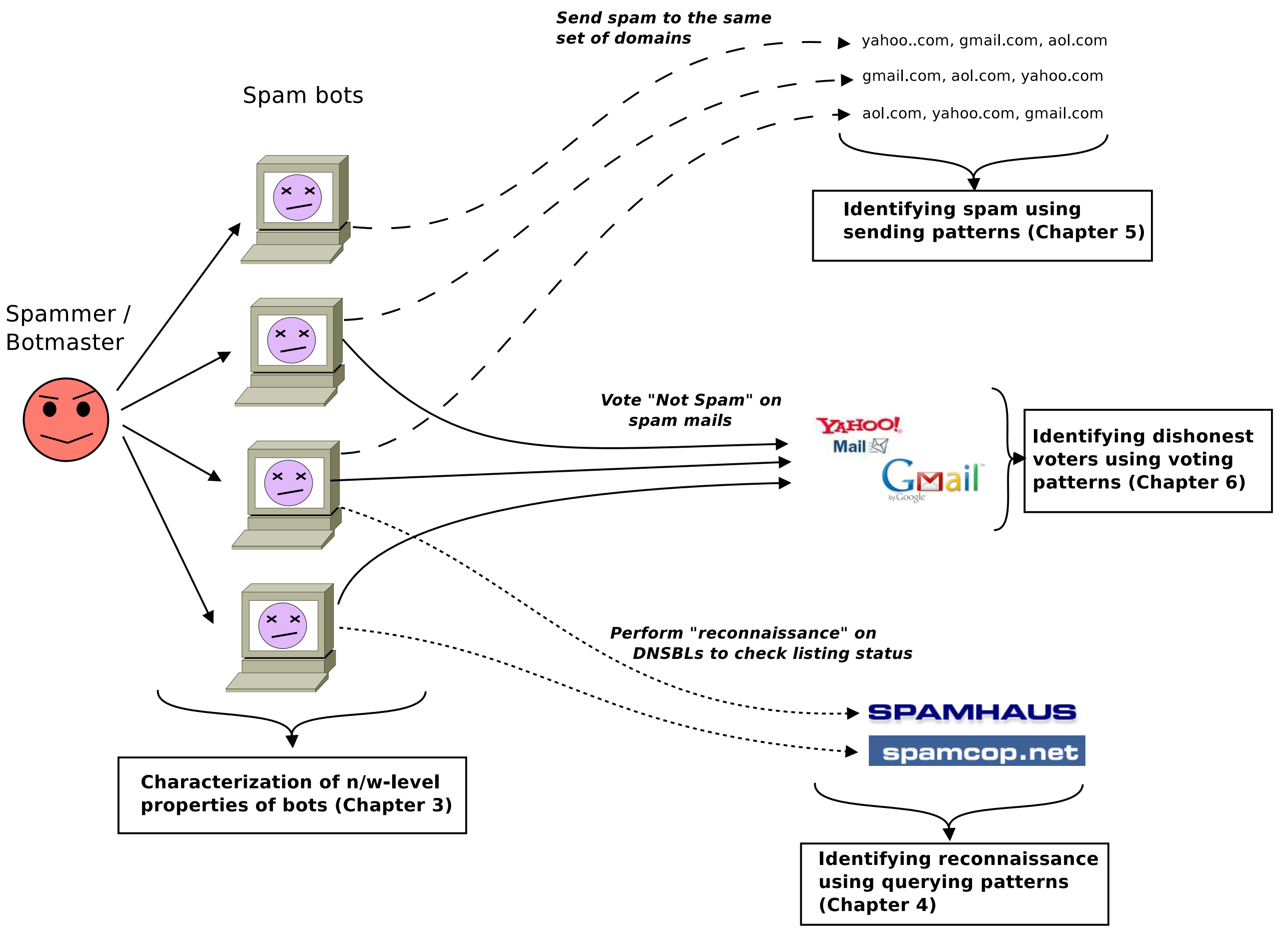

Send spam to the same set of domains
yahoo..com, gmail.com, aol.com
gmail.com, aol.com, yahoo.com
aol.com, yahoo.com, gmail.com
Spam bots
Identifying spam using sending patterns (Chapter 5)
Spammer / Botmaster
Vote "Not Spam" on spam mails
Identifying dishonest voters using voting patterns (Chapter 6)
Perform "reconnaissance" on DNSBLs to check listing status
Characterization of n/w-level properties of bots (Chapter 3)
Identifying reconnaissance using querying patterns (Chapter 4)

Figure 1.2: Overview of the attacks addressed in this dissertation

call BGP "spectrum agility", in which spammers hijack large BGP prefixes for short periods to send spam.

Chapter 4. Driven by our finding in Chapter 3 that over 30% of spammers are not listed in any DNS-based IP blacklist (DNSBL), we investigate a new attack in this chapter, where spammers attempt to proactively detect when a particular bot IP becomes listed in a popular DNSBL by making stealthy queries to the DNSBL for spam bot IP addresses. We then leverage the coordinated behavior of bots in performing such reconnaissance to develop a method to detect reconnaissance queries and to opportunistically enumerate bot identities that perform the reconnaissance. This method has resulted in a commercial patent, and has been has been used to detect bots in practice.

Chapter 5. We established in Chapter 3 that blacklisting based on specific IP addresses may not be able to identify many spammers, especially given the fraction of previously-unseen spammers that sent spam in a single burst to our spam sinkhole. We hypothesize that spammers likely send email to multiple domains, and a network-wide view from multiple vantage points may help identify such spammers based on *behavior* rather than their IP address. We confirm this hypothesis and present a clustering-based algorithm to detect coordinated spammer behavior in this chapter. Our technique, called *SpamTracker*, clusters email senders based on the set of recipient domains to which messages are sent. We hypothesize that, because bots in a given botnet are likely to send email to the same subset of domains, the sending patterns of bots will cluster well. We use spectral clustering to construct a pattern of sending behavior for known spammers, which can later be used to identify new, previously-unknown spam senders *only using their sending pattern*. We evaluate our algorithm using mail logs from a large email service provider that sinks email for over 115 recipient domains; we find that our algorithm is able to discover clusters of spam bots, and also show that our technique can be applied in a real-time blacklist.

Chapter 6. In Chapter 3, we found that certain large IP address blocks that predominantly consist of legitimate email senders occasionally contain spam senders, and hypothesized that these senders were likely to be large email service providers, who were being abused by a few spammers to relay their spam. In this chapter, we look at the problem of such compromised accounts in Yahoo! Mail, the largest Web-based email service provider. Webmail services rely on user feedback to quickly build consensus on the status of an email; thus, most Webmail services allow users to report a message as “spam” or “not spam”. To influence this decision and to prolong the true classification of spam messages, spammers use bot-controlled user accounts to dishonestly vote "not spam" on spam messages. We present an algorithm that uses canopy-based clustering to detect Webmail user accounts used solely for dishonest voting. The clustering algorithm is based on the insight that accounts used for dishonest voting are accessed from many different bot IP addresses, while legitimate user accounts are typically accessed from very few IP addresses. We performed this study using real data from Yahoo! Mail, the largest Web-based email provider; our algorithm is being used by Yahoo! in practice.

Chapter 7. This chapter presents tools and techniques in support of our findings. First, we present *SpamSpotter*, a high-performance dynamic, real-time blacklisting system that presents mail recipients with a simple, uniform interface to query network-level spam filtering algorithms such as SpamTracker (Chapter 5 and SNARE [51]. SpamSpotter offers a framework that presents developers of new network-level algorithms with a simple set of abstractions to develop and deploy their algorithms. To make these algorithms easily usable by recipients, SpamSpotter uses a modified version of the widely-used DNS-based blacklist query structure. Next, we present a SpamLoJack, a tool and a Web-based service that joins live feeds of BGP updates and spam data to discover spam that is being sent from short-lived or other suspicious BGP announcements; SpamLoJack is also integrated into the query interface of SpamSpotter. Spam from hijacked routes is stealthy and a regular occurrence [6], and we expect SpamLoJack can provide early warning against route hijacks deliberately used for spamming.

Chapter 8. This chapter concludes this dissertation. We will summarize the lessons learned in this dissertation and present various avenues for future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter provides an overview of techniques both for sending and for mitigating spam, and discusses related work in these areas, focusing on network-level methods for spam filtering.

2.1 Spamming Methods

Spammers use various techniques to send large volumes of mail while attempting to remain untraceable. We describe several of these techniques, beginning with "conventional" methods and progressing to more intricate techniques.

2.1.1 Direct Spamming

In the "early" days of spam, when broadband connections were not as prevalent as they are today, spammers purchased upstream connectivity from "spam-friendly" ISPs, set up powerful servers, and churned out large amounts of spam from these servers. Occasionally, spammers would buy connectivity and send spam from ISPs that did not condone this activity and were forced to change ISPs. Ordinarily, changing from one ISP to another would require a spammer to renumber the IP addresses of their mail relays. To remain untraceable and avoid renumbering headaches, spammers sometimes obtain a pool of dispensable dialup IP addresses, send outgoing traffic from a high-bandwidth host after spoofing its own IP address to appear as if it came from the dialup connection, proxying the reverse traffic through the dialup connection back to the spamming host [110]. Unfortunately, the downside of direct spamming is that the spam sending machines are limited in number and represent a single point of failure for the spammer: once listed in an IP or IP prefix blacklist, spammers would have no way of churning out spam. Moreover, direct spamming usually involves traceable money trails to the owner (*i.e.*, spammer), for example, through the purchase of the hosting service. As botnets rise to prominence due to their resilience and aggregate bandwidth, direct spamming has seen a major decline.

2.1.2 Open Relays and Proxies

Open relays are mail servers that allow unauthenticated Internet hosts to connect and relay email through them. Originally intended for user convenience (*e.g.*, to let users send mail from a particular relay while they are traveling or otherwise in a different network), open relays have been exploited by spammers due to the anonymity and amplification offered by the extra level of indirection. Spammers use tools to scan for open relays by identifying servers that listen on SMTP port 25, and then by sending a test email to an account *under the spammers' own control* [90], which, if received, confirms the existence of an open relay. Fortunately, open relay blacklists such as SORBS [119] and ORDB [88] have forced most legitimate open relays to cease service. However, as a recent study by Pathak *et al.* showed, spammers continue to scan for open relays and exploit any relays that they find by sending high volumes of spam through them.

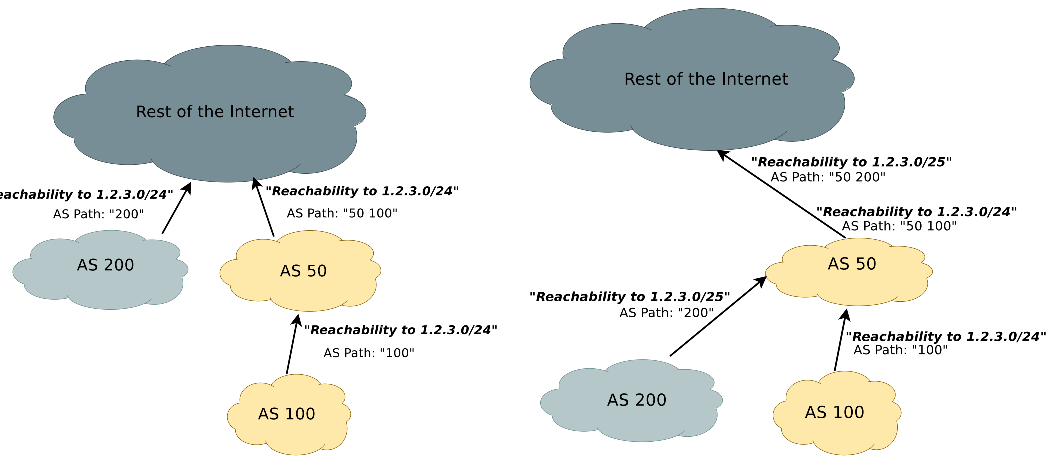
Open proxies are services that merely route application-level traffic, allowing a sender to anonymously access or use Internet services. When used for sending email, the open proxy allows a spammer to mask their IP address: the recipient mail server typically only uses the IP address of the host that made a TCP connection to it, and the spammer is free to use fake IP addresses in the SMTP headers of emails it sends. As with open relays, open proxies have found themselves ending up in blacklists such as Blitzed Open Proxy Monitor [141].

2.1.3 Spam Using Unorthodox BGP Route Announcements

BGP [114], the Internet's inter-domain routing protocol, is designed to be robust: technically, any Autonomous System (AS) can announce connectivity to an IP prefix, and the routers that receive many route announcements for a prefix are free to choose the "best path" using attributes such as the number of ASes in the path (AS path length). Unfortunately, because BGP has no built-in security, *prefix hijacks*—the announcement of a route to an IP prefix by an AS that causes traffic to not reach the true owner of the prefix—are quite common.

The most common reason for prefix hijacks is a misconfiguration: configuring router policies is a complex process that involves technical and policy decisions, and operators may make mistakes that cause them to unintentionally hijack another IP prefix. In a recent example of this nature, an ISP from Pakistan accidentally announced a highly specific prefix, a /24, that belonged to YouTube [115]. Because YouTube's legitimate route announcement was a /22 that encompassed the /24 announced by the Pakistani ISP, the more specific prefix announced by the Pakistani ISP became the "best path" to some YouTube services for a large portion of the Internet, thus disrupting connectivity to YouTube. Deliberately hijacking a prefix, however, requires a malicious ISP who must convince its upstream provider to accept a fraudulent route announcement and re-announce it to the rest of the Internet. This kind of activity is less common, and if discovered, can potentially result in the provider losing upstream connectivity. Another potential approach for deliberately hijacking a prefix is to compromise another ISP's router and then announce stealthy routes.

Figure 2.1 illustrates the two ways in which prefix hijacks typically occur: either because a hijacker AS announces a shorter AS path to a prefix, or because it announces a more specific sub-prefix of the hijacked prefix. Figure 1(a) shows a shorter-AS-path hijack: in this scenario, AS 100, which owns the prefix 1.2.3.0/24, announces the


Rest of the Internet
"Reachability to 1.2.3.0/24"
AS Path: "200"
"Reachability to 1.2.3.0/24"
AS Path: "50 100"
AS 200
AS 50
"Reachability to 1.2.3.0/24"
AS Path: "100"
AS 100
Rest of the Internet
"Reachability to 1.2.3.0/25"
AS Path: "50 200"
"Reachability to 1.2.3.0/24"
AS Path: "50 100"
AS 50
"Reachability to 1.2.3.0/25"
AS Path: "200"
"Reachability to 1.2.3.0/24"
AS Path: "100"
AS 200
AS 100

(a) Hijack using a shorter AS Path (b) Hijack using a more specific prefix

Figure 2.1: Example of a malicious AS, AS 200, hijacking AS 100's prefix.

prefix to its upstream provider, AS 50, and also prepends its own AS number to the AS path—which, at this point, merely consists of a single number, "100". AS 50 reannounces the 1.2.3.0/24 prefix to the rest of the Internet after appending its own AS path ("50"), making the total AS path length from anywhere in the Internet at least 2 or more. If a malicious entity, AS 200, is somehow able to announce AS 100's prefix to the Internet with a smaller AS path length, many ASes which see the smaller path length will accept AS 200's announcement as the best path to 1.2.3.0/24. Thus, AS 200 can send traffic—including spam—from an IP address within 1.2.3.0/24 to any AS that accepted its announcement. Figure 1(b) shows a case where a hijack occurs because the hijacker announces a more specific sub-prefix: in this scenario, both AS 200 and AS 100 have equal-length AS paths to the prefix in question, but AS 200 announces a /25 instead of a /24. Thus, due to BGP's longest-prefix-match algorithm, traffic to IP addresses within 1.2.3.0/25 will be routed to AS 200 and not to AS 100.

Although deliberate prefix hijacks and attacks using hijacked routes are not common, we have discovered a stealthy and advanced spamming method distinct from the two hijack scenarios presented above, which we call **BGP spectrum agility**

(Chapter 3.5). This new mechanism to cloak spam involves spammers briefly announcing hijacked less-specific IP prefixes from which they send spam. Although we had observed this behavior informally several years ago [31], and subsequent anecdotal evidence has suggested that spammers may use this technique [9], our study thoroughly documents this activity, and further finds that spammers may be using spectrum agility to complement spamming by other methods. Recent reports about hijacked prefixes used for spam from the North American Network Operators' Group (NANOG) indicate that spam from hijacked may be an increasing problem [6].

2.2 Spam from Botnets

Botnets have been in use as vehicles of cybercrime for quite some time: their activities range from sending massive amounts of spam, mounting denial of service attacks on commercial Web services for monetary gain (*e.g.*, for ransom), hosting and spreading viruses, malware, pornography, and scam pages, click fraud, and identity theft from the owners of compromised computers. Previous research has traced the history of botnets [111, 127, 136] and common modes of botnet operation [22].

Previous work has identified bots by examining the communication protocols used by botnets (*e.g.*, for "rallying"), most notably Internet Relay Chat (IRC) [28, 148]. Some have suggested the use of such protocols to identify and remediate botnets. For example, researchers have joined IRC-based botnets and enumerated victims using IRC commands [33]; others have used network traffic to identify IRC zombies [100]. Some researchers have identified bot victims by observing the unwanted traffic they generate, *e.g.*, the `RST` storms or backscatter generated by DDoS attacks using forged source addresses [86]. Recent network-level techniques for identifying bots using their Command & Control traffic include BotHunter [46], and BotSniffer [47].

Most early methods of controlling and marshalling botnets involved a centralized "Command & Control" (C&C) server—usually using the IRC or HTTP protocols.

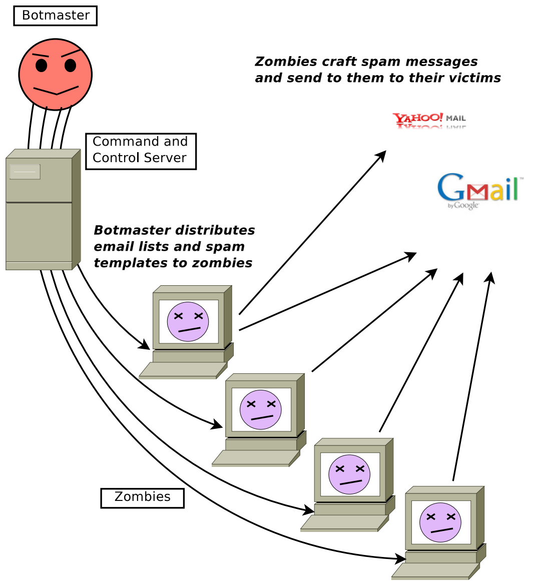
Centrally controlled botnets generally operate according to the pattern shown in Figure 2(a) [23]. Studies show that although the C&C channels to manage bots were predominantly IRC-based [22, 136], newer botnets are almost entirely controlled by peer-to-peer (P2P) protocols [72,128,129]. This development was in response to early botnet mitigation strategies, which largely centered on identifying and neutralizing a centralized C&C. P2P makes a botnet more robust as there is no single point of failure. Another recent development has been the use of encryption for all botnet C&C and data traffic using keys distributed within the bot binary itself.

Attempts have been made to detect such botnets using misuse-detection or basic intrusion detection analysis [15, 49]. Dagon *et al.* used DNS redirection to monitor botnets [25]. Botminer [45] is a more recent work that takes a first-principles approach to detecting bots using a protocol-independent model and passive monitoring of botnet activity. Botlab [62] is a recent effort to capture and study spamming botnets in the wild.

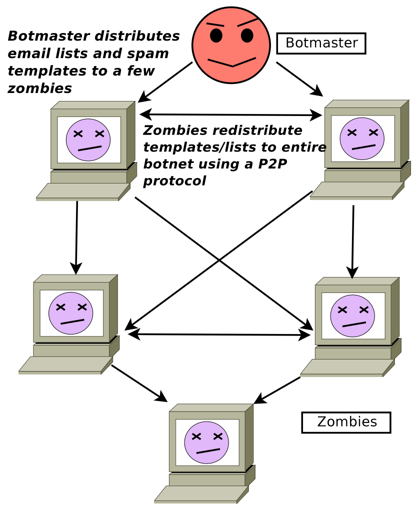
2.2.1 Direct Spamming to the Recipient

Conventional wisdom suggests that the majority of spam on the Internet today is sent by botnets [19, 23, 146]. The amount of spam attributed to botnets over the past few years has been estimated at as high as 95% [130]; the most recent figure from Symantec Labs estimates that 88.2% of spam on the Internet are attributable to botnets [131]. In Chapter 3, we show that although the overall volume of spam from bots is reputed to be high, our spam trap receives low volumes of spam from each bot, which is likely a trick that spammers use to not "trip" volume-based spam filters at any given recipient domain.

Most botnets include code that allows them to send spam using templates supplied by the botmaster. In Chapter 3, we investigate the spamming behavior of the `W32/Bobax` ("Bobax") worm. Bobax (of which there are many variants) exploits the


Botmaster
Command and Control Server
Zombies craft spam messages and send to them to their victims
YAHOO! MAIL
GMail by Google
Botmaster distributes email lists and spam templates to zombies
Zombies

(a) Operation of a centrally controlled botnet, such as the Bobax botnet army


Botmaster distributes email lists and spam templates to a few zombies
Botmaster
Zombies redistribute templates/lists to entire botnet using a P2P protocol
Zombies

(b) Operation of a peer-to-peer botnet, such as the Peacomm worm

Figure 2.2: Differences in operation between centralized and peer-to-peer botnets

DCOM and LSASS vulnerabilities on Windows systems [83], allows infected hosts to be used as a mail relay, and attempts to spread itself to other machines affected by the above vulnerabilities, as well as over email. Agobot and SDBot are two other bots purported to send spam [56] using the centralized C&C model.

Newer peer-to-peer botnets such as the Storm Worm [128] (also known as Trojan.Peacomm, illustrated in Figure 2(b)) and Waledac [129] have more complex infection, rallying, and update behavior, making their eradication all that much harder. In addition to heavy use of encryption and obfuscation techniques to confound researchers, these botnets are rallied using peer-to-peer mechanisms. Each Storm bot, for example, connects to a small subset of 30–35 other bots in the botnet; thus, there is no centralized controller that has control of every bot, which eliminates the threat of the typical approach to botnet takedown of neutralizing the C&C domain name.

2.2.2 Spamming via Webmail Services

Web-based email accounts provided by Gmail, Yahoo! Mail, and Hotmail have also brought new spam threats: spammers have begun using compromised Web mail accounts to send spam. Recent estimates suggest that about 5.2% of accounts that logged in to Hotmail were bots [149]. Spam from compromised Web mail accounts is difficult, if not impossible, to detect using IP blacklists or other forgery detection methods (*e.g.*, domain-key based authentication methods such as DKIM [29]). Web mail providers attempt to detect compromised accounts used to send spam, but these providers handle hundreds of millions of user accounts (193 million users at Gmail [37] and 275 million at Yahoo [58]) and deliver nearly a billion messages each day [139]. Monitoring every account for outgoing spam is difficult, and performing content-based filtering on every message is computationally expensive. Automated monitoring systems may not be able to differentiate a spam sender from a legitimate, high-volume sender.

In Chapter 6, we present a unique attack that is directly used to sustain spamming through Webmail systems. Webmail providers rely on users to quickly mark spam messages as "Spam", so that they can remove copies of the spam messages from other users' Inboxes and also blacklist the sender of the spam. Spammers, in an attempt to pollute these votes, have begun voting "Not Spam" on spam email using *a separate set of Webmail accounts controlled by bots.* Chapter 6 analyzes this attack, and presents an efficient way to discover such dishonest voters.

2.3 Mitigation techniques

Techniques for mitigating spam are as varied as techniques to send spam. In this section, we discuss the most common methods with emphasis on network-level methods.

2.3.1 Content-based Filtering

One of the earliest and most widely used anti-spam techniques is *content-based filtering*, which typically classifies email based on its content; content-based filtering uses features of the contents of an email's headers or body to determine whether it is likely to be spam. Content-based filters, such as those incorporated by popular spam filters like SpamAssassin [120], successfully reduce the amount of spam that actually reaches a user's Inbox. On the other hand, content-based filtering has drawbacks. Users and system administrators must continually update their filtering rules and use large corpuses of spam for training; in response, spammers devise new ways of altering the contents of an email to circumvent these filters. The cost of evading content-based filters for spammers is negligible, since spammers can easily alter content to attempt to evade these filters. Many commercial spam filtering services also use finely tuned content-based filters to perform accurate classification of their email; for example, Yahoo! Mail uses the Sparta system which includes a finely-tuned content-based classifier that gives high accuracy [144]. Unfortunately, they too are vulnerable to the problems of content-based filters—the high cost of running a resource-intensive service, and the need for constant updates or retraining of the classifiers.

Recent large-scale content-based spam filtering techniques have concentrated on using URLs embedded in spam messages—URLs are the sole way spammers can lure unsuspecting users to their spam campaigns, thus, they can be clustered to identify potential spam messages. Li *et al.* focus on clustering spam senders to predict whether a known spammer will send spam in the future [78], and Anderson *et al.* cluster spam according to URLs to better understand the relationship between the senders spam messages that advertise phishing and scam sites and the Web servers that host the scams themselves [8]. Xie *et al.* developed AutoRE [142] to automatically extract regular expressions based on URLs in spam emails. Pathak [96] *et al.* builds upon this work to develop an entirely unsupervised content-based URL clustering algorithm.

Botnet Judo [91] is a system that uses captive bot binaries to capture spam at the source, extract templates of spam emails, and use these to filter spam at recipient domains. Monarch is a recent real-time URL filtering service that uses Amazon's Cloud infrastructure to build a scalable system capable of crawling URLs in email and Twitter feeds and returning a scalable classification decision [132].

2.3.2 IP Blacklists

In addition to performing content-based checks, many mail filters, including SpamAssassin, also perform lookups to determine whether the sending IP address is in a "blacklist". Conventional blacklists constitute lists of IP addresses of likely spammers and are intended to help spam filters make better decisions about whether to block a piece of email based on the sender [121, 122]. Some blacklists are policy-based (*e.g.*, they list all IP addresses that belong to a certain class, such as dialup addresses [119, 123]). Other IP-based blacklists are "reactive": they attempt to keep track of whether an IP address is a spammer, bot, phisher, etc. and keep this list up-to-date as hosts are renumbered, botnets move, and so forth [81, 121, 122, 134]. These blacklists must be vigilantly maintained so as to not going out of date or contain false positives.

IP blacklists are also often referred to as DNSBLs (short for DNS-based IP blacklists), because the method of querying these blacklists is usually through a specially-crafted DNS query. DNSBLs are a "hack" on the DNS name resolution infrastructure to allow users to query for blacklisted IP addresses using existing DNS client and server protocols and utilities. A DNSBL maintainer keeps blacklisted IP addresses in a zone file; the server responds to a query for a listed IP address (encoded in a domain name) with another IP address (usually an address such as `127.0.0.2` that has no meaning in the DNS resolution infrastructure) but returns an `NXDOMAIN` for an unlisted address.

Blacklists of known spammers, open relays and open proxies remain one of today's predominant spam filtering techniques. There are more than 30 widely used blacklists in use today; each of these lists is separately maintained, and insertion into these lists is based on many different types of observations (*e.g.*, operating an open relay, sending mail to a spam trap, etc.). The results presented in Chapter 3—in particular, that IP address space is often "stolen" to send spam and that many bot IP addresses are short-lived—indicate that this long-standing method for filtering spam could become much less effective as spammers adopt more sophisticated techniques.

2.3.3 Other Filtering Methods

Sender Policy Framework (SPF) attempts to prevent IP addresses from sending mail on behalf of a domain for which they are not authorized to send mail [140], and domain keys associate a responsible identity with each mail [7]. Although both frameworks make it more difficult for an arbitrary IP address to send mail, they do not allow a recipient to classify an email sender with an unknown reputation.

Many existing systems perform *collaborative filtering* and whitelisting, which takes inputs from many distributed sources to build information about known spam (or spammers). Some of the most widely deployed collaborative filtering systems characterize known spam based on the contents of a piece of spam that was reported or submitted by another user or mail server [18, 26, 70, 92, 95, 138]. These systems allow mail servers to compare the *contents* of an arriving piece of email to the contents of an email that has been confirmed as spam; they do not incorporate any information about network-level behavior.

Other systems collect information from distributed sets of users either to help filter spam or decrease the probability that legitimate mail is mistakenly filtered. IronPort [60] and Secure Computing [117] sell spam filtering appliances to domains which then pass information about both legitimate mail and spam back to a central processing engine that in turn improves the filters. The widespread deployment of these products and systems make them ideal candidates for the deployment of algorithms such as SpamTracker (Chapter 5).

2.3.4 Analysis of the Economics of Spam

Researchers primarily from the University of California, San Diego have conducted a series of detailed experiments analyzing large-scale botnet-orchestrated spamming operations. Kreibich *et al.* analyzed the logistics of a spam campaign by infiltrating a Storm Worm botnet; their results present an overview of how spammers distribute spamming tasks to bots, how spam templates are created and distributed, how addresses are harvested, etc. [73,74]. Another line of research investigates the conversion rate of spam: by injecting spam messages into a botnet's spam output, the researchers attempt to put a dollar figure to the conversion rate of spam [65]; their results suggest that the conversion rate of spam is very low.

2.4 Related Work in Network-level Spam Filtering

In this section, we survey research that is most related to the research presented in this dissertation, *i.e.*, techniques and algorithms that use network-level features to filter spam.

2.4.1 Characterization of Network-level Properties

Early studies on network-level spam filtering include research by Jung *et al.* that inspects DNS blacklist (DNSBL) traffic and the effectiveness of blacklists [64]; they observed that 80% of the IP addresses that were sending spam were listed in DNSBLs two months after the collection of the traffic trace. Cursory studies before ours have suggested that spammers advertise routes to hijacked IP prefixes for short amounts of time to send spam [31, 124, 133].

Several previous and ongoing projects have studied spammers' attempts to harvest email addresses for the purposes of spamming. For instance, Project Honeypot sinks email traffic for unused MX records and hands out "trap" email addresses to investigate harvesting behavior and to help identify spammers [94]. A previous study has used the data from Project Honeypot to analyze the methods employed by spammers; monitor the time it takes from when an email address is harvested to the time when that address first receives spam; the countries where most harvesting infrastructure is located; and the persistence (across time) of various harvesters [93]. We present preliminary results from a similar study in a technical report version of Chapter 3 [104]. Moore *et al.* found that the majority of hosts—and more than 80% of the hosts in Asia—did not patch the relevant vulnerability until well after actual outbreak [85], which makes it more reasonable to assume that IP addresses of Bobax drones remain infected for the duration of our spam trace.

Anderson *et al.* mines emails in real time and follows URLs embedded in them to cluster spam campaigns and the servers that these campaigns are hosted on [8]; they find that though many hosts are used to send spam about various scams, the scams themselves are hosted on very few machines. Konte *et al.* also analyze the hosting infrastructure of scam and find similar results: the scam campaigns and the server they are hosted on tend to few in number and long-lasting. Botlab [62] studies spamming botnets by capturing bots in a virtual machine and analyzing the spam that they send. The authors captured six spam bot variants and discovered that these were responsible for 79% of all incoming spam at the authors' location.

Table 2.1 summarizes recent research in the characterization of network-level properties of spammers.

Table 2.1: Research in Characterization of Network-level Properties of Spammers

Year	Author	Approach	Results
2006	Ramachandran *et al.* [103]	Measurement-based characterization using a spam sinkhole	Spam senders exhibit high similarity in terms of the IP prefixes from which they originate, number of messages sent, the operating systems used, the size of messages, etc. Spam bots often send a large amount of spam in a "single shot" Some spammers adopt a stealthy technique of sending spam by hijacking large IP prefixes using fake BGP announcements.
2007	Xie *et al.* [143]	Correlating User login data and IPs from Hotmail to detect dynamic IP address regions	97% of mail senders from dynamic IP address regions send only spam 42% of spam arriving at Hotmail is due to spammers in dynamic IP ranges
2007	Collins *et al.* [20]	Reports of the IP addresses of bots, spammers, and phishing hosts	IP addresses that are "unclean" (*i.e.*, have spam/phishing/scan reports against them) tend to cluster with other unclean IP addresses in IP space ("spatial uncleanliness") IP addresses ranges that have a relatively large number of unclean hosts continue to contain unclean hosts for a longer period of time ("temporal uncleanliness")
2008	Pathak *et al.* [90]	Connection data collected at an open relay	Spammers actively scan and "test" open relays before sending spam through them Spammers are either high-volume or low-volume; high-volume spammers send large amounts of spam on their own, while low-volume each send low volumes of spam but act in coordination with other low-volume spammers
2010	Qian *et al.* [98]	Probing experiments to discover the extent of Triangular Spamming	Studies the extent of a well-known attack [110] and finds that 97% of ISPs that disallow outgoing SMTP messages may be vulnerable to it.

2.4.2 Network-level Spam Mitigation Techniques

Behavioral modeling of email sending patterns. Hershkop *et al.* suggested techniques for analyzing email by looking at behavioral features of users (*e.g.*, sending patterns of individual users), as well as n-gram analysis and keyword spotting [54,126]. However, these techniques still rely to some extent on analysis of email contents and focus on the spread of email viruses; the work also proposes an offline analysis toolkit, whereas SpamSpotter (Chapter 7) is a scalable system for online, real-time detection.

Filtering using network-level features. Recent years have seen work that builds on this behavioral modeling by studying the *network-level behavior* of spammers. Our previous work studied the network-level behavior of spammers, with an eye towards developing filters that are based on behavioral features (*i.e.*, how the spam was sent, as opposed to the contents of individual messages) [105]. Clayton *et al.* 's spamHINTS project has also recently been developing techniques for distinguishing spammers from legitimate senders [17]; SpamSpotter could also be a deployment platform for these algorithms and others. Xie *et al.* [143] discovered that a vast majority of mail servers running on dynamic IP address were used solely to send spam. Hao *et al.* developed SNARE, a Spatio-temporal Automated Reputation System that uses a combination of network-level features to build a robust classifier [52]. Beverly and Sollins built a sender-reputation classifier based on transport-level characteristics (*e.g.*, round-trip times, congestion windows) [10] using a support vector machine.

Clustering for Spam classification. Due to the unique properties by which spam is generated and sent, spam data is highly suitable for clustering. As mentioned earlier, previous research has attempted to cluster spam content in various forms: templates of spam message, URLs, and the sites hosted on these URLs, etc.; there have been fewer attempts to perform purely network-level clustering. Our work on SpamTracker [108] clusters both legitimate and spam senders based on the subsets of domains to which they send email. Other work has also attempted to group senders based on recipient [40, 61, 77]. Venkataraman [137] suggest using *network-aware clusters* (*i.e.*, IP prefixes with high propensity for spam) to detect spam from

Table 2.2: Research in Classifiers and Blacklists using Network-level Features

Year	Author	Approach	Results
2007	Venkataraman *et al.* [137]	Measurements using spam and legitimate email received at 700 user mailboxes	IP address blocks are a stable indicator of the nature of message (*i.e.*, spam or legitimate email) Legitimate email mainly comes from long-lived IP addresses, while spam originates from long-lived BGP prefixes (called *network-aware clusters*)
2007	Ramachandran *et al.* [108]	Measurements on sending patterns of legitimate and spam emails to over 100 recipient domains	Bots in a botnet target the same subset of recipient domains as other bots Legitimate senders do not have a discernible pattern in the subset of domains to which they send email.
2008	Zhang *et al.* [147]	Logs from a large-scale security log-sharing system, DShield	IP Blacklists can be made highly predictive and targeted to consumers of the blacklist by correlating multiple reports of an IP address using a link-rank like scheme
2009	Hao *et al.* [52]	Legitimate and spam data for 1 month from a security vendor	A classifier built using only network-level features such as message size, number of recipients, sender's AS number, etc. shows good classification performance
2009	Zhao *et al.* [149]	User login IDs and login IPs from Hotmail	User IDs of bots can be clustered using the IPs that they login from
2010	Qian *et al.* [97]	Email collected from a University	As a continuation of [137], this study shows that filtering based on IP-prefix clusters can be augmented using DNS responses corresponding sender IP addresses.

legitimate email. Qian *et al.* extend this notion of clusters to also include clusters of rDNS servers corresponding to email sender IPs [97].

Other clustering approaches for spam filtering includes BotGraph [149], which attempts to cluster bots that log in to Hotmail using the number of IPs shared by them: accounts controlled by bots are likely to log in from more IPs than a legitimate user account. We use inspiration from this work to use clustering to analyze voting fraud in a large Webmail service provider [102].

Table 2.2 highlights recent and closely-related research that uses network-level features to develop spam mitigation techniques.

2.5 Where This Dissertation Fits in

- Our work on understanding the network-level behavior of spammers was the first exhaustive study in the area [103]. Subsequent research has both analyzed and proved various results presented in this paper. For example, our observation that certain areas of IP space send disproportionately more spam (and contain disproportionate numbers of bots) has been confirmed in later studies [20, 137, 143]; these studies confirm our finding that a sender's IP address ranges can be used as a predictor for the likelihood of its "spamminess".

- Our work was the first to highlight the shortcomings of blacklists [103, 107, 108], and we have extended this study using a longitudinal investigation of the blacklisting status of spammers and a study of dishonest voting patterns in Webmail systems [102].

- We have introduced three novel attacks that were previously unknown or not analyzed in detail [102, 103, 107]. Each of these attacks shed insight into unique facets of the behavior of spammers, and adds to a growing body of work in understanding the intricacies of the spamming process.

- The theme of this dissertation—that IP-based reputation is becoming ineffective, but that spammers can be classified using their coordinated network-level behavior (and not their IP address)—is also well-supported: similar research includes studies on dynamic IP regions [143], network-aware clusters of spam [97, 137], uncleanliness of IP prefixes [20], and BotGraph [149]. Of course, network-level filters alone cannot provide classification of sufficient quality; they must be combined with various content-based filters such as SpamAssassin, or one of the template- or URL-based filters mentioned above [91, 132, 142]. We, however, restrict our work to investigating spam filtering only using network-level features.

- We have built SpamSpotter to address the lack of practical deployment for new network-level classification methods. Although DNS-based IP blacklists are widely used, most new spam-filtering techniques never see public deployment. SpamSpotter implements three network-level spam filtering algorithms: SpamTracker, SNARE [52] and Trinity [13] into a familiar DNSBL-like query interface that can be easily integrated into existing spam-filtering pipelines. We have also implemented SpamLoJack, a tool that identifies spam from potentially hijacked BGP routes, using a DNSBL like interface.

CHAPTER 3

CHARACTERIZING THE NETWORK-LEVEL BEHAVIOR OF SPAMMERS

3.1 Introduction

This chapter studies the *network-level* behavior of spammers, including: IP address ranges that send the most spam, common spamming modes (*e.g.*, bots, direct spamming, BGP route hijacking, etc.), how persistent across time each spamming host is, and characteristics of spamming botnets. We present the results from two kinds of studies of the network-level characteristics of spammers:

1. A focused study using a 17-month trace of over 10 million spam messages collected at an Internet "spam sinkhole" between August 2004 and December 2005. This study establishes the network-level characteristics of spammers and spam bots, the IP ranges from which they originate, their listing status in major blacklists, and their correlation with behavior exhibited by known spam bots.
2. A longitudinal study that studies the "staying power" of network-level attributes using data collected from spam sinkholes between 2006–2011. This study investigates the *variations over time* of several network-level characteristics of spam, including the prefixes from which spam is sent, the size of spam messages, the amount of previously unseen "fresh" bots, the number of messages sent by bots, and their listing status in blacklists.

Our results show that most spam is being sent from a few regions of IP address space, and that spammers appear to be using transient "bots" that send only a few pieces of email over very short periods of time. We also find that a large fraction of spammers are "fresh", *i.e.*, not seen before in the months preceding their spam activity. We find that an increasing fraction of fresh spammers are not listed in any IP blacklists, indicating that spammer tactics may be successful at evading existing network-level defenses. We also found a small, yet non-negligible, amount of spam is received from IP addresses that correspond to short-lived BGP routes, typically for hijacked prefixes.

These trends suggest that filtering email messages based on *network-level* properties beyond just the IP address (which are less variable than email content), and improving the security of the Internet routing infrastructure, may prove to be extremely effective for combating spam. Our longitudinal analysis shows that though variable over the course of 6 years, many characteristics—for example, the prefixes originating the most spam, the average size of emails, the average number of emails sent by a spam bot, etc—are stable and can be used in filtering spam.

Beyond merely exposing spammers' behavior, characterizing the network-level behavior of spam could be a major asset for designing spam filters that are based on spammers' network-level behavior (presuming that the network-level characteristics of spam are sufficiently different than those of legitimate mail, a question we explore further in Section 3.3). Whereas spammers have the flexibility to alter the content of emails—both per-recipient and over time as users update spam filters—they have far less flexibility when it comes to altering the network-level properties of the spam they send. It is far easier for a spammer to alter the content of email messages to evade spam filters than it is for that spammer to change the ISP, IP address space, or botnet from which spam is sent.

We draw the following conclusions from our study:

- *The vast majority of received spam arrives from a few concentrated portions of IP address space (Section 3.3).* Spam filtering techniques currently make no assumptions about the distribution of spam across IP address space. In a related area, many worm propagation models assume a uniform distribution of vulnerable hosts across IP address space (*e.g.*, [125]). In contrast, we find that the vast majority of spamming hosts—and, perhaps not coincidentally, most Bobax-infected hosts—lie within a small number of IP address space regions. This finding has been further investigated in related work [137, 143]

- *Most received spam is sent from Windows hosts, each of which sends a relatively small volume of spam to our domain (Section 3.4).* Most bots send a relatively small volume of spam to our sinkhole (*i.e.*, less than 100 pieces of spam over 17 months), and about three-quarters of them are only active for a single time period of less than two minutes (65% of them send all spam in a "single shot"). Although we have not been able to join bot IPs with more recent data spam data, other researchers [97] and our own later research discussed in Chapter 5 have confirmed that many spammers are indeed "single-shot", with a large fraction of spam each day being sent by previously-unseen IP addresses.

- *A small set of spammers continually use short-lived route announcements to remain untraceable (Section 3.5).* A small portion of spam is sent by sophisticated spammers, who briefly advertise IP prefixes, establish a connection to the victim's mail relay, and withdraw the route to that IP address space after spam is sent. Anecdotal evidence has suggested that spammers might be exploiting the routing infrastructure to remain untraceable [6, 9, 133]; this chapter quantifies and documents this activity for the first time. To our surprise, we discovered a new class of attack, where spammers attempt to evade detection by hijacking *large* IP address blocks (*e.g.*, /8s) and sending spam from widely dispersed "dark" (*i.e.*, unused or unallocated) IP addresses within this space.

- *Many network-level features of spam, such as the IP ranges from which spammers send spam, the average size of messages, etc. remain surprisingly stable over a period of even 6 years*, indicating that these features are suitable for designing robust spam filters.

Beyond these findings, this chapter's joint analysis of several datasets provides a unique window into the network-level characteristics of spam. To our knowledge, our study is the first that examined the interplay between spam, botnets, and the Internet routing infrastructure. We acknowledge that our spam corpus represents only a single vantage point and does not represent conclusive figures about Internet-wide characteristics of spam. On the other hand, the spam we have collected reflects the *complete set of spam emails received by a single Internet domain.* This dataset exposes spamming as a typical network operator for some Internet domain might also witness it. This unique view can help us better understand whether the features of spam that any single network operator observes could be useful in developing more effective filtering techniques.

With these goals in mind and an understanding of the context of our data, we offer the following additional observations on the implications of our results for the design of more effective techniques for spam mitigation, which we revisit in more detail in Section 3.6. First, the distribution of spam and botnet activity across IP space suggests that, for some IP address ranges and networks, spam filters might monitor *network-wide* spam arrival patterns and attribute higher levels of suspicion to spam originating from networks with higher spam activity. Second, spammer techniques such as using previously-unseen IPs and sending a number of spam emails in a "single-shot" may be adversely affecting IP-based reputations systems such as DNS-based blacklists. Finally, the ability to trace the identities of spammers hinges on securing the routing infrastructure. Given the highly variable nature of the content of spam messages, incorporating general network-level properties of spam into filters may ultimately provide significant gains over more traditional methods (*e.g.*, content-based filtering), both through increased robustness and the ability to stop spam closer to its source.

The rest of this chapter is organized as follows. In Section 3.2, we describe our data collection techniques and the datasets we used in our analysis. In Section 3.3, we present focused and longitudinal studies of the distribution of spammers, spamming botnets, and legitimate mail senders across IP address space. Section 3.4 presents our findings regarding the relationship between the spam received at our sinkholes and known spamming bots. Section 3.5 examines the extent to which spammers use IP addresses that are generally unreachable (*e.g.*, using short-lived BGP route announcements) to send spam untraceably. Based on our findings, Section 3.6 offers positive recommendations for designing more effective mitigation techniques. We conclude in Section 3.7.

3.2 Data Collection

This section describes the datasets that we use in our analysis. Our primary dataset consists of the actual spam email messages collected at a large spam "trap" or "sinkhole", over a 6.5-year period between August 2004 and December 2011. We perform a focused study on a 17-month aggregate of this dataset from August 2004–December 2005. We use the remainder of the dataset to perform longitudinal studies of certain network-level features. We call this sinkhole "Spamtrap #1".

Unfortunately, due to problems with the collection server, Spamtrap #1 has had intermittent outages between 2006–2009, some of which lasted many months; thus, there were several gaps in spam collection. Thus, we augment the spam feed for our longitudinal study using spam data from a second spam trap that receives a similar amount of spam per day ("Spamtrap #2"). Both spamtraps sink email for predominantly US-based domain names, receive approximately similar amounts of email, and have a similar collection setups: the spamtraps are physically located in US universities, and both run a "catch-all" setup that accepts any email coming to the mail server.

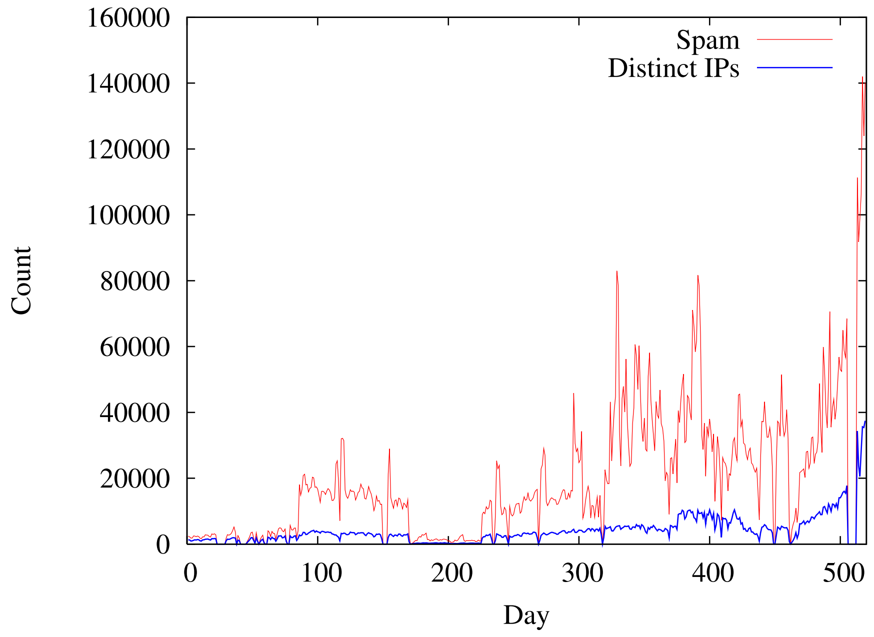
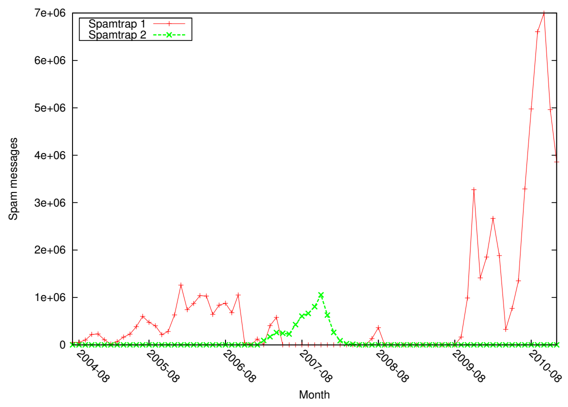


Figure 3.1: The amount of spam received per day at our sinkhole from August 2004 through December 2005.

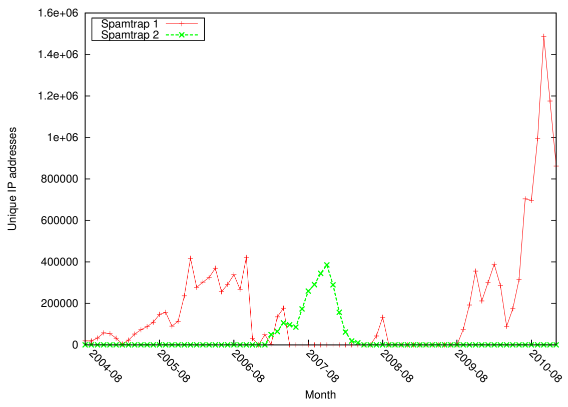
To study the specific characteristics of certain subsets of spammers, we augment this dataset with three other data sources. First, to identify the regions of IP space from which spam arrives and to confirm that these regions are distinct from regions where legitimate mail arrives, we use legitimate email data from a large email security services vendor. Second, we intercept the "command and control" traffic from a Bobax botnet at a sinkhole to identify IP addresses that were infected with the Bobax worm (and, hence, are likely members of botnets that are used for the sole purpose of sending spam). Third, we collect BGP routing data at the upstream border router *of the same network where we are receiving spam* and monitor the routing activity for the IP prefixes corresponding to the IP addresses from which spam was sent.

3.2.1 Spam Email Traces

To obtain a sample of spam, we registered a domain with *no legitimate email addresses* and established a DNS Mail Exchange (MX) record for it. Hence, all mail received by this server is spam. The "sinkhole" has been capturing spam since August 5, 2004.



(a) Spam received per month for each spam trap from August 2004–December 2010.



(b) The number of unique IPs seen each month at the two spam traps.

Figure 3.2: The spam received and the number of IPs seen per month at the two spam traps between August 2004–December 2010.

In addition to simply collecting spam traces, the sinkhole runs Mail Avenger [80], a customizable Simple Mail Transfer Protocol (SMTP) server that allows us to take specific actions upon receiving email from a mail relay (*e.g.*, running traceroute to the mail relay sending the mail, performing DNSBL lookups for the relay's IP address, performing a passive TCP fingerprint of the relay). We have configured Mail Avenger to (1) accept all mail, regardless of the username for which the mail was destined and (2) gather network-level properties about the mail relay from which spam is received. In particular, the mail server collects the following information about the mail relay *when the spam is received*:

- the IP address of the relay that established the SMTP connection to the sinkhole
- a traceroute to that IP address, to help us estimate the network location of the mail relay
- a passive "`p0f`" TCP fingerprint, based on properties of the TCP stack, to allow us to determine the operating system of the mail relay
- the result of DNS blacklist (DNSBL) lookups for that mail relay at eight different DNSBLs.

Note that, unlike many features of the SMTP header, these features are not easily forged.

Figure 3.1 shows the amount of spam that this sinkhole received per day through January 6, 2006 (the 17-month period of time over which we conduct our focused analysis). Although the total amount of spam received on any given day is rather erratic, the data indicates two unsettling trends. First, the amount of spam that the sinkhole is receiving generally appears to be increasing. Second, and perhaps more troubling, the number of distinct IP addresses from which we see spam on any given day also appears to be on the rise.

Figure 2(a) shows the data we use in our longitudinal study: the amounts of spam received by the two spamtraps *each month* over the 6.5-year period, which shows the gaps in collection. Correspondingly, Figure 2(b) shows the number of unique IPs seen each month. The hardware hosting Spamtrap 1 suffered occasional crashes, causing it to not collect spam for months at a time. Whenever collection setup was resumed, we notice that spam "ramps up", indicating that spammers continuously monitor whether mail servers for domains are active even after long periods of inactivity, and correspondingly ramp up mails to domains that accept email.

To make longitudinal analysis easier, we choose 1-month periods of spam roughly one year apart for each year from 2004–2010. Table 3.1 describes the months of data we choose to perform this longitudinal analysis, and the number of unique messages and spammer IP addresses for each month. We expect that, by using data of one month at a time instead of aggregating data from a longer period, we can avoid problems with reallocation of IP addresses, IP address blocks, effects of transient spam campaigns, etc.

Table 3.1: Statistics for the dataset used in our longitudinal study

Month	Spam messages	Distinct IPs	Spam Feed
11/2004	221,131	57,807	Spamtrap 1
11/2005	281,119	113,839	Spamtrap 1
10/2006	1,052,139	421,526	Spamtrap 1
11/2007	1,054,980	385,152	Spamtrap 2
08/2008	363,161	133,206	Spamtrap 1
11/2009	3,271,885	355,610	Spamtrap 1
11/2010	4,963,155	1,175,786	Spamtrap 1

3.2.2 Legitimate Email Traces

One of the motivations for our study was to determine whether the network-level characteristics of spam differ markedly from those of legitimate email. To perform this comparison, we obtained a corpus of the features of various mail received by a large email security appliance vendor that has appliances deployed at hundreds of enterprises. This corpus contains, among other fields, the IP address of each email sender, and an accurate classification of whether the email was classified as spam or not by the vendor's (content-based and network-level) spam-filtering algorithms. Because the logs are post-spam-filtering, we can assume that the incidence of false positives or false negatives is low (*e.g.*, the vendor advertises a false positive rate of lower than 0.01%). We possess this trace only for a period of one day in 2005, but due to the large volume of email handled by the vendor, this trace includes over 4.8 million distinct IP addresses (of which over 94% are spammers).

3.2.3 Botnet Command and Control Data

To identify a set of hosts that are sending email from botnets, we used a trace of hosts infected by the `W32/Bobax` ("Bobax") worm from April 28-29, 2005. This trace was captured by hijacking the authoritative DNS server for the domain running the command and control of the botnet and redirecting it to a machine at a large campus network. This method was only possible because (1) the Bobax drones contacted a centralized controller using a domain name, and (2) the researchers who obtained the trace were able to obtain the trust of the network operators hosting the authoritative DNS for that domain name. This technique directs control of the botnet to the honeypot, which effectively disables it for spamming for this period. On the upside, because all Bobax drones now attempt to contact our command-and-control sinkhole rather than the intended command-and-control host, we can collect a packet trace to determine the members of the botnet.

To obtain a sample of spamming behavior from known botnets, we correlate Bobax botnet membership from the 1.5-day trace of Bobax drones with the IP addresses from which we receive spam in the sinkhole trace. This technique, of course, is not perfect: over the course of our spam trace, hosts may be patched; the hosts' IP addresses may also have changed if they use DHCP. Although we cannot precisely determine the extent to which the transience of bots affects our analysis, previous work suggests that, even for highly publicized worms, the rate at which vulnerable hosts are patched is slow enough to expect that many of these infected hosts remain unpatched [85]; more recent research on the "uncleanliness" of prefixes [20] confirms this fact. However, we believe that the resulting inaccuracies are small: We observe a significantly higher percentage of Windows hosts in the subset of spam messages sent by IP addresses in our Bobax trace than in the complete spam dataset, which indirectly suggests that the hosts with IP addresses from the Bobax trace were indeed part of a spamming botnet when they spammed our sinkhole.

3.2.4 BGP Routing Measurements

We wish to identify cases of spamming where a route for a spammer is reachable for only a short period of time, coinciding with time spam was sent. To measure network-layer reachability from the network where spam was received, we co-located a "BGP monitor" in the same network as our spam sinkhole, similar to that in our

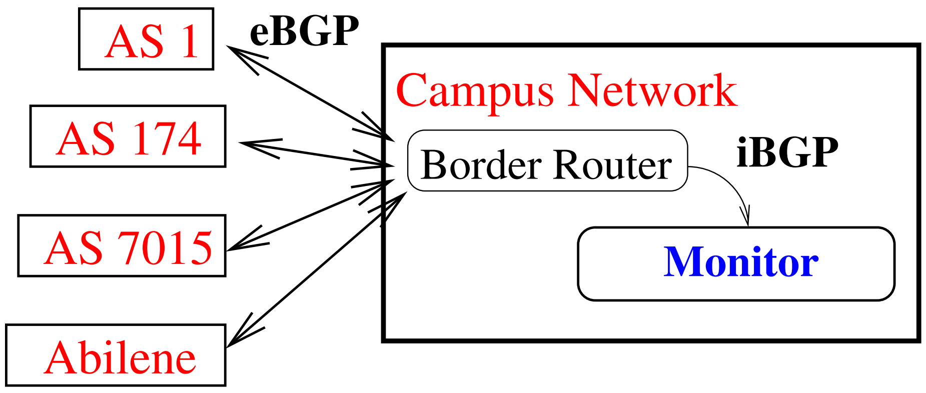


Figure 3.3: At each collection host, we collect BGP messages from the network's border router. The figure shows the configuration for a large campus network, which obtains upstream connectivity from Genuity (AS 1) Cogent (AS 174), Comcast (AS 7015), and Internet2 via Abilene.

previous work [32]. Figure 3.3 shows the placement of the BGP collection host in relation to the border router of its own network (the MIT campus network) and its upstream connection to the rest of the Internet. The monitor receives BGP updates from the border router, and our analysis includes a BGP update stream that overlaps with our spam trace. Since the monitor has an internal BGP session to the network's border router, it will see only those BGP updates that cause a change in the border router's choice of *best* route to a prefix. Despite not observing all BGP updates, the monitor receives enough information to allow us to study the properties of *short-lived BGP route announcements*: the monitor will have *no* route to the prefix at all if the prefix is unreachable.

Because our spam sinkhole was moved since mid-2009 to Georgia Tech, we require BGP routing measurements also from Georgia Tech to perform joint studies of spam and BGP routes. Due to recent incidents of hijacked prefixes being used for spamming, we have developed a joint collection setup in the Georgia Tech campus network to detect spam from hijacked routes with our tool; we present this setup and our tool, SpamLoJack, in Chapter 7.

3.3 Network-level Characteristics of Spammers

In this section, we study some first-order network-level characteristics of spam sources. We survey the portions of IP address space from which our sinkhole received spam and the ASes that sent spam to the sinkhole. To determine whether these network level characteristics could be suitable for filtering spam, we compare the network-level characteristics of spam and legitimate email received by a security appliance vendor.

We find that the distribution of spam across IP address space is quite different from the arrival of legitimate email, especially when observed at the granularity of larger blocks of IP addresses (*e.g.*, /24s). Still, the distribution of spam senders across IP address space is far from uniform, and spam arrival by *IP address range* is much more pronounced, persistent, and concentrated than similar characteristics by IP address. Additionally, we find that a large fraction of spam was received from just a handful of ASes: nearly 12% of all received spam originated from mail relays in just two ASes (from Korea and China, respectively), and the top 20 ASes were responsible for sending nearly 37% of all spam. This distribution (as well as the main perpetrators) is also persistent over time. This heavily skewed distribution suggests that spam filtering efforts might better focus on identifying high-volume, persistent groups of spammers (*e.g.*, by IP block or AS number), rather than on blacklisting individual IP addresses, many of which are transient.

3.3.1 Distribution Across Networks

To determine the address space from which spam was arriving ("prevalence") and whether the distribution across IP addresses changes over time ("persistence"), we tabulated the spam in our trace by IP address space. We find that spam arrivals across IP space are far from uniform.

Finding 3.3.1 (Distribution across IP address space) *The majority of spam is sent from a relatively small fraction of IP address space.*

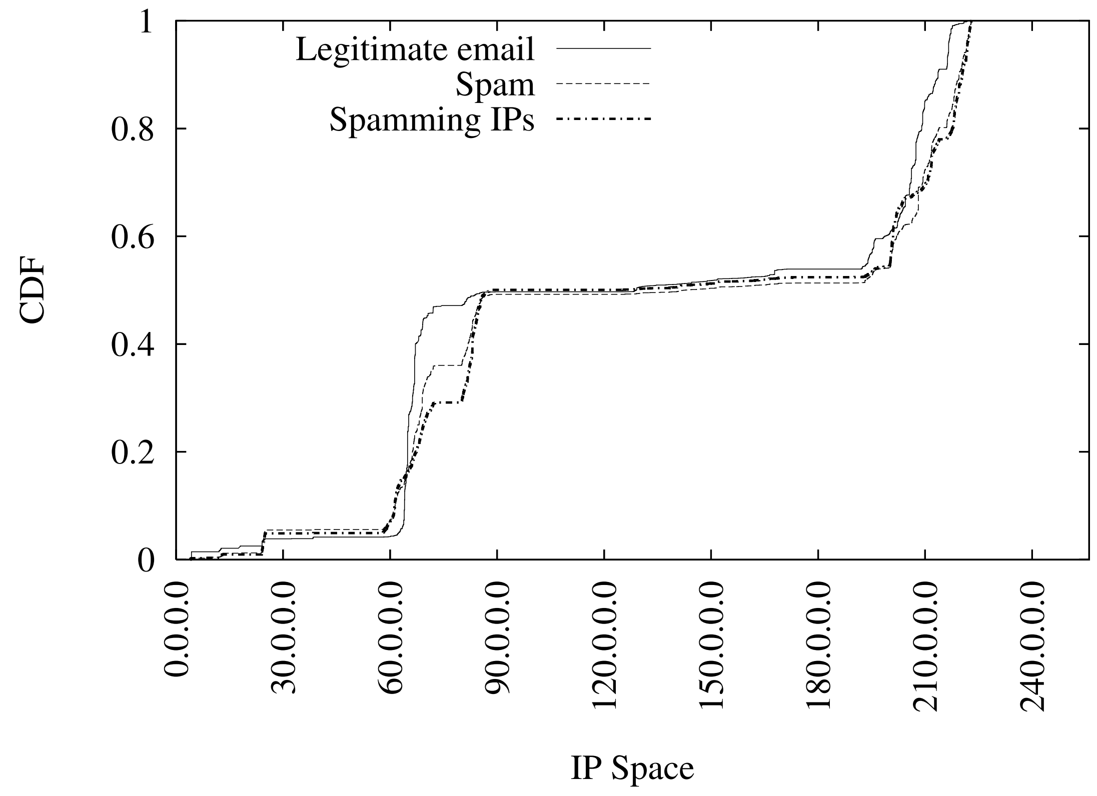


Figure 3.4: Fraction of spam email messages and comparison with legitimate email received (as a function of IP address space); also, fraction of client IP addresses that sent spam, binned by /24.

Figure 3.4 shows the number of spam email messages received over the course of the entire trace, as a function of IP address space. A few IP address ranges have significantly more spam than legitimate mail (*e.g.*, 80.*–90.*), and vice versa (*e.g.*, 60.*–70.*). Although Figure 3.4 may indicate that legitimate email and spam email arrives from largely the same prefixes, the plot does not show variations in sending behavior smaller IP blocks adjacent to each other (*e.g.*, /24s).

To investigate the nature of /24s better, we plot the IP addresses of legitimate and spam email address senders using a Hilbert curve representation of IP address space. A Hilbert curve is a space-filling curve with the property that when used to represent numbers, adjacent numbers always occupy adjacent positions on the curve. To draw a Hilbert curve that represents each /24 block as a single pixel, we used a

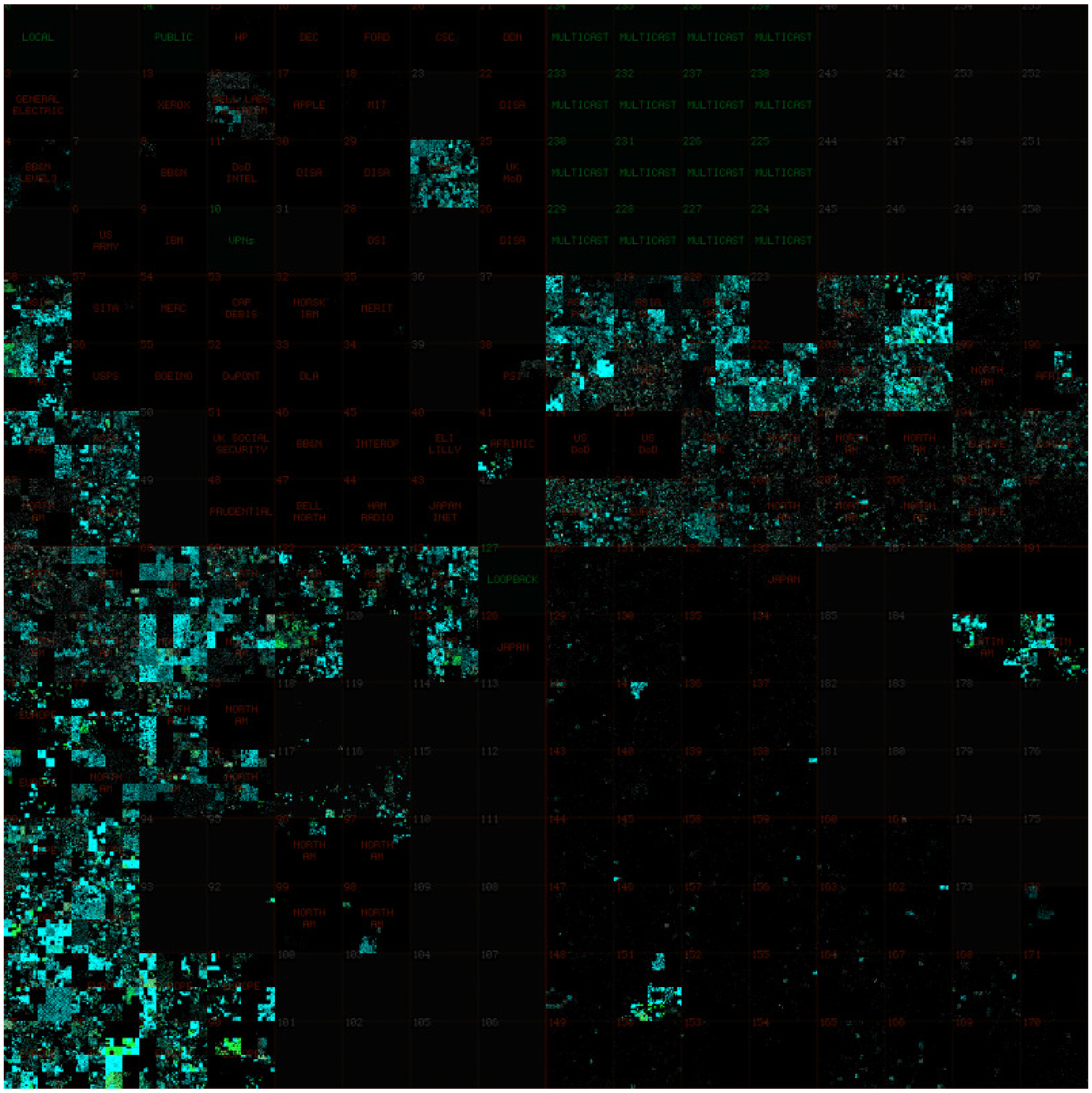


Figure 3.5: Hilbert space representation of the prefixes that send only spam (cyan), only legitimate email (green), or both (white). Each pixel corresponds to one /24.

12th order Hilbert curve—*i.e.*, a square with 2^{12} pixels in each dimension, where each pixel represents an IP address.

Figure 3.5 shows the Hilbert curve [55] representation of the entire IP address space. /24s that sent only spam email are denoted in cyan, and /24s that send only legitimate email are in green. /24s which include IPs which sent both spam and legitimate email are in white. This graph confirms our hypothesis that email senders

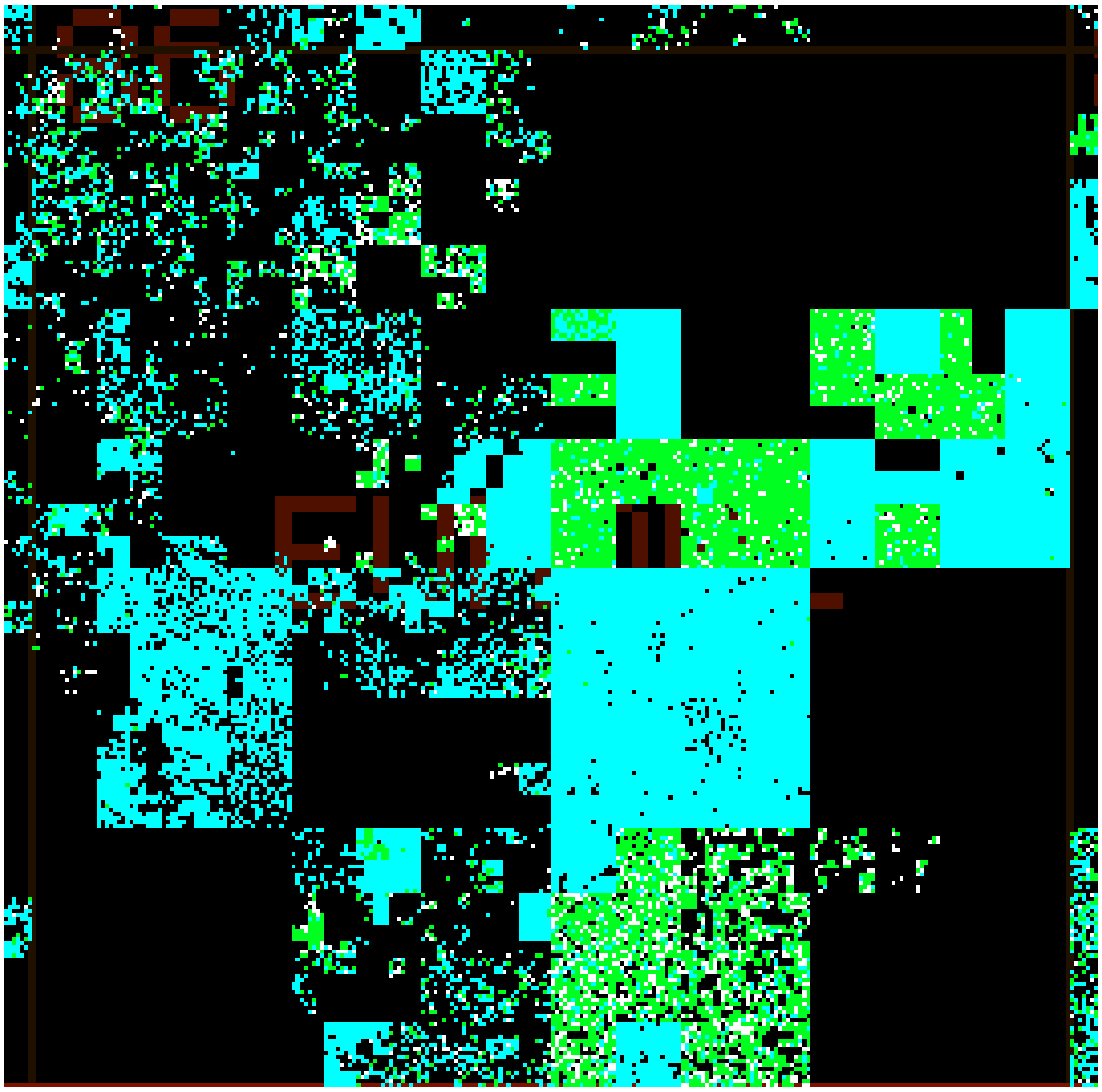


Figure 3.6: Zoomed-in view of a single /8 from Figure 3.5, 86.0.0.0/8.

from large, adjacent IP blocks tend to be either predominantly spam or predominantly legitimate, as indicated by multi-pixel square blocks of green and cyan, and the relative absence of white pixels.

Figure 3.6 shows a close-up only of a specific block, 86.0.0.0/8—one of the "spammier" /8s in our corpus, which offers a detailed look at the sending behavior of adjacent /24s. We see many blocks larger than a /24 that have entirely spam or legitimate senders. An interesting observation is that nearly all of the false positive

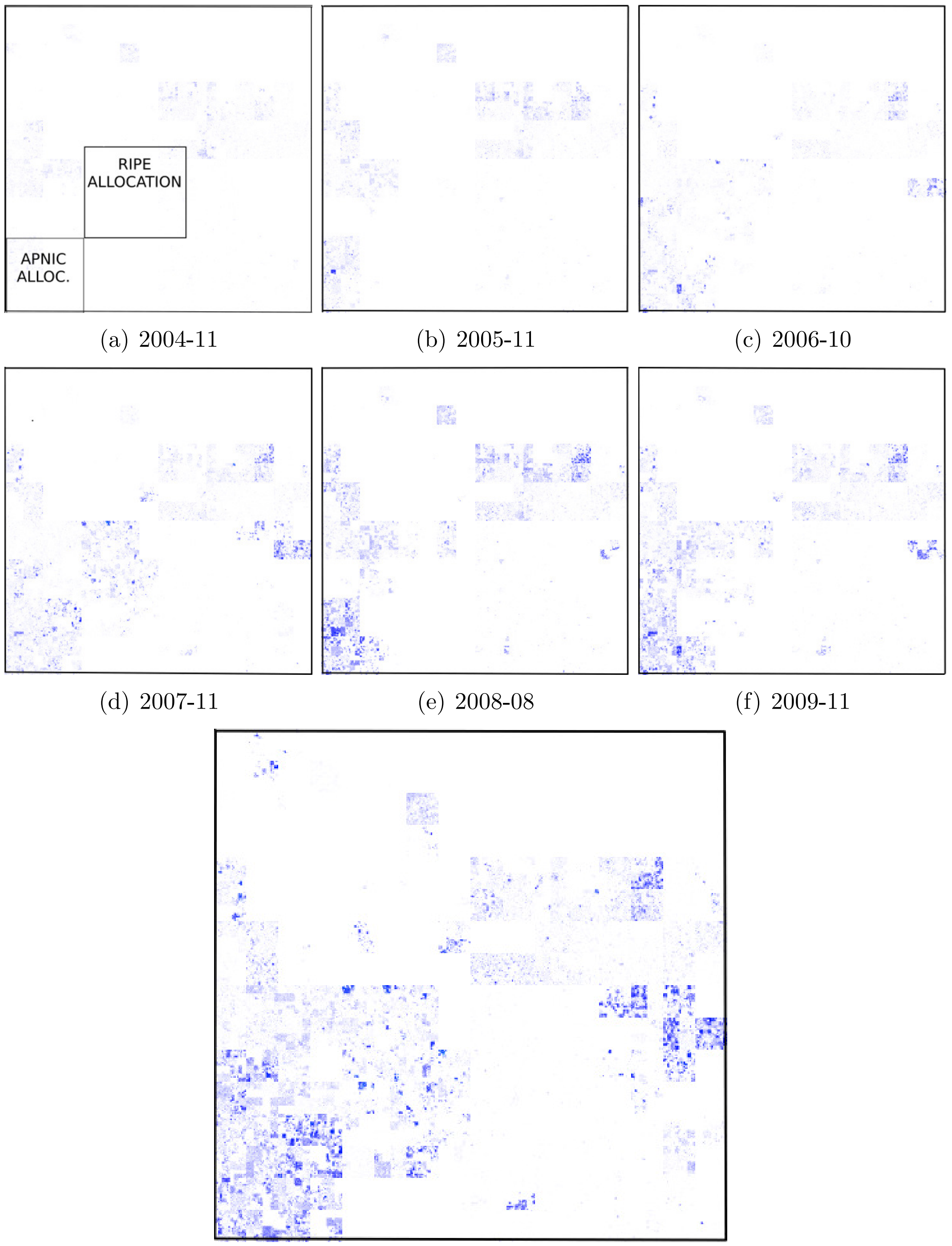
/24s (white pixels) appear within blocks of green (*i.e.*, legitimate senders only). We hypothesize that most of these false positives correspond to true legitimate mail service providers; although most email sent from these providers is legitimate, spammers often compromise accounts and attempt to send using these legitimate servers that thwarts blacklisting approaches [149]. This emerging threat of compromised accounts at legitimate mail service providers (*e.g.*, Webmail providers such as Yahoo! Mail and Gmail) is further complicated by the existence of *vote gaming*—an attack that stymies the provider's efforts to quickly identify spam messages and senders using feedback from its legitimate users. We present this attack and a mitigation technique in Chapter 6.

Longitudinal Study. We wish to see whether the regions that send the most spam have changed over time; a knowledge of the regions that have a propensity for spam can help mail recipients have some prior idea whether a particular previously-unseen IP address is likely to be a spammer or not.

Finding 3.3.2 (Distribution Across IP space (Longitudinal Study)) *Over time, historically spam-heavy IP address ranges continue to grow, and new spammer IP address ranges tend to be close to past spammer ranges.*

Figure 3.7 shows the Hilbert space representations of spammer IP addresses for the 7 months in our longitudinal dataset. We can see that there are increasing numbers of spam bots each year, but these are concentrated in certain regions of IP space. For example, the bottom left octant the map roughly corresponds to the European RIPE NCC allocation (highlighted in the first figure). Some European ISPs are traditionally known to be spam-friendly, and we see that the number of such spam bots has increased in these IP address ranges over the years. We also see that the Asia Pacific allocation (APNIC) has grown from nearly no spammers in 2004 to a large number of spammers in 2010. Most of this IP address range (110/8–126/8) was

Figure 3.7: Hilbert curve representations of IP space sending the most spam (indicated in colored pixels) for different points of time over 6 years.



(a) 2004-11 (b) 2005-11 (c) 2006-10

(d) 2007-11 (e) 2008-08 (f) 2009-11

(g) 2010-11

allocated between 2007–2009, and the increase in spam from these IP addresses also supports at the large amount of spam arriving from Asia (India, China etc.).

These results indicate that although the number of spammers are increasing globally, we can make a few observations about the future growth of spam senders in IP space. (1) they tend to concentrate in regions of IP space that correspond to ISPs (and geographical locations) that historically have large amounts of spam; and (2) new spammer IPs tend to be close in IP space to past spammer IPs. These observations can be help in the better design of spam filters: because ISPs do not change or sell their IP allocations often, and becuase spam-friendly ISPs are unlikely to take an anti-spam stance all of a sudden, we can assume that if an IP block is known to contain spammers, it is quite likely that it will contain spammers in future—an observation also confirmed by Collins *et al.* [20]. Similarly, because new spammer IPs (and blocks of IPs) tend to be close in IP space to previous spammer IPs, a spam filter could automatically "grow" a previously-known spammer IP address block if it sees spam from adjacent blocks.

3.3.2 Distribution Across ASes and Countries

In many cases, IP address ranges are not adequate for distinguishing spam from legitimate email. To determine whether other network-level properties, such as the AS from which the email was sent, could serve as better classifiers, we examined the distribution of spam across ASes and compared this feature to the distribution of legitimate email across ASes.

Finding 3.3.3 (Distribution across ASes) *More than 10% of spam received at our sinkhole originated from mail relays in two ASes, and 36% of all received spam originated from only 20 ASes. With a few exceptions, the ASes containing hosts responsible for sending large quantities of spam differ from those sending large quantities of legitimate email.*

The concentration of spammers in a small collection of offending ASes—and the fact that this collection of ASes differs from the ASes responsible for sending legitimate

Table 3.2: Amount of spam received by our spam trap between August 2004–December 2005 from mail relays in the top 20 spammy ASes. 11 of the top 20 networks from which we received spam are primarily based in the United States.

AS Number	*# Spam*	*AS Name*	*Primary Country*
766	580559	Korean Internet Exchange	Korea
4134	560765	China Telecom	China
1239	437660	Sprint	United States
4837	236434	China Network Communications	China
9318	225830	Hanaro Telecom	Japan
32311	198185	JKS Media, LLC	United States
5617	181270	Polish Telecom	Poland
6478	152671	AT&T WorldNet Services	United States
19262	142237	Verizon Global Networks	United States
8075	107056	Microsoft	United States
7132	99585	SBC Internet Services	United States
6517	94600	Yipes Communications, Inc.	United States
31797	89698	GalaxyVisions	United States
12322	87340	PROXAD AS for Proxad ISP	France
3356	87042	Level 3 Communications, LLC	United States
22909	86150	Comcast Cable Corporation	United States
8151	81721	UniNet S.A. de C.V.	Mexico
3320	79987	Deutsche Telekom AG	Germany
7018	74320	AT&T WorldNet Services	United States
4814	74266	China Telecom	China

Table 3.3: Top 10 ASes (by email volume) in our legitimate email trace.

AS Number	*# Email*	*AS Name*	*Primary Country*
15169	49500	Google Inc.	United States
5731	38238	AT&T WorldNet Services	United States
26101	30406	Yahoo	United States
3561	22730	Savvis	United States
4355	17381	Earthlink, Inc	United States
8560	16666	Schlund Partner AG	Germany
8075	14699	Microsoft Corp	United States
14779	13115	Inktomi Corporation	United States
6541	12493	GTE.net LLC	United States
14780	11597	Inktomi Corporation	United States

email (with the exception of ASes 5731 and 8075)—suggests that spam filters should attribute more suspicion to email coming from ASes where spam commonly originates. This observation begs the question about why Figure 3.4 does not show similar differences. Indeed, the spamming behavior of specific IP address ranges deserves further study, since Figure 3.4 really only exposes macro-level behavior of IP address ranges (*i.e.*, differences for small IP address ranges may not be visible in the figure). We are studying the behavior of fine-grained address ranges in ongoing work.

Recent reports have claimed that most spam originates in the United States [63]. On the other hand, Figure 3.4 suggests that many spamming hosts reside in IP address space that is allocated to the Asia-Pacific region (*e.g.*, 61.0.0.0/8). To perform a rough estimate of the amount of spam originating from each country, we associated the ASes from which we received spam to the countries where those ASes were based.[1] Table 3.2 also shows the distribution of hosts that sent spam to the sinkhole by country, for the top 20 ASes from which we received spam.

Finding 3.3.4 (Distribution by country) *Although the top two ASes from which we received spam were from Asia, 11 of the top 20 ASes from which we received spam were from the United States and comprised over 40% of all spam from the top 20 ASes.*

We mapped the most prolific IP address (*i.e.*, the top 11.6% of IP addresses, responsible for 65% of all spam received at the sinkhole) to their respective countries. Our analysis indicates that nearly three times as much spam in our trace originates from ISPs based in the United States than from either of the next two most prolific countries (Korea and China, respectively). This conclusion does differ from other reports, which also indicate that most spam comes from the U.S., but to a much lesser degree.

[1] Although some ASes span multiple countries, typically even large transit providers have different AS numbers for backbone networks in different countries. In any case, we use the *primary* country where the AS is based.

The distribution of spam by country, when compared to the statistics for legitimate email (Table 3.3), also suggests that, in some cases, assigning a higher level of suspicion according to an email's *country* of origin may be an effective filtering technique for some networks.

3.3.3 Sending Patterns of Spammers

Finding 3.3.5 (Transience of Spammers) *Individual spammer IP addresses are transient: a large majority send emails to our sinkholes only once.*

Individual IP addresses are far more transient than IP address ranges. Figure 3.8 shows that even though a few IP addresses sent more than 10,000 emails to our spam trap, about 85% of client IP addresses sent less than 10 emails to the sinkhole in the entire 17 month period, indicating that targeting an individual IP address might not help mitigate spam without sharing information across domains. This finding has an important implication for spam filter design: Though the individual IP addresses from which spam is received changes from day-to-day, the fact that spam continually comes from the same IP address *space* suggests that incorporating these more persistent features may be more effective, particularly in portions of the IP address space that send either mostly spam or mostly legitimate email.

Longitudinal Study. In this Section, we investigate how the sending patterns of spammers change over the 6.5 year longitudinal trace. Specifically, we investigate two artifacts: (1) what fraction of spammers are "fresh", *i.e.*, ones that send email to our sinkhole for the first time in a long period (*e.g.*, one month); and (2) what fraction of spammers maintain the behavior of sending spam to our sinkhole just once within the 1-month observation period. Both techniques would hinder the ability to reliably blacklist a spammer's IP address.

Finding 3.3.6 (Fraction of "Fresh" Spammers (Longitudinal Study)) *Well over*

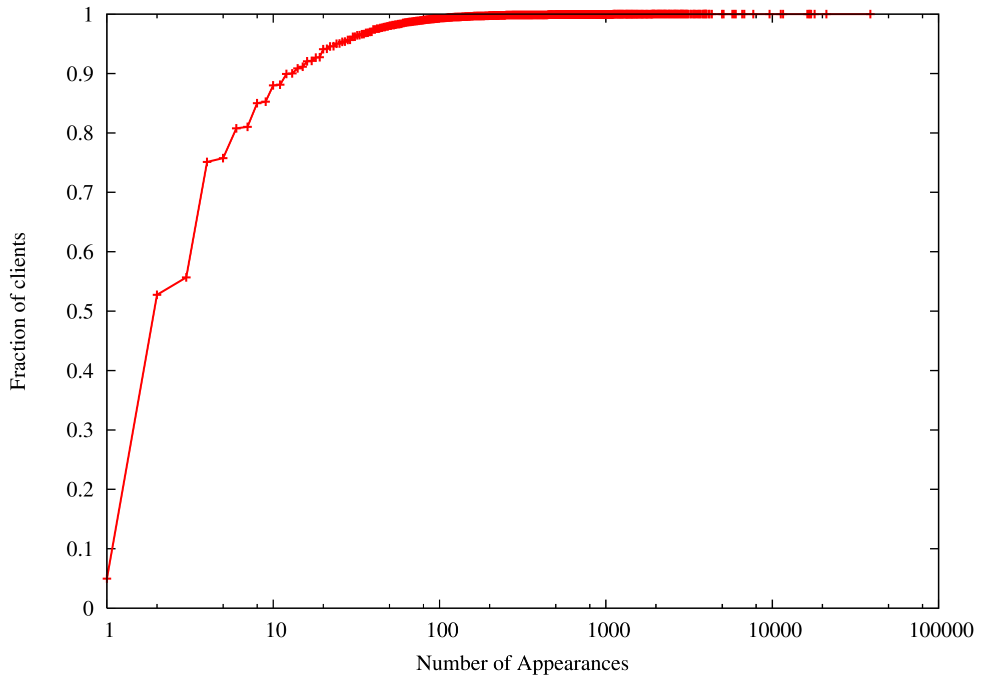


Figure 3.8: The number of distinct times that each client IP sent mail to our sinkhole (regardless of the number emails sent in each batch).

90% of spammers we see at our spam trap in a month were not observed for the previous one month, and approximately 90% were not seen for the previous two months.

Table 3.4 shows the fraction of spamming IP addresses we did not observe at our sinkhole in the preceding month before the spam was received, and the amount of spam due to such fresh IPs each month. 90% or more of the IPs that sent spam to our sinkhole are "fresh" by this measure; even if we "look back" for two months instead of one month, this fraction does not decrease. For example, 96.5% of IPs seen during October 2006 at our spam trap were not seen for at least 1 month before, and 92.9% were not seen for at least 2 months. The overwhelming number of such fresh addresses indicates that a single domain—or even a small group of domains—are unlikely to be able to filter spam effectively using just IP-based blacklisting. We also note that the fraction of fresh IPs remains consistent over time, indicating that blacklists will continue to have a hard time keeping up with such fresh IP addresses.

Table 3.4: The fraction of spamming IPs that have not been seen in the previous month ("fresh" IPs), and the fraction of spam from such IPs.

Month	% IPs that are "fresh"	% Spam from fresh IPs
11/2004	88.9%	28.1%
11/2005	92.3%	44.0%
10/2006	96.5%	49.6%
11/2007	92.2%	38.1%
08/2008	94.2%	43.7%
11/2009	94.0%	31.3%
11/2010	91.5%	32.3%

An interesting observation from Table 3.4 is the relatively small amount of spam from such fresh IP addresses. As we show in Chapter 5, these fresh IP addresses each send small amounts of spam to many domains, implying that only a view from multiple vantage points can reveal the true magnitude of the volume of spam sent by these IPs. The majority of spam to our spam trap is being sent by a small fraction (less than 10%) of spammers. We expect, however, that such high-volume spammers will be easily filtered at large mail service providers, implying that the remaining "fresh" IPs are the ones that contribute to the majority of the hard-to-filter spam.

Finding 3.3.7 (Fraction of "single-shot" Spammers (Longitudinal Study)) *More than 90% of spammers we see at our spam trap in a month sent multiple spam messages over a single 10-second window; such single-shot spammers amounted to 80.1% of all spam received in 2010.*

Spam from single-shot senders. Next, we investigated whether the single-shot nature of senders that we observed in Figure 3.14 persists over time. We calculate the number of IPs in each month that sent email in a single short burst to our spam trap, which is defined as a 10-second interval to account for delays at the sender and our spam trap setup. Table 3.5 shows this finding. There is a clear trend of increasing amounts of IPs that hit mail recipients in a single shot, and also an increasing trend

Table 3.5: The fraction of IPs in our trace that sent email in a single shot, and the amount of spam attributable to single-shot IPs. We define "single-shot" as an IP sending email only within a 10-second interval.

Month	% single-shot IPs	% spam from single-shot IPs
11/2004	70.4%	25.0%
11/2005	78.4%	43.6%
10/2006	86.5%	67.0%
11/2007	80.1%	46.0%
08/2008	89.5%	74.2%
11/2009	90.6%	80.0%
11/2010	94.7%	80.1%

of the amount of spam attributable to such IPs. We believe that this trend further supports anecdotal evidence about a bot behavior: spammers, in an attempt to avoid blacklisting, use their bots to hit one domain at a time and do not return to the same domain for months. Moreover, to maximize the amount of spam delivered, nearly all these single-shot bots send multiple messages at a time, which amounts to 80.1% of received spam by the end of 2010.

3.3.4 The Effectiveness of Blacklists

Given the transience of each IP address sending spam to our sinkhole (*i.e.*, the results shown in Figure 3.8), we suspected that filtering based on IP address, a method commonly employed by DNSBLs, would be affected by these techniques. To test this hypothesis, we used the results from real-time DNSBL lookups performed by Mail Avenger to 8 different blacklists *at the time the mail was received* .

Figure 3.9 indicates that IP-based blacklisting is still working reasonably well *if many blacklists are consulted simultaneously*: Although 20% of spam came from IP addresses that were not listed in *any* blacklist, (as shown by the middle line "All spam", where about 80% spam was listed in at least one blacklist), more than 50% of such spam was listed in two or more blacklists, and 80% was listed in two or more blacklists. Unfortunately, the 20% which were not listed in any blacklist are likely to

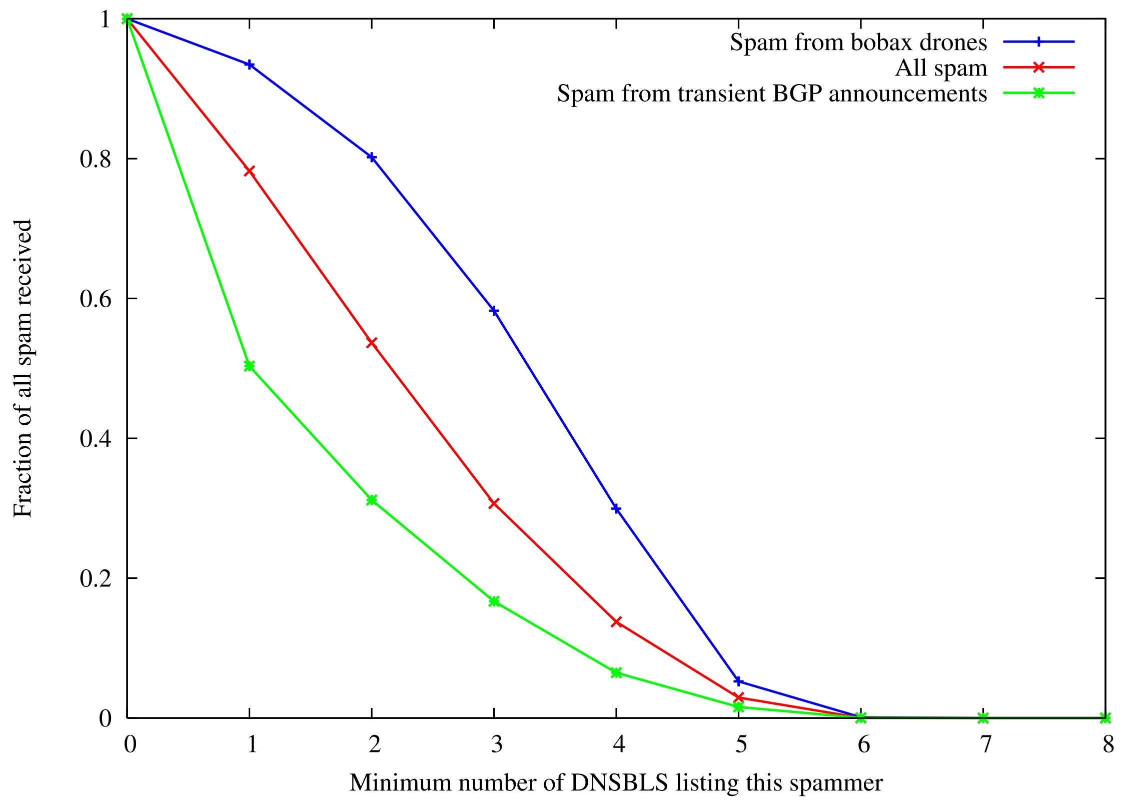


Figure 3.9: The fraction of spam emails that were listed in a certain number of blacklists or more, *at the time each mail was received.*

cause the most uncaught spam; this was our motivation in designing SpamTracker, a *behavioral* blacklisting system described in Chapter 5.

More troubling, however, is that the spam that we received from spammers using "BGP spectrum agility" techniques (Section 3.5) are not blacklisted nearly as much: half of these IP addresses do not appear in *any* blacklist, and only about 30% of these IP addresses appear in more than one blacklist.

Finding 3.3.8 (Effectiveness of blacklists) *Nearly 80% of all spam was received from mail relays that appear in at least one of eight blacklists. A relatively higher fraction of Bobax drones were blacklisted, but relatively fewer IP addresses sending spam from short-lived BGP routes were blacklisted—only half of these mail relays appeared in any blacklist.*

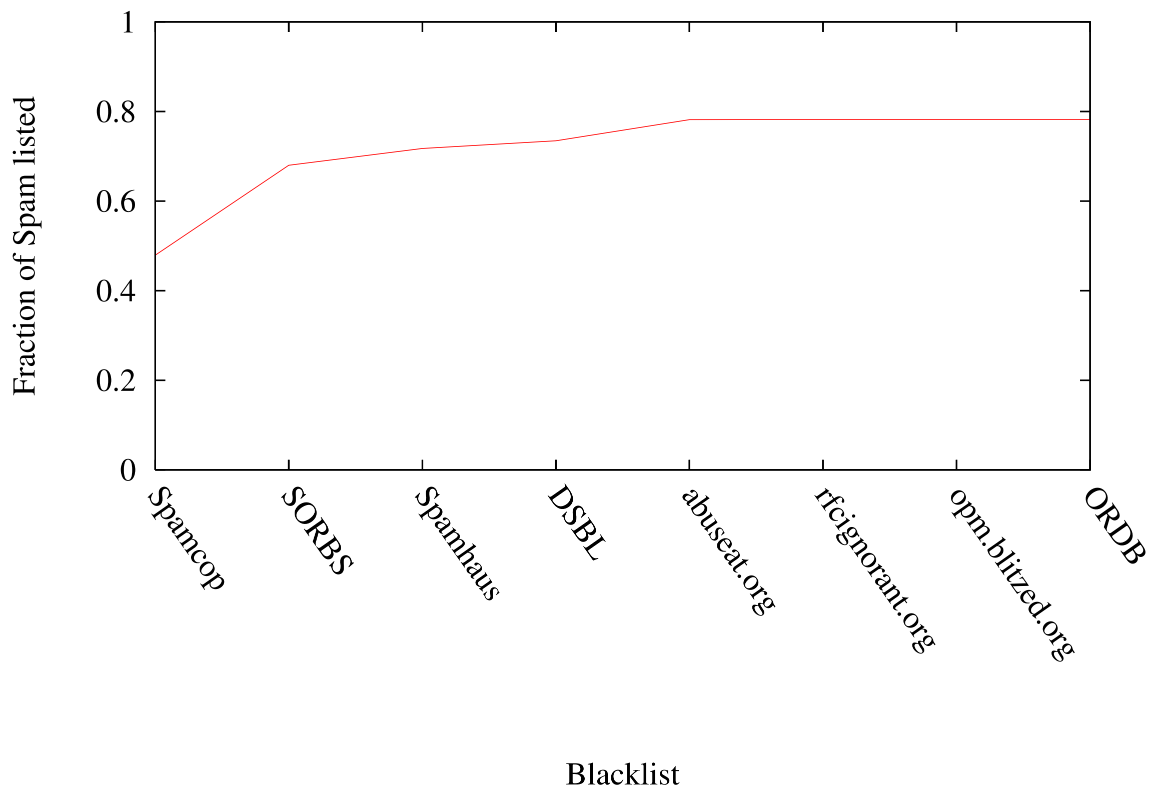


Figure 3.10: The cumulative fraction of spam emails that were listed in each blacklist *at the time each mail was received*, sorted from most aggressive to least aggressive blacklist.

Although this finding appears to suggest that DNSBLs are effective at identifying most types of spam based on IP address, the reality is actually not as bright as it appears. First, this result is based on an aggressive approach that sends queries to *eight* blacklists; Figure 3.10 shows the cumulative fraction of spam listed in each blacklist, from most aggressive DNSBL to least aggressive and shows that even the most aggressive blacklist, Spamcop, only lists about half of all spam received. Second, many of the more aggressive blacklists are known to have a significant number of false positives. Finally, even aggressive mechanisms, such as querying eight different blacklists, are fairly ineffective at identifying IP addresses using more sophisticated cloaking techniques (*e.g.*, the BGP spectrum agility technique, which we discuss in more detail in Section 3.5).

Longitudinal Study. We expect that spammers use techniques such as fresh IPs and single-shot spamming to escape blacklisting. Because our spam collection infrastructure automatically queries the recipient IP address in one of six blacklists.[2], we can quantify exactly how well spammers were escaping blacklisting with fresh IP addresses versus previously seen IPs.

We studied the difference in listing behavior between "fresh", previously unseen IPs (as defined earlier) and IPs that have been seen in the past. Figure 3.11 plots the percentage of IPs that were not listed in one of six DNS blacklists on arrival. The three plots correspond to sender IPs in the "fresh" set, the remaining "seen" sender IPs, and finally, fresh IPs which were only listed by Spamhaus's policy blacklist (PBL) [123].

Finding 3.3.9 (Listing status of "Fresh" IPs (Longitudinal Study)) *A significant fraction of spamming IPs—nearly 40% of all fresh IPs in 2007—are not listed in* any *of 6 blacklists at the time the mail was received. This fraction goes up to 50% if we also discount fresh IPs only listed by policy (and not by actual confirmation of spam), indicating that spammer techniques are successfully avoiding listing in blacklists, and that coordinated detection methods are necessary.*

The IPs listed in the PBL are put on the blacklist not due to evidence of malicious behavior, but because they were identified by their ISP as belonging to a dynamic IP address space that should *typically* not be making direct SMTP connections to remote mail recipients. Because of the chance of false positives due to legitimate residential senders being blacklisted, many mail server administrators do not reject email using the PBL alone. Hence, we include a third plot that captures the listing status of fresh IP addresses that were listed only in the PBL. Note that we only have PBL

[2]Two out of the eight blacklists we queried in 2004 have shut down since, and we exclude these blacklists in our longitudinal study.

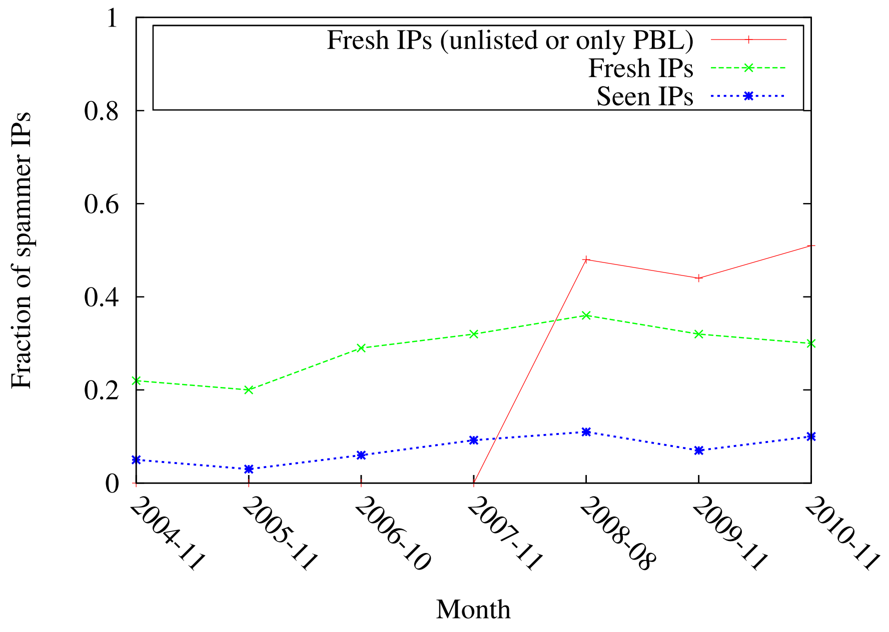


Figure 3.11: The percentage of IPs that are not listed in any blacklist at the time it hits our spam trap. "Fresh" IPs belong to senders which have not been seen at our spam trap for at least a month, and "Seen" IPs include all other senders. The third plot represents fresh IPs that were either unlisted or only listed in Spamhaus's policy blacklist [123].

lookup information from 2008 onwards. We see that fresh IPs have a significantly lower listing rate than "seen" IPs, and this listing rate is even worse when PBL is not consulted: In November 2010, about 32% of fresh IPs were not listed in any blacklist at all, and when adding fresh IPs listed only in the PBL, the fraction jumps to over 51%.

3.3.5 Message Size Distribution of Spam

In this section, we study the average message size distribution of spam across our longitudinal trace. As illustrated in Figure 3.12, we find that most months have an average of 1000 bytes or less, with a significantly higher month correlating with the peak of PDF spam. Although we do not have a longitudinal legitimate email trace to compare message size growth, we believe that legitimate email will exhibit a higher average and a higher standard deviation; indeed, the SNARE algorithm [51] uses

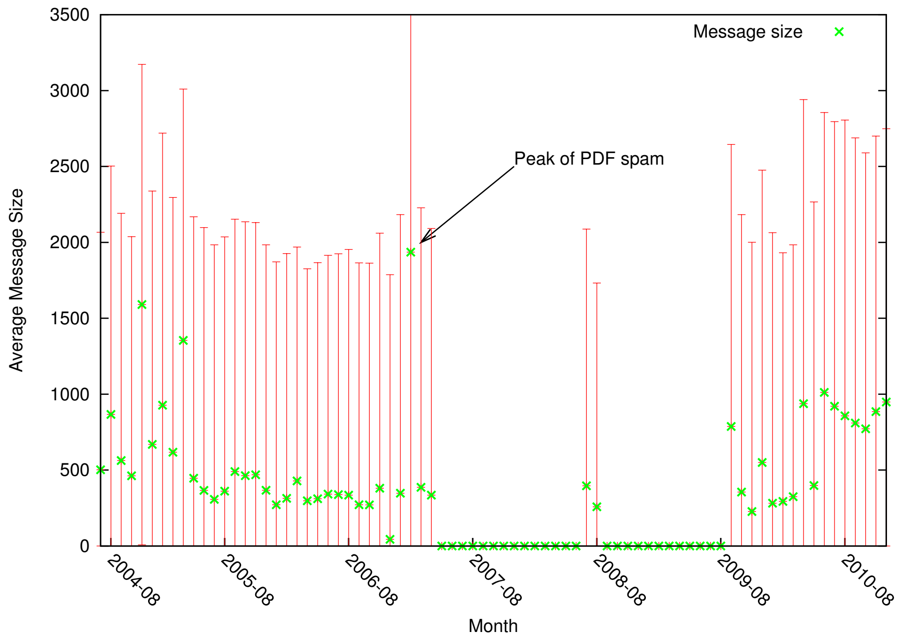


Figure 3.12: The average message size (in bytes) per month in our spam trap 1 trace, with error bars showing one standard deviation.

message size as one of the discriminating features in designing its spam classifier.

Finding 3.3.10 (Message Size Distribution of Spam (Longitudinal Study)) *Average message sizes of spam remain small and approximately constant over time, indicating that bots continue to churn out large volumes of small spam messages to efficiently use the traditionally slow uplinks in home Internet connections.*

3.4 Spam from Botnets

In this section, we amass circumstantial evidence that suggests that a majority of spam originates from bots. Although, given our limited datasets, we cannot determine a precise fraction of the total amount of spam that is coming from bots, we use our trace of "Bobax" command and control data to study the patterns of spam that are being sent from hosts that are known to be bots. First, we study the activity profile of drones from the "Bobax" botnet and find that the IP address space where we observe worm activity bears close similarity to the IP address space where we

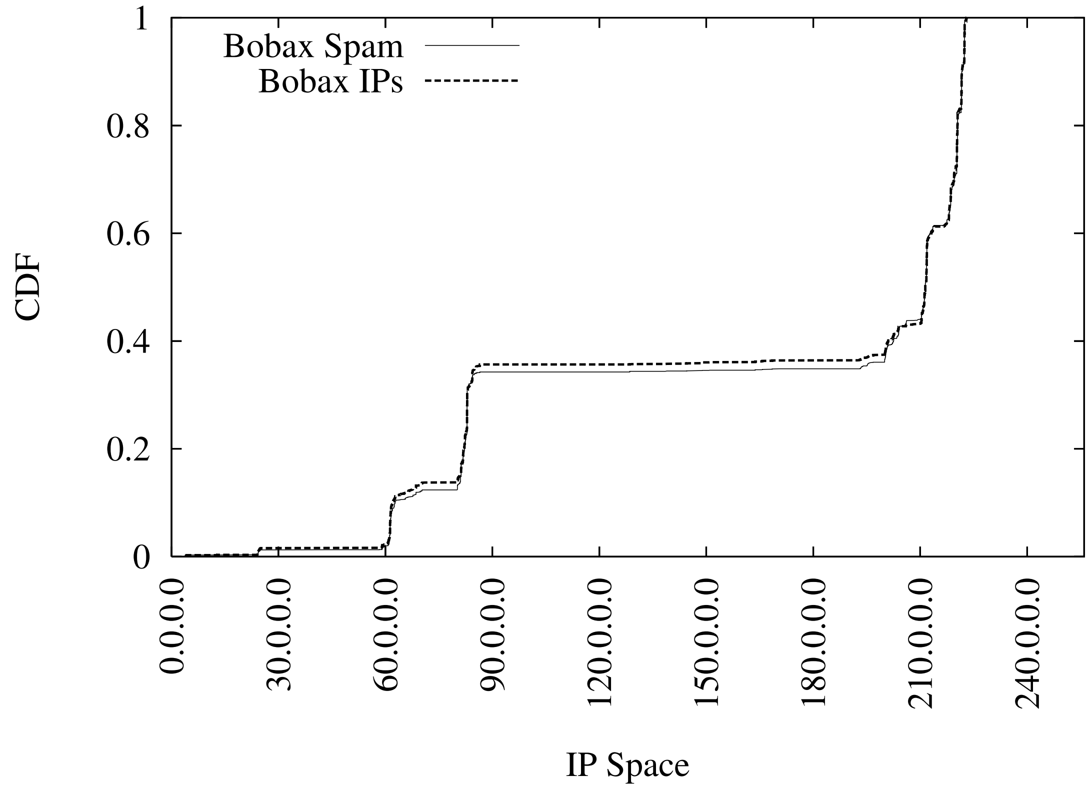


Figure 3.13: The number of *all* Bobax drones, and the amount of spam received from those drones at the sinkhole, as a function of IP address space. On the x-axis, IP address space is binned by /24.

observed spamming activity (Finding 3.3.5). Second, we observe that about 70% of all remote hosts spamming our sinkhole—and 95% of hosts for which we could attribute some operating system—appear to be running Windows; additionally, these hosts each send relatively low volumes of spam to the sinkhole, regardless of their persistence.

3.4.1 Bobax Topology

We studied the prevalence of spamming hosts versus the prevalence of known Bobax drones to better understand how the distribution of IP addresses of Bobax-infected hosts compared to the IP distribution of spammers in general. Figure 3.13 shows the results of this analysis; the distribution of *all* Bobax-infected hosts is quite similar to that of the distribution of all spammers (Figure 3.4).

Finding 3.4.1 (Bobax vs. spammer distribution) *Spamming hosts and Bobax drones have similar distributions across IP address space, which indirectly suggests that much of the spam received at the sinkhole may be due to botnets such as Bobax.*

This similarity provides evidence of correlation, not causality, but the fact that the distribution of IP addresses from which spam is received more closely resembles botnet activity than the spread of IP addresses of legitimate email suggests that a significant amount of spam activity may be due to botnet activity.

Although the range 60.* – 67.* has a significant fraction of spamming IP addresses (Figure 3.4), we see relatively less spam from Bobax drones from this space, which led us to suspect that spammers may be using techniques other than botnets for sending spam from many of the hosts in this range. Indeed, in Section 3.5, we present findings that suggest that one or more sophisticated groups of spammers appear to be sending spam from a large number of machines (or, perhaps, a smaller number of machines with changing IP addresses), numbered from portions of unused IP space within this range that are unroutable except for when they are sending spam.

3.4.2 Operating Systems of Spamming Hosts

In this section, we investigate the prevalence of each operating system among the spam we received, as well as the total amount of spam we received from hosts of each type. For this purpose, we used the passive OS fingerprinting tool, `p0f`, which is incorporated into Mail Avenger; thus, we can attribute an operating system to each remote host that sends us spam. Using this technique, we were able to identify the operating system for about 75% of all hosts from which we received spam. Table 3.6 shows the results of this study. Roughly 70% of the hosts from which we receive spam, and 95% of these hosts to which we could attribute an operating system, run Windows; this fraction is consistent with the fact that roughly 95% of all hosts on the Internet run Windows [87].

More striking is that, while only about 4% of the hosts from which we receive spam are from hosts are running operating systems other than Windows, this small set of hosts appears to be responsible for at least 8% of the spam we receive. The fraction, while not overwhelmingly large, is notable because of the conventional wisdom that most spam today originates from compromised Windows machines that are serving as botnet drones. As recent research has shown, spammers continue to seek out open relays and proxies to relay large volumes of spam [90]; we expect that these non-Windows machines are likely such relays or "high-volume" spammers hosted on spammer-friendly hosting providers.

Finding 3.4.2 (Prevalence of spam relays by OS type) *About 4% of the hosts sending spam to the sinkhole are not Windows hosts but our sinkhole receives about 8% of all spam from these hosts.*

A significant fraction of the spamming infrastructure is apparently still Unix-based, likely indicating open relay servers or high-volume spam servers hosted in rogue hosting providerrs.[3]

3.4.3 Spamming Bot Activity Profile

The results in Section 3.4.2 indicate that an overwhelming fraction of spam is sent from Windows hosts. Because a very large fraction of spam comes from Windows hosts, our hypothesis is that many of these machines are infected hosts that are bots. In this section, we investigate the characteristics of spamming hosts that are known to be Bobax drones. Specifically, we seek to answer the following three questions:

1. **Intersection:** *How many of the known Bobax drones send spam to our sinkhole?*

[3]Alternatively, this spam might be sent from Windows machines whose stacks have been modified to emulate those of other operating systems. Although we doubt that this is likely, since most spam filters today do not employ `p0f` checks, we acknowledge that it may become more common in the future, especially as spammers incorporate these techniques.

Table 3.6: The operating system of each unique sender of received spam, as determined by passive OS fingerprinting.

Operating System	*Clients*	*Total Spam*
Windows	854404 (70%)	5863112 (58%)
- Windows 2000 or XP	604252 (49%)	4060290 (40.2%)
- Windows 98	13727 (1.1%)	54856 (0.54%)
- Windows 95	559 (<0.1%)	2797 (<0.1%)
- Windows (other/unconfirmed)	235866 (19%)	1745169 (17.2%)
Linux	28132 (2.3%)	557377 (5.5%)
FreeBSD	6584 (0.5%)	152456 (1.5%)
MacOS	2944 (0.2%)	46151 (0.4%)
Solaris	1275 (< 0.1%)	18084 (0.2%)
OpenBSD	797 (< 0.1%)	21496 (0.2%)
Cisco IOS	736 (< 0.1%)	5949 (<0.1%)
NetBSD	44 (< 0.1%)	327 (<0.1%)
HP-UX	31 (< 0.1%)	120 (<0.1%)
Tru64	26 (< 0.1%)	143 (<0.1%)
AIX	23 (< 0.1%)	366 (<0.1%)
OpenVMS	18 (< 0.1%)	62 (<0.1%)
IRIX	7 (< 0.1%)	62 (<0.1%)
Other/Unidentified	128580 (10.4%)	1212722 (12%)
No Fingerprint	204802 (16.7%)	2225410 (22%)
Total	1228403	10103837

2. **Persistence:** *For how long does any particular Bobax drone send spam?*[4]
3. **Volume:** *How much of the spam from Bobax drones originates from hosts that are only active for a short period of time?*

The rest of this section explores these three questions. Although our trace sees spam from only a small fraction of all Bobax-infected drones, this sample nevertheless can offer insight into the behavior of spamming bots.

Intersection and prevalence. To satisfy our curiosity (and to compare with other claims about the amount of spam coming from botnets [19]), we wanted to determine the total fraction of received spam that originated from botnets versus other mechanisms. The circumstantial evidence in Sections 3.4.1 and 3.4.2 suggests that the fraction of spam that originates from botnets is quite high. Unfortunately, there are no techniques for isolating botnets from mail logs alone; we can only determine whether a particular piece of spam originated from a botnet based on whether the IP address of the relay sending the spam appears in our trace of machines known to be infected with Bobax.

Even this information is not sufficient to answer questions about the amount of spam coming from botnets, since machines other than Bobax-infected hosts may be enlisted in spamming botnets. Indeed, good answers to this question depend on both additional vantage points (*i.e.*, sinkhole domains) and better botnet detection heuristics and algorithms. Not only will more vantage points and better detection algorithms aid analysis, but they may also prove useful for massively collaborative spam filtering—identification of botnet membership, for example, could prove a very effective feature for identifying spammers.

[4]Previous work has noted that the "DHCP effect" can create errors in estimation for both persistence and prevalence (*e.g.*, a single host could dynamically be assigned different IP addresses over time) [85]. Although the DHCP effect can introduce problems for estimating the total population of a group of spammers, it is not as problematic for the questions we study in this chapter. Since one of our objectives is to study the effectiveness of IP-based filtering (rather than, say, count the total number of hosts), we are interested more in measuring the persistence of *IP addresses*, not hosts.

At our spam sinkhole, we receive spam from only 4,693 of the 117,268 Bobax-infected hosts in our command-and-control trace. This small (though certainly non-negligible) view into the Bobax botnet emphasizes the need for observing spamming behavior at multiple domains to observe more significant spamming patterns of a botnet. Nevertheless, this set of hosts that appear both in our spam logs and in the Bobax trace can provide useful insight into the spamming behavior and network-level properties of *individual* bots; it also appears to be a reasonable cross-section of all spamming bots (Figure 3.13 indicates that the IP distribution of bots from which our sinkhole receives spam is quite similar to the distribution of all spamming hosts across IP address space as shown in Figure 3.4).

Persistence. Figure 3.14 shows the persistence of each Bobax-infected IP address that sent spam to the sinkhole. The figure indicates that the majority of botnets make only a single appearance in our trace; these "single shot" bots account for roughly 25% of all spam that is known to be coming from Bobax drones.

Finding 3.4.3 (Single-shot bots) *More than 65% of IP addresses of hosts known to be infected with Bobax send spam only once, and nearly 75% of these addresses send spam to our sinkholed domain for less than two minutes, although many of them send several emails during their brief appearance.*

Of the spam received from Bobax-infected hosts, about 25% originated from hosts that only sent mail from IP addresses that only appeared once. The persistence of Bobax-infected hosts appears to be mildly bimodal: although roughly 75% of Bobax drones persist for less than two minutes, the remainder persist for a day or longer, about 50 persist for about six months, and 10 persist for entire length of the trace. Although these short-lived bots do not yet send the majority of spam coming from botnets, this "single shot" technique may become more prominent over time as network-level filtering techniques improve and spammers employ more sophisticated

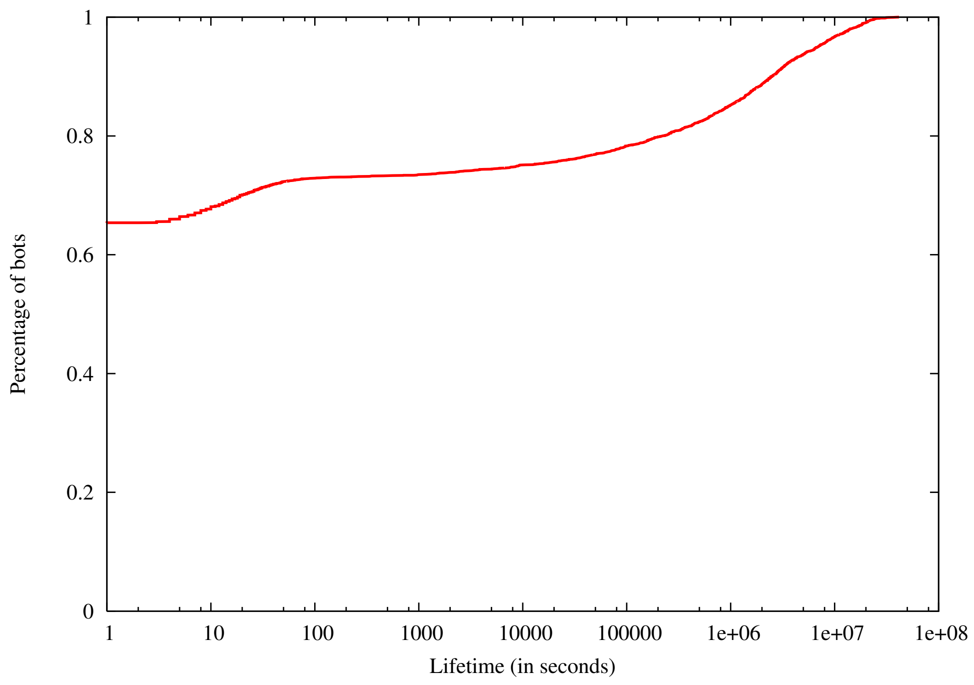


Figure 3.14: Bobax drone persistence.

evasion techniques.

Because most bot IP addresses are short-lived, we hypothesized that IP-based blacklists (*e.g.*, DNSBL filtering) would be somewhat ineffective for blocking spam. To our surprise, Figure 3.9 shows that the botnet hosts from which we received spam were actually *more* likely to be listed than the typical spamming mail relay (although, as we describe in Section 3.3.4, the technique appears to be somewhat ineffective in general). Intuitively, this result is justifiable, because other domains likely received spam from drones with the same IP addresses. This result also demonstrates the benefits of collaborative spam filtering, which facilitates the identification of spammers that send only a single piece of spam but send spam to multiple domains.

Volume and Rate. Figure 3.15 shows the amount of spam sent for each Bobax drone, plotted against the persistence of each drone. This graph shows that most Bobax drones do not send a large amount of spam, *regardless of how long the drone was active*. Indeed, nearly all of the Bobax drones observed in our trace send fewer

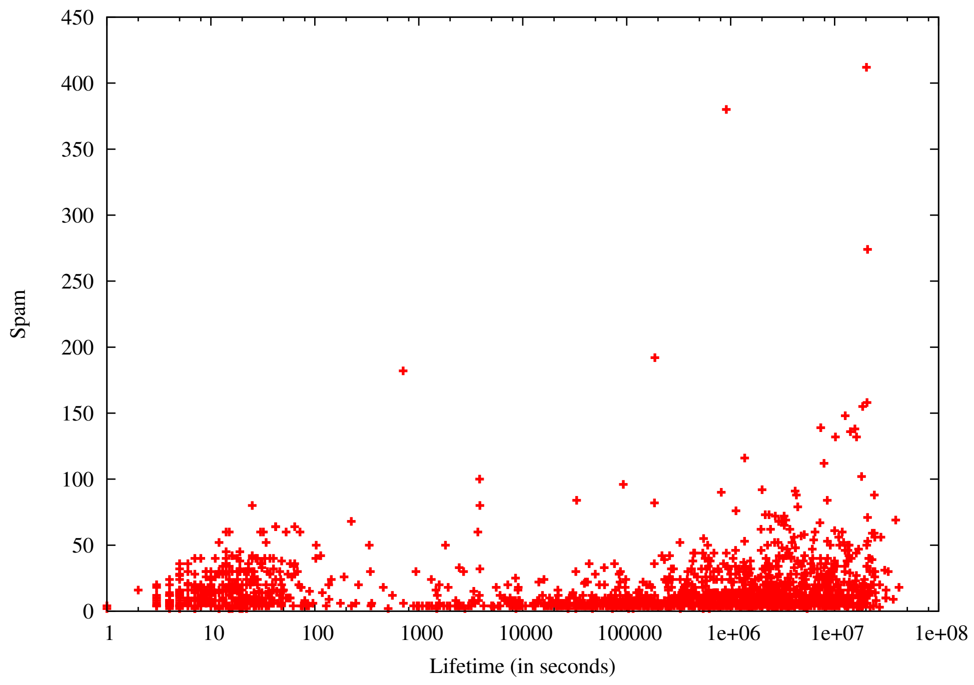


Figure 3.15: Number of spam email messages received vs. bobax drone persistence.

than 100 pieces of spam over the entire period of the trace. This finding suggests that spammers have the ability to send spam from a large number of hosts, each of which is typically used for a short period of time and nearly always used to send only a relatively small amount of spam. Thus, not only are IP-based filtering schemes likely to be ineffective, but *volume-based* detection schemes for spamming botnets may also be ineffective.

Finding 3.4.4 (Spam arrives from bots at very low rates) *Regardless of persistence, 99% of bots sent fewer than 100 pieces of spam to our domain over the entire trace.*

Most persistent bots sent fewer than 100 pieces of spam to our sinkhole, indicating that typical rates of spam from Bobax drones, *for spam received by a single domain*, are less than a single piece of spam per bot per day.

3.5 Spam from Short-lived BGP Announcements

Many spam filtering techniques leverage the ability to positively identify a spammer by its IP address. For example, DNS blacklists catalog the IP addresses of likely spammers so that spam filters may later send queries to determine whether an email was sent by a likely spammer. Of course, this technique implicitly assumes a connection between an IP address and the physical infrastructure that a spammer uses to distribute email. In this section, we study the extent to which spammers use such transient identities by examining spam received by the sinkhole domain that coincides with short-lived BGP route announcements.

Informal anecdotes have claimed that some spammers briefly advertise portions of IP address space, send spam from mail relays with IP addresses in that space, and subsequently withdraw the routes for that space after the relays have sent spam [9, 124, 133]. This practice makes it difficult for end users and system administrators to track spam sources because the network from which a piece of spam was sent is likely to be unreachable at the time a user lodges a complaint. Although it is technically possible to log BGP routing announcements and mine them to perform post-mortem analysis, the relative difficulty of doing so (especially since most network operators do not monitor interdomain routes in real time) essentially makes these spammers untraceable.

Little is known about (1) whether the technique is used much in practice (and how widespread it is), (2) what IP space spammers tend to use to mount these types of attacks and (3) the announcement patterns of these attacks. This study seeks to answer two sets of questions about the use of short-lived BGP routing announcements for sending spam:

- *Prevalence across ASes and persistence across time.* How many ASes use short-lived BGP routing announcements to send spam? Which ASes are the most guilty, in terms of number of pieces of spam sent, and in terms of persistence across time?

- *Length of short-lived BGP announcements.* How long do short-lived BGP announcements last (*i.e.*, long enough for an operator to catch)?

As we will see, sending spam from IP address space corresponding to short-lived route announcements is *not*, by any means, the dominant technique that spam is sent today (when this technique is actively being used, it accounts for no more than 10% of all spam we receive, and it generally accounts for much less). Nevertheless, because our domain only observes spamming behavior from a single vantage point, this technique may be more common than we are observing. Additionally, because this technique is not well defended against today, and because it is complementary to other spamming techniques (*e.g.*, it could conceivably be used to cloak botnets), we believe that this behavior is worth attention, particularly since some of the techniques we observe (*i.e.*, hijacking large prefixes) represents a significant departure from conventional wisdom on prefix hijacking.

3.5.1 BGP Spectrum Agility

Figure 3.16 shows an example of `61.0.0.0/8` being announced by AS 4678 for a brief period of time on September 30, 2005, during which spam was also sent from IP addresses contained within this prefix.

To investigate further the extent to which this technique is used in practice, we performed a joint analysis of BGP routing data (described in Section 3.2.4) and the spam received at our sinkhole, which is co-located with the BGP monitor. Given the sophistication required to send spam under the protection of short-lived routing announcements (especially compared with the relative simplicity of purchasing access to a botnet), we doubted that it was particularly prevalent. To our surprise, a small number of parties appear to be using this technique to send spam quite regularly. In

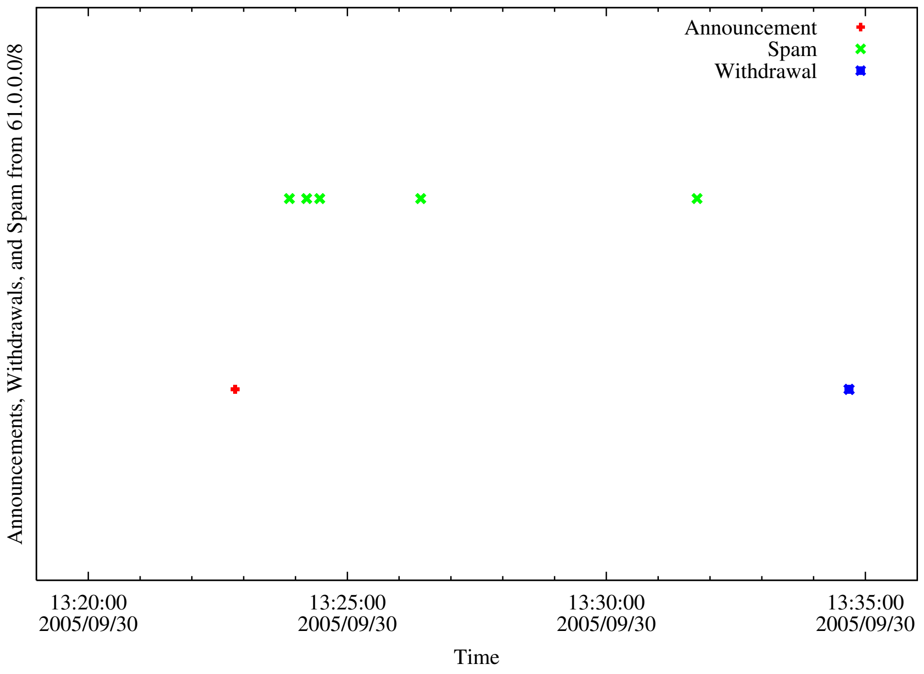


Figure 3.16: Observation of a short-lived BGP route announcement for `61.0.0.0/8`, spam arriving from mail relays in that prefix, and the subsequent withdrawal of that prefix.

fact, looking in further detail at the several (prefix, AS) combinations, we observed the following remarkable patterns:

- AS 21562, an Internet service provider (ISP) in Indianapolis, Indiana (according to `ra.net` and `arin.net`), originated routing announcements for `66.0.0.0/8`.
- AS 8717, an ISP in Sofia, Bulgaria, originated announcements for `82.0.0.0/8`.
- In a third, less persistent case, AS 4678, an ISP in Japan, Canon Network Communications (according to `apnic.net`), originated routing announcements for `61.0.0.0/8`.

We were surprised that three of the most persistent prefixes involved in short-lived

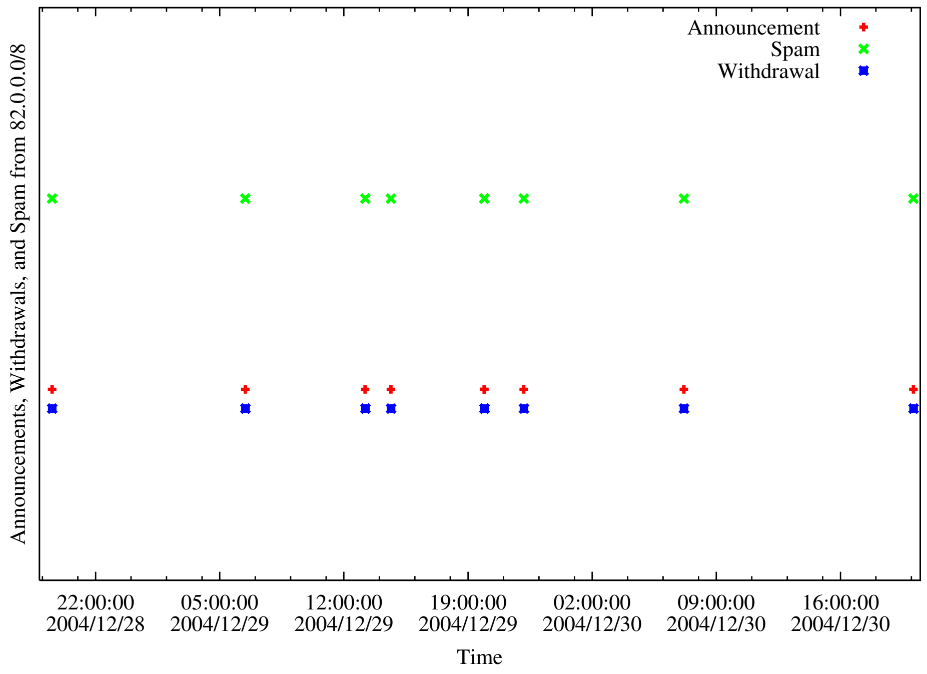


Figure 3.17: Observation of a short-lived BGP route announcement for `82.0.0.0/8`, spam arriving from mail relays in that prefix, and the subsequent withdrawal of that prefix.

BGP routing announcements were so large. Although some short-lived routing announcements may be misconfigurations [79], the fact that these routing announcements continually appear, that they are for large address blocks, and that they typically coincide with spam arrivals (as shown in Figure 3.16) raised our suspicion about the veracity of these announcements. Indeed, not only are these route announcements short-lived and hijacked, but they are also for large address blocks. Although the use of large address blocks might initially seem surprising, the distribution of the IP addresses of hosts sending spam using this technique suggests the following theory.

Finding 3.5.1 (Spectrum Agility) *A small, but persistent, group of spammers appear to send spam by (1) advertising (in fact, hijacking) large blocks of IP address space (i.e., /8s), (2) sending spam from IP addresses that are scattered throughout that space, and (3) withdrawing the route for the IP address space shortly after the spam is sent.*

We have called this technique "spectrum agility" because it allows a spammer the flexibility to use a wide variety of IP addresses within a very large block from which to send spam. The large IP address block allows the mail relays to "hop" between a large number of IP addresses, thereby evading IP-based filtering techniques like DNSBLs. Judging from Figure 3.9 and our analysis in Section 3.3.4, the technique seems to be rather effective. As an added benefit, route announcements for shorter IP prefixes (*i.e.*, larger blocks of IP addresses) are less likely to be blocked by ISPs' route filters than route announcements or hijacks for longer prefixes.

Upon further inspection, we also discovered the following interesting features: (1) the IP addresses of the mail relays sending this spam are widely distributed across the IP address space; (2) the IP addresses from which we see spam in this address space typically appear only once; (3) on February 6, 2006, attempts to contact the mail relays that we observed using this technique revealed that that roughly 60-80% of these hosts were not reachable by `traceroute`; (4) many of the IP addresses of these mail relays were located in allocated, albeit unannounced and unused IP address space; and (5) many of the AS paths for these announcements contained reserved (*i.e.*, to-date unallocated AS numbers), suggesting a possible attempt to further hamper traceability by forging elements of the AS path. We are at a loss to explain certain aspects of this behavior, such as why some of the machines appear to have IP addresses from allocated space, when it would be simpler to "step around" the allocated prefix blocks, but, needless to say, the spammers using this technique appear to be very sophisticated.

Whether spammers are increasingly using this technique is inconclusive. Still, many of the ASes that send the most spam with this technique also appear to be relative newcomers. Variants of this type of technique may be used in the future to make it more difficult to track and blacklist spamming hosts, particularly since the technique allows a spammer to relatively undetectably commandeer a very large

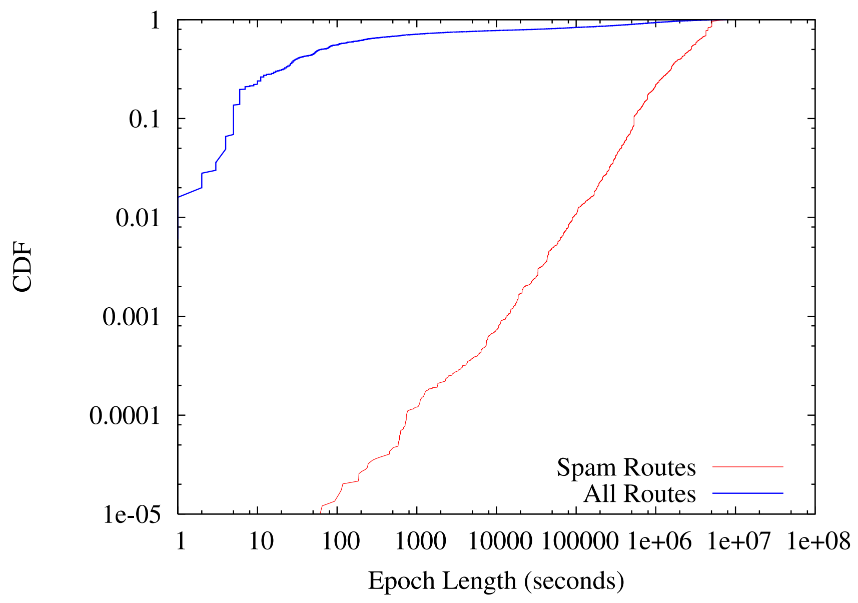


Figure 3.18: CDF of the length of each short-lived BGP episode, from September 2005–December 2005.

number of IP addresses.

3.5.2 Prevalence of BGP Spectrum Agility

Because of the volume of data and the relatively high cost of performing longest-prefix match queries, we performed a more extensive analysis on a subset of our trace, from September 2005 till December 2005, to detect the fraction of spam coming from short-lived announcements and to determine a reasonable threshold for studying short-lived announcements across the entire trace. Figure 3.18 shows that, for all of the IP addresses for which we received spam over the course of these four months, almost 99% of the corresponding BGP routing announcements were announced continuously for at least a day. In other words, most of the received spam corresponded to routing advertisements that were *not* short-lived. On the other hand, this technique appears to be used intermittently, and during time periods when this activity was more prevalent, as much as 10% of all received spam coincides with routing announcements that lasted less than a day.

Finding 3.5.2 (Prevalence: Spam from Short-Lived Routes) *Only about 1% of spam was received from route that persisted for less than a single day, although during intervals when this technique was used more commonly, as much as 10% of all spam coincided with routes that lasted less than a day.*

Unfortunately for traditional filtering techniques, the spammers who are the most persistent across time are, for the most part, *not* the spammers who send the most spam using this technique. Indeed, only two ASes—AS 4788 (Telekom Malaysia) and AS 4678 (Canon Network Communications, in Japan)—appear among both the top-10 most persistent and most voluminous spammers using short-lived BGP routing announcements.

3.5.3 How Much Spam from Spectrum Agility?

A comparatively small fraction of spam originates from IP addresses that correspond to short-lived BGP route announcements (*i.e.*, routing announcements that persist for less than a day) that coincide with spam arrival. The total amount of spam received as a result of this technique seems to pale in comparison to other techniques: no more than 10% of all spam—and more likely as little as 1%—appears to be sent using this technique. Although this technique is not apparent for most of the spam we receive (after all, a botnet makes traceability difficult enough), the few groups of spammers that employ this technique typically use it quite regularly. We also observed that many of the ASes using this technique for the longest period of time do *not*, in fact, rely on this technique for sending most of their spam. Even the most prolific spamming AS in this group, Malaysia Telekom, appears to send only about 15% of their spam in this fashion.

Finding 3.5.3 (Persistence vs. Volume) *The ASes from where spammers most continually use short-lived route announcements to send spam are not the same ASes from which the most spam originates via this technique.*

Many ASes that advertise short-lived BGP routing announcements and send large volumes of spam from these routes do not appear to be hijacking IP prefixes. In the case where spam volume is high, these short-lived routing announcements may simply coincide with spam being sent via another means (*e.g.*, from a botnet). The ASes that persistently advertise short prefixes, however, appear to be doing so intentionally.

3.6 Lessons for Better Spam Mitigation

Existing spam mitigation techniques have focused on either throttling senders (*e.g.*, recent attention has focused on cost-based schemes [42, 50]) or having receivers filter spam according to the *content* of a message. The results of this chapter, however, highlight several important lessons that strongly indicate that devoting more attention to the network-level properties of spammers that may be a useful addition to today's spam mitigation techniques. Using network-level information to help mitigate spam not only provides a veritable font of new features for spam filters, but network-level properties have two important properties that could potentially lead to more robust filtering.

1. Network-level properties are *less malleable* than those based on an email's contents.
2. Network-level properties may be *observable in the middle of the network*, or closer to the source of the spam, which may allow spam to be quarantined or disposed of before it ever reaches a destination mail server.

From our findings, we derive the following lessons regarding the network-level behavior of spammers that could help in designing better mitigation techniques.

Lesson 1 *Spam filtering requires a better notion of host identity.*

We observed a significant amount of spam from "fresh" bots, "single-shot" bots, and spammers using spectrum agility. Short-lived bots, short-lived BGP route hijacks, and dynamic addressing effects foil the common practice of using a host's IP address as its identity, *e.g.*, in IP blacklisting. This finding suggests that observations from a single vantage point cannot be used to identify who is a spammer and who is not.

Lesson 2 *Detection techniques based on aggregate behavior are more likely to expose nefarious behavior than techniques based on observations of a single IP address.*

Although comprehensive IP-based blacklisting is somewhat effective, blacklisting techniques may also benefit by exploiting other network-level properties such as IP address *ranges* or prefixes, many of which (*e.g.*, 80.*–90.*) send mostly spam. This indicates that IP prefix-based or ownership-based blacklisting—already followed to some extent by Spamhaus, through its "Registry of Known Spammers" (ROKSO) list—would be effective.

Lesson 3 *Securing the Internet routing infrastructure is a necessary step for bolstering identity and traceability of email senders.*

Although BGP spectrum agility is by no means responsible for most received spam, several characteristics make the technique extremely troubling. Most notably, the technique can be combined with other spamming techniques (possibly even spamming with botnets) to give spammers more agility in evading IP-based blacklists. Indeed, our analysis of DNSBLs indicates that spammers may already be doing this. A routing infrastructure that instead provided protection against route hijacking (specifically, unauthorized announcement of IP address blocks) would make BGP spectrum agility attacks more difficult to mount.

Lesson 4 *Many network-level features are robust even across a period of 6 years, indicating that these features can be incorporated relatively easily into spam filters and can be quite effective at detecting spam that is missed by other techniques.*

We found that features such as the network prefix and AS that originated an email, its size distribution, sending behavior, listing status in DNSBLs, etc. can potentially be used as features to classifiers.

Given the benefits that network-wide analysis could provide for stemming spam, we imagine that the ability to witness the network-level behavior of spammers *across* multiple distinct domains could also expose patterns that are not evident from a single domain. One organization might be able amass such a dataset either by sinkholing a large number of domains; for example, Project Honeypot [94] solicits donations of MX records for registered domains that do not receive email (though its corpus is still significantly smaller than ours). As we have discovered thus far from our initial experiences establishing new sinkholes, attracting spam to a new domain takes some effort (we found some amusement in the difficulty of attracting spam when we actually wanted to receive it). In addition to using sinkholes, network operators might share network-level statistics of received email from *real* network domains to pre-emptively detect and filter spamming hosts.

3.7 Summary

This chapter has studied the network-level behavior of spammers using a joint analysis of a unique combination of datasets—a 17-month-long trace of all spam sent to a single domain with real-time traceroutes, passive TCP fingerprints, and DNSBL lookup results; BGP routing announcements for the network where the sinkholes are located; command and control traces from the Bobax spamming botnet; and mail logs from a large commercial email provider.

This analysis allowed us to study some new and interesting questions that should guide the design of better spam filters in the future, based on the lessons in Section 3.6. We studied network-level behavior of spammers and compared these characteristics to those of legitimate email, noting some differences that could help identify spammers by IP address space or AS. We also used “ground truth” Bobax drones to better understand the characteristics of spamming botnets, and we found that most of these drones do not appear to revisit the same domain twice. While this property does not appear to hamper the use of blacklists for identifying bots (emphasizing the benefits of collaborative spam filtering), we found that blacklists were remarkably ineffective at detecting spamming relays that sent spam from IP addresses scattered throughout a briefly announced (and typically hijacked) IP address block—a new technique we call “BGP spectrum agility”. This technique is lethal because it makes traceability and blacklisting significantly more difficult. Spam filters that incorporate *network-level* behavior could not only mitigate this class of attack and many others, but they could also prove to be more resistant to evasion than content-based filters.

We complete this analysis by looking at a longitudinal trace of 6 years of spam data from the same sinkhole to investigate how well certain network-level features are useful in terms of robustness and longevity. We find that many unique network-level features, such as the IP blocks from which spammers operate, the mean size of spam messages, the ASes and countries most responsible for spam, ther sending behavior (*e.g.*, whether they have been seen before, and the number of emails sent in a “single shot”, etc. all provide good indicators of the nature

CHAPTER 4

IDENTIFYING BOTNETS THAT PERFORM IP BLACKLIST RECONNAISSANCE

4.1 Introduction

Botnets are the engines behind much malicious activity on the Internet, ranging from spam to denial of service to click fraud [33], since they allow attackers to distribute tasks over thousands of hosts distributed across the Internet. A botnet is network of compromised hosts ("bots") connected to the Internet under the control of a single entity ("botmaster", "controller", or *command and control*) [22]. The large cumulative bandwidth and relatively untraceable nature of spam from bots makes botnets an attractive choice for large-scale spamming. Previous work provides further background on botnets [22, 25].

Identifying members of botnets could help stem these attacks, but *passively* detecting botnet membership (*i.e.*, without disrupting the operation of the botnet) would be more desirable. The last chapter also demonstrated that botmasters use their bots cleverly: they ensure that bots "cycle" between their targets, and techniques such as hitting a target (*e.g.*, a recipient mail server) from previously unseen IP addresses, or by sending multiple spam messages at one time, are effective.

Seeing that a significant fraction of bots that send email to our spam trap were not listed in any DNS-based blacklist, we wondered whether botmasters used these unlisted bots intentionally to send spam—*i.e.*, whether the botmaster actually queried the same DNS blacklists we used to figure out which bots in a botnet were listed and which ones were not. If this activity did indeed occur, we hypothesize that we might be able to identify members of the botnet not using attack data, but by passively monitoring queries to the blacklist.

We performed *counter-intelligence* based on the insight that botmasters themselves perform DNSBL lookups to determine whether their spamming bots are blacklisted. Using heuristics to identify which DNSBL lookups are perpetrated by a botmaster performing such reconnaissance, we were able to compile a list of likely bots. This chapter studies the prevalence of DNSBL reconnaissance observed at a mirror of a well-known blacklist for a 45-day period, identifies the means by which botmasters are performing reconnaissance, and suggests the possibility of using counter-intelligence to discover likely bots. We find that bots are performing reconnaissance on behalf of other bots. Based on this finding, we suggest counter-intelligence techniques that may be useful for early bot detection.

If network operators and system administrators could reliably determine whether a host is a member of a botnet, they could take appropriate steps towards mitigating the attacks they perpetrate. Although previous work has described an *active* detection technique using DNS hijacking technique and social engineering [25], there are few efficient methods to *passively* detect and identify bots (*i.e.*, without disrupting the operation of the botnet). Indeed, detecting botnets proves to be very challenging: a victim of a botnet attack can typically only observe the attack from a single network, from which point the attack traffic may closely resemble the traffic of legitimate users. Regrettably, the state-of-the-art in botnet identification is based on user complaints, localized honeypots and intrusion detection systems, or through the complex correlation of data collected through darknets [71].

Using *passive* analysis of lookup traffic to a DNS-based blackhole list (DNSBL), we find evidence of a new, stealthy phenomenon: we find that spammers themselves are looking up the listing status of bots in DNSBLs. Many Internet Service Providers (ISPs) and enterprise networks use DNSBLs to track IP addresses that originate spam, so that future emails sent from these IP addresses can be rejected. For the same reason, botmasters are known to sell "clean" bots (*i.e.*, not listed in any DNSBL) at a premium. This chapter addresses the possibility of performing opportunistic *counter-intelligence* to help us discover identities of bots, based on the insight that *botmasters themselves must perform "reconnaissance" lookups to determine their bots' blacklist status.* We propose a set of techniques to identify botnets using passive analysis of DNS-based blackhole list (DNSBL) lookup traffic.

The contributions of this chapter include:

1. **Study of DNSBL reconnaissance techniques.** We study the prevalence of DNSBL reconnaissance by analyzing logs from a mirror of a well-known blackhole list for a 45-day period from November 17, 2005 to December 31, 2005. Section 4.4 discusses the prevalence of the different types of reconnaissance techniques that we observed. Much to our surprise, we find that bots are performing reconnaissance on behalf of other (possibly newly infected) bots. Although some bots perform a large number of reconnaissance queries, it appears that much of the reconnaissance activity is spread across many bots each of which issue few queries, thus making detection more difficult.

2. **Passive heuristics for counter-intelligence.** We develop heuristics to distinguish DNSBL reconnaissance queries for a botnet from legitimate DNSBL traffic (either offline or in real-time), to identify likely bots. These heuristics are based on an enumeration of possible lookup techniques that botmasters are likely to use to perform reconnaissance, which we detail in Section 4.2. Unlike previous detection schemes, our techniques are *covert* and do not disrupt the botnet's activity.

3. **Identification of new bots.** We analyze DNSBL queries that are likely being performed by botmasters to identify "clean" bots. Such reconnaissance usually precedes the use of bots in an attack, suggesting the possibility that

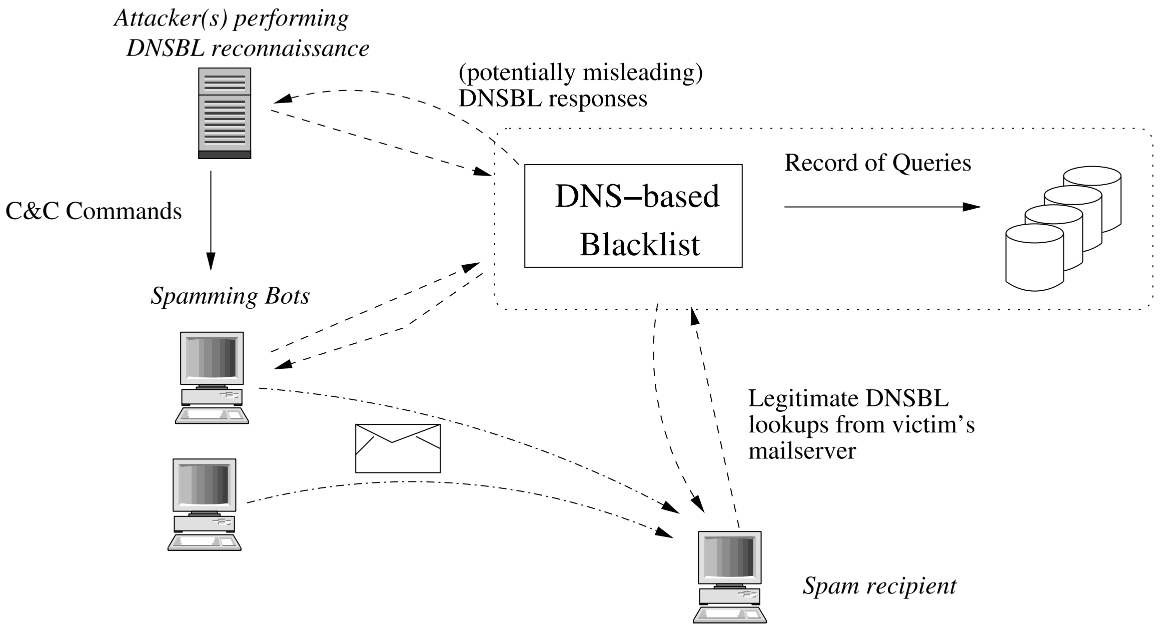

Attacker(s) performing DNSBL reconnaissance
(potentially misleading) DNSBL responses
C&C Commands
DNS−based Blacklist
Record of Queries
Spamming Bots
Legitimate DNSBL lookups from victim's mailserver
Spam recipient

Figure 4.1: DNSBL-based Spam Mitigation Architecture.

this DNSBL counter-intelligence can be used to bolster responses. Section 4.3 demonstrates the possibility of such early warning. To validate our detection scheme, we correlate the IP addresses of these likely bots with data collected at a botnet sinkhole (sinkholing technique explained in previous work [25]) over the same time period (this dataset has been used as "ground truth" for botnet membership in previous studies [25, 106]).

4. **4. DNSBL-based countermeasures.** Our heuristics could be used to detect reconnaissance in real-time. This ability potentially allows for active countermeasures, such as returning misleading responses to reconnaissance lookups, as shown in Figure 4.1. We revisit this topic in Section 4.5.

4.2 Model of Reconnaiassance Techniques

This section describes our model for DNSBL reconnaissance techniques (*i.e.*, the techniques that botmasters may be using to determine whether bots have been blacklisted). Our goal in developing these models and heuristics is to distinguish DNSBL queries issued by botmasters from those performed by legitimate mail servers.[1]

4.2.1 Properties of Reconnaissance Queries

Our detection heuristics are based on the construction of a *DNSBL query graph*, where an edge in the graph from node A to node B indicates that node A has issued a query to a DNSBL to determine whether node B is listed. After constructing this graph, we develop detection heuristics based on the expected *spatial* and *temporal* characteristics of legitimate lookups versus reconnaissance-based lookups. These characteristics hold primarily in cases when members of the botnet are not performing queries on behalf of each other, a case that makes detecting reconnaissance more difficult, as we explain in Section 4.2.2. As we describe below, our detection heuristics exploit both spatial and temporal properties of the DNSBL query graph.

Property 1 (Spatial relationships) *A legitimate mail server will perform queries* and *be the object of queries. In contrast, hosts performing reconnaissance-based lookups will only perform queries; they will* not *be queried by other hosts.*[2]

In other words, legitimate mail servers are likely to be queried by other mail servers that are receiving mail from that server. On the other hand, a host that is not itself being looked up by any other mail servers is, in all likelihood, not a mail server. We can use this observation to identify hosts that are likely performing reconnaissance: lookups from hosts that have a high *out-degree* in the DNSBL query graph (*i.e.*, hosts that are performing many lookups) but have a low *in-degree* are likely unrelated to

[1]DNSBL queries issued by mail servers are often performed by directly querying the DNSBL, rather than relying on a local resolver. For example, SpamAssassin [120] implements its own recursive DNS resolver. Hosts performing reconnaissance are also unlikely to query DNSBLs using local resolvers. Thus, in both cases, the querying IP address observed at the DNSBL correctly reflects the end-host performing the query.

[2]*This heuristic assumes that networks generally use the same host for both inbound and outbound mail servers. Although this configuration is common, some large networks separate the hosts responsible for inbound and outbound mail servers. In this case, queries from the inbound mail server might be misinterpreted as a reconnaissance attempt.*

the delivery of legitimate mail. To quantify this effect, we define the *lookup ratio*, λ, of some node n as follows:

$$\lambda_n = \frac{d_{n,out}}{d_{n,in}}$$

where d_{out} is the number of distinct IP addresses that node n queries, and d_{in} is the number of distinct IP addresses that issue a query for node n.[3] This metric is most effective when hosts performing reconnaissance are disjoint from hosts that are actually used to spam, which appears to the case today.However, as reconnaissance techniques become increasingly more sophisticated (as we describe in Section 4.2.2), this metric may become less useful. Still, we find that this metric proves to be quite useful in detecting many instances of DNSBL-based reconnaissance.

The *temporal arrival pattern* of queries at the DNSBL by hosts performing reconnaissance may differ from temporal characteristics of queries performed by legitimate hosts. We expect this to be the case because, whereas legitimate DNSBL lookups are driven by the arrival of actual email, reconnaissance queries will not reflect any realistic arrival patterns of actual email.

Property 2 (Temporal relationships) *A legitimate mail server's DNSBL lookups reflect actual arrival patterns of real email messages: legitimate lookups are typically driven automatically when emails arrive at the mail server and will thus arrive at a rate that mirrors the arrival rates of emails. Reconnaissance-based lookups, on the other hand, will not mirror the arrival patterns of legitimate email.*

We may be able to exploit the fact that email traffic tends to be diurnal [41] to tease apart DNSBL lookups that are driven by actual mail arrival from those that are driven by reconnaissance. Discovering reconnaissance activity using this method is a topic for future work.

[3]When $d_{n,in}$ is zero (which is commonly the case), we can simply consider λ_n to be a very large number.

4.2.2 Reconnaissance Techniques

In this section, we describe three classes of DNSBL reconnaissance techniques that may be performed by botmasters: *single-host, or third-party, reconnaissance*; *self-reconnaissance*; and *reconnaissance using other bots.* For each case, we describe the basic mechanism, the heuristics that we can use to detect reconnaissance in each of these cases, and how each technique may complicate detection.

Third-party Reconnaissance. In *third-party reconnaissance*, the botmaster performs DNSBL lookups from a single host for a list of spamming bots; this host may be the command-and-control of the botnet, or it might be another dedicated machine. In any case, we hypothesize that the machine performing the lookups in these cases is not likely to be a mail server. Single-host reconnaissance, if performed by a machine other than a mail server, is easily detected, because the node performing reconnaissance will have a high value of λ_n.

Once detected, single-host reconnaissance may provide useful information to aid us in revealing botnet membership. First, once we have identified a single host performing such lookups, the operator of the DNSBL can monitor the lookups issued by that host over time to track the identity of hosts that are likely bots. If the identity of this querying host is relatively static (*i.e.*, if its IP address does not change over time, or if it changes slowly enough so that its movements can be tracked in real-time), the DNSBL operator could take active countermeasures, such as intentionally returning incorrect information about bots' status in the blacklist, a possibility we discuss in more detail in Section 4.5.

Self-Reconnaissance. Single-host reconnaissance is simple, but it is susceptible to detection. To remain more stealthy, and to distribute the workload of performing DNSBL reconnaissance, botmasters may begin to distribute these lookups *across the botnet itself.* A simple (albeit sub-optimal) way to distribute these queries is to have a bot perform reconnaissance on its own behalf ("self-reconnaissance"); in other words, each bot could issue a DNSBL query to itself (*i.e.*, to determine whether it was listed) before sending spam to the victim.

In this case, identifying a reconnaissance-based DNSBL query is fairly straightforward, because, except in cases of misconfiguration, a legitimate mail server is unlikely to issue a DNSBL lookup for itself. Even though this technique has the advantage of distributing the load of reconnaissance across the botnet, we did not observe this technique being used in practice, likely because a self-query is a dead giveaway.

Distributed Reconnaissance. A more stealthy way to distribute the operation across the botnet is to have each bot perform reconnaissance on behalf of other bots either in the same botnet or in other botnets. For instance, note that Property 1 is unlikely to hold: in this case, the nodes performing reconnaissance will also be queried by other mail servers to which they send spam. As a result, these nodes are likely to have a high $d_{n,in}$, unlike nodes performing single-host reconnaissance. Ultimately, detecting this type of reconnaissance activity may require mining temporal properties (*e.g.*, Property 2).

Although using the botnet itself for DNSBL reconnaissance is more discreet than performing this reconnaissance from a single host, a network operator who positively identifies a small number of bots (*e.g.*, starting with a small hit-list of known bots, probably by using a honeynet with known infected machines). As discussed in Section 4.4, if this *seed list* of bots performs queries for other hosts, it is likely that these machines are also bots.

We suspected that this mode of reconnaissance would be uncommon, possibly because of the complexity involved in implementing and operating such a system (*e.g.*, keeping track of nodes in the looked-up botnet, disseminating this information to the querying nodes etc.). Much to our surprise, we did witness this behavior; we present these results in Section 4.4.

4.3 Data and Analysis

This section describes our data collection and analysis. We first describe our DNSBL dataset and its limitations. Then, we describe how this dataset is used to construct the DNSBL query graph described in Section 4.2.

4.3.1 Data Collection and Processing

Our study primarily involves two datasets collected from the same time period (November 17, 2005 to December 31, 2005): (1) the DNSBL query logs to a mirror of a large DNSBL, and (2) the logs of bot connections to a sinkhole for a Bobax botnet [12]. Unlike most botnets, the Bobax bot is designed solely for spamming [2], increasing the likelihood that a query for known Bobax host is the consequence of the querying mail server having received spam from that host.

Ground Truth. Because DNS blacklists list an IP address only on confirmed evidence of it belonging to a spammer, we use positive responses to DNSBL lookups as evidence for an IP address belonging to a confirmed spammer. To verify whether the scheme we propose is indeed able to discover *additional bots not already listed in blacklists*, we compared the IP addresses in the DNSBL query graph against the IP addresses of spammers in a large spam corpus collected at a spam honeypot (the setup of this honeypot is described in our earlier work [106]). This spam trap has been functioning since August 2004 and has no legitimate email addresses whatsoever; thus, we can use the IPs of any sender that sends email to our spam trap as ground truth.

4.3.2 Analysis and Detection

In this section, we describe how the DNSBL query graph is constructed. Definitions for the terminology used in our algorithm follow: (1) B, the set of IP addresses that attempted to connect to the Bobax sinkhole during the observation period (November

17, 2005–December 31, 2005); (2) *querier*, the IP address of the host that performs a given DNSBL query; (3) *queried*, the IP address of the host that is looked up in a DNSBL query; and (4) G, the DNSBL query graph constructed as a result of the algorithm.

The graph construction algorithm takes as input a set of DNSBL query logs (we use `tcpdump` for packet captures) and the set B and outputs a directed graph G. The algorithm, summarized in Figure 4.2, consists of two main steps: *parsing* and *pruning*. As the algorithm suggests, we prune DNSBL queries to only include edges which have at least one end (either *querier* or *queried*) present in the set B. Pruning is performed for efficiency reasons: the full DNSBL query logs mostly contain queries from legitimate mail servers. Using B to prune the complete query graph allows us to concentrate on a subgraph which has a higher percentage of reconnaissance lookups than the unpruned graph. We recognize that our analysis will overlook reconnaissance activity where both the *querier* or *queried* nodes are not members of B. To address this shortcoming, we perform a *query graph extrapolation* after the algorithm is run. In this step, we make a second pass over the DNSBL query logs and add edges if at least one of the endpoints of the edge (*i.e.*, either *querier* or *queried*) is already present in the graph. Query graph extrapolation is repeated until no new edges are added to G.

We then compute λ_n for each node in the graph (Property 1), which allows us to identify nodes involved in reconnaissance techniques described in Section 4.2. Although the results in Section 4.4 suggest that some bots have large values of λ_n, techniques that use a large number bots to look each other up may be undetectable with this metric. We are developing techniques based on Property 2 to further improve our detection.

```
CONSTRUCTGRAPH()
create empty directed graph $G$

/* Parsing */
for each DNSBL query:
    Identify querier and queried

    /* Pruning */
    if $querier \in B$ or $queried \in B$ then
        add querier and queried to $G$ if they
            are not already members of $G$
        if there exists an edge $E(querier, queried) \in G$ then
            increment the weight of $E(querier, queried)$
        else
            add $E(querier, queried)$ to $G$ with weight 1
```

Figure 4.2: Algorithm to construct a DNSBL query graph

4.4 Preliminary Results

This section presents preliminary results using Property 1 to identify DNSBL reconnaissance activity on the observed DNSBL query graph. We emphasize that the reconnaissance being performed by bots is distinctly under the radar as far as total DNSBL traffic is concerned: the pruned traffic amounts to less than 1% of the total DNSBL traffic. In this section, we present two surprising results: First, botnets are being used to perform DNSBL reconnaissance on behalf of bots in other botnets, which has implications for botnet detection. Second, the distribution of these queries across bots suggests that some DNSBL reconnaissance activities may be detectable in real-time, which has implications for early detection and mitigation.

Attempts to validate our hypotheses from Section 4.2 resulted in some interesting discoveries, including the discovery of new bots. We initially expected that most DNSBL lookups would be third-party lookups, as described in Section 4.2.2, and that we would be able to validate the queried nodes as being known bots. Instead, we discovered the opposite: the nodes with the highest values of λ_n in the pruned graph were *known* bots, while *the queried nodes in the graph were new, previously*

Table 4.1: AS numbers of hosts which have the highest out-degrees. The last column shows the number of hosts queried by this node that are known spammers (verified using logs from our spam sinkhole).

Node #	ASN of Node	Out-degree	known spammers
1	Everyone's Internet (AS 13749)	36,875	12
2	IQuest (AS 7332)	32,159	7
3	UUNet (AS 701)	31,682	5
4	UPC Broadband (AS 6830)	26,502	8
5	E-xpedient (AS 17054)	19,530	4

unknown bots. Further, using data from our spam sinkhole [106], we found that some of these nodes were Windows machines and confirmed spam originators. This finding suggests that, in general, it may be possible to start with a set of known bots and use the DNSBL graph to "bootstrap" the discovery of new bots.

Table 4.1 shows five of the top queriers (*i.e.*, high out-degree nodes), *all* of which are known bots from our Bobax trace. Even more interesting is the fact that a few IP addresses queried by these nodes actually sent spam to our spam honeypot. Moreover, nearly all of IP addresses that sent spam to our honeypot were *not* present in our list of known bots. Due to the fact that our honeypot only captures a small portion of the Internet's spam, the fraction of total reconnaissance queries that we can confirm as spamming bots is small. Still, we believe it strongly suggests evidence of a known bot performing DNSBL reconnaissance on a distinct (and possibly newly compromised) botnet.

Figure 4.3 shows the distribution of out-degrees for all querying nodes present in the pruned DNSBL query graph. The long tail also confirms that bots already have the capability to distribute these queries, which is cause for concern. Our view of DNSBL queries is narrow (most querying nodes are geographically close to the DNSBL mirror), so we expect that more vantage points of DNSBL lookups would reveal other prominent "players". The fact that the prominent players in our analysis

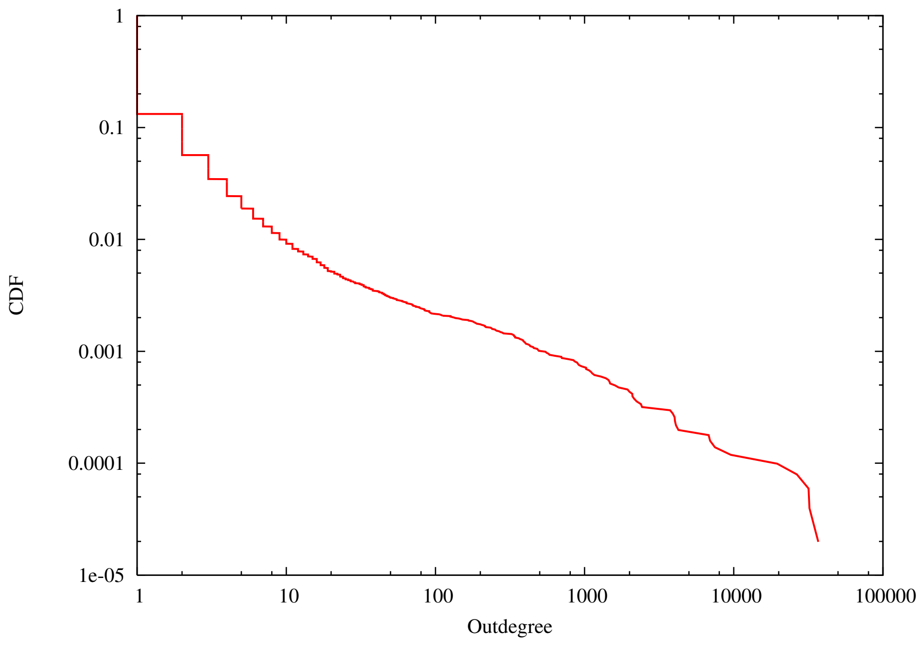


Figure 4.3: CDF of the distribution of out-degrees for querying IP addresses.

were also bots suggests that these nodes may also be obvious candidates for the mitigation techniques described in Section 4.5.

4.5 Countermeasures

In Section 4.4, we found that the known bots in our Bobax trace were not the targets of lookups, but instead were issuing lookups for other, possibly newly compromised bots. This finding suggests a possible technique that could be used for the discovery of new bots, even without an initial list of suspects: an initial set of suspect IP addresses could be constructed by establishing a spam trap, which according to both previous work [106] and the observations in this chapter, appear to be largely bots. Alternatively, a suspect node could be detected simply by identifying nodes in the DNSBL query graph with a high value of λ_n. Beginning with this initial suspect list, an operator may be able to conclude that, not only are the nodes that this node is querying likely bots, but also the node itself is likely a bot. If there are other high-degree nodes also querying the same bots, a detection algorithm might be able to "walk" the DNSBL graph (*e.g.*, from parent to parent) to discover multiple distinct botnets.

We believe that using such techniques to aggressively monitor botnet-based DNSBL reconnaissance may prove to be useful for mitigating spam: as noted in our previous work [106], most bots send a very low volume of spam to any single domain; thus, reporting a bot to blacklists *after* the spam is received may not be effective.

With the ability to distinguish reconnaissance queries from legitimate queries, a DNSBL operator might be able to mitigate spam more effectively. We speculate one possibility as follows: an operator could tune the behavior of the blackhole list server to mislead a botmaster, using a class of techniques we call *reconnaissance poisoning.* On one hand, the DNSBL could trick the botmaster into thinking that a particular bot was "clean" (*i.e.*, unlisted) when in fact it was listed, which would induce the botmaster to unwittingly send spam from blacklisted machines. On the other hand, the DNSBL could also reply to a reconnaissance query with an indication that a host was listed, even though it was not listed, thereby discouraging a botmaster from using a machine that would likely be capable of successfully sending spam.

Of course, active countermeasures such as reconnaissance poisoning do run the risk of false positives: if we mistakenly attribute a legitimate DNSBL query to a reconnaissance-based query, we could mislead a legitimate mail server into either mistakenly accepting spam that would have otherwise been rejected or, more regrettably, rejecting legitimate email. Such techniques could also be defeated if the botmaster queries multiple blacklist providers that maintain independent lists. Investigating the extent to which our detection metrics are subject to false positives, as well as the extent to which these false positives interfere with a legitimate mail server's filtering techniques, is part of our ongoing work.

4.6 Summary

This chapter has developed techniques and heuristics for detecting DNSBL reconnaissance activity, whereby botmasters perform lookups against the DNSBL to determine whether their spamming bots have been blacklisted. We first developed heuristics for counter-intelligence based on several possible ways we figured reconnaissance was being performed. We then studied the prevalence of each of these reconnaissance techniques. Much to our surprise, we found that bots were in fact performing reconnaissance on IP addresses for bots in other botnets. Based on this finding, we have outlined possibilities for new botnet detection techniques using a traversal of the DNSBL query graph, and we have suggested techniques that DNSBL operators might use to more effectively stem the spam originating from botnets. We are investigating the effectiveness of these detection and mitigation techniques as part of our ongoing work.

CHAPTER 5

FILTERING SPAM USING BEHAVIORAL BLACKLISTING

5.1 Introduction

Spam-blocking efforts have taken two main approaches: (1) content-based filtering and (2) IP-based blacklisting. Both of these techniques are losing their potency as spammers become more agile. To evade content-based filters, spammers have adopted techniques such as image spam and emails explicitly designed to mislead filters that "learn" certain keyword patterns. In Chapter 3, we demonstrated that a spammers are also successful at avoiding IP blacklists: a significant fraction of the IP addresses that send spam to our spam sinkhole are not listed in any IP blacklist.

This "low and slow" spam sending pattern and the ease with which spammers can quickly change the IP addresses from which they send spam has rendered today's methods of blacklisting spamming IP addresses less effective than they once were [21]. For example, our study in Section 5.2 shows that, of the spam received at our spam traps, as much as 35% was sent from IP addresses that were not listed by either Spamhaus [122] or SpamCop [121], two reputable blacklists. Further, 20% of these IP addresses remained unlisted even after one month. Most of the IP addresses that were eventually blacklisted evaded the blacklist for about two weeks, and some evaded the blacklists for almost two months.

Two characteristics make it difficult for conventional blacklists to keep pace with spammers' dynamism. First, *existing blacklists are based on non-persistent identifiers.* An IP address does not suffice as a persistent identifier for a host: many hosts obtain IP addresses from dynamic address pools, which can cause aliasing both of hosts (*i.e.*, a single host may assume different IP addresses over time) and of IP addresses (*i.e.*, a single IP address may represent different hosts over time). Spammers also intentionally use IP addresses that have not been seen by a domain to send spam, and, as demonstrated in Chapter 4, may perform blacklist lookups to find which bots are unlisted. Second, *information about email-sending behavior is compartmentalized by domain and not analyzed across domains.* With large botnets at their disposal, spammers now disperse their "jobs" so that each IP address sends spam at a low rate to any single domain. By doing so, spammers can remain below the radar, since no single domain may deem any single spamming IP address as suspicious.

As a result, IP blacklists must be continually updated to keep pace with campaigns mounted by armies of "fresh" IP addresses. Unfortunately, a spam campaign may complete by the time the IP addresses are blacklisted, at which time a new campaign with new IP addresses is imminent. Blacklisting all new IP addresses is not an option, either: it creates a nuisance for administrators when legitimate mail relays are renumbered. Thus, blacklist administrators choose to not list every IP address that sends spam, allowing spam to get through.

We hypothesized that many of the IP addresses we observed at our spamtrap are "fresh" due to observations from a single vantage point; looking at emails received from many vantage points *might* allow us to see the same senders hitting multiple vantage points. This chapter presents SpamTracker, a spam filtering system that uses a new technique called *behavioral blacklisting* to classify email senders based on their sending *behavior* to multiple vantage points, rather than using their identity(*i.e.*, IP address). Behavioral blacklisting complements existing blacklists by categorizing spammers based on *how* they send email, rather than the IP address (or address range) from which they are sending it. The intuition behind behavioral blacklisting is that, while IP addresses are ephemeral as identifiers, spam campaigns, spam lists, and spamming techniques are more persistent. If we can identify email-sending patterns that are characteristic of spamming behavior, then we can continue to classify IP addresses as spammers even as spammers change their IP addresses.

We design a behavioral blacklisting algorithm that uses the *set of target domains* that a particular IP address sends mail to as the primary indicator of its behavior and incorporate this algorithm into a system called SpamTracker. We use the set of domains that an IP address targets within a fixed time window as the feature for clustering IP addresses that behave similarly. Our clustering algorithm takes as input an $n \times d \times t$ tensor, where n is the number of IP addresses that sent email to any of d domains within one of t time windows. The algorithm outputs clusters of IP addresses that exhibit similar sending patterns. Our evaluation of these clusters shows that spamming IP addresses form large clusters that are highly similar to each other but distinct from the behavior of IP addresses in other clusters. IP addresses of legitimate senders, on the other hand, do not form large clusters. SpamTracker can classify a "fresh" IP address as a spammer or a legitimate sender based on how closely its sending behavior (*i.e.*, the set of domains that it targets) maps to a cluster that has been marked as known spamming behavior. Using logs from an organization that manages email for over 115 domains, we find that SpamTracker detects many spammers *before they are listed in any blacklist*, suggesting that SpamTracker can complement today's IP-based blacklists by catching some spammers earlier than they would otherwise be caught.

SpamTracker requires little auxiliary information about whether an email sender is a spammer or a legitimate sender: it takes as input the email-sending patterns of all senders, builds clusters based on the sending behaviors of (a possibly small set of) known spammers, and classifies each sender based on whether its behavior is similar to a cluster that resembles known spamming behavior. Unlike conventional approaches which track individual IP addresses, SpamTracker tracks behavioral patterns to quickly identify whether a new IP address exhibits similar patterns to other previously seen IP addresses. Its ability to track behavior of *groups*, rather than individual IP addresses, allows it to adapt more quickly to ephemeral IP addresses that may not exhibit strong patterns from the perspective of any single domain.

Because SpamTracker classifies email based on sending behavior rather than on more malleable properties of email (*e.g.*, content, or even IP address), we believe that spammers will have considerably more difficulty in evading SpamTracker's classification methods. Nevertheless, SpamTracker must be agile enough to adapt to spammers' changing behaviors: spamming patterns (*i.e.*, which domains are targeted, and how they are targeted) will change over time, and adversaries that are aware of the SpamTracker algorithm may adjust their sending patterns to avoid falling into a particular cluster. We believe, however, that automated, large-scale behavior such as spamming will always give rise to clustering, and the challenge is to design SpamTracker to adapt the clusters it uses for classification, even as the spammers themselves attempt to evade them.

The chapter is organized as follows. Section 5.2 motivates behavioral blacklisting. Section 5.3 presents a brief background on clustering techniques and describes EigenCluster [16], the clustering algorithm that we use in SpamTracker. Section 5.4 describes the design and implementation of SpamTracker, and Section 5.5 presents our validation results and compares the performance of SpamTracker to state-of-the-art IP-based blacklists and spam trap deployments. In Section 5.6, we discuss various extensions of SpamTracker and deployment-related concerns. Section 5.7 concludes.

5.2 Motivation

This section provides background on current email spamming practices and the performance of blacklists. In Section 5.2.1, we present the volumes and rates at which IP addresses in our traces send spam to each domain; we find that spammers exhibit sending patterns that make it difficult to reliably detect and track spamming IP addresses. In Section 5.2.2, we provide background on current IP-based blacklisting techniques (*e.g.*, DNS-based blacklists) and present a study of their effectiveness.

5.2.1 The Behavior of Spamming IP Addresses

We present statistics on the network-level behavior of spammers, focusing on the techniques that make building the reputation of any particular IP address difficult. We study two aspects in particular: (1) *Persistence:* How much spam does a particular IP address send in a day, and how does the set of IP addresses change over time? (2) *Distribution:* What is the distribution of spam *across target domains* for any particular IP address, and how does this distribution change over time?

Persistence: "New" IP addresses every day. To determine the extent to which spamming IP addresses remain stable, we study the IP addresses that send spam to over 115 distinct domains, which collectively received 33 million pieces of spam during March 2007.[1]

Figure 5.1 shows the number of "new" IP addresses that these domains observed per day over the course of a month. The top line shows the fraction of IP addresses that were seen in the trace for a particular day that were *never* seen before in the trace (other lines show fraction of spam from IP addresses that appeared on the immediately preceding day, or within the month). Indeed, spam is coming from different IP addresses every day, and about 10% of IP addresses seen on any particular day were never seen before at any of the target domains. Note that although 10% is a much smaller figure than the nearly 50% of spamming IP addresses that were never seen before at our spamtrap—likely due to observing from a single vantage point—it indicates that spammers also have a significant pool of *globally "fresh"* IP addresses. Thus, even given perfect mechanisms for maintaining reputation about email senders

[1]Section 5.5.1 describes this dataset (as well as the others that we used in our evaluation) in more detail.

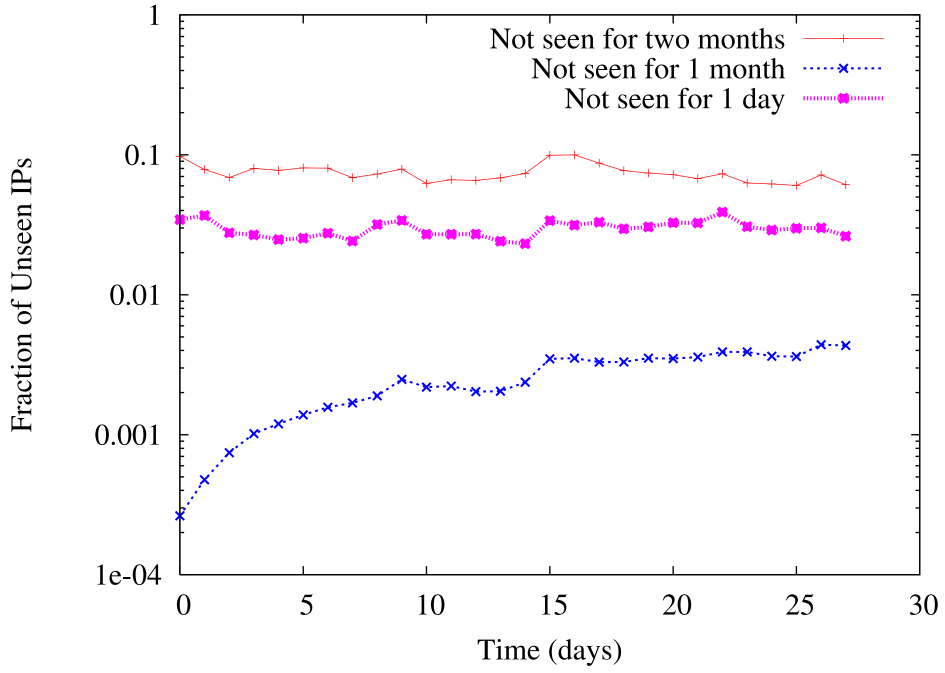


Figure 5.1: Fraction of spamming IP addresses that were not observed at any of 115 domains for the past 1 day, past month, and past 2 months.

and relatively widespread observation, a significant number of IP addresses that have never been seen before are sending spam on any given day.

Lack of persistence in spamming IP addresses makes maintaining reputation about spammers based solely on IP addresses difficult, since the blacklisted IP addresses keep changing. Given no previous information about the activity of an IP address, a conventional blacklist will not be able to reliably block spam from that address.

Distribution: Some IP addresses target many domains. Existing blacklisting techniques collect reputation information about spam or spam senders based on the activity observed at a single domain (*e.g.*, if a spammer sends a significant amount of spam to a single IP address, if it hits a spam trap, etc.) [121, 122]. Although some existing systems collect information from a large number of distributed domains, few, if any, build reputation based on observed patterns *across* domains. Thus, an IP address that distributes spam evenly across target domains may evade a blacklist

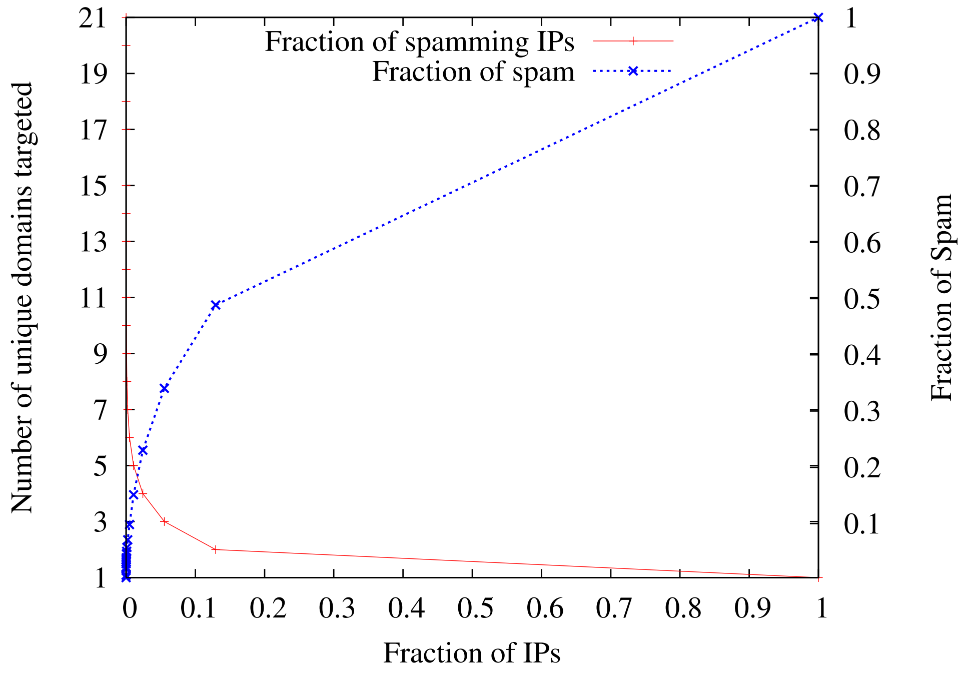


Figure 5.2: Fraction of spam sent (y' axis), and number of domains targeted (y axis), by spamming IP addresses for a typical day's worth of traffic at the email provider's servers. The IP addresses are reverse sorted by number of spam messages produced.

entirely: maintenance of these lists typically requires explicit reports from a network about a "loud" IP address, so an IP address that is "low and slow" to any particular domain may be able to escape detection and blacklisting.

Previous work has shown that many bots that send spam are comparatively low-volume if observed at any one domain [106], but each of these IP addresses must send low volumes of spam to *many* domains for them to be "useful" to the spammer. Our analysis confirms this conjecture: Figure 5.2 shows that about half of all spam (y' axis) comes from the top 15% spamming IP addresses (x-axis); this subset of IP addresses targets two or more domains (y-axis). Similarly, the top spamming IP addresses responsible for up to 35% of spam target three or more domains. Thus, *observing email sending patterns across domains could help expose sending patterns that are responsible for sending a significant amount of spam.*

5.2.2 The Performance of IP-Based Blacklists

After presenting a brief overview of IP-based blacklists and their most common operating mode (DNS-based blacklists, or "DNSBLs"), we briefly survey the performance of currently used DNS-based blacklists in terms of two metrics:

- *Completeness.* The fraction of spamming IP addresses (and fraction of spam from spammers) that is listed in a blacklist at the time the spam was received.
- *Responsiveness.* For the IP addresses eventually listed by a blacklist, the time for a blacklist to eventually list spamming IP addresses after they first send spam to *any* target domain.

Our results demonstrate that DNSBLs can be both incomplete and unresponsive in response to dynamism in IP addresses. We present additional data that suggests that the email sending characteristics of spammers—in particular, their transience and the low volume of spam that they send to any single domain—make it difficult for blacklists to track the IP addresses of some spammers.

Background: DNS-Based Blacklists (DNSBLs). DNSBLs offer a lightweight mechanism for querying a list of IP addresses, but *the list membership must be maintained at least semi-manually.* Maintenance entails not only deciding when a particular IP address should be added to a blacklist, but also when it should be removed. Blacklist maintainers typically add an IP address to a blacklist based on reports from network operators (which requires the spammer to raise the attention of an operator) or by sending spam to a particular spam trap or traps (which may not see the spam in the first place, particularly if spammers know to avoid them). Because reputation information about IP addresses can become "stale" (*e.g.*, due to IP address dynamism, renumbering, etc.), the blacklist maintainer must determine how long an IP address should remain listed; this duration ranges from 30 minutes to more than

Table 5.1: Fraction of spam at two spam traps from IP addresses that were unlisted in either Spamhaus or SpamCop, both at the time the message was received, and the fraction of spam from IP addresses that remained unlisted after 1 month.

			Spam from unlisted IP addresses	
Data Source	*Spam*	*IP addresses*	*At Receipt*	*After 1 Month*
Trap 1	384,521	129,243	134,120 (35%)	79,532 (20%)
Trap 2	172,143	64,386	17,132 (10%)	14,534 (8.5%)

a year, depending on the nature of the problem and resolution.

Completeness. We study the completeness of "reactive" blacklists (*i.e.*, those that only list IP addresses based on observed spamming activity or user reports as opposed to policy (*e.g.*, SORBS [119]) lists all dynamic IP addresses irrespective of whether they were observed spamming or not). We consider the two most popular reactive blacklists, Spamhaus [122] (specifically the XBL and SBL zones) and SpamCop [121]. To assess the completeness of existing DNSBLs, we first examine whether blacklists identify the spammers that we observe in one month of spam from two spam traps. We then observe mail received at a server that hosts email for hundreds of independent domains to determine how much of the mail that this provider accepted could have been blocked earlier if the provider had more complete blacklists at its disposal.

Experiment 1: Are emails to spam traps blacklisted? We first studied whether spammers were listed when they sent spam to two large spam traps during March 2007. The two traps serve independent domains and they have no real email addresses, so we can consider all mail that these domains receive to be spam.[2] Both run the MailAvenger [80] SMTP server, which we have instrumented to measure whether a sender's IP address is blacklisted at any of 8 blacklists *at the time the email was received.*

[2]One of the domains serves eight legitimate users. We exclude this legitimate mail from our analysis and do not expect the presence of these addresses to have an effect on the spam received at the domain.

Trap 1 received 384,521 pieces of spam, of which 134,120 (35%) were received from IP addresses that were not listed in either Spamhaus or SpamCop when the spam was received. Trap 2 received 172,143 pieces of spam, of which 10% came from IP addresses that were not blacklisted. The significant fraction of spam coming from unlisted IP addresses suggests that *complementary* blacklisting techniques could significantly reduce spam. Additionally, *blacklists may remain incomplete* even one month after each of these IP addresses sent spam: Unlisted IP addresses that accounted for 20% of spam at Trap 1, and 8.5% of spam at Trap 2, remained unlisted in Spamhaus blacklist one month after they were seen in our spam traps (see Table 5.1), suggesting that there is still a significant fraction of spam from senders that successfully evade conventional blacklisting techniques.

Experiment 2: Are accepted senders blacklisted later? The second set of logs are from an organization that hosts email service for over 700 domains, about 85 of which were active during March 2007 (our observation period). This provider's mail servers reject or accept email based on a combination of techniques, including multiple blacklist lookups (Spamhaus [122], SORBS [119], etc.) and a large collection of customized heuristics. This provider blocks up to twice as much spam as any single blacklist.

Using our daily snapshot of the Spamhaus blacklist as a basis for comparison, we study the effectiveness of this email provider's blocking heuristics by determining the fraction of mail that the provider accepts. Our results show that even this provider's advanced filtering does not ensnare a significant collection of spammers: Of the 5,084,771 senders that passed the spam filters, *only* 110,542 (2%) became listed in the Spamhaus blacklist during the following month. This fraction is significantly lower than the 15% quoted by this provider as the fraction of accepted email that is later reported as spam, which suggests that current blacklists remain incomplete, even after long periods of time.

Responsiveness. Many DNSBLs do not list an IP address before they receive multiple end-user reports about a spam sender; some even perform manual verification. Meticulous verification can reduce the likelihood of blacklisting "good" senders, but doing so also limits responsiveness. In this section, we quantify the responsiveness of the Spamhaus DNSBL by determining, for the IP addresses that were eventually listed in April 2007, how long those IP addresses were active before they eventually were blacklisted.

As before, we use snapshots of the Spamhaus blacklist, but we also use hourly "diffs" of the blacklist to determine when a new IP address was added. We examine email logs from March 1–31, 2007 and blacklist data from April 1–30, 2007. For each IP address that was not listed when it first sent spam to one of our spam traps but was eventually listed at some later point in April 2007, we compute the delay between the first occurrence of the IP at our trace to the first time that the IP address became listed in Spamhaus.[3]

Even when blacklists *do* list spamming IP addresses, the process of updating the blacklist may be slow. Figure 5.3 shows the time-to-listing for all IP addresses that were unlisted during the receipt of the email but eventually appeared at the blacklist in April 2007. In the case of the spam traps, 10–15% of spam senders that were unlisted at receipt of spam remained so 30 days after spam was received. The fraction is a strong indicator of the sluggishness of blacklists, because sending email to a spam trap automatically labels the sender as a spammer. In the case of the provider that serves millions of real customers ("Organization"), almost 20% of senders that were

[3]Because we only have the Spamhaus database for April, we cannot determine the exact listing time for IP addresses that were in the database on April 1, 2007; rather, we only know that they were listed between the time the spam was observed in March and April 1, 2007 ("less than 30 days" in Figure 5.3). If the IP address was not listed by April 1, 2007, we assume that whenever the IP becomes listed in April is the first time Spamhaus listed it. This assumption is reasonable as Spamhaus lists *persistent* spammers for a minimum of 30 days [4].

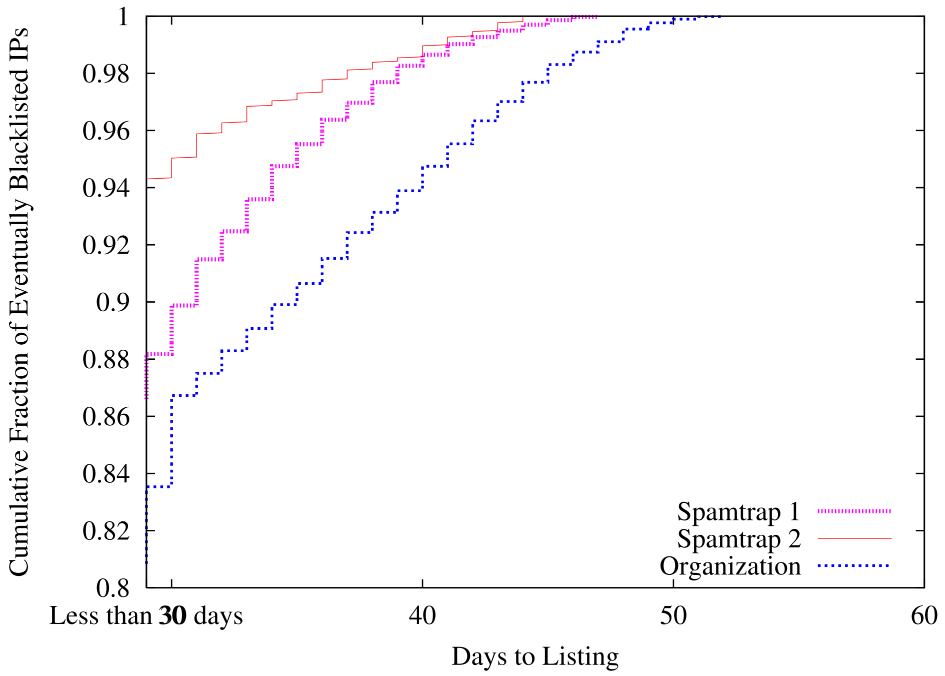


Figure 5.3: Time-to-listing for the Spamhaus blacklist for IP addresses that were unlisted at the time spam was received, but were eventually blacklisted. *Note: y*-axis starts at 0.8.

unlisted when email was received *remain unlisted for over 30 days* before eventually appearing in the blacklist.

This analysis indicates that reactive blacklists are sometimes slow to respond, even for confirmed spammers; this slow responsiveness, coupled with the ability to continually send spam from "fresh" IP addresses (Section 5.2.1) represents a significant "window of opportunity" for spammers to send spam from non-blacklisted IP addresses. Motivated by this slow responsiveness, the next section proposes a complementary approach to blacklisting that is based on email sending patterns, rather than the reputation of an IP address alone.

5.2.3 The Case for Behavioral Blacklisting

Although individual IP addresses' sending behavior may change across time, we posit that (1) the *sending patterns* exhibited by spammers are sufficiently different from those of legitimate senders; and (2) those patterns become more evident when email senders can be observed across many receiving domains. Based on these two hypotheses, the rest of the chapter proposes a system called SpamTracker, which proactively blacklists email senders based on the set of domains they target. SpamTracker relies on a technique that we call *behavioral blacklisting*, which attempts to classify based on their network behavior, rather than their identity or the contents of the emails they send. While individual IP addresses may be ephemeral, they may exhibit "familiar" spamming patterns (*i.e.*, similar to those of already well-known spamming IP addresses) that become evident when sending patterns are observed across multiple domains.

5.3 Clustering Algorithm

SpamTracker uses a spectral clustering algorithm proposed and analyzed by Kannan *et al.* [66] and made efficient in practice by Cheng *et al.* [16]. Section 5.3.1 presents an overview of the spectral clustering approach, and Section 5.3.2 describes how we apply spectral clustering within SpamTracker.

5.3.1 Spectral Clustering

Spectral clustering refers to partitioning algorithms that rely on the principal components of the input. There are generally two basic variants which can be viewed as (a) one-shot or (b) recursive. Given an object-feature matrix A with the goal of clustering the objects (rows) of A, a one-shot algorithm would find the top few singular vectors of A (say k) and either project to their span or create a cluster for each one by assigning each row to that vector in which it has the largest component. A recursive algorithm, on the other hand, uses one singular vector to partition the rows and recurses on the two parts. We focus on this type of algorithm.

The method in Cheng *et al.* [16] ("EigenCluster") has two phases: a top-down divide phase and a bottom-up merge phase. In the divide phase, the algorithm normalizes a given nonnegative input matrix so that all rows have the same sum, then computes the second largest right singular vector. It sorts the rows according to their components in this vector and partitions this sequence at the point where the corresponding cut has minimum *conductance* (among the $n-1$ possible cuts of the sequence). The conductance of a partition is the total weight of entries across the partition divided by the smaller of the total weights incident to each side [66, 118]. After finding the partition, it recurses on each side until only singletons remain. This completes the divide phase, whose end result is a tree (the root represents all the rows, the leaves are individual rows). The merge phase finds a tree-respecting partition, *i.e.*, one where every cluster corresponds to the entire subtree attached at some node of the tree. For many objective functions, it does this by dynamic programming, in a bottom-up fashion. The specific function we use for the merge phase is called *correlation clustering* [16].

5.3.2 SpamTracker: Clustering Email Senders

SpamTracker classifies an email sender purely based on its sending behavior, ignoring content and variable handles for classification such as dynamically-allocated IP addresses. The intuition behind SpamTracker is that sending patterns of spamming hosts are similar to other senders and remain relatively stable, even as the IP addresses (or actual systems) that are sending the emails change. Consider the case of a spamming bot: Whatever the particular spamming behavior of a spamming bot, it is likely to be similar to other bots in its own botnet. Because botmasters in large botnets have only coarse-grained control over their bots [89], spamming patterns of bots will typically be similar across targeted domains *even if each bot sends low volumes of spam to each domain.* Thus, clustering spammers based on their sending patterns provides a way for their early detection, irrespective of their particular identities (*e.g.*, the IP address) or blacklisting status. It follows from the above that, spam sent from

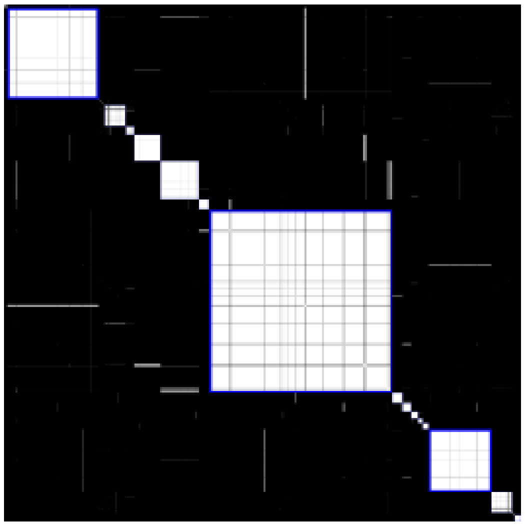


Figure 5.4: An IP× IP matrix of related spam senders; IP addresses that send mail to similar sets of domains are grouped into distinct clusters; the intensity of a pixel at (i,j) indicates i's similarity to j.

even a newly-enlisted bot (*i.e.*, from an IP address that has not been observed to send spam) will likely be caught by SpamTracker because its behavior will cluster it with other known bots in the botnet.

The SpamTracker algorithm proceeds in two stages: (1) clustering and (2) classification. In the unsupervised clustering stage, SpamTracker accepts as input a $n \times d \times t$ tensor M, where n is the number of IP addresses that sent email to any of d domains within any of t particular time windows. Thus, $M(i,j,k)$ denotes the number of times IP address i sent email to domain j in time slot k. SpamTracker first collapses the time axis to obtain an $n \times d$ matrix M':

$$M'(i,j) = \sum_{k=1}^{t} M(i,j,k).$$

It clusters the matrix M' using the spectral clustering algorithm described in Section 5.3.1. The output of the clustering stage is the set of clusters of IP addresses $C = C_1, C_2, \ldots, C_k$, where $\cup_{i=1}^{k} C_i =$ IP addresses in M and $C_i \cap C_j = \phi$ for $i \neq j$. Logically, the set C consists of groups of IP addresses in M that have similar behavior in their target domains. Each cluster is associated with a traffic pattern, obtained by averaging the rows corresponding to IP addresses that fall in the cluster. For a cluster c, we call this vector c_{avg}.

$$c_{avg} = \frac{\sum_{i=1}^{|c|} M'_c(i)}{|c|}$$

where $M'_c(i)$ is the submatrix comprising the rows of cluster c. In the classification stage, SpamTracker accepts a $1 \times d$ vector r that corresponds to the recent behavior of an IP. It then calculates a score $S(r)$ for this queried IP address using the following equation.

$$sim(r, c) = \frac{r \cdot c_{avg}}{|c_{avg}|} \tag{1}$$

Intuitively, $sim(r, c)$ measures the similarity of the row vector r to cluster c by performing an inner product of r with the normalized average of rows in cluster c. A cluster that has a similar set of target domains as r would have a large inner product.

We calculate the spam score $S(r)$ as the maximum similarity of r with any of the clusters.

$$S(r) = \max_c sim(r, c). \tag{2}$$

S can be used to filter or *greylist* (*i.e.*, temporarily reject with the assumption that a legitimate mail sender will eventually retry) spam by a mail service provider at or before the SMTP dialogue stage. We set a threshold such that if the row for an IP that is looked up has score higher than the threshold, it is flagged as spam. The threshold can be different for each cluster.

Querying an IP address is inexpensive: only Equations 1 and 2 need to be computed per lookup. The next section explains the design of SpamTracker in detail and the optimizations we use to improve the lookup speed and the overall robustness of the system.

5.4 Design

This section describes how SpamTracker can be integrated into an existing email infrastructure. We present a brief overview of the system and then describe in detail its two basic operations: (1) computing the clusters that form the basis of the classifier; and (2) classifying a new IP address when it arrives.

5.4.1 Overview

The spectral clustering algorithm in Section 5.3.2 serves as the back-end of SpamTracker. The behavioral classifier that accepts lookups from mail servers and assigns scores to the queried senders forms the front-end. Figure 5.5 shows the high-level design of SpamTracker and the interaction between the back-end (which performs clustering and classification operations) and the interface to mail servers (which receives email sending patterns as input to the clustering algorithm and answers queries about the status of any particular IP address); to an ordinary mail server, the interface to SpamTracker looks like any other DNS-based blacklist, which has the advantage that existing mail servers need only to be reconfigured to incorporate SpamTracker into spam filtering decisions. We discuss how SpamTracker can be incorporated into existing infrastructure in Section 5.6.2.

SpamTracker's clustering algorithms rely on the assumption that the set of domains that each spammer targets is often more stable than the IP addresses of machines that the spammer uses to send the mail. Rather than maintaining reputations of senders according to their IP addresses, SpamTracker uses the vector representing how a sender sends traffic across domains, r, as a "behavioral fingerprint" and determines whether this fingerprint resembles a known spamming cluster. Section 5.4.2 describes how SpamTracker builds clusters of known spammers, and Section 5.4.3

explains how SpamTracker determines whether an email sender's sending patterns resemble one of these clusters.

5.4.2 Clustering

SpamTracker uses the spectral clustering algorithm from Section 5.3.1 to construct the initial set of clusters. SpamTracker's clustering takes as input email sending patterns about confirmed spammers (*i.e.*, the volume of email that each confirmed spamming IP address sends across some set of domains) over some time window to construct the matrix $M(i, j, k)$. This input requires two components: (1) an initial "seed list" of bad IP addresses; and (2) email sending patterns for those IP addresses. This section describes in turn how SpamTracker might be able to acquire this type of data.

Data about spamming IP addresses is easy to obtain, and SpamTracker could use any such initial list of IP addresses to "bootstrap" its initial clusters. For example, an Internet Service Provider (ISP) that uses conventional SpamAssassin [120] filters to filter spam could use that list of IP addresses as its initial spammer IP addresses to be used for the basis for clustering.

The sending patterns of each of the spamming IP addresses is more difficult to obtain because it requires visibility into the emails that many domains have received. Our evaluation of SpamTracker (Section 5.5) uses an email hosting provider's decisions about early mail rejects from hundreds of domains to compute these clusters, but, in practice, other systems like SpamTracker could also likely gain access to such data.

To build the rows in M for each spamming IP address, participating domains could submit IP addresses that they have confirmed to be spammers as they do with blacklists, but based on our findings of the "low and slow" sending patterns of spammers (Section 5.2), SpamTracker will be most effective if it maintains sending patterns across domains for as many IP addresses as possible and subsequently clusters based on some subset of those that are labelled as spam by at least one domain. Fortunately, SpamTracker could obtain these sending patterns from receiving mail servers' queries to the classifier[4], at least from some subset of trusted domains.[5] Specifically, a lookup for IP address a from domain d is a reasonable indicator that a has sent email to d, so SpamTracker can build vectors for *all* such addresses a and later build the matrix M from just those addresses that are confirmed to be spammers.

5.4.3 Classification

SpamTracker maintains a vector representing the sending pattern, r, for each IP address a compiled from reports from the mail servers of participating domains. SpamTracker collects these sending patterns as mail servers from trusted participating domains perform lookups to SpamTracker on address a, using the same method for collecting these patterns for all IP addresses during the clustering phase (described in Section 5.4.2).

Given an r for some IP address a, SpamTracker returns a score $S(r)$ (computed using Equation 2, Section 5.3.2) whose magnitude determines how closely this fingerprint resembles a confirmed spamming pattern (*i.e.*, cluster). SpamTracker can simply return $S(r)$ to the querying mail server, which can then incorporate this score into its existing mail filtering rules. An important benefit of the classification process is that $S(r)$ can be computed using only an IP address's r vector and the c_{avg} rows for the spam clusters, both of which can be replicated and distributed (providing robustness against attack, as well as load balance). Clustering requires 'r' vectors from as many IP addresses as possible; even though it requires aggregating sending information from many sending domains (and, hence, from potentially many SpamTracker

[4]Note that the query mechanism needs a way of finding the email domain name of the organization performing the query. DNS reverse lookups, or extra information in the query packets, could provide such a mechanism.

[5]Because previous work has observed that bots occasionally perform reconnaissance queries against blacklists [101], we cannot assume that *all* queries to the blacklist reflect the receipt of email by a mail server.

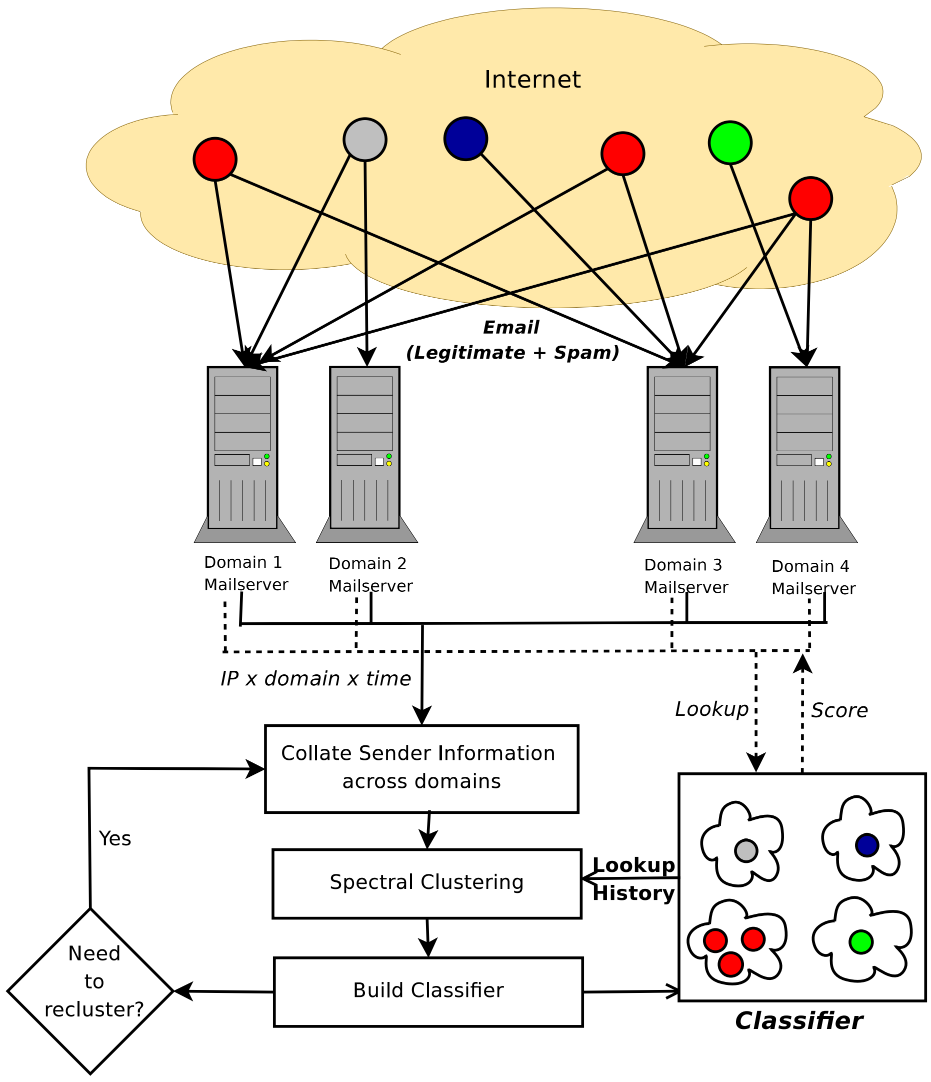


Figure 5.5: The high-level design of SpamTracker. The *clustering* component of SpamTracker accepts information about email senders as an $IP \times domain \times time$ tensor and computes clusters of related senders (and corresponding average vectors). The *classification* component accepts queries for IP addresses and returns a score, $S(r)$, for the IP's behavior.

replicas), this aggregation and clustering can be performed on a slower timescale than classification.

5.4.4 Tracking Changes in Sending Patterns

SpamTracker must recompute new clusters as sending patterns change. Our implementation of SpamTracker reclusters at fixed intervals, but in practice SpamTracker might only recluster when sending patterns no longer map to any existing clusters. Reclustering cost (time, memory, CPU) increases with larger input matrices, so clustering on very large time windows may be impractical. We use an efficient re-clustering method that preserves historical information but keeps clustering cost approximately constant. At the beginning of each clustering phase, we add all average rows from the previous clustering stage scaled by the size of the cluster each row represents, which produces the effect of clustering on the input of both stages without the added cost.

5.5 Evaluation

This section describes the evaluation of SpamTracker. In a real deployment, SpamTracker could compute clusters based on sending patterns across many domains for some time interval. To emulate this scenario, we construct the SpamTracker classifier by constructing $M(i, j, k)$ from the email logs of a large organization that manages mail servers for hundreds of domains. We use the matrix for time window at $[t, t+\Delta t)$ to build the classifier, and the data in the window $[t + \Delta t, t + 2\ \Delta t)$ to validate our classification. Section 5.5.1 summarizes the data sets used in our evaluation. Section 5.5.2 describes the properties of the resulting clusters and the validation results, and Section 5.5.3 describes our evaluation of SpamTracker's ability to improve upon existing blacklisting and blocking techniques by classifying spammers ahead of blacklists.

5.5.1 Data

Table 5.2 summarizes the traces, their duration, and the data fields each trace provides. Our primary data is a set of email logs from a provider ("Organization") that hosts and manages mail servers for over 115 domains. The trace also contains an indication of whether it rejected the SMTP connection or not. We also use the full database of Spamhaus [122] for one month, including all additions that happened within the month ("Blacklist"), to help us evaluate the performance of SpamTracker

Table 5.2: Data sets used in evaluation.

Trace	Date Range	Fields
Organization	Mar. 1 – 31, 2007	Received time, remote IP, targeted domain, whether rejected
Blacklist	Apr. 1 – 30, 2007	IP address (or range), time of listing

relative to existing blacklists. We choose the Blacklist traces for the time period immediately after the email traces end so that we can discover the first time an IP address, unlisted at the time email from it observed in the Organization trace, was added to Blacklist trace.

Ground Truth. As in the previous chapter, we rely on blacklist listing status as an indicator of whether an IP address indeed belongs to a confirmed spammer. Blacklists are known for their low false positive rates, thus, if an IP address is listed, it is highly likely that it belongs to a spammer.

We possess daily logs of listed IP addresses in a large blacklist *following* our email logs, which allows us to observe *new* IPs being listed. IP addresses either stay listed for a period of many months in blacklists, or are de-listed for a similar period. Thus, if we see an IP address that is newly listed in the month immediately following our email logs, it is likely that that IP address was not listed as a spammer in the preceding month. We use this fact as evidence that our algorithm can detect many spammmers in advance of their being listed in blacklists.

5.5.2 Clustering and Classification

To study the properties of the clusters that SpamTracker computes, we build the SpamTracker classifier using data for a window Δt at time t, and use it to assign a spam score $S(r)$ senders in the window $[t + \Delta t, t + 2\ \Delta t)$. We set Δt to be 6 hours; clustering using different time intervals (which we intend to explore in future work) may also help SpamTracker perform better.

Figure 6(a) shows the distribution of these scores for all IP addresses in a 6-hour window, separated into two plots based on whether the Organization decided to reject the mail early or accept it for delivery. A high score implies that the sending pattern for the classified IP is similar to a known spamming pattern. The low-score region (where $S(r) < 1$) comprises IP addresses whose patterns are unknown to the classifier. Senders that map into this range should not necessarily be considered legitimate; rather they simply do not have a recognized, blacklisted sending pattern. High scores reflect IP addresses whose sending patterns are very similar to the average rows of the classifier. As expected, the distribution of mails rejected by the organization tend towards larger values of $S(r)$. We suspect that because legitimate email senders likely will not mimic each other's sending patterns, the IP addresses in this region—both in the "accepted" and "rejected" plots—are likely to contain spammers. Indeed, in Section 5.5.3, we show that SpamTracker correctly classified IP addresses in that were accepted by the Organization but were eventually blacklisted.

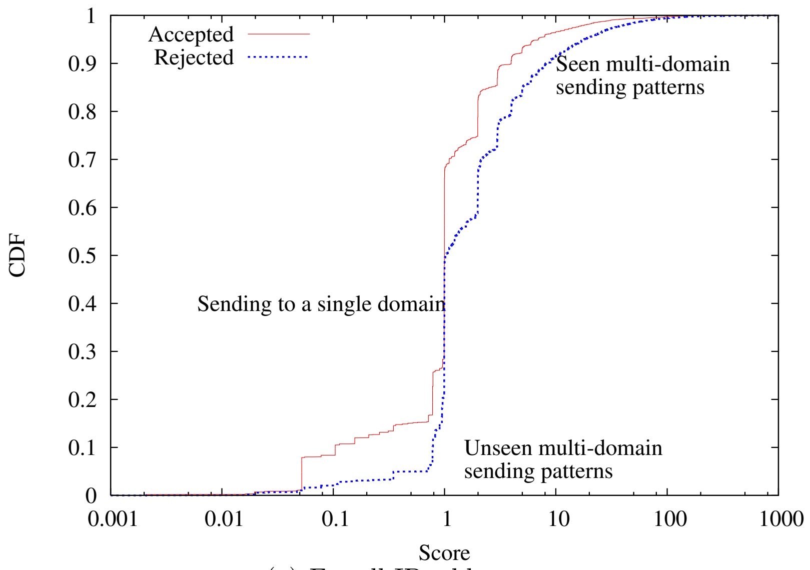
Ideally, users of SpamTracker should be able to set a single threshold for $S(r)$ that clearly separates the majority of legitimate email from the majority of spam, but setting a single threshold for the experiment shown in Figure 6(a) could result in misclassifying a large fraction of received mail. For example, though setting a threshold of 10 would blacklist only about 5% of the Organization's accepted mail, it would only correctly classify 10% of all of the rejected mail. In fact, a lower threshold may be more appropriate: as we describe in Section 5.5.3 below, a significant fraction of accepted mail is still spam, and, in many cases, SpamTracker captures this spam before the Organization or Spamhaus does. However, *without ground truth data, it is difficult to determine a precise false positive rate, because "accepted" mail may simply be misclassified spam.*

We believe that the quality of data (rather than the classification algorithm itself) is affecting our ability to separate the accepted and rejected mail with a single spam score. First, the data set is not cleanly labelled: the decisions of the Organization concerning whether to accept or reject a mail are not in fact a ground truth indicator as to whether mail is legitimate: The Organization estimates that as much as 15% of accepted mail is spam, and, as we show in Section 5.5.3, the emails that were accepted by the Organization for which SpamTracker assigns high scores may in fact be undetected spammers. Second, SpamTracker performs best when the representative sending behavior for a cluster is distributed across multiple domains, rather than concentrated in a single domain. Figure 6(a) shows that many emails have a spam score of 1, which implies that the classified IP address's pattern is *similar to a cluster whose average row is dominant in one column.* According to Equations 1 and 2, this pattern will return a similarity of about $|r|$. Because, in our dataset, a majority of senders in most small time windows send email to only a single domain, $|r|$ is 1 for 50% of accepted email and 30% of rejected email. Our dataset often has email senders that send mail to only a single domain in a time window.

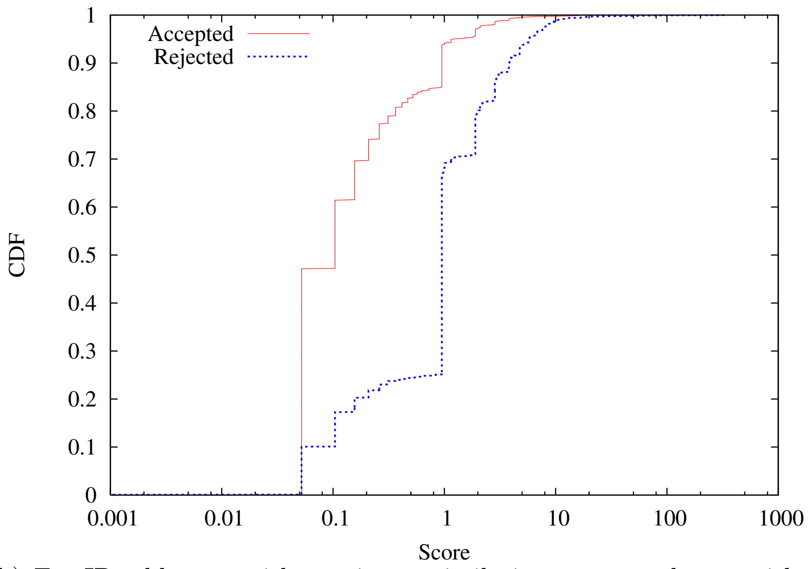
Figure 5.6.5 shows the distribution of $S(r)$ for IP addresses that have maximum similarity with a single cluster whose c_{avg} is not dominated by a single column. The "accepted" and "rejected" distributions separate more cleanly because legitimate IP addresses that have maximum similarity with this cluster will likely not have sent mail to all domains comprising the average row of this cluster (although the spammers in this cluster will likely hit all or most domains). A better distribution of monitors might result in a more even observation of sending patterns, which should result in a distribution of $S(r)$ that more closely resembles that shown in Figure 5.6.5.

5.5.3 Detecting "New" Spammers

To estimate the fraction of spammers that SpamTracker's clustering techniques can detect in advance of conventional blacklisting techniques, we study a subset of the email with the highest spam scores and determine whether any emails from this subset were eventually reported as spam (*e.g.*, by users, or some other auxiliary



(a) For all IP addresses.



(b) For IP addresses with maximum similarity to some cluster with a distributed sending pattern.

Figure 5.6: Score distribution for SpamTracker's classification for (a) All IP addresses in a "round" of classification, and (b) IP addresses that have maximum similarity with a cluster whos c_{avg} is not dominated by a single column. Evaluation for a 6-hour period using a classifier trained using the previous 6-hour window.

technique). Operators at the Organization acknowledge that about 15% of email that is initially accepted falls into this category. To estimate how well SpamTracker would perform at catching this 15% of misclassified mail, we examine the 620 emails were initially missed by the Organization's filters but eventually blacklisted in the following month. Of these, 65 emails (about 10%) had a spam score of $S(r) > 5$ (from Figure 6(a)), suggesting that SpamTracker could complement existing filtering mechanisms by capturing additional spam that existing filters miss.

5.6 Discussion

Although the general method of *behavioral blacklisting* shows promise for fast classification of spammers, there is much room for improvement, particularly with respect to the classification algorithms (which could, for example, incorporate other features as input). This section proposes specific areas where the classification algorithms could be improved, surveys how filtering techniques based on behavioral blacklisting could ultimately be deployed in an operational network, and presents our ongoing efforts to do so. We also discuss how behavioral blacklisting scores might be integrated into existing spam filtering systems and some of the issues that may arise in implementation and deployment.

5.6.1 Improving Classification

IP addresses that are most similar to a single spamming cluster can be classified more accurately. In order to achieve this separation for *all* new IP addresses, we propose two improvements to SpamTracker that may result in better clusters.

Using more features for clustering. Although SpamTracker uses target domains to construct the initial object-feature matrix (Section 5.3.2), other behavioral features may be able to better classify spammers. Temporal patterns such as the time interval between successive emails received from an IP (or alternatively, the sending frequency of the IP) is one such feature. Botmasters often manage all their bots using unified interfaces that may also be used to disseminate spam templates and mailing lists to bots [89], so these bots may exhibit similar temporal behavior (perhaps spamming frequencies) in addition to their similarity in target domains.

Improved similarity computation. In Equation 1, all columns of IP's "fingerprint" vector, r, are weighted equally. Some domains may be better at distinguishing one cluster of spammers from another. For example, spammers targeting victims in different countries may send email to country specific domains as well as to ubiquitous domains (*e.g.*, `gmail.com`). In this case, the country-specific domains may be more helpful in distinguishing the two sets of spammers. Our ongoing work includes experimenting with an algorithm that weights each column (domain) differently.

5.6.2 Incorporating with Existing Systems

We discuss how SpamTracker can be incorporated to complement the existing deployments of mail servers and spam filters. We describe two possibilities below: integration with existing filters and on the wire deployment. In either case, the back-end of SpamTracker can remain the same: it only needs to run a DNS server (or another popular query interface such as XML-RPC) that accepts requests for IP addresses, retrieves the classification score $S(r)$ from the SpamTracker classification engine, and returns the score to the client. In this sense, SpamTracker is a stand-alone system that can even be used internally within an organization.

Option 1: Integration with existing infrastructure. SpamTracker could be incorporated into existing filtering systems on mail servers by providing an additional "confidence score" for these filters that help them determine whether a particular piece of email is spam in terms of sender behavior. Because SpamTracker provides a simple interface (*i.e.*, it takes as input an IP address and returns a score), it can be incorporated into any existing spam filtering engine (*e.g.*, SpamAssassin [120], MailAvenger [80]) in the same way that any other blacklist information would be added as a filtering criterion. Using this system would be easy: the addition of one line to the configuration of most mail filtering software should allow users to benefit from SpamTracker's filtering strategy.

The disadvantage, however, is that it does not stop email traffic close to the source: the mail server that receives the spam drops the mail only after the traffic has already traversed the network and consumed resources on the receiving mail server.

Option 2: "On the wire" deployment. Unlike most existing spam filtering or classification systems, SpamTracker has the unique advantage that it can classify email senders solely based on the source IP address and destination domain of the mail being sent (*i.e.*, it does not require examining or analyzing an email's contents). Thus, another possibility for deploying SpamTracker involves deploying a network element that can examine traffic "on the wire" and identify connections to mail servers from IP addresses that fall into clusters with high spam scores. Such a system could be deployed *anywhere* in the network, not just at the receiving mail server.

The disadvantage to this strategy is that deployment involves several additional steps: in particular, such a filtering element would need a channel to receive up-to-date information about both the email sending clusters (*i.e.*, their average vectors, and their "spamminess") and the vector for any particular sending IP address (*i.e.*, to which domains it has sent). Maintaining up-to-date information about clusters and sending IP addresses in such a distributed, dynamic setting may prove challenging in practice.

5.6.3 Deployment Challenges

SpamTracker must be able to handle a large volume of email and senders, be resistant to attack, and must remain highly available. To achieve these goals, we believe that SpamTracker could ultimately be distributed: many servers (possibly the same ones who manage mail for various domains) report sender behavior to a centralized location that performs the clustering. SpamTracker must aggregate data from many domains, compute the corresponding clusters of email senders, and return scores from many sources; in doing so, it faces scalability and reliability challenges that could be addressed with the following enhancements.

Better scalability with data compression. SpamTracker's clustering algorithm is centralized, which raises scalability concerns, both for bandwidth (to exchange information between domains) and in terms of processing power (clustering complexity increases with input size). One way to reduce load is by distributing the clustering process. For example, compressing cluster information into average rows before sending this information to a centralized server may reduce bandwidth consumption: SpamTracker requires the full $IP \times domain$ matrix from each source to perform clustering, but requires only the average row vectors for each cluster (*i.e.*, the output of the algorithm) for classification.

Better reliability with replication and anycast. To improve availability, SpamTracker servers could be replicated and placed in different locations or on independent networks. Multiple servers might be anycasted or managed by different organizations (much like the DNS root nameserver infrastructure today), all of which perform the same computation and disseminate average rows to second-level servers, which in turn respond to user lookups.

5.6.4 Evasion

SpamTracker must be resistant to attacks that mislead the clustering engine in ways that can cause spam to be misclassified as legitimate email, and vice versa. To improve classification robustness, SpamTracker could form clusters based on email sending patterns from a smaller number of trusted email recipients (*e.g.*, a few hundred trusted domains), each of which communicates with the SpamTracker system over a secure channel. Although SpamTracker's clustering benefits from more inputs about email senders, it can serve as a classifier for a much larger set of domains that it does not necessarily trust to provide data for forming the clusters.

If spamming bots in a botnet select target domains from the *same distribution*, SpamTracker's clustering algorithm will include these spammers in the same cluster. Still, SpamTracker is limited by the time window used for clustering (*e.g.*, 6 hours, as in Section 5.5), and a spammer might exploit this weakness to evade SpamTracker. We are improving SpamTracker to automatically adjust the window in response to the fraction of received email in the last window that was classified as spam. The intuition is that the fraction of spam does not change much over short timeframes, and a decrease in the fraction of flagged email indicates that the window is too small to cluster similar IP addresses together. Spamming bots might also try to emulate the distribution of target domains (or other behavioral features) of normal senders, but by doing so, they may be inherently less effective (*e.g.*, they may have to reduce their sending rate or the expansiveness of their target list).

5.6.5 Sensor Placement

A set of domains that observes more even sending behavior across domains may be able to better distinguish spammers from legitimate senders. Recall from Section 5.2.1 that 90% of the spam we observe is received by only 84 of the 115 domains from which we observe email, and that only about 15% of the senders in our traces target more than one of the domains from which we can observe sending patterns at the email hosting provider. Based on our experiments using only clusters where the average vectors are less "skewed" towards a single domain (Figure), we expect that a more even distribution of sensors email would further improve the SpamTracker classifier. Many commercial spam filtering companies (*e.g.*, IronPort [59], Secure Computing [116]) may already have this data. Another option for sensors would be ubiquitous Web mail domains such as `hotmail.com`, `gmail.com`, etc.

5.7 Conclusion

This chapter presented SpamTracker, a system that classifies email senders using a technique we call *behavioral blacklisting*. Rather than classifying email senders according to their IP addresses, behavioral blacklisting classifies senders based on their sending patterns. Behavioral blacklisting is based on the premise that many spammers exhibit similar, stable sending patterns that can act as “fingerprints” for spamming behavior.

SpamTracker clusters email senders based on the set of domains that they target. SpamTracker uses these sending patterns of confirmed spammers to build “blacklist clusters”, each of which has an average vector that represents a spamming fingerprint for that cluster. SpamTracker tracks sending patterns of other senders and computes the similarity of their sending patterns to that of a known spam cluster as the basis for a “spam score”. Our evaluation using email logs from an email provider that hosts over 115 independent domains shows that SpamTracker can complement existing blacklists: it can distinguish spam from legitimate mail and also detects many spammers *before they are listed in any blacklist*. SpamTracker’s design makes it easy to replicate and distribute, and deploying it requires only small modifications to the configurations of existing mail servers. Our ongoing work involves gathering data from a wider set of domains, improving the behavioral classification algorithms (*e.g.*, by using other features of email senders), and deploying the system to allow us to evaluate it in practice.

CHAPTER 6

VOTE GAMING ATTACKS ON WEBMAIL SYSTEMS

6.1 Introduction

In previous chapters, we have dealt with spam sent by bots directly from the compromised host to recipient mail servers. While difficult to filter due to constantly evolving content templates and fresh IP addresses, spam detection methods such as DKIM [29] (which cryptographically signs the "From: " address of an email) and SPF [44] (which is used by ISPs and mail service providers to indicate the IPs which may legitimately send email) have had an impact on reducing the amount of spam and forged emails sent by bots directly to their recipients. The massive popularity of web-based email accounts provided by Gmail, Yahoo! Mail, and Hotmail, however, have also brought new threats: spammers have begun using bots to compromise Web mail accounts to send spam.

Recent estimates suggest that about 5.2% of accounts that logged in to Hotmail were bots [149]. Spammers use these compromised accounts chiefly to send spam or conduct targeted social engineering attacks. Spam from compromised Web mail accounts is difficult, if not impossible, to detect using IP blacklists or other spam and forgery detection methods (*e.g.*, domain-key based authentication methods such as DKIM). Web mail providers attempt to detect compromised accounts used to send spam, but these providers handle hundreds of millions of user accounts (193 million users at Gmail [37] and 275 million at Yahoo [58]) and deliver nearly a billion messages each day [139]. Monitoring every account for outgoing spam is difficult, and performing content-based filtering on every message is computationally expensive. Automated monitoring systems may not be able to differentiate a spam sender from a legitimate, high-volume sender.

In this chapter, we present an alternate way that spammers use compromised accounts: for casting dishonest “Not Spam” votes on spam email. This problem arises due to efforts of Web mail providers in trying to stem *incoming* spam. Due to the massive amounts of email that reach their domaisn, Web mail providers chiefly rely on users to “vote” on whether an email delivered to the inbox is spam or not, and conversely, whether an email delivered to the spam folder has been mistakenly flagged as spam. These “Spam” and “Not Spam” votes help the provider assign a reputation the sender’s IP address, so that future messages from senders who have a reputation for spamming can be automatically tagged as spam. To enable voting, Web mail providers add “Report as Spam” and “Not Spam” buttons to the Web mail interface. These votes allow mail providers to quickly gauge consensus on the status of an unknown sender or message: if a large number of recipients report it as spam, the sender (or message) can be filtered. These votes from users, sometimes referred to as “community clicks” or “community filtering”, are in most cases the best defense against spam for large Web mail providers [39].

We discovered that spammers use compromised Web mail accounts not only to send spam, but also *to cast votes that raise the reputation of spam senders*. We call this type of attack a *vote gaming attack*. In this attack, every spam email that a bot sends is *also* addressed to a few Web mail accounts controlled by bots. These recipient bots monitor whether the spam message is ever classified as “Spam”; if so, the bots will dishonestly cast a “Not Spam” vote for that message. Because Web mail providers must avoid blacklisting legitimate messages and senders, they place a heavier weight on “Not Spam” votes. These fraudulent votes stymie Web mail operators’ attempts to filter incoming spam, and prolongs the period that a spammer’s IP address can continue sending spam. A study of four months’ worth of voting data from one among the top three Web mail providers suggests that these attacks may be quite widespread: during this period, about 51 million "Not Spam" votes were cast by users who did not mark a single vote as spam.

Ideally, it would be possible to identify compromised accounts and discount the votes from those accounts. Unfortunately, we find that spammers use a *different set of compromised accounts* to cast fraudulent votes than they use to send spam, so techniques for detecting compromised accounts that are based on per-user or per-IP features cannot solve this problem. Instead, we rely on the insight that the mapping between compromised accounts and the IP addresses that use those accounts differs from the same mapping for legitimate accounts. Accounts that cast fraudulent votes tend to have two properties: (1) the same bot IP address accesses multiple accounts, and (2) multiple bot IP addresses access each compromised account.

In this chapter, using four months of email data from a large Web mail provider that serves tens of millions of users, we study (1) *the extent of vote gaming attacks*; and (2) *techniques to detect vote gaming attacks.* To the best of our knowledge, this is the first study that characterizes vote gaming attacks at a leading Web mail provider. To detect this new class of attacks, we develop a high-dimensional, parallelizable clustering algorithm that identifies about 1.1 million compromised accounts that cast fraudulent votes (most of which were previously undetected), with few false positives. We compare our technique to a graph-based clustering algorithm, BotGraph [149], that has been used to detect compromised accounts. We show that our technique, which is now deployed in production at a large Web mail provider, detects *10 times as many vote gaming user accounts* (approximately 1.13 million accounts), with a *10× reduction in the false positive rate* (approximately 0.17%). We also describe how to implement variants of our technique on a grid processing infrastructure such as Hadoop [48]—a key requirement when dealing with data at the scale of a production Web mail service.

Although we focus on vote gaming attacks that were mounted on a large Web mail provider, vote gaming has occurred in other Web-based services as well, such as online polls [1] and story ranking on social news sites [3]. Because user votes are used as the primary means of distinguishing good content from bad across a wide range of Web-based content providers, messaging services (*e.g.*, Twitter), video-sharing sites (*e.g.*, YouTube), etc., vote gaming is a threat for these applications as well. Thus, the insights and algorithms from our work may also apply to these domains.

The rest of this chapter is organized as follows. Section 6.2 provides details on vote gaming attacks. Section 6.3 presents a model of the vote gaming attack, which we use to design our detection mechanisms (Section 6.4). Section 6.5 evaluates the techniques, and Section 6.6 describes scalable, distributed implementations of the detection techniques and evaluates the speed of the two implementations. Section 6.7 evaluates the sensitivity of the algorithms to parameter settings. Section 6.8 discusses open issues and avenues for future work, and Section 6.9 concludes.

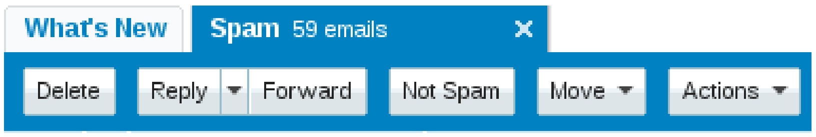
6.2 Vote Gaming Attacks

Spam from Compromised Web Mail Accounts. Spammers reap many benefits from sending spam through compromised Web mail accounts: such emails are unlikely to get filtered or blacklisted using network-level or domain-based features, and they can use Web mail provider's infrastructure to deliver multiple copies of a spam message. These advantages have inspired botmasters to acquire many user accounts either by "phishing" the passwords of trustworthy customers, or through automated registrations by cracking CAPTCHAs [38].

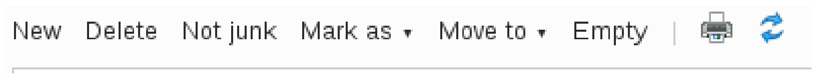
A recent study by Microsoft researchers found 26 million botnet-created user accounts in Hotmail [149]. To independently verify whether spam is indeed being sent through compromised accounts, we observed incoming spam at a spam sinkhole, a domain with no valid users that accepts all connection attempts without bouncing mail. We collected 1.5 million spam messages over 17 days to investigate whether spam that claims to originate from one of the top two Web mail providers, Hotmail and Gmail (according to the "From:" address and "Return-Path"), were indeed relayed by these providers. Using SPF verification [44], we found that nearly *10% of spam from* `gmail.com` *and nearly 50% of spam from* `hotmail.com` *are sent through these provider's servers.* Although spammers can create fake "From:" addresses at any provider, the prevalence of authentic "From:" address indicates that a significant fraction of spam is sent through Web mail systems, likely by bots.

User Voting as a Spam-filtering Mechanism. Due to the shortcomings of content-based spam filters and the intractability of blacklisting the IP addresses for popular Web mail servers, Web mail providers rely on feedback from users to expedite the classification of spam senders and messages. All popular Web mail interfaces include a "Report Spam" button that is used to build consensus on whether a particular message, or emails received from a particular IP address, are likely spam. Figure 6.1 shows the prominent position of the "Not Spam" button on the reading panes of Yahoo! Mail, Windows Live Mail, and Gmail. Soliciting user feedback is effective [39]: when a number of users report a spam message, the system detects consensus and can automatically learn to filter further messages from the sender. Web forums and other media services also rely on similar approaches.

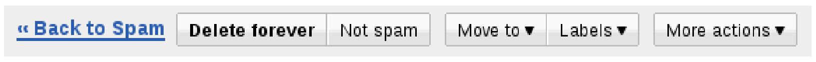
Fraudulent Voting. Figure 6.2 represents a typical pattern of vote gaming attacks at a large Web mail provider. Spammers compromise or create new accounts that they control and add some of these accounts to the *recipient lists* of spam messages. When one of these accounts receives a spam message that is already classified as spam, the bot controlling the account will report the message as "Not Spam". When a number of bots report the message as "Not Spam", the spam filtering system will notice the lack of consensus and refrain from automatically filtering the message into a user's spam folder, since misclassifying legitimate mail as spam is considerably detrimental.



(a) Yahoo! Mail



(b) Windows Live



(c) Gmail

Figure 6.1: "Not Spam" buttons appear on the interfaces of popular Web mail services when reading a message already classified as spam.

To make detection more difficult, botmasters do not typically use voting bots to send spam, which maximizes the number of "not spam" votes that each voting bot can cast before being detected. Figure 6.3 shows an example of the series of votes cast on messages sent by a likely spammer IP address over the course of 19 days at a large Web mail provider.

6.3 Modeling Vote Gaming Attacks

I1n this section, we develop a model for vote gaming attacks and explain how the behavior of accounts used for vote gaming differ from that of legitimate users.

Consider a dataset that consists of:

- a set of "Not Spam" (NS) votes,
- the identities of the users who cast the votes ($\{U\}$)
- the IP addresses that sent messages on which these votes were cast ($\{P\}$).

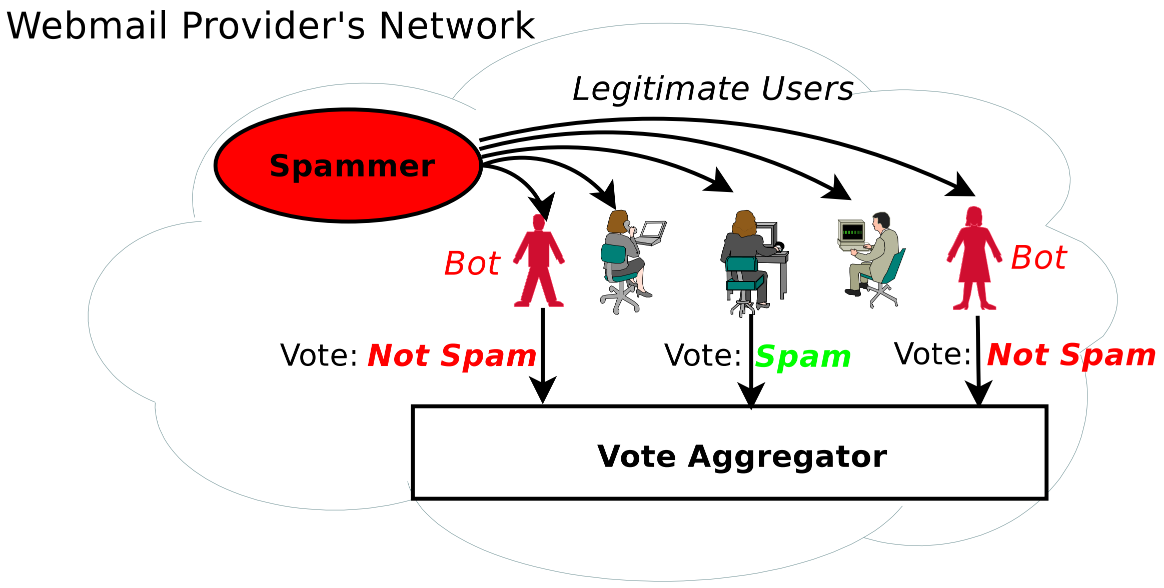


Figure 6.2: A spammer sends mail to many legitimate user accounts, as well as a few accounts controlled by voting bots. If the message is classified as Spam, bots will report it as "Not Spam", prolonging the true classification of the message.

We can represent voting as a *bipartite graph*, where each NS vote is an edge from a user ID to an IP address, as shown in Figure 6.4. In practice, this dataset is *unlabeled* (*i.e.*, identities of the bots and spammers are unknown) even though they are labeled in the figure for clarity.

Two properties of vote gaming attacks help detection:

1. *Volume:* Compromised user accounts cast NS votes to *many different* IP addresses
2. *Collusion:* Spammer IP addresses receive "not spam" votes from *many different* compromised accounts.

Of course, legitimate users also cast NS votes, and a legitimate user may also incorrectly cast a NS vote on a spam sender. Legitimate users may also cast many NS votes, either because they receive a large amount of email, or perhaps because they have subscribed to mailing lists whose messages are frequently marked as spam by other users. However, legitimate users tend to not cast collections of NS votes

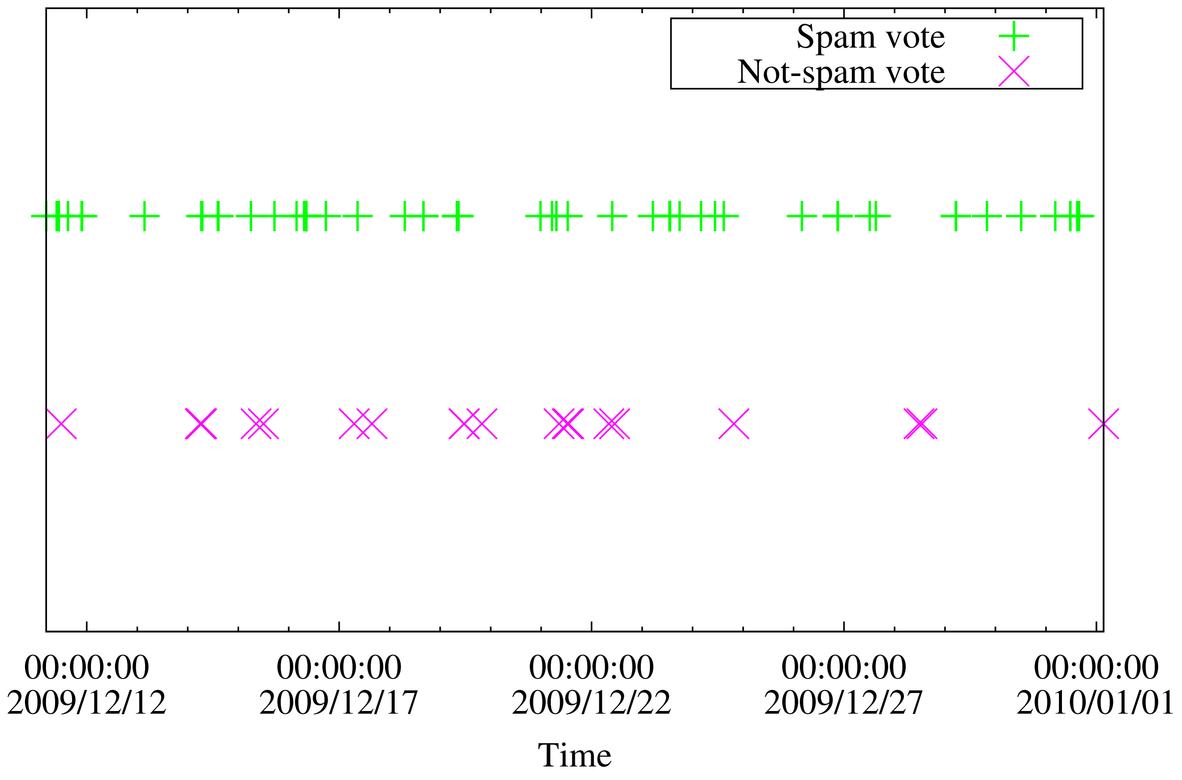


Figure 6.3: Timeseries of Spam and Not Spam votes cast on a likely spammer IP address over 19 days. Fraudulent voters need to cast fewer votes to annul the "spam" classification of a message.

on a *specific set* of IP addresses, because it is extremely unlikely for multiple legitimate users to receive a spam message from the same IP address *and* proceed to vote NS on the same message. Thus, in combination with the second feature—that a large fraction of IP addresses that a bot votes on will also be voted on by *other* bot accounts—we can detect compromised accounts with very few false positives. Because legitimate users do not cast NS votes on messages because of the IP that sent the message, they are unlikely to share a large set of their voted-on IPs with other legitimate users.

Using these insights, we can apply unsupervised learning to the model of voting data to extract sets of likely gaming accounts. To enable unsupervised learning, we first represent the bipartite graph as a *document-feature matrix*, with user accounts

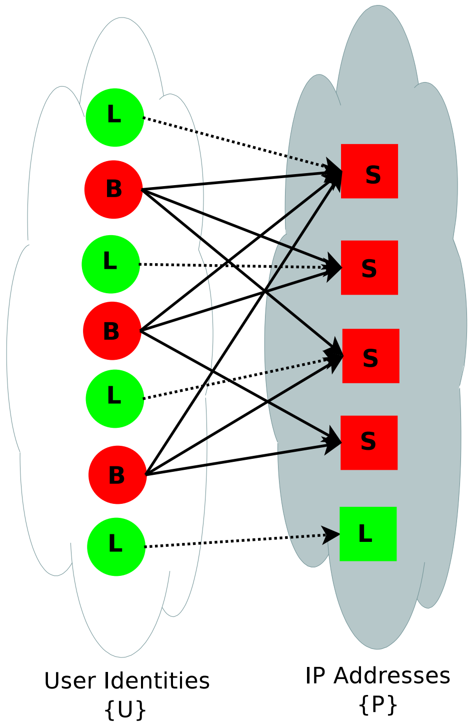


Figure 6.4: NS votes as a bipartite graph matching voting user IDs ($\{U\}$) to sender IP addresses ($\{P\}$). Dotted edges represent legitimate NS votes; thick edges represent fraudulent NS votes. L: legitimate voter/sender; B: bot voter; S: Spam sender.

as documents and the IP addresses that are voted on as features. We then cluster accounts that have high similarity with each other based on the number of IP addresses they share. Section 6.4 describes our clustering approaches, and how it outperforms a similar approach used in BotGraph [149].

Our detection methods rely on three assumptions:

A1 Compromising accounts is sufficiently costly to require spammers to reuse accounts in U.

A2 A single user ID in U can vote on a specific IP address in P at most m times.

A3 The majority of votes on a spammer's IP address are "Spam" votes from legitimate users.

All of these assumptions typically hold in reality. A1 holds because most Web mail providers follow a reputation system with regards to voting. To prevent spammers from creating large amounts of accounts and using them only to cast NS votes, users need to build up a voting reputation in order to be accounted for. This requires spammers to compromise existing accounts with good voting reputation, which is time-consuming. A2 holds because the Web mail provider must reach a consensus across many users. Thus, most providers only allow a few votes per IP address (we assume $m = 1$). A3 holds because legitimate recipients outnumber compromised accounts. This assumption is inherent in the business model of spammers, who want to reach as many users as possible and have fewer compromised accounts than target “clients”. A3 implies that each spammer must cast several NS votes to affect the consensus for an IP address. If each compromised account can only cast a single vote per IP address, to achieve a critical number of NS votes, the spammer must cast multiple NS votes from different accounts.

6.4 Detecting Vote Gaming Attacks

We now develop detection methods for vote gaming attacks. We review an existing graph-based clustering algorithm from Kumar *et al.* [75] and later applied in Bot-Graph [149]. We explain why this approach is not optimal for detecting vote gaming attacks; we then present a new clustering approach using *canopy-based clustering.*

6.4.1 Problem: Clustering Voting Behavior

Figure 6.5 shows how we can represent a sample voting graph as the input document-feature matrix M for a clustering algorithm. Let U be the set of users who voted and P be the set of IPs they voted on. $M \subseteq U \times P$, and each $M(i, j)$ denotes the number of votes given by user i to an email sent from IP j. The matrix M consists of all users who have voted and all IPs that have received a non-spam vote. Our goal is to extract groups of fraudulent user identities from M with few false-positives.

Large email providers have tens of millions of active users per month, and the number of voted-on IPs is on the order of millions. We wish to identify the user IDs

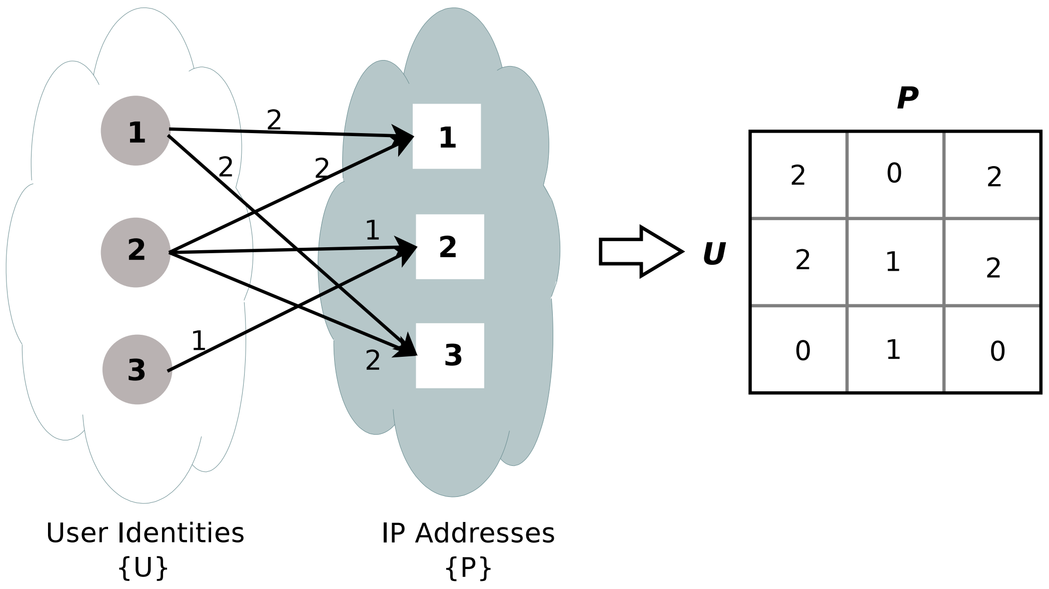


Figure 6.5: Representing the NS voting graph as an adjacency matrix. Labels on edges represent the number of times a user votes on an IP.

that behave similarly by clustering in this high-dimensional space. Our setting differs from conventional clustering setups [53] in the following ways:

1. **Lack of cluster structure.** Unlike the usual settings in which clustering is performed, there are no clear clusters in our data. In fact, with a normal set of users, two users will rarely receive emails from the same IP, and even more rarely will they cast the same vote on the same IP. Thus, any form of tightly connected clusters in our data is a signal of anomaly—as we shall see later, we instead end up with a large number of clusters at various scales.

2. **Sparsity.** On average, users cast *less than one* non-spam vote during the entire month, although we also observe a significant number of users with large numbers of non-spam votes.

3. **Data scale.** Our data has many users and IPs. While many traditional clustering algorithms are quadratic time, our data's scale requires linear-time or near-linear-time algorithms.

4. **Adversarial nature.** The data is generated adversarially. The spammers can

succeed only if they can remain undetected by the anti-spam filters. This means that we rarely get spammers casting a large number of non-spam votes from the same ID. Instead, campaigns to vote "Not Spam" are distributed over a large number of user IDs.

These features make the choice of clustering algorithm and distance metric critical. As a simple example, clustering based on distance metrics such as the Euclidean metric will erroneously show high similarity between IDs which have few IPs in common as long as the common IPs have high weight.[1] Consequently, we need to develop clustering strategies specifically for our problem setting.

6.4.2 Baseline: Graph-based Clustering

As a baseline for comparison, we apply a graph-based clustering method that is similar to the technique introduced by Kumar *et al.* [75] and later applied by BotGraph [149]. We choose this algorithm to enable direct comparison of methods used in previous work, and with our second approach, canopy-based clustering. Kumar *et al.* [75] proposed the k-neighborhood plot as a way to study the similarities between entities using Web data. Given a bipartite graph $G = (A, B, E)$, Kumar *et al.* define the *k-NC graph H* corresponding to G as follows: H is defined over the vertex set A; we include edge (u, u') in H if there exist k distinct nodes $\{v_1 \ldots v_k\} \subseteq B$ such that for each i, both (u, v_i) and (u', v_i) are in G. Figure 6.6 illustrates the construction of a k-neighborhood graph from a bipartite graph. Zhao *et al.* use the same construct in BotGraph to discover botnets by working with the bipartite graph of users versus the Autonomous System (AS) numbers of the IPs from which users log in [149]. We make one improvement to the clustering approach in BotGraph: rather than mapping user

[1]Consider two vectors $\mathbf{A} = [1, 1, 1, 10]$, $\mathbf{B} = [0, 0, 0, 10]$, and $\mathbf{C} = [1, 1, 1, 3]$. The distance between $\mathbf{A}$ and $\mathbf{B}$ is $d_{\text{Euclidean}}(\mathbf{A}, \mathbf{B}) = 1.73$, although $\mathbf{A}$ and $\mathbf{B}$ have only one feature in common. The distance $d_{\text{Euclidean}}(\mathbf{A}, \mathbf{C}) = 7.0$, *i.e.*, *greater than* $d_{\text{Euclidean}}(\mathbf{A}, \mathbf{B})$, even though $\mathbf{A}$ and $\mathbf{C}$ vote on the same set of IPs. The high-valued feature influences the Euclidean metric more than, for example, the Jaccard metric.

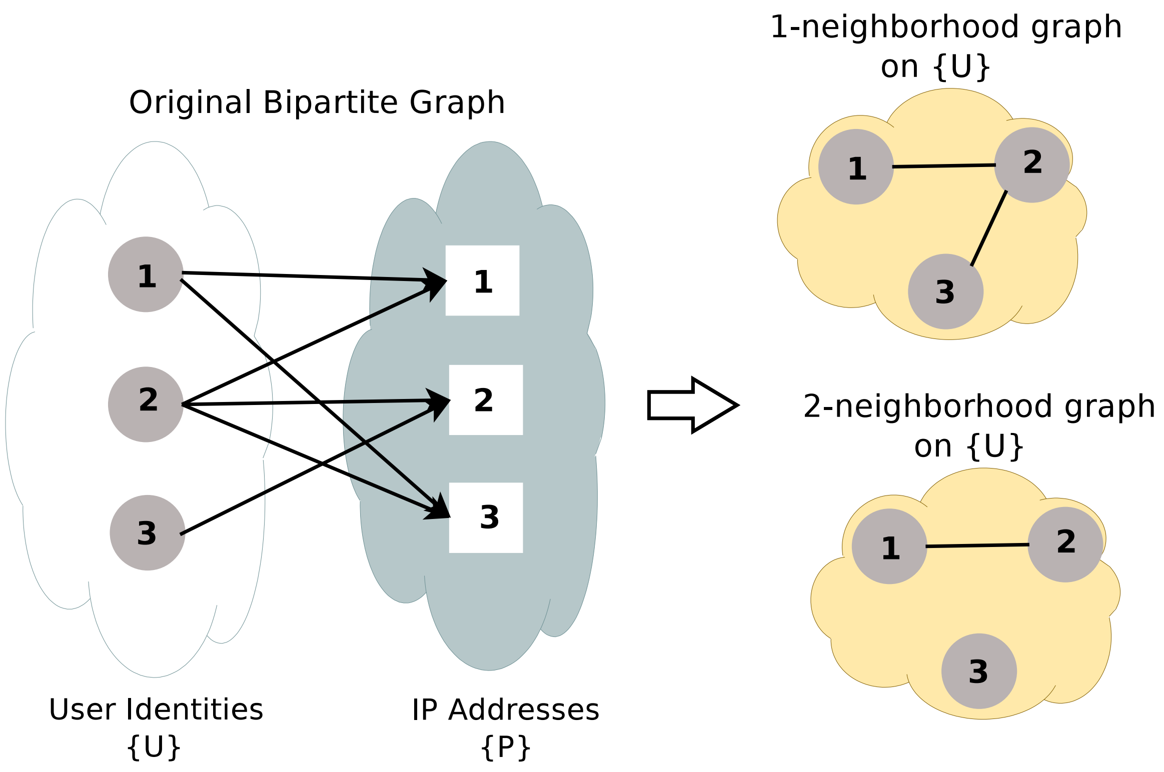


Figure 6.6: k-neighborhood representation of $\{U\}$ from Figure 6.5.

accounts to AS numbers, we map them to IP addresses, since mapping user accounts to AS numbers hides the fact that a user account is accessed from multiple locations.

Efficiently finding a value for k. Two users voting NS on the same k sender IPs is indicative of suspicious coordinated behavior. The success of this approach depends on efficiently finding a value of k that identifies a significant number of attackers with no false positives. A low value for k may retain some legitimate users in components that mostly have bots. On the other hand, a high value for k produces components whose voting behaviors are highly coordinated, although the sizes of the components—and hence the number of bots identified—decrease.

A simple way to construct the k-NC graph for any fixed value of k first creates the weighted graph G' with vertex set U where for each (u, u') the weight $w(u, u')$ equals the number of common neighbors of u and u' in G. Then, we can create the threshold using the value of k that we desire and apply standard component finding algorithms.

This takes time $O(\min(|U|^2, i_{\max}|E|))$ where $i_{\max}$ is the maximum number of users who vote on an IP. This approach is infeasible when the size of $|U|$ is on the order of tens of millions, and $i_{\max}$ is typically of the order of thousands as well. For a fixed value of k, Kumar *et al.* [75] show how to compute the k-NC graph in time $O(n^{2-1/k})$ where $n = |U|$, which is significant gain for small k. Our setting, however, requires a larger k to ensure we do not create edges between normal users and bot accounts so this algorithm is impractical in our setting. Furthermore, as with BotGraph [149], we need to run the component-finding algorithm at various values of k to find the right threshold.

To create components at various thresholds, we have developed a new technique using dynamic graph algorithms for maintaining components under edge additions and deletions. Although it is difficult to maintain components under edge deletions, it is easier to do so under edge additions. Thus, we start with a maximum value $k_{\max}$, find components with threshold $k = k_{\max}$, and then decrease k by 1. At each step that we decrement k, the graph gains a new set of edges, and these could change the component structure by joining some previously disconnected components. Updating the component list efficiently only requires maintaining a union-find data structure, and the whole process takes total time $O(k_{\max}(|U| + |E|.\alpha(E, U)))$, where $\alpha(E, U)$ is the inverse Ackermann function, an extremely slow-growing function which is a small value—less than 5—for almost all practical values of $|E|$ and $|U|$.

Graph-based Clustering Produces False Positives. The most significant shortcoming of graph-based clustering such as BotGraph [149] for detecting bot-controlled accounts is its false positive rate, which are typically unacceptable for email. Intuitively, graph-based clustering disconnects edges lower than a certain weight and labels *all* nodes in a large connected component as bots; it does not pay attention to the absolute degree of a node in a connected component *when compared with other nodes in the component.* This behavior produces false positives.

Figure 6.7 illustrates why graph-based clustering may produce false positives. The nodes (*i.e.*, user accounts) shown outside the cloud are legitimate, but the nodes inside the cloud are controlled by bots. All the legitimate accounts share two IP addresses between each other (*e.g.*, perhaps due to a company proxy server that cycles between two public IP addresses), as shown by the edges with weight two. Unfortunately, one legitimate user has also logged in from two IP addresses that have bot programs running on them. This scenario could be a false positive—for example, the legitimate user's IP address could have been recycled with DHCP to a botted machine—or it could have occurred accidentally, because the legitimate user has a bot program on his computer while he continues to use it. In either case, this legitimate user acts as a "bridge" that connects a component of true voting bots, and a number of legitimate users that would otherwise have been disconnected. A clustering algorithm based on pairwise similarity comparisons is unlikely to make this mistake because it would compare all-pairs similarity, and discover that the true bots have a much higher similarity to each other than other pairs. Although this particular false positive could have been avoided by increasing the value of the threshold k to 3, the BotGraph algorithm would stop the component finding process at $k = 2$, because the component sizes between successive steps differs by an order of magnitude: the component of 14 nodes breaks to a largest component of 3 nodes if k is increased to 3.

6.4.3 Our Approach: Canopy-based Clustering

To reduce false positives and cope with high dimensionality, we adapt a two-stage clustering technique by McCallum *et al.* called *canopy clustering* [82]. Canopy clustering is a divide-and-conquer approach for clustering high-dimensional data sets. Canopy clustering is more practical than graph-based clustering for detecting vote-gaming attacks, because it produces fewer false positives and is more scalable. The

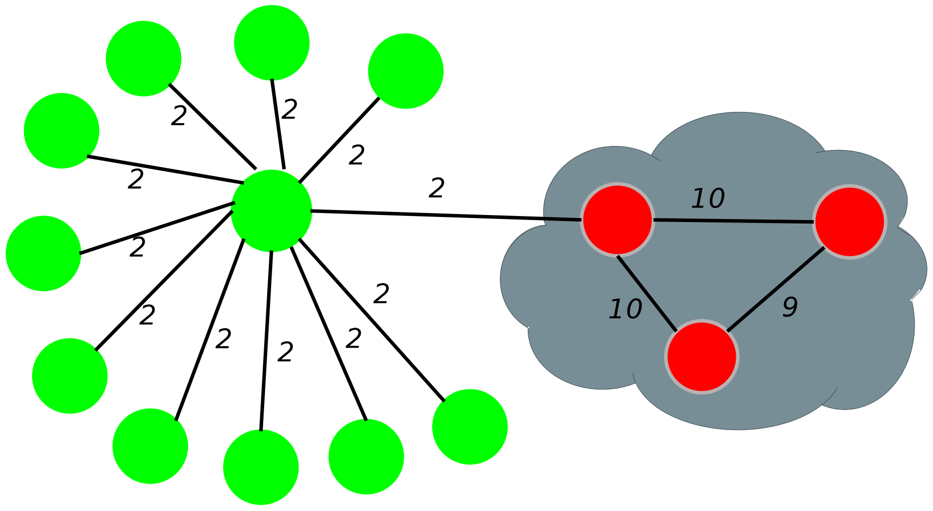


Figure 6.7: Shortcoming of graph-based clustering: one false-positive edge can connect a bot component (shown within the cloud) to a number of unrelated, almost-disconnected legitimate users (outside the cloud). Edge labels are the edge-weights. Here, the threshold $k = 2$.

algorithm proceeds in two stages:

Step 1: Canopy Formation. First, we partition the raw data into overlapping subsets called canopies, using an inexpensive similarity metric and very few similarity comparisons. We construct canopies such that all elements in a cluster in the output of a traditional clustering algorithm will be within the same canopy. Thus, the second stage of canopy clustering need only conduct more rigorous similarity comparisons for elements that are within the same canopy. Provided that the number of elements in the largest canopies are much smaller than in the raw data, this method typically reduces the number of expensive similarity measurements by many orders of magnitude.

The choice of metric used to create the initial partition of the raw data into canopies is important: a good metric is inexpensive (*i.e.*, does not involve operations such as division or multiplication), and minimizes the size of the largest canopy. Following McCallum *et al.*'s suggestion of using the number of common features between elements as an inexpensive metric, we use the *number of common IPs voted* *on by two users* as our canopy metric. We explain this metric in Section 6.6, and how its parameter settings affect detection and false positive rates in Section 6.7.

Step 2: Conventional Clustering. The output of the first step are canopies of tractable sizes, such that we can directly perform clustering on each canopy. For this stage, we use a well-known hierarchical clustering scheme, greedy agglomerative clustering (GAC), using α-*Jaccard* similarity[2] as the metric. We choose GAC using the Jaccard metric because it is appropriate for clustering user IDs where the similarity metric should take into account the fraction of shared IPs. In Section 6.6, we introduce an approximation of this method that works in a cluster computing infrastructure such as Hadoop. We also discuss how to parallelize this clustering using techniques from locality sensitive hashing [36].

GAC is an iterative method, where initially, each element in the data set is in a cluster of its own. At each iteration, we find the similarity between every pair of clusters using the Jaccard metric, and merge the two clusters that are the most similar to each other, provided this similarity is greater than a threshold, α. We compute the Jaccard metric between two clusters using the mean distance between elements in the cluster. If $\mathcal{C}_1$ and $\mathcal{C}_2$ are two clusters of elements, the mean distance is

$$d_{\text{mean}}(\mathcal{C}_1, \mathcal{C}_2) = \frac{1}{|\mathcal{C}_1|\ |\mathcal{C}_2|} \sum_{x \in \mathcal{A}} \sum_{y \in \mathcal{B}} d_{\text{Jaccard}}(x, y)$$

Iteration stops when either (1) only a single cluster remains, or (2) the similarity between the two most-similar clusters is less than α. Because canopies are overlapping, an element may be clustered into multiple clusters. To resolve this issue, after we perform GAC on each canopy independently, we assign any element that is in

[2]Let x and y be two user identities, with X and Y representing the sets of IP addresses on which they voted "not spam". x and y will be clustered together only if

$$\frac{|X \cap Y|}{|X \cup Y|} \geq \alpha$$

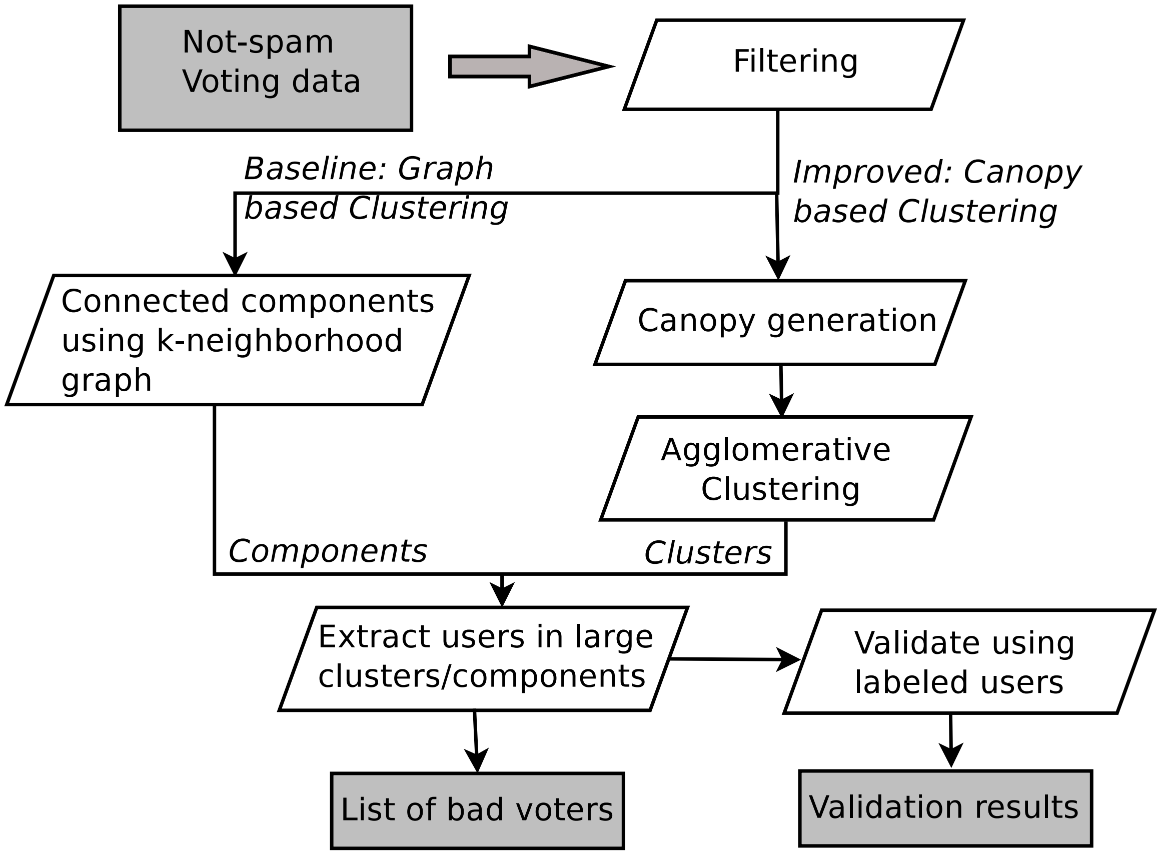

Not-spam
Voting data
Filtering
Baseline: Graph based Clustering
Improved: Canopy based Clustering
Connected components using k-neighborhood graph
Canopy generation
Agglomerative Clustering
Components
Clusters
Extract users in large clusters/components
Validate using labeled users
List of bad voters
Validation results

Figure 6.8: Workflow for finding and validating fraudulent voters from unlabeled voting data.

multiple clusters solely to the largest cluster; we find that this choice does not incur false positives because most large clusters are likely comprised of bot accounts.

6.5 Evaluation

We evaluate the accuracy and precision of the clustering algorithms for detecting vote gaming attacks. Section 6.5.1 describes our dataset; Section 6.5.2 describes the metrics used to evaluate the quality of the clustering algorithms, and presents the basic performance of each algorithm for identifying vote gaming attacks. Figure 6.8 explains the workflow of our evaluation and validation technique.

Main Result. Although both canopy-based greedy agglomerative clustering (GAC) and graph-based clustering both can detect vote gaming attacks, GAC finds more

Table 6.1: Description of voting dataset.

Period	4 months (Jul.–Oct. 2009)
Total Voting Users	35 million
↪ *Total only-NS voters*	↪ 13.93 million (39.8%)
↪ *Users labeled "good"*	↪ 1.3 million (3.71%)
↪ *only-NS voters*	↪ 22,853 (1.76%)
↪ *Users labeled "bad"*	↪ 2.41 million (6.91%)
↪ *only-NS voters*	↪ 164,941 (6.82%)
Total Spam votes	357 million
Total Not-spam votes	82 million
↪ *By only-NS voters*	↪ 51.7 million (63%)
Voted-on IPs	5.1 million
↪ *Voted-on as NS*	↪1.7 million (33.3%)

potentially dishonest voters overall (1.1 million vs. 160,101), has a higher detection rate of confirmed dishonest voters (10% vs. 3%) and a lower false positive rate (0.17% vs. 1.09%). (Section 6.5.2, Table 6.2)

6.5.1 Data

Our dataset consists of the logs of votes cast by the users of a large Web mail service provider on mail that they receive, extending for four months from July–October 2009. Each line corresponds to one vote; the fields included are: (1) the ID of the user who cast the vote, (2) the IP address of the sender of the email on which the vote was cast (the *"voted-on" IP*), and (3) the type of vote—"S" for spam and "NS" for not spam. Section 6.6 describes the filtering stage of our workflow.

Ground Truth. To validate whether the clusters of voters we obtain contain fraudulent voting accounts, we use labels of confirmed fraudulent voting accounts obtained from the Yahoo! Mail anti-spam team. These accounts were enumerated using techniques other than the ones we described in this chapter (*e.g.*, erratic clicking behavior, voting too many "not spam", etc.). To evaluate the percentage of false positives, we use a list of users known to engage in reputable behavior; this list contains users who have long-standing accounts with the provider, or users who have purchased items

Table 6.2: Comparison of Greedy Agglomerative Clustering (GAC) and Graph-based clustering that shows the median cluster (or component) size, the number of potential dishonest voters detected, the percent of confirmed dishonest voters detected, and the false positive rate.

Method	*Median clust. size*	*Detection*	*Detection rate*	*FP rate*
Canopy Clustering	109	1,138,368	**10.24%**	**0.17%**
Graph-based	32	157,177	3.51%	1.09%

from e-commerce sites also owned by the provider's parent company. Because the set of labeled users was collated independently by the anti-spam team at the large Web mail provider, only a subset of these labeled accounts intersect with our 4-month dataset of NS votes. Historically, these labeled lists are known to be highly accurate.

Table 6.1 summarizes the voting dataset and its intersection with user labels. We have observed empirically that, although some NS votes are legitimate (*e.g.*, there are cases where a legitimate email contained keywords that triggered a content filter for spam), the majority of NS votes are performed by bots to delay the identification of spam sent by other bots: 63% of NS votes are cast by users who only cast NS votes. Although we derive data labels using independent verification methods (*e.g.*, manual inspection, suspicious account activity), these labels can often only be attributed to the users *after* they have performed a significant amount of malicious activity and have been de-activated. Our goal is to identify as many *undiscovered* fraudulent voters as possible, so we use accounts that are labeled after the time period during which we evaluate our clustering methods.

6.5.2 Detection and False Positive Rates

Our aim is to identify large groups of bots without incurring many false positives. Thus, for either clustering algorithms, we consider any user account that falls into clusters (or components) above a threshold size to be a bot. We compare the two techniques in terms of three metrics: (1) *detection*, *i.e.*, the number of users that are classified into clusters larger than the x^{th} percentile cluster size (x being variable); (2)

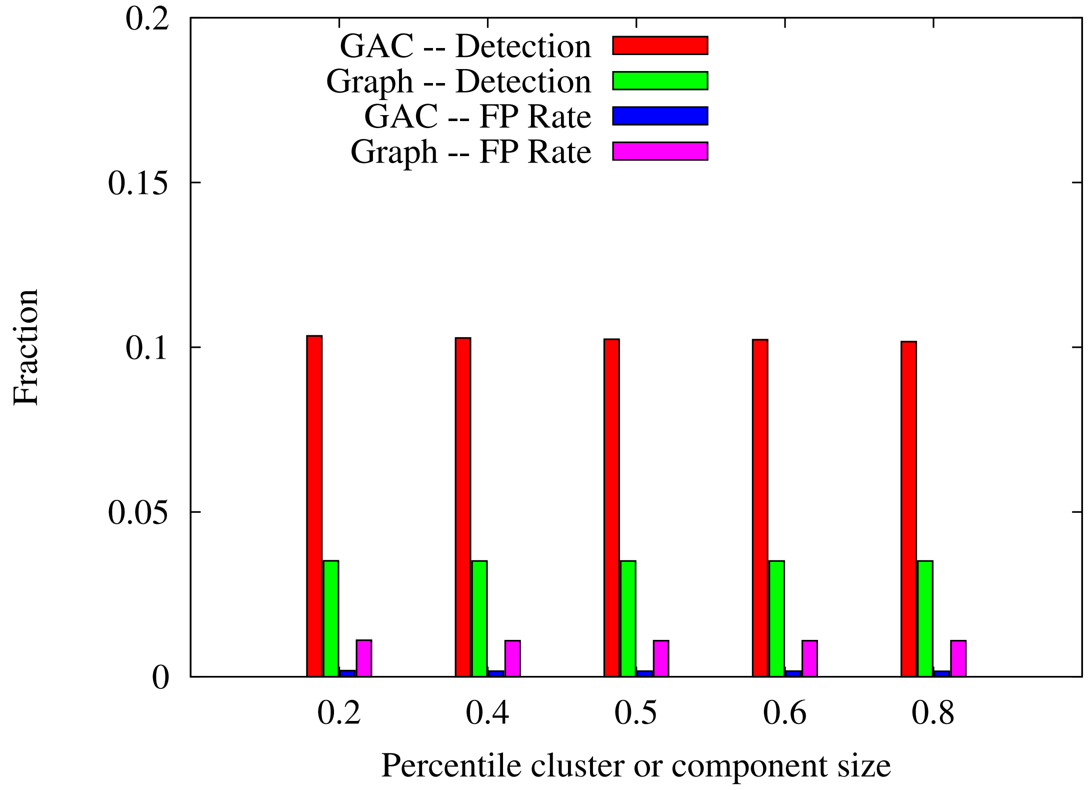


Figure 6.9: Performance of GAC and Graph-based clustering for various percentiles of cluster/component sizes. The x-axis shows the percentile cluster size above which all clusters are considered to contain only bots. The y-axis shows the detection and false positive rates.

detection rate, *i.e.*, the fraction of users labeled "bad" (*i.e.*, fraudulent voters) who are in clusters larger than the x^{th} percentile cluster size, and (2) *false positives (FPs)*, which we quantify as the ratio of good users in clusters larger than the x^{th} percentile cluster size to all good users, for various values of x. Table 6.2 presents these statistics for the median (*i.e.*, $x = 0.5$) cluster size, and Figure 6.9 shows the detection and FP rates for various percentile values (x). Neither GAC nor graph-based clustering vary much in terms of detection or false positive rates with respect to x; thus, even a small-sized cluster is likely to contain mostly bots. Graph-based clustering results use $k = 5$, and canopy-based GAC uses a Jaccard similarity threshold of 0.85. Section 6.7 explains our parameter choices for both algorithms in detail.

Canopy-based GAC outperforms graph-based clustering in all metrics: the number of potential dishonest votes detected, the detection rate of confirmed dishonest voters, and the false positive rate. GAC detects more potential dishonest voters because, unlike graph-based clustering, it does not create clusters based on a discrete parameter (*i.e.*, k): graph-based clustering at k = 5 has a few large clusters with a steep drop in cluster size, while the drop in cluster sizes is more gradual with GAC. GAC is also more precise than graph-based clustering. in graph-based clustering, a large connected component at some k may contain two or more sub-components which are connected only by an edge of weight exactly k. Even if the users in one sub-component do not vote on the same IPs as users in the other, they will be categorized into one large component, potentially increasing false positives if some of these users are legitimate. GAC performs all-pairs similarity comparison between users, which results in clusters where all users are similar to one another.

One of the top three large Web mail providers is using our detection technique in production. Although a 10% detection rate of confirmed dishonest voters may seem low, even single-percentage-point gains are significant for a for large-scale Web mail providers, given the high volumes of spam seen by Web mail providers. Any increase in detection rates can help these providers make more accurate decisions about which email connection attempts to reject early, and which mail can be more quickly and efficiently classified as spam (*e.g.*, without inspecting the message's contents); indeed, clustering is being applied in practice at the large Web mail provider to detect fraudulent voters. Our techniques also identified fraudulent voters more quickly than other methods: many of the bots we discovered were identified by the anti-spam team as bots only well *after* our dataset was collected. We also note that the actual detection rate may be higher that 10% in practice, because at least some of the users labeled "bad" may have had the bulk of their malicious activity before or after the time period of our dataset.

6.6 Scalable Distributed Implementation

We describe scalable implementations of the distributed graph-based clustering (Section 6.6.2) and canopy-based clustering (Section 6.6.3). We evaluate the performance of the two methods in Section 6.6.4.

Main Result. Both implementations run on our 4-month dataset in only a few hours, making it practical to run on a sliding window that includes new voting data. GAC is slower than graph-based clustering due to the overhead of all-pairs comparisons (Section 6.6.4, Table 6.3).

6.6.1 Overview

At the scale of large Web mail providers, raw voting data totals tens of millions of unique identities that map to millions of IP addresses. At this scale, analyzing data on a single machine is often infeasible. Many large organizations such as Yahoo!, Google, and Microsoft use distributed computing to analyze Web-scale datasets, by storing the data on distributed filesystems and using methods such as MapReduce [27] to process them.

MapReduce is appropriate for tasks that are inherently parallelizable, such as searching and sorting, but solving clustering tasks using MapReduce poses a number of challenges. First, because individual rows of the matrix M may be split across different mappers and reducers, MapReduce clustering algorithms often take many iterations to converge to a high-quality clustering. Second, between each iteration of clustering, there could be a large amount of inter-node communication in the distributed filesystem as potentially similar rows of M are sent to the same mapper/reducer. Finally, the intermediate output containing the results of comparing every pair of rows may sometimes be much larger than the raw dataset. Although some clustering algorithms, such as k-means [53], are parallelizable, they are ill-suited for our problem.[3] Unfortunately, our clustering algorithms expect a shared-memory architecture and are not inherently parallelizable. Below, we present efficient approaches to implementing both graph-based clustering and canopy-based clustering using MapReduce that trade off accuracy for efficiency.

6.6.2 Distributed Graph-based Clustering

Step 1: Creating an approximate user-user graph using MapReduce. In a distributed infrastructure, computing the k-neighborhood graph is challenging due to the amount of intermediate output it generates. Suppose the original bipartite graph is stored in the following format:

```
<user ID> <list of (IP, NS votes) pairs>
```

Because this file is split across many machines, the straightforward approach to construct the k-neighborhood graph uses two MapReduce iterations. The first iteration's Map phase outputs the *inverse* edge file where each line has an IP address as the key and a user ID that voted on it as the value. The Reduce phase will then collect all lines with the same key and output all pairs of users who have the same key. The second iteration counts the number of time a specific user-user pair has been written out, which yields the number of IPs shared between the two users—the edge weight in the user-user graph. The main bottleneck in this process is the size of intermediate output between the two iterations: for example, an IP that has been voted on by 1000 users will produce $\binom{1000}{2}$ pairs of user-user entries, and when repeated for many high-degree IPs can overflow even the terabytes of space on a distributed filesystem.[4]

[3] k-means, although widely applied, has flaws: (1) every point in the data is forced into a cluster, which may affect the cluster quality if points are outliers; (2) as mentioned before, the euclidean distance metric is both expensive to compute, and gives weight to the larger-valued features than the number of common features; (3) the number of clusters, k, may not be easy to determine beforehand.

[4] Zhao *et al.* also face this problem, but alleviate it using DryadLINQ [145] that offers a "merge" capability to reduce intermediate output size; we use the more widely-used MapReduce platform.

We apply approximations to filter the number of intermediate user-user edges that must be output. We first filter users who have voted on very few IPs. Next, because we are interested only in users who fall into large components at reasonably high values of k, we suppress user-user edges where the two users are unlikely to have many IPs in common. To do so, we hash the IPs that are voted on by a user into a fixed-size bit-vector, essentially a variant of a count-min sketch [24]. Before outputting a user-user edge, we compare the overlap between the two users' bit vectors and proceed only if the overlap is greater than a certain threshold (which we set to lower than k_{max} because hashing different IPs to a fixed-size bit vector could create collisions). Similarly, when outputting all the user-user pairs for a certain IP that has a large number—say p users, voting for it—instead of outputting all $\binom{p}{2}$ pairs, we select a random subset of size αp and output them only. It is possible to tune the value of α with respect to the threshold k desired to ensure that we do not break apart large connected components in the resulting user-user graph.

Step 2: Finding connected components on the user-user graph. Finding connected components using MapReduce needs at least $O(d)$ iterations, where d is the diameter of the graph (*i.e.*, maximum length shortest-path between any two vertices). In this approach, the input is the edge file of the user-user graph and a vertex-component mapping that maps each vertex to its "component ID", initially set to the ID of the vertex itself. In each iteration, a mapper processes each edge $e(u, v)$ in the edge file and outputs two lines $< u, i >$ and $< v, i >$ where i is the minimum component ID of vertices u and v. This output becomes the new vertex-component mapping. The process is repeated until no vertex changes its component ID. In the case that the set of vertices fits into memory, we can employ the algorithms outlined in [67] to actually find components in a constant number of passes.

6.6.3 Distributed Canopy-based Clustering

Step 1: Creating Canopies. Although our dataset comprises tens of millions of user accounts that cast votes on millions of IP addresses, the graph is sparsely connected. Because the adjacency matrix M is sparse, we choose a sparse matrix representation, M', where each row $M'(i)$ is a set of t tuples, where t is the number of IP addresses that ID i has cast votes on. M' is constructed such that, if an entry $(j, k) \in M'(i)$, then $M(i, j) = k$.

We create canopies using an inexpensive similarity metric and use the number of common IP addresses to measure similarity between two rows of M. Adapting the method by McCallum *et al.* [82], we first create an inverted index N that maps IP addresses to the set of users who vote on them. To create a new canopy, we pick a random row i from M and add it to the canopy as the first row. For each non-zero column j in $M(i)$, we find the other rows in M that also vote on IP j using the row $N(j)$. Using the inverted index allows us to ignore all rows of M and only compare with the rows from $N(j)$. We use upper and lower thresholds—T_{high} and T_{low} ($T_{high} > T_{low}$)—to measure similarity: if the similarity of a given row in M to $M(i)$ is greater than T_{high}, we *remove* the row from M and add it to the canopy. If the similarity is less than T_{high} but greater than T_{low}, we add the row to the canopy but *do not remove it from* M. This procedure explains why canopies can be overlapping: if a row is removed from M, it will not be considered for inclusion in any more canopies. In our implementation, we set T_{high} to 7 and T_{low} to 5; *i.e.*, a row is added to a canopy removed from M if it has at least 5 rows in common with the first row in the canopy, and it is also removed from M if it has at least 7 rows in common with the first row. We explain how we obtain these numbers in Section 6.7.2.

Step 2: Greedy Agglomerative Clustering. After computing canopies, we read each canopy and cluster only the rows in that canopy. To reduce the workload, we skip canopies smaller than 10 rows and canopies where the first has fewer than two non-zero columns. We use the average-linkage clustering metric to decide the similarity between rows in a canopy. If a row is a member of multiple canopies, we include that row in the clustering input for all canopies. In the final output, we include such rows as members of the largest cluster among different canopies. In a distributed setting such as MapReduce, accurate canopy clustering can be quicker than an accurate graph-based component-finding algorithm: provided the largest canopy can be clustered by a single node, agglomerative clustering of canopies can be done entirely in parallel in one step, without involving the inter-node overhead or the $O(d)$ iterations of graph-based component-finding.

Although for our dataset, the naïve implementation that compares every pair of clusters within a canopy before merging the two most similar clusters is sufficient, locality sensitive hashing (LSH) makes this step faster [36]. With LSH, we can create a hash-function on the vectors of the IPs that two users vote on, such that with high probability, two users with Jaccard coefficient above α are going to fall in the same hash-bucket. The threshold α and the probability desired will control the parameters of the hash-function. We compare pairwise all user IDs that fall within each bucket, and choose the most similar pair of IDs to merge as one cluster. Once we form a new cluster by merging two user IDs, we can repeat the process using the vector representation of the new cluster using the same hash function. This process ensures that at any step, we find the nearest neighbors with high probability.

6.6.4 Comparison: Clustering Speed

To evaluate the speed of each approach, we implemented and tested each approach on an unloaded 8-core Intel Xeon 2Ghz machine (4MB L2 cache) with 36GB of main memory running Linux 2.6.32. Both implementations were single-threaded. In addition, we tested our approximate graph-based clustering implementation on a

Table 6.3: Speed and memory consumption of our GAC and graph-based clustering implementations. Times for graph-based clustering include the multiple iterations of finding connected components, from $k = 20$ to $k = 7$. We could not measure the system time or RSS for our Hadoop implementation.

Method	*WC time*	*Sys. time*	*Max RSS*
Graph-based	86.7 min	6.8 sec	5944 MB
$\hookrightarrow$ Hadoop	14 min	N.A.	N.A
GAC	5.5 hrs	2.3 min	8221 MB
$\hookrightarrow$ Canopy formation	30.1 min	2.7 sec	3109 MB

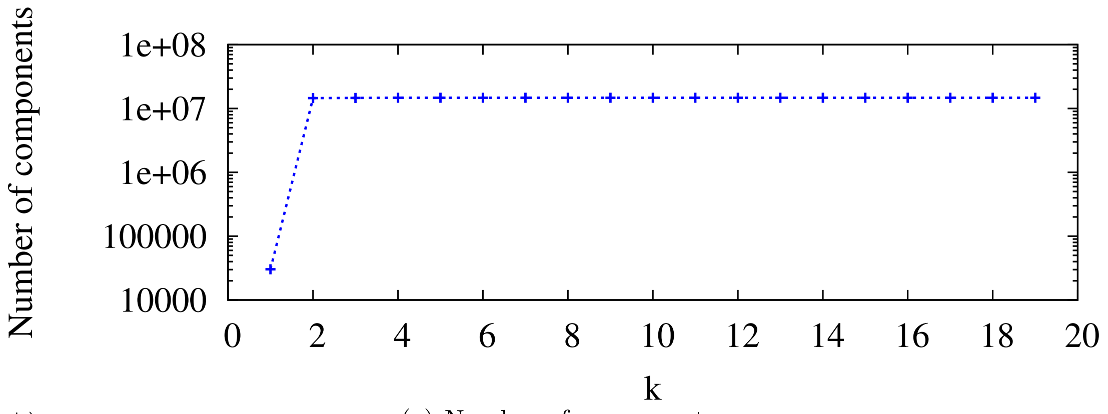
distributed cluster using the Hadoop MapReduce framework. The input was the edge file for the bipartite graph that maps users to the IPs that they vote on.

Table 6.3 presents the times taken and maximum resident set size for each method. Although GAC performs better than graph-based clustering, GAC takes longer and consumes more CPU time because of many all-pairs similarity computations between users in a canopy. The GAC phase does not require more memory consumption than the canopy formation; the extra memory usage is likely due to the memoization used to speed up our implementation. Canopy-based clustering can be easily parallellized, so with a multi-threaded application, we expect to gain a speedup proportional to the number of cores. Table 6.3 also shows the large improvement in running time for our approximate graph-based clustering algorithm on a grid infrastructure such as Hadoop [48]. Although we could not implement canopy clustering on the same infrastructure, we expect a significant speedup for that method as well.

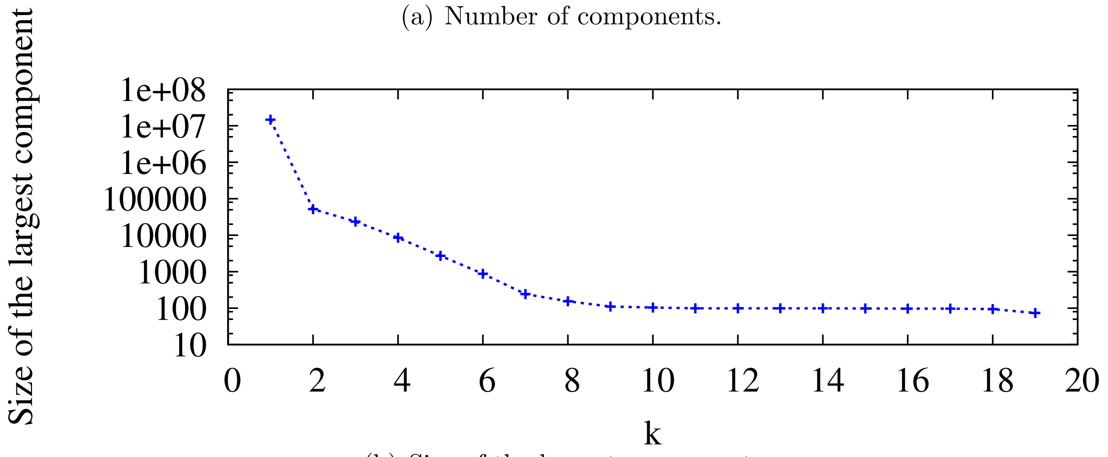
6.7 Sensitivity Analysis

In this section, we analyze the sensitivity of the detection and false positive rates for the algorithms evaluated in Section 6.5.

Main Result. The effectiveness of both techniques depends on parameter settings. Because graph-based clustering has a single parameter (the neighborhood density, k), its cluster sizes are more sensitive to the setting of k (Section 6.7, Figure 6.10).



(a) Number of components.

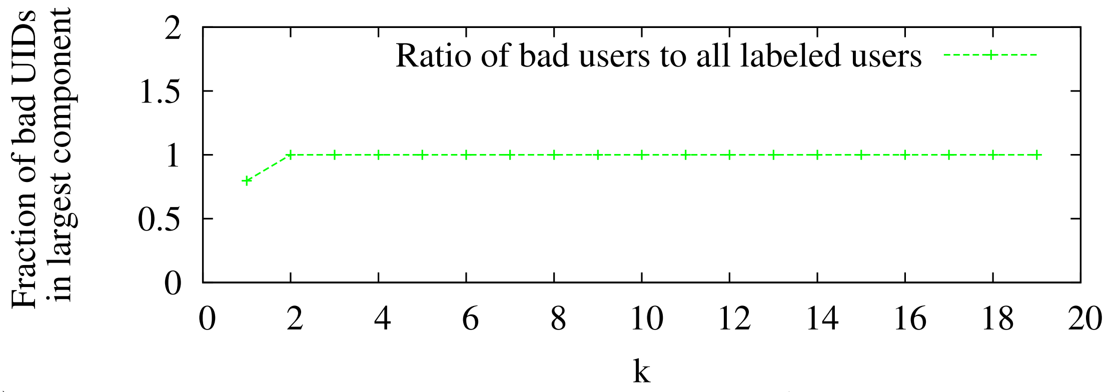


(b) Size of the largest component.

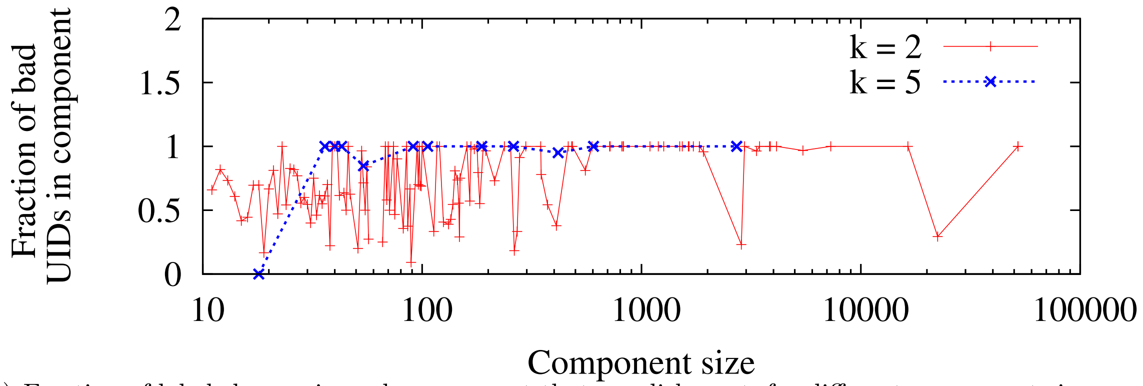
Figure 6.10: Variation of the number of components and the size of the largest component as the value of k increases from 1 through 20. The number of components do not increase much past $k = 2$, but the size of the largest component decreases exponentially from $k = 2$ to $k = 8$. We pick a value of k that gives a good tradeoff between the component size and number of components ($k = 5$).

6.7.1 Graph-Based Clustering

Our goal is to find a value of k that yields clusters that are as large as possible with few false positives. This task is challenging: selecting the smallest value of k where the largest component fragments might yield $k = 2$. However, $k = 2$ may not yield large components containing only bots with no false positives, because to be in a connected component at $k = 2$, a legitimate user only needs to vote "not spam" on two IPs that a voting bot also votes on as "not spam"; this event may occur either



(a) Fraction of fraudulent voters in *largest* component at various k. A value of 1 indicates zero false positives.



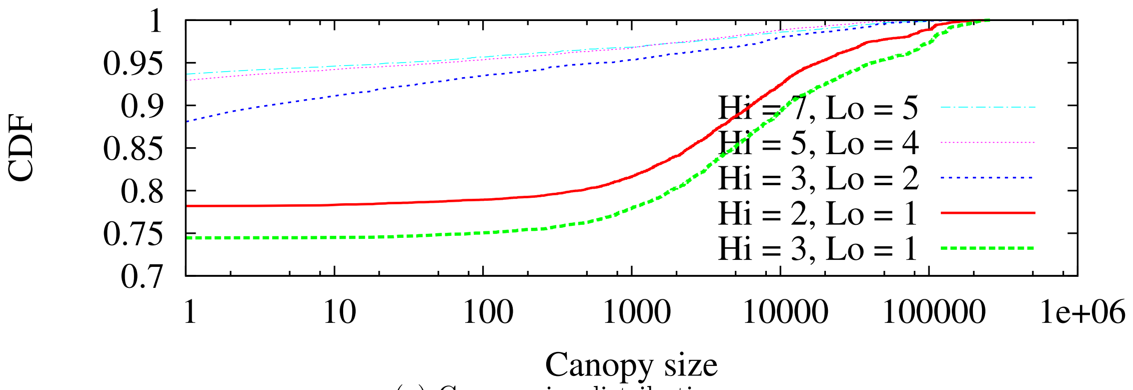
(b) Fraction of labeled users in each component that are dishonest, for different component sizes and at different values of k. Low values on the y-axis indicate a higher false positive rate.

Figure 6.11: (a) Fraction of "bad" users in the largest component as k is varied; and (b) the fraction of "bad" users as component size varies for two specific values of k. The largest component only contains users labeled "bad" above $k = 2$, but there is higher variability in the false positive rate for smaller-sized components at $k = 2$ than at $k = 5$.

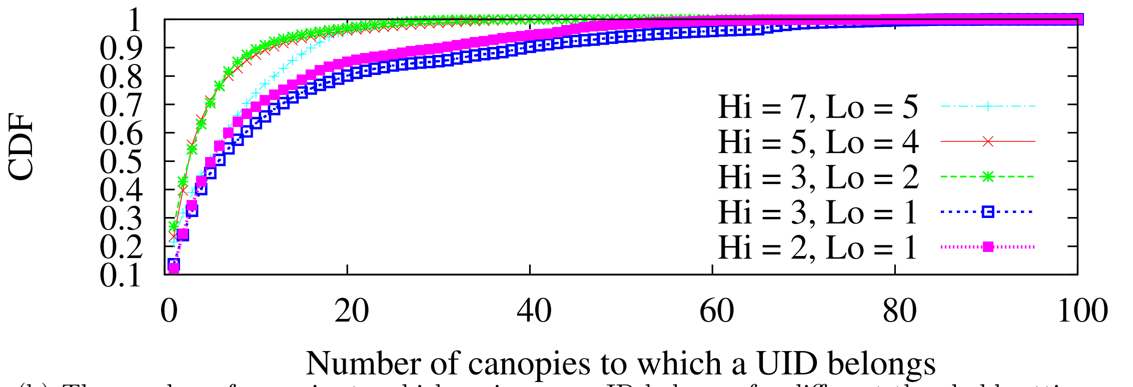
if a user votes "not spam" by accident or because the voted-on IPs were re-assigned during our data timeframe due to DHCP reassignment. Thus, instead of choosing the stopping value of k only using the decrease in size of the largest component, we stop when a large fraction of labeled users in the largest components are known dishonest voters.

Figure 6.10 shows the number of components and the size of the largest component as k increases from 1 to 19. As Figure 10(a) shows that at $k = 1$, almost all nodes are in a giant component that includes nearly all nodes in the user-user graph, but just by increasing k to 2, the giant component fragments from over 14.6 million nodes to just 52,006 nodes, and the number of components increases from 30,225 to over 14.5 million. Figure 10(b) highlights the decrease in the size of the largest component, echoing the structure of the Web pages-vs.-ads bipartite graph in Kumar *et al.*'s work [75].[5]

Even for low values of k, the largest component consists mostly of "bad" users. Figure 11(a) shows how the fraction of users labeled as fraudulent in the largest component varies as a fraction of all labeled users, for various values of k. Even at $k = 2$, the largest component has no users labeled "good" (*i.e.*, no false positives). This characteristic holds as k increases: there are no false positive "good" users in the largest component at any value of k greater than two. However, the *minimum component size* above which there are no false positives is dependent on k. We examine the size of the largest component and the fraction of dishonest voters in each component (among labeled users). Figure 11(b) shows the number of false positives in each component, rank-ordered by the size of the component, for $k = 2$ and $k = 5$. Smaller components for small values of k often include many "good" users; at $k = 2$, even the second-largest component contains more than half good users. As we increase k to 5, the good-user portion of the large component fragments, resulting in smaller components with even fewer false positives, which is why we picked this threshold for our evaluation.



(a) Canopy size distribution.



(b) The number of canopies to which a given user ID belongs, for different threshold settings.

Figure 6.12: Canopy characteristics for various upper and lower thresholds, T_{high} and T_{low}.

6.7.2 Canopy-Based Clustering

Choosing thresholds for canopy formation. The first step in canopy-based agglomerative clustering is canopy formation, which is parameterized by the thresholds T_{high} and T_{low} (Section 6.6.3). These thresholds control the extent to which the data is partitioned and the extent to which canopies overlap with one another. Because we apply canopy clustering to reduce the size of our input dataset, we must pick values of

[5]This work illustrates the similarity of Web pages based on the number of advertisements they share; they found that sharing even 5 advertisements did not say much about the connection between Web pages, but six or more shared advertisements implied a stronger notion of similarity. Similarly, we find that two users in the same component at $k = 2$ or $k = 3$ are not necessarily similar but connections at a slightly higher value of $k = 6$ or $k = 7$ implies high similarity.

Table 6.4: Sensitivity of the detection and false positive rates to the choice of the similarity threshold. We chose 0.85 (highlighted).

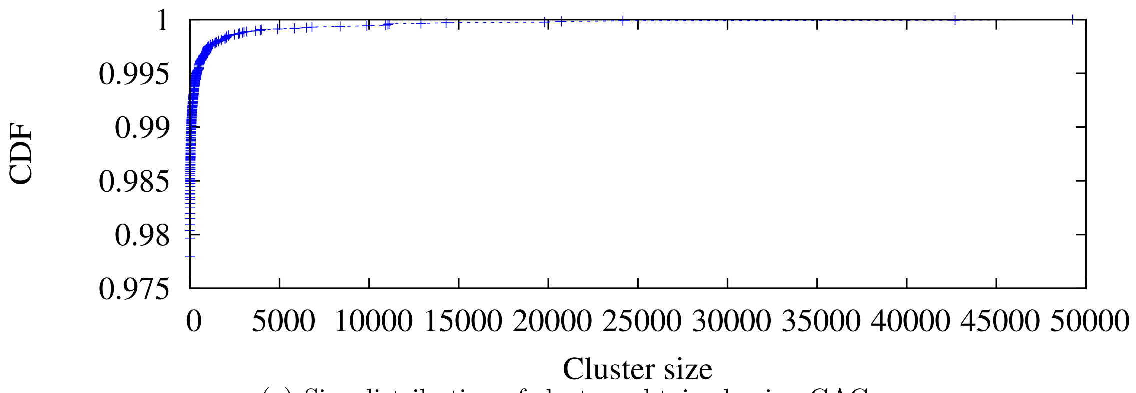
Sim. Threshold	*Detection Rate*	*FP rate*
0.90	8.74%	0.14%
0.87	9.01%	0.15%
0.85	**10.24%**	**0.172%**
0.82	15.52%	0.217%
0.78	17.52%	0.244%
0.76	19.24%	0.35%
0.74	21.70%	0.328%
0.72	23.29%	0.499%

T_{high} and T_{low} such that: (1) the average size of canopies are reduced, (2) the overlap between canopies is reduced, and (3) the total number of canopies are reduced. Low values of T_{high} reduce overlap, and high values of T_{low} decrease the size of canopies. However, if both T_{high} and T_{low} are too large, all but highly similar rows will be in non-singleton canopies.

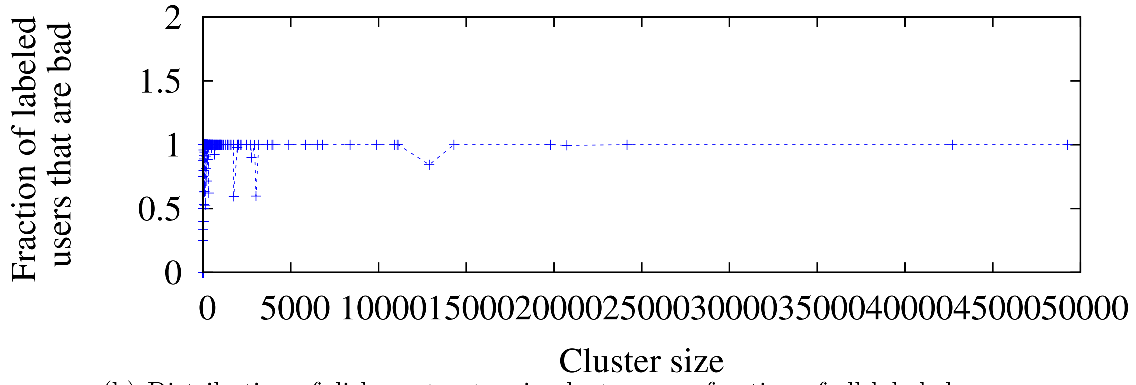
Figure 12(a) plots the size distribution of canopies on varying T_{high} and T_{low}, and Figure 12(b) plots the CDF of the user IDs which are mapped onto multiple canopies. These figures show that setting $T_{high} = 7$ and $T_{low} = 5$ partitions the users into distinct canopies into a few small canopies with minimal overlap.

Choosing a threshold for the Jaccard Metric. We cluster each canopy using average-linkage similarity (Section 6.4.3). For each canopy, GAC iteratively performs all-pairs similarity computation and merges the most similar clusters if their Jaccard similarity exceeds a similarity threshold. Table 6.4 shows how the detection rate and false positive rates change for other settings of the similarity threshold. A similarity threshold of 0.85 yields a high detection rate and a low false positive rate.

Figure 13(a) shows the size distribution of the clusters we obtained. More than 99% of clusters are singletons (*i.e.*, likely legitimate users). Figure 13(b) shows the distribution of dishonest voters for various cluster sizes, presented as a fraction of



(a) Size distribution of clusters obtained using GAC



(b) Distribution of dishonest voters in clusters as a fraction of all labeled users.

Figure 6.13: Analysis of Greedy Agglomerative Clustering: (a) shows that over 99% of clusters are singletons, and (b) shows that in the clustering output at our chosen parameter settings, most clusters over size 2 (with very few exceptions, as explained in text) have only users that are labeled "bad".

labeled users in the cluster. All large clusters except for one have almost no false positives. The exception—a cluster of 12,890 users—has 517 users labeled "good" 2,776 users labeled "bad". Considering that all of these false positives fall into a single cluster, these users are likely compromised users that were mislabeled.

6.8 Discussion

We present the results of identifying voting bots using a complementary dataset, where we map user accounts to the login IP address of the user who cast a not-spam vote (*i.e.*, the IP address of the host from which the user logged in to the Web mail

Table 6.5: Results of applying graph-based clustering on login IP data, and extracting the largest 4 components. Because this dataset has different characteristics than our primary 4-month dataset, we found that a neighborhood density of $k = 8$ gave the best results.

Users	IPs	Validated as Voting Bots	NS Votes
102991	56	102991 (100%)	6.11m
69710	32	64629 (92.7%)	5.14m
59077	39	26592 (45%)	2.58m
49045	65	49045 (100%)	4.5m

service). We also discuss potential limitations of our approach and our evaluation.

Clustering Using Login IPs. We have an additional dataset from May–June 2009 that has the login IP address of the user (recall that the dataset in Section 6.5.1 has the IP address of the sender of the email on which the user cast a vote). We expect that the IP addresses from which a dishonest NS-voting user logs in should also follow the model of Section 6.3. Table 6.5 summarizes the results of graph-based clustering applied to the graph that maps user IDs to these login-IPs. Indeed, a large number of IP addresses shared a given bot account (specifically, larger on average than the number of IP addresses a bot account votes *on*); hence, a higher neighborhood density of $k = 8$ yields the best results. As expected, most users in the largest components were identified as bot-controlled. Certain components have significant fractions of accounts not yet labeled (*e.g.*, the third-largest component has 55% accounts not yet labeled), which represents significant savings in terms of the number of fraudulent NS votes that can be prevented. Because we only had access to this data for a limited time, we were unable to compare the results of graph-based clustering with canopy-based clustering.

Low Detection Rate of Known Dishonest Voters. Although our 10.24% detection rate may appear low, this number amounts to nearly 26,000 fraudulent voters that were previously undetected by other methods, with only 0.17% false positives. As the sensitivity analysis in Table 6.4 illustrates, if the operators find a slightly higher false positive rate of 0.5% acceptable, they can detect up to 23.29% of the labeled bad users. Another reason for this seemingly low detection rate is that many users labeled "bad" in the set of labeled users may have had the bulk of their NS votes before or after the timeframe of our data set; such users will not have enough NS voting activity to cluster well with other heavy NS voters. Recall that our actual *detection*—defined to be the number of user accounts that were in clusters above a particular size—is much larger: we detected 1.13 million accounts belonging to clusters above size 109, which amounts to nearly 10% of *all* not-spam voters.

As the false positive rate analysis in Figure 13(b) shows, large clusters have *zero false positives* (with one exception that is likely due to mislabeling). Because these clusters likely consist of only bot accounts, the actual number of bot accounts detected by our technique will be much greater. For example, the largest cluster in Figure 13(b) alone has nearly 50,000 users, all of which are likely bots.

Dataset Limitations. Because the data that we used in our study was not timestamps, we could not analyze datasets on smaller timeframes. However, our analysis using login IPs shows that smaller timescales also work to identify voting bot accounts. Regardless, our approach can be used for day-to-day detection of bots: because both clustering methods complete in a few hours, an operator could run the analysis daily on a sliding historical window of voting data.

Using Voting Clusters for Real-time Detection. From clusters of dishonest voting accounts, one can go back to the original user-IP graph to retrieve the IP addresses shared by users in the cluster. The IPs and user accounts corresponding to large clusters can then be put on a "watch list", and any new users or IPs that map to a watched user or IP can be investigated before they cause much damage. A second avenue for using our approach in real-time filtering is to combine information obtained using clustering to improve other classifiers. Clustering extracts macroscopic patterns from the activity graph of voting. A traditional *supervised* classifier for voting would use features at the level of each user (*e.g.*, the user account's age, it's reputation, etc.) and might miss accounts that can be discovered by clustering. As an example, consider a reputable user account that becomes compromised and used for dishonest voting. The traditional classifier will likely continue to classify the account as "good", but our clustering approach could instead discover that the account falls into large clusters and raise an alert.

6.9 Summary

Web mail providers rely heavily on user votes to identify spam, so preserving the integrity of user voting is crucial. We have studied a new attack on Web mail systems that we call a *vote gaming attack*, whereby spammers use compromised Web mail accounts to thwart Web mail operators' attempts to identify spam based on user votes. Using four months of voting data from a large Web mail provider, we found that vote gaming attacks are prevalent in today's Web mail voting systems. As a first step towards defending against these attacks, we have developed and implemented a clustering-based detection method to identify fraudulent voters. Our method identifies 1.1 million dishonest voters, over the course of several months (most of whom were previously undetected), while yielding almost no false positives. The techniques presented in this chapter are an important step in stemming the tide of this new class of attacks and are already being used in production as part of a large Web mail provider's techniques to detect fraudulent votes. We believe that these techniques may also be applicable to other online Web forums where bots perform vote gaming, such as user-generated content sites or online polls. We intend to explore the applicability of our methods to these other settings as part of our future work.

CHAPTER 7

TOOLS: SPAMSPOTTER AND SPAMLOJACK

7.1 Introduction

This chapter presents the design, implementation, evaluation, and initial deployment of two tools that resulted from this dissertation: SpamSpotter and SpamLoJack.

SpamSpotter. First, we present SpamSpotter, an open, large-scale, real-time reputation system for filtering spam. Existing blacklists (*e.g.*, SpamHaus [122]) are based on static lists; as we demonstrated in Chapters 4 and 5, they have come under attack, and cannot keep pace with spammers' ability to send spam from "fresh" IP addresses. The inherent flaw of IP blacklists is their reliance on IP addresses as persistent identifiers. Despite new spammer tactics and attacks presented in this dissertation that erode the effectiveness of IP reputation, IP blacklists continue to be the only form of practical network-level filtering accessible to users and mail server administrators. Although there have been techniques such as SpamTracker, SNARE [52], network-aware clusters [97], etc. that can identify spammers using network-level features *other than* their specific IP addresses, these techniques tend to remain in the fringes: most such research is conducted using offline logs, and seldom see practical use.

We designed SpamSpotter to bridge the vast gap between the technology that exists in spam filtering research and the technology that is actually used for spam filtering in the industry. Instead of trying to build a new algorithm or technique to classify spammers, we concentrate on building a scalable framework that designers of new spam filtering algorithms can leverage to easily build and deploy their algorithms. In addition to making novel spam-filtering techniques available to a wider audience, SpamSpotter helps researchers better tune their methods: the real-time performance of an algorithm on a varied dataset from many recipient domains may be very different from its performance on an offline dataset from a single domain, and we expect SpamSpotter deployment will enable the development of faster and more accurate network-level spam filters.

SpamSpotter currently incorporates three network-level spam filtering techniques: SpamTracker (Chapter 5), SNARE [52], and Trinity [13]. SpamSpotter's framework allows combining various spam-filtering algorithms and deploying and testing new email classification algorithms. We tackle significant design challenges involving scalability, speed, and accuracy. We have evaluated the performance and accuracy of SpamSpotter using traces from a spam-appliance vendor and a large email-hosting provider. Due to its familiar DNSBL-like query interface, network administrators can easily incorporate SpamSpotter's blacklist into their spam filtering systems in the same way that they would use any other static blacklist, with only minor configuration changes. This chapter presents the motivation and design behind SpamSpotter; a full description and detailed evaluation is available in our technical report [109].

SpamLoJack. Second, we present SpamLoJack, a tool that identifies potential BGP prefix hijacks in real-time using a joint feed of BGP updates combined with a feed of spam from a spam sinkhole. Spam from hijacked BGP prefixes is hard to defend against using IP-based or behavioral approaches: if a miscreant is able to hijack a large IP prefix, he may be able to send each spam message from a new IP address, thwarting any potential IP-based or behavioral detection approach.

Because spam from hijacked prefixes continues to occur [6], we leverage our co-located collection setups—the spam trap to collect spam and the BGP speaker to collect routing updates—to identify spam from potentially hijacked prefixes. The key idea in our approach is to continuously monitor incoming BGP updates for route changes, and to maintain the history of the various Autonomous Systems that announce each prefix. When our spam sinkhole receives a message from an IP address in a "suspicious" prefix—*e.g.*, a prefix whose origin AS does not match IANA allocation records—we log an alert that includes information such as the route corresponding to the received spam, the AS that originated the route, the AS that legitimately owns the prefix according to records, and historical and statistical information concerning the prefix. With incidents of spam from hijacked prefixes being reported regularly, we believe this service will be of use to many network operators both for spam filtering and to quickly identify hijacked prefixes. We make SpamLoJack available through the same DNSBL interface provided by SpamSpotter; we believe this interface makes it easy for operators to integrate SpamLoJack into their production systems. We plan to design a Web-based front-end to SpamLoJack in future work.

We acknowledge that SpamLoJack cannot possibly detect all hijacked prefixes; in fact, a remote vantage point such as SpamLoJack's collection servers cannot even reliably determine whether the change of a route or even an origin AS is malicious: Internet link failures regularly cause route changes, and multiple-origin ASes may legitimately have different origin ASes. Despite these issues, SpamLoJack still provides valuable intelligence to IP prefix owners and mail server operators: from the perspective of a AS such as our collection server, paths and origin ASes for prefixes are stable, and any change in those might indicate a likely hijack. In addition, because SpamLoJack performs joint analysis only on the arrival of a spam email—a clear sign of malicious activity—SpamLoJack is more likely to detect truly malicious prefix hijacks than other approaches such as PHAS [76].

The rest of this chapter is organized as follows. The design of SpamSpotter is presented in Section 7.2, and its implementation in Section 7.3 (for a full evaluation of SpamSpotter's performance and and discussion of its use-cases, we refer the reader to the SpamSpotter technical report [109]). Section 7.5 motivates the need for SpamLoJack, and Section 7.6 presents the design and implementation of SpamLoJack. Section 7.7 concludes.

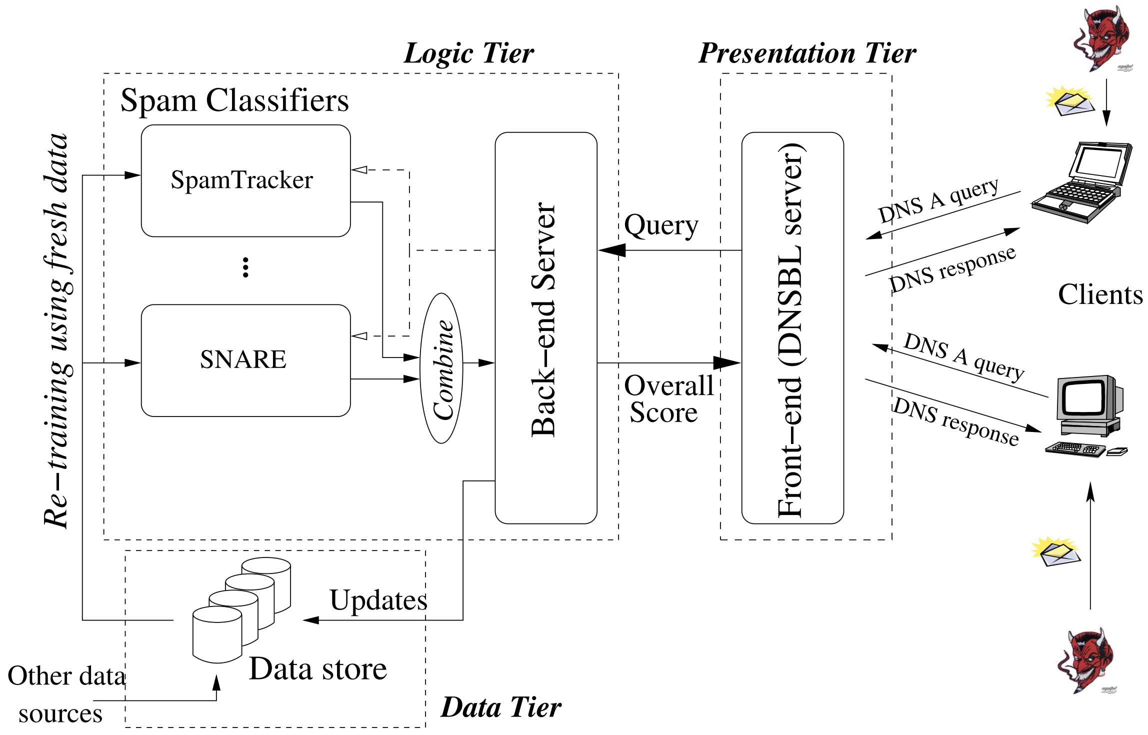


Figure 7.1: High-level architecture of a deployment of SpamSpotter. The data store is updated using incoming queries and external data sources.

7.2 SpamSpotter Design

This section presents the design of SpamSpotter. Figure 7.1 shows SpamSpotter's three-tier architecture, which has a front-end interface to clients (*i.e.*, mail servers) and back-end interfaces to data sources and various network-level classification algorithms. We discuss each component in turn. We have designed SpamSpotter so that the presentation tier looks like a standard DNSBL-based interface. The rest of this section presents each component of SpamSpotter in detail.

7.2.1 Front-End Interface: Augmented DNSBL

The client-facing interface to SpamSpotter is a DNS-based lookup interface similar to those offered by popular IP blacklists such as Spamhaus [122] and SpamCop [121]. DNS-based IP blacklists (DNSBLs) operate using a "hack" on the DNS message format: the domain names do not correspond to a real domain name, but instead encode the IP address of an email sender. Responses contain a score that is embedded within a 32-bit IPv4 address. For example, to query an IP address such as `12.34.56.78` at SpamHaus, the client issues a DNS "A" record query for `78.56.34.12.zen.spamhaus.org`. At the SpamHaus DNSBL, the IP address is compared with blacklist entries (reversing the order of quads in the queried IP address helps here); if the IP is blacklisted, the server returns an IP address such as `127.0.0.2`; otherwise, it returns `NXDOMAIN`. Typically, only the last octet of the returned IP address is meaningful to the client (*e.g.*, for SpamHaus, '2' refers to unsolicited commercial email senders, '4–8' refer to exploit senders).

1. Use DNSBL queries as a data source. Mail servers and end-hosts typically issue DNSBL queries immediately after an email is received. Most DNSBL providers discard this data, but SpamSpotter uses these queries as a real-time data feed that contains the IP addresses of the sender and recipient and the time at which the email was received; unsupervised algorithms such as SpamTracker can construct clusters of email senders from unlabeled input data; using labels on a small fraction of the data or using domain knowledge (*e.g.*, only spam senders form large clusters based on sending behavior), this data can be used to construct a behavioral classifier. In the process of constructing this classifier, SpamSpotter automatically "summarizes" input data thereby reducing its volume. SpamSpotter may still be vulnerable to malicious DNS query sequences designed to disrupt classification; please see our technical report [109] for a full description of attack defenses.

Using query data as a source of information about email activity poses a few other challenges. (1) The data is not labelled into spam senders and legitimate senders (if it were, DNSBL clients would not need to issue queries in the first place), and thus, the DNSBL provider cannot use query data to augment their blacklist; (2) queries to DNSBLs are typically not authenticated, and if query data was used to update blacklists, the blacklists would be susceptible to bias through targeted queries by malicious entities; (3) query volume is massive, and even leading DNSBLs today do not have a method to summarize or capture interesting information in incoming queries. We address these challenges using unsupervised learning, using only a subset of queries from certain trusted mail recipients to update blacklists, and through implementation decisions that allow SpamSpotter to scale to large query volumes.

2. Embed additional metadata into queries and responses. Although DNS message size limits are large enough to allow senders to encode many attributes of emails in a DNSBL query, DNSBL queries today contain only the IP address of the email sender. To make it easy for existing mail servers to use SpamSpotter exactly as they would use a DNSBL, SpamSpotter supports the conventional DNSBL format of prepending the sender's IP address to the DNSBL's domain name (*i.e.*, `sender-ip.dnsbl-domain-name`). To support behavioral filtering algorithms that use other network-level features as input, SpamSpotter also allows other encoding attributes of emails such as the target domain of the email, the size of the email body, or the BGP route used advertised by the sender, etc. SpamSpotter currently supports the following attributes, in addition to the sender's IP address:

- `rcpt_ip`: this attribute allows SpamSpotter to construct features such as geodesic distance [52] between sender and receiver, time-of-day at the sender and receiver, etc. The IP address issuing a DNSBL query is often the server that receives the email, so this field may be inferred from the DNS request packet and need not be explicitly specified.
- `rcpt_domain`: this format allows SpamSpotter to also enable SpamTracker classification by computing features based on recipient (*i.e.*, target) domains.
- `msg_size`: SNARE [52] also includes a classifier based on message size; prefixing the message size to the DNSBL query allows SpamSpotter to also use the message size classifier in computing the response.

- `nrcpt`: The number of recipients each message receives is yet another feature used by SNARE; this value can also be prefixed to a DNSBL query to SpamSpotter.

Any of these fields can be prepended to a single DNSBL query; the accuracy of classification increases with the level of detail of the query. Each input query is parsed by SpamSpotter's DNSBL server, and the fields of the query are marshalled into a message that is forwarded to the back-end server.

SpamSpotter's DNSBL responses include a score as a fixed-point number, encoded as an IPv4 address and returned as a DNS "A" response (DNSBLs today return scores as an IP address whose first three octets are fixed at 127.0.0). The response can also include richer information, such as a numeric score for each lookup, which permits clients to enforce fine-grained, customized filtering rules. Currently, the responses issued by SpamSpotter's DNSBL server use the last three octets of an IPv4 address. The first octet is always 127. The second octet is 0 if the score is positive, and 1 if it is negative. The third octet is the integer portion of the score; the maximum possible score is 255. The fourth octet is the decimal part of the score to two decimal places. For example, a score of -310.279 would be represented by the IP address 127.1.255.28. SpamSpotter's precision allows various spam filtering tools to filter messages with varying levels of confidence. SpamSpotter can also return scores without encoding them, using DNS "TXT" records, but for compatibility with existing DNS lookup tools (*e.g.*, `dig`) and lookup systems that understand only the dotted-quad notation, SpamSpotter outputs the score in a format that is easy to read as an IPv4 address.

7.2.2 Logic: Spam Classifiers

We built the logic tier of SpamSpotter such that individual algorithm designers do not need to worry about input or output formatting—merely the specifics of their algorithm. We assume that each classification algorithm accepts a chunk of input data, builds a classifier, and makes the classifier available for lookup. When a query arrives from the DNSBL, the back-end sends the relevant metadata to each classification algorithm. Each algorithm responds with a score normalized between 0 and 1, where higher values indicate greater likelihood of spam according to an algorithm's classifier. The back-end server may also combine these responses to obtain a single real value (*e.g.*, using weighted averaging). The back-end also makes all decisions about the amount of data to provide to each algorithm, the spam-ham thresholds of scores returned by the algorithms, when to trigger re-training of each algorithm, etc.

SpamSpotter's back-end already incorporates three statistical spam-filtering algorithms; we present a brief overview of each algorithm below. To demonstrate generality, we show how SpamSpotter can incorporate both unsupervised and supervised learning algorithms, as well as simple heuristics. We chose these algorithms due to our familiarity with their implementations or their relative ease of implementation, but SpamSpotter's interfaces are general and could presumably incorporate and synthesize additional behavior-based classification algorithms (*e.g.*, [10]).

1. Unsupervised Learning: SpamTracker [108]. SpamTracker constructs clusters of senders who have similar sending patterns (*i.e.*, set of domains targeted within a short period of time). Our hypothesis is that large-scale spamming occurs in an organized fashion, with many infected machines (bots) acting in concert to send spam to a set of email addresses. The behavior of such bots, when observed across multiple large domains, will appear as a single "fingerprint". Legitimate email senders, on the other hand, do not act in a coordinated fashion, and different legitimate senders will usually not have similar patterns. To look up a new IP address, SpamTracker constructs a fingerprint for the IP address and compare it with patterns obtained from the clustering process. The score is the similarity of the new IP address's recent sending behavior to the best matching pattern. SpamTracker is unsupervised: it can form clusters of sending patterns without requiring labels on the data; a good match to a pattern formed by a large cluster of IP addresses indicates that the queried IP address is a spammer. SpamTracker's accuracy can also be improved if even a small fraction of input data has reliable labels.

2. Supervised Learning: SNARE [52]. SpamTracker has some shortcomings. First, the feature that SpamTracker uses is evadable; second, it requires coordination and aggregation of received email data across many recipient domains. To counter these weaknesses, we designed *SNARE (Spatio-Temporal Automated Reputation Engine)*, which uses supervised learning to construct a classifier based on robust yet lightweight network-level features that can often be gleaned from a single packet, and with minimal communication with other recipient domains. In this work, we identify a collection of ten lightweight network-level features that together can be used as inputs to a robust, lightweight sender reputation system based on network-level classifiers. Table 7.1 summarizes these features and categorizes them into three: those that can be determined from a single packet, those that require inspecting the message or SMTP header, and those that require aggregation over time. SNARE gathers these features from the DNS queries from mail servers, as described in Section 7.2.1. SpamSpotter trains using a labelled sample of this data.

3. Heuristic: Trinity [13]. Trinity is a heuristic-based unsupervised algorithm developed by Brodsky *et al.* The algorithm maintains a database of the number of emails each IP address sends in a short time-period. The hypothesis is that, to maximize their "productivity", spam bots send a large amount of email (to many domains) in a short period of time; thus, a system that collates the number of emails each IP address sends to a large set of domains can be queried to determine whether an IP address is a potential spam sender. Trinity discretizes the count over four 15-minute intervals and "ages" the counters every 15 minutes; in our implementation,

Table 7.1: Features used for network-level sender reputation in SNARE.

Feature	*Description*
	Single-Packet Features
AS number	the autonomous system number of the sender
Neighborhood density	number of IP addresses in neighborhood also sending email
Sender-receiver distance	geographic distance between sending IP and receiving mail server
Port status	open service ports on sender IP
Time of day	ratio of spam/ham for hour-of-day
	Single-Message Features
Recipients	number of recipients in “To” field
Size	message size in bytes
	Aggregate Features (24+ Hour History)
Average length	average message length
Average density	average neighborhood density
Std. density	standard deviation of density

we use the same time-periods.

Synthesis. SpamSpotter can not only run each classification algorithm independently, but it can also combine the outputs from a combination of supervised (*e.g.*, SNARE) and unsupervised (*e.g.*, SpamTracker) learning algorithms to produce a more accurate classifier that takes input from each classification algorithm. We explore two ways of synthesizing the outputs from multiple classifiers:

1. Incorporate the SpamTracker score as a feature in a supervised learning algorithm like SNARE.
2. To reduce false positives, only consider a sender a spammer if some number of classification algorithms have classified the sender as a spammer.

We evaluate synthesis techniques in the accompanying tech report, and in future work in Section 8.4.

Table 7.2: Set of attributes for each message in SpamSpotter's data store. Italicized entries may not be available for all data feeds. Fields such as *rblstatus* and *result* are updated as new information becomes available. There may be multiple *score* fields for each algorithm.

Field	*Description*
feed_name	The name of the data feed where this record was obtained
timestamp	UNIX timestamp
target_domain	The domain name in the 'RCPT TO:' SMTP command
sender_ip	IP address of the email sender
rcpt_ip	IP of the email recipient
result	Email classification (*i.e.*, spam or ham)
score	A normalized [0, 1] score for the message; higher values indicate greater likelihood of spam.
msg_size	The size of the email in bytes
rblstatus	The listing status of the sender in multiple blacklists (encoded as a bit-vector)

7.2.3 Data: Aggregation and Storage

SpamSpotter's *data tier* consists of the data store and access routines that allow the back-end server to fetch data from, and insert data into the data store. The back-end expects one record per email, with all records sorted by arrival time of the email. The data store integrates many types of email feeds, including data from spam traps, data from a large mail service provider who manages hundreds of domains, and a spam appliance vendor whose appliances observe over 300 million emails per day. It also logs all incoming queries and the scores returned by each algorithm. These logs serve two purposes: (1) they can be used as future training data for the algorithms, and (2) they can be used to evaluate the performance and false positive rates of each algorithm when ground truth data (*e.g.*, human reports a mail as spam, highly accurate IP-based blacklists begins listing the sender IP address, etc.) eventually become available, which can help assess the need for re-training. These logs, or the output of each classifier, can also ultimately be shared with back-ends run by other parties. Table 7.2 shows set of columns for the data store. The data store is typically

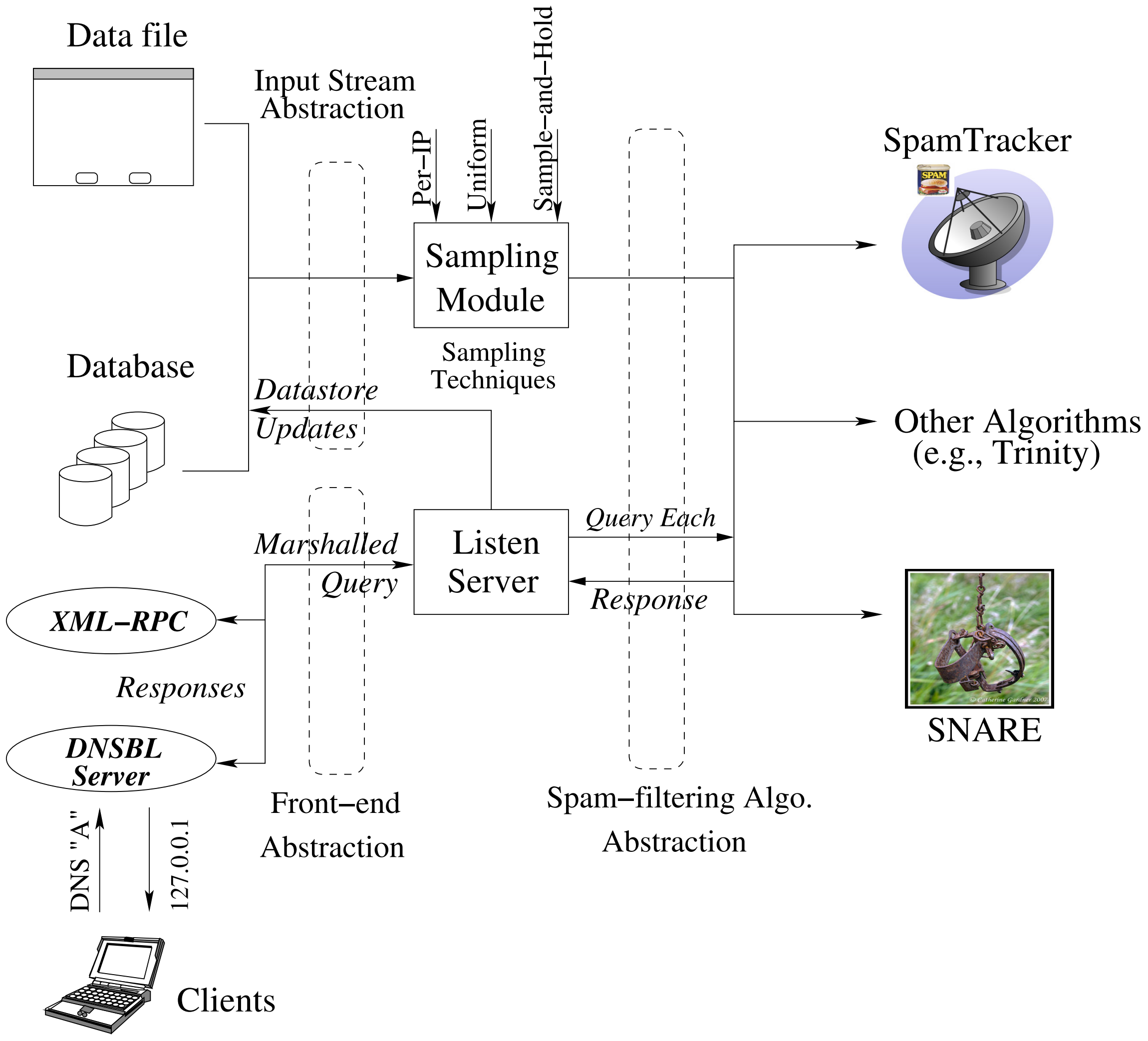


Figure 7.2: SpamSpotter implementation.

a database, but SpamSpotter can interpret many data representations (*e.g.*, flat file, XML feed, etc.).

7.3 SpamSpotter: Implementation

This section describes the implementation of SpamSpotter. Figure 7.2 shows the various components of SpamSpotter and the interfaces between them. We briefly describe how we implement each of the three components of our design.

Table 7.3: DNS queries that show how to piggyback email features on the DNSBL query string. SpamSpotter can also support more expressive responses using DNS TXT records.

DNS Query	*Response*	*Explanation*
5.6.7.8.-.gmail.com.-.4.3.2.1.rbl.gtnoise.net	127.0.1.97	The IP address 1.2.3.4 sent email to the domain `gmail.com` which was received by the server IP address 5.6.7.8. The returned score is +1.97.
5000.-.3.-.gmail.com.-.4.3.2.1.rbl.gtnoise.net	127.1.4.50	As before, but the receiver IP is inferred from IP address of the machine sending the query to our server. This query also specifies two extra features: the message size (5000 bytes), and the number of recipients for the message (3). The score returned is -4.50.

7.3.1 Implementation Overview

Front-end: Augmented DNSBL and client plugins. Our prototype uses a modified version of *rbldnsd* [112], an open-source DNSBL server, to serve DNSBL queries using SpamSpotter as a back-end (instead of a static blacklist). *rbldnsd* marshalls metadata in the queries it receives and forwards it to the back-end server, and returns the score to the client embedded within an IP address. This interface requires minimal changes to client-side software: for SpamAssassin [120], the popular client-end spam filter, SpamSpotter-style DNS queries can be incorporated either with plugins or with only two lines of edits[1]. The back-end server communicates to the DNSBL server using an interface that can be easily adapted for other forms of query such as XML-RPC or HTTP. Our installation can be queried using DNS as shown in Table 7.3.

Logic: Integrating classification algorithms. The back-end server controls all algorithms in SpamSpotter. When executed, the server first spawns separate threads

[1]At line 255 in the file Plugin/DNSEval.pm for SpamAssassin v3.2.5.

to begin training each supported algorithm with input data. Each algorithm further spawns one or more low-priority threads to perform training while waiting to serve lookup requests. Input data is multiplexed from many sources (*e.g.*, a DB, a file, etc.) and can be passed through sampling modules before being sent to an algorithm. Finally, the server also spawns a pool of worker threads to quickly process incoming reputation lookup queries. For each query, the back-end server performs the following operations: (1) Un-marshall the query parameters, re-pack them in another format, and forward the queries to each algorithm in parallel; (2) Collect responses, perform any operations on the scores, and forward a single real value to the front-end; (3) Log the query and computed score to the data store for use in later stages of training, or for performance validation.

The SpamSpotter prototype and patches are about 10,000 lines of C and C++ code. SpamTracker is implemented using EigenCluster [16]. SNARE is implemented using an off-the-shelf Decision Tree classifier, C4.5 [99]; we are currently upgrading the decision-tree based classifier to a method with better predictive power, such as Boosted Decision Trees [34] or RuleFit [35]. Integrating new behavioral classification algorithms into SpamSpotter is simple: each algorithm is implemented according to an interface that provides it a `stream` object that can be read to retrieve message features (in order of message arrival). In addition, each algorithm must define one `retrain` function callback that is executed periodically by the back-end server, and one `lookup` function – called by the back-end server with the contents of incoming DNS queries – that returns a score for the DNS query attributes.

Data Store. We use a MySQL database for data from the mail service provider and the spam-collection facility ("Spamtrap"), and a formatted flat text file for the spam appliance vendor's data. Because each file in the latter data source is massive (65 GB or greater), it is pointless to store the file in a database. Both the database and flat files are multiplexed (according to message arrival time) and made accessible to

Table 7.4: Summary of evaluation results for SpamSpotter.

Goal	*Result*
Speed	*(Training Time)* Training typically requires about 10 minutes, and no more than one hour; training time is linear in the input size.
	(Memory Requirements) No more than 1.5 GB of memory for training; memory requirements are linear in the input size.
Accuracy	*(Detection)* *SNARE*: $\approx$ 98.6% detection rate for $\approx$ 3.4% false positive rate; *Trinity*: $\approx$ 99.5% with $\approx$ 0.5% false positive rate.
Scale	*(Federation)* Significant reduction in false positives with low detection rate penalty
	(Sampling) Accurate classification requires as little as a few thousand samples of spam and legitimate mail.
Deployability	*(Response Time)* SpamSpotter returns reputation scores in about 10 ms or less on a local network, even for workloads that are equivalent to today's blacklist mirrors.

algorithms through a simple interface: a `read` function that reads the next message record in order of message arrival time, and a `seek` function that sets the data position indicator (*e.g.*, in order to start reading data from a different time-period).

7.3.2 Performance

Table 7.4 summarizes our evaluation of SpamSpotter. Please refer to our technical report for a full description of the evaluation [109].

7.4 Limitations of SpamSpotter

The chief limitation of SpamSpotter is in obtaining high-quality, labeled data that can be used to train various network-level algorithms. Obtaining spam data is relatively easy; we use the IP addresses and other network-level features of emails that arrive at our spam trap. Obtaining legitimate email, however, is more difficult: we must either rely on another spam filtering method to determine if an email is legitimate, or rely on human-labeled legitimate email. The shortcoming on relying on another approach is that errors can propagate due to that method's false positives and negatives; as we showed in Chapter 5, some of the 15% of misclassified spam from an email service provider were correctly classified by SpamTracker, but we had no way to verify whether all email classified by SpamTracker were indeed spammers.

The alternative approach is to use hand-labeled legitimate email from a few trusted users to build the classifier. This approach, while highly accurate, raises two concerns: (1) *privacy:* users may not be willing to divulge all necessary network-level features for certain algorithms, and (2) *lack of variety:* a small set of users at a single institution (*e.g.*, a university campus) may receive mail from a limited set of senders; training using such data may affect a classifier's ability to correctly identify legitimate email from other types of senders.

To make SpamSpotter viable in practice, we plan to ask groups of legitimate users to run a plugin or script on their email clients (*e.g.*, Mozilla Thunderbird, Microsoft Outlook) or Webmail client instances (*e.g.*, as a browser extension) that will automatically extract certain features from the user's Inbox and forward the information to SpamSpotter.

7.5 SpamLoJack: Motivation

SpamLoJack is based on our observations in Chapter 3, Section 3.5: certain stealthy spammers hijack BGP routes, send quick bursts of spam, and then withdraw the route soon after spam is sent. This activity is malicious and extremely hard to defend against: once miscreants gain access to a BGP-speaking router (either through compromise, or by colluding with a rogue ISP), they can hijack nearly any prefix—even those belonging to legitimate mail servers such as Yahoo! Mail or Gmail—for at least portions of the Internet.

Spam via hijacked routes has always been "under the radar": because botnets continue to deliver large amounts of spam direct to their destinations, spammers have not adopted hijacking as a significant means of delivering spam. Indeed, in our study from 2006, we found that only about 10% of spam could have been attributed to suspicious, short-lived routes. However, as the Internet's routing infrastructure and its security are no better today than it was 5 years ago, spam from hijacked routes continues to be a significant source of worry. Recent events indicate that spammers continue to hijack routes to send spam. In a widely-discussed event on the North American Network Operators' group mailing list [6], a spammer friendly US-based ISP, Circle Internet, was found to have hijacked the IP space belonging to a defunct chemical company's unused IP address space. Although, in this particular instance, the Spamhaus DSNBL had listed the spammer's IP prefix, many hijacks—especially short-lived ones—may go undiscovered, especially since blacklists usually take 30 days or more to list a new spammer IP prefix. Moreover, because a large amount of IPv4 space is "dark"—*i.e.*, either not routed or not used for services, spammers and spammer-friendly ISPs have a large amount of IP address blocks to hijack at their leisure. Advanced techniques such as BGP spectrum agility (described in Section 3.5) further complicate the detection of prefix hijacks.

Our goal in designing SpamLoJack is to allow mail recipients to quickly discover whether a mail that they received might have originated from a hijacked route. Detecting whether a route is hijacked is fraught with problems—multiple-origin ASes may legitimately announce multiple routes, different vantage points in the Internet may observe different routes and origin ASes, etc.—thus, SpamLoJack cannot guarantee that a particular email indeed arrived from a hijacked route. Instead, SpamLoJack uses heuristics such as the historical records of ASes that originated the route and the records of spam received from these prefixes to arrive at an estimate of whether a route qualifies as "suspicious". In addition to recipients of email, SpamLoJack can also be useful to network operators: whenever SpamLoJack discovers a suspicious route that appears to be hijacked and sending spam, SpamLoJack can notify the true owner of the prefix such that immediate action can be taken; in this respect, SpamLoJack resembles Lad *et al*'s previous work on the Prefix Hijack Alert System (PHAS) [76]. Unlike PHAS, however, we use malicious events (*i.e.*, the appearance of spam) as a trigger to look for hijacked routes; thus, our system may potentially trigger fewer false alerts.

7.6 SpamLoJack: Design & Implementation

This section describes the design and implementation of SpamLoJack. Figure 7.3 describes the data collection setup used in SpamLoJack.

7.6.1 Data Collection

The spam we collect comes from a sinkhole set at Autonomous System number 2637 that belongs to the Georgia Institute of Technology. Ideally, to identify the routes by which spam arrives to our sinkhole, we would require a feed from the border routers of Georgia Tech. However, due to the practical difficulties of getting a real-time feed from a production setup such as Georgia Tech, we use an alternate setup: we leverage the routes received to a stub AS (AS 47065) that has Georgia Tech as its sole upstream provider. This AS has been set up by researchers in our lab for the GENI Transit Portal (TP) project [135].

Georgia Tech forwards all route announcements and withdrawals received form its upstreams—which include Tier 1 ISPs Cogent and Qwest—to the TP router. As updates arrive, the TP router appends them in real time to a file on disk. SpamLoJack runs an agent on the TP router that will push any modifications to the BGP updates file to the host that runs the SpamLoJack code. A similar agent runs on our spam sinkhole to push the details of incoming spam to the primary SpamLoJack machine.

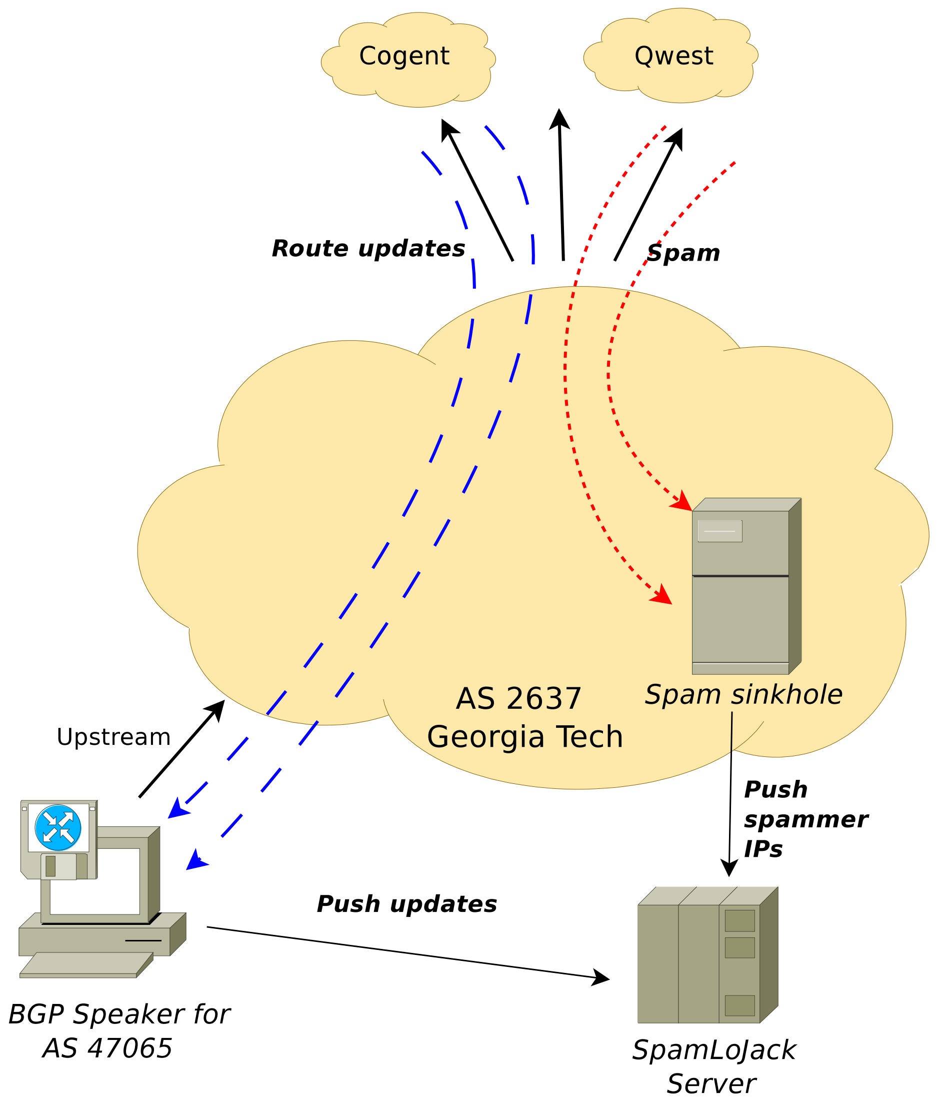


Figure 7.3: Setup used to collect BGP routing updates and spam feeds at AS 2637 (Georgia Institute of Technology)

7.6.2 Design

The design of SpamLoJack is motivated by its essential functionalities, which are as follows.

1. Maintain a lookup table for all active prefixes such that SpamLoJack can rapidly look up the route corresponding to the IP address from which a spam email arrived.

2. Maintain a record of the details of current and past routes for all prefixes including the AS path, the community attributes, the times of announcement and withdrawal, and a historical record of the number of spam emails received from these routes as well the number of unique spammer IPs in these routes.

3. Implement an alert mechanism based on heuristics that indicate whether a particular route appears suspicious. For example, SpamLoJack currently implements an alert if spam is received from a route that is announced shortly before spam is sent, and withdrawn soon afterwards. A related alert is raised when the origin AS for the best path for a prefix is different from the origin AS for the prefix according to Internet Registrars, or if it is different from the origin AS for the best path that persists for the majority portion of a time period.

4. Allow users to query the historical records of each prefix, includng amount of spam received from each prefix and corresponding origin ASes. SpamLoJack should also strive to make this information easily available; currently, we offer a front-end similar to the DNS-based query interface used in SpamSpotter.

7.6.3 Implementation

SpamLoJack Agents on Data Collection Machines. We run agents on both the TP router and the spam sinkhole that monitor updates to log files and immediately pushes the update over an RPC interface to the SpamLoJack core server. These agents use the Linux `inotify(7)` interface that efficiently monitors updates to files.[2] Both use Google Protocol Buffers [43] to push information to the SpamLoJack server via Remote Procedure Calls (RPC). Both agents are written in C++ and total approximately 500 lines of code.

SpamLoJack Core Processing. The core SpamLoJack code is implemented in

[2]The `inotify` interface is used to implement the `tail(1)` functionality in newer Linux systems.

C++. There are four components to the server code.

1. *RPC server.* This server runs as a separate threads and processes and serializes updates that arrive from the agents on the spam sinkhole and the TP BGP router. The updates it receives are queued into a buffer which is processed by the thread that performs the core processing.

2. *Core processing.* The thread that performs core processing maintains routes in a Patricia trie [69]. Given an IP address and a set of prefixes, a Patricia trie allows fast lookup of the longest prefix that matches any given IP address. For each spam email that matches a certain prefix, the thread logs the event to a database. If the prefix is withdrawn in future, the database is updated to reflect the time of withdrawal.

3. *Database.* The database maintains records of each BGP prefix and various attributes associated with each prefix. It maps each BGP prefix to (1) its owner AS (according to IANA records); (2) the various best AS paths that were seen at various points in the past 1 month; (3) the periods of time for which each AS path was the best path, and the number of spam emails received while each AS path was the best path. The database also logs whenever spam is received when the existing AS path configuration appears "suspicious", for example, when the origin AS in the best path does not match the AS that owns the prefix, or when route that receives spam is short-lived.

4. *Front-end.* This component accepts queries from the *rbldnsd* front-end for SpamSpotter for a specialized zone corresponding to SpamLoJack. For example, to look up an IP address 1.2.3.4's route behavior, the querier can issue a DNS query for a TXT record as follows:

```
$ dig +short 4.3.2.1.spamlojack.rbl.gtnoise.net TXT

1.2.3.4       p:1.2.3.0/21   orig_asn:25000   cc:US
p:1.2.0.0/21  path:2637,701,100,200,25000 spam:10    active_percent:90
p:1.2.0.0/22  path:2637,701,100,300,4000  spam:5000  active_percent:10
```

This sample lookup response indicates a number of useful features. The first line of the response comprises the queried IP and information about the IP according to IANA records, such as the IP allocation, origin AS, and the origin AS's country code. Each following line includes details of each best path that was seen at our BGP speaker for this IP address when spam was received from this prefix. For example the second line indicates that our spam sinkhole received spam from the allocation corresponding to the queried IP address for two distinct best paths. The first, likely the "real" AS path, shows a path that ends in the true owner of the /21 allocation; this path was active for 90% of time during the past month, and this route was used by spammers to send 10 spam emails. The second line indicates a more suspicious route: this route's origin AS is AS 400—different from the AS that owns the /21 allocation. Moreover, we note that this route announces a more specific /22 allocation, effectively hijacking half of AS 25000's IP space. Further, we see that although this route has been active for only 10% of time in the past month, it originated 5000 spam messages that arrived at our sinkhole. We plan to extend SpamLoJack's query/response interface with more expressive queries and responses. For example, a query could specifically ask for spam only from short-lived routes and the response could be augmented to indicate such routes.

Although we have not yet discovered examples that are as clearly malicious as above using SpamLoJack, we hope that SpamLoJack can allow network operators to keep tabs on their prefixes, and mail server administrators to discard email (or nearly any of unwanted traffic) from prefixes that appear to be hijacked. A final use of SpamLoJack is similar to the Prefix Hijack Alert System by Lad *et al* [76]: operators can make regular queries of their prefixes for suspicious activity (or sign up to be notified using a modified interface to SpamLoJack) so that they can easily discover if their IP blocks have been hijacked, especially using stealthy techniques such as spectrum agility.

SpamLoJack Performance. Because SpamLoJack is integrated into the SpamSpotter system, its performance is in line with SpamSpotter: on a local network SpamLoJack also returns queries in 10ms or less. Because we maintain the entire routing table in memory, running SpamLoJack incurs additional memory overhead: we find that maintaining a full routing table and additional metadata (*e.g.*, historical records of spam received from each prefix) uses between 3–4 gigabytes of memory.

We expect to extend SpamLoJack to include integrating a detailed database that keeps track of the historical records of spam received from each prefix and AS path on a longer timescale. Because using a traditional disk-backed relational database may cause unacceptable overhead (*i.e.*, takes more than 100ms on average), we plan to use one of the many new high-performance key-value stores based on the noSQL paradigm that all maintain a large in-memory cache in addition to an on-disk database [84,113].

7.7 Summary

In this section, we presented two tools—SpamSpotter and SpamLoJack—to help network operators and mail server administrators discover spammers and filter spam using network-level features. SpamSpotter is a framework that allows designers of new network-level spam-filtering techniques to easily build, integrate and deploy their services without worrying about the details of data collection, extraction of features, or deployment. SpamSpotter presents a unified interface to data for algorithms, is machine-learning aware (*i.e.*, understands the need of periodic retraining), and is easily integrated into production spam-filtering systems using the popular DNSBL query interface. We also presented SpamLoJack, a tool that joins data feeds from a co-located BGP router and spam sinkhole to identify spam arriving via potentially hijacked routes. SpamLoJack is also easily usable through the same DNS-based query interface that is used by SpamSpotter as well as popular spam filtering mechanisms such as SPF. We believe that SpamSpotter and SpamLoJack lay the foundations for an extensible network-level spam-filtering framework that that can prove useful for mail server administrators, network operators, and security researchers.

CHAPTER 8

CONCLUDING REMARKS

The Internet becomes a larger portion of our daily lives each year, and people spend increasing amounts of time online communicating with their colleagues, friends, and family. Email continues to be one of the major modes for personal and business communication, but other methods such as messaging on Web forums, syndicated messaging using platforms such as Facebook Connect, and micro-messaging platforms such as Twitter, have also gained widespread use. Spam is the chief menace for all these modes of communication: if people are spending enough time on a service, then it instantly becomes of interest to spammers.

Email spam is a well-acknowledged and well-studied problem, and many solutions have been proposed over the years. Most early efforts hinged on content-based classification: because humans could easily distinguish spam emails from legitimate ones with a quick glance, researchers implemented algorithms to do the same: construct features from email, learn the differences between spam and legitimate email, and use it for classification. Content-based filters continue to be widely used in all areas that are affected by spam, and tools such as SpamAssassin [120] and proprietary algorithms from security vendors exhibit high detection rates and low false positive rates. Unfortunately, *content-based* approaches suffer from two major shortcomings.

First, spammers do not idly sit by while researchers develop classifiers to filter spam; instead, they constantly monitor the performance of open-source and commercial filters to identify how much of their spam is reaching their intended targets, and easily develop new spam message formats to circumvent classifiers. For example, if a content-based classifier is extremely good at identifying a keyword and all its variations (*e.g.*, "VIAGRA" spelled as "V14GR4"), the spammer may attempt to embed the word "VIAGRA" in an image file and send the image file as an attachment; researchers must now expend time and effort to extend their classifier to perform optical character recognition (OCR) on *all* image attachments in email. This never-ending "arms-race" drives up costs and drives down productivity, and any effort in this direction is but a stop-gap measure until spammers develop their next trick.

Second, content-based filters are expensive. Each email must be accepted by the mail server, spooled on disk, and processed through a spam filter before a decision can be made to accept or reject the email. Not only does this increase the processing time per email, it also greatly affects scalability: with 90% of email being spam, the majority of the delay and computational power is spent analyzing and classifying spam. To scale with the increasing volumes of email each year, mail server administrators either must constantly upgrade their systems or face long queuing delays even for legitimate email.

8.1 The Need for Network-level Spam Filtering

This dissertation has introduced network-level features as an alternative and robust basis for creating classifiers and filters to stem the tide of spam. Network-level features are attributes of the sender that can be easily measured or observed even from the network layer (as opposed to the application layer), such as the IP address of the sender, its AS and origin country, and its historical behavior.

Although IP blacklists are a popular and effective network-level method of filtering known spam senders, we have demonstrated that this method has many shortcomings: for example, they cannot include IPs that send spam from dynamic address ranges for fear of false positives, or are ineffective from any spam originating from hijacked prefixes. Although collecting IP addresses that send to spam traps is one source of identifying spammer IPs that are active at any instant, they are not sufficient: many IPs that hit a spam trap may be reassigned to legitimate hosts in future, and a blacklist vendor cannot reliably add the IP address to his list without proof that the IP address will remain assigned to a spam bot's machine for some period of time. Because the blacklist vendor have no assurance that an IP address will continue to map to a spam bot, they typically do not list every IP address that appears at their spam trap for fear of false positives.

Although an IP address is a transient identifier of a spammer, the activity of the spammer is a more stable identifier. The techniques introduced in this dissertation do not rely on the IP address of a spammer remaining assigned to them for a long time; we merely use the IP address as an identifier for a short period such that we can identify the spammer's behavioral pattern. In support of our thesis, we have demonstrated three unique attacks against IP-based reputation systems, and ways to defend against these attacks only using network-level features and models of coordinated behavior of spammers.

8.2 Summary of Contributions

This dissertation established that network-level properties of spammers are sufficiently different from those of legitimate users, and that these features can be incorporated into practical systems both to detect and prevent various spammer attacks. More specifically, the contributions of this dissertations are as follows.

- *Comprehensive Characterization of the Network-level Behavior of Spammers.* We analyze the sending IP address ranges, sending patterns, chief ASes and countries responsible for sending spam, the chief operating systems that spammers use, the listing characteristics of spammers in DNSBLs, etc. Further, we analyze many of these features over a longitudinal trace over 6 years from 2004–2010; we find that many distinguishing network-level features of spammers remain stable over time.

- *Filtering Spam using Behavioral Blacklisting.* We found in Chapters 3 and 5 that IP blacklists perform poorly against a significant fraction of spammers, particularly ones that send spam from "fresh" IP addresses. Thus, we design a system to detect spammers not using their identity, but using their behavior: because spam bots in a botnet likely send email to the same set of domains, we cluster senders based on the set of domains to which they send email. We find that groups of spam senders cluster well because they hit the same set of domains, while legitimate senders do not cluster into large groups. Our system, *SpamTracker*, can detect many spam senders months before they become listed in blacklists.

- *Real-time Dynamic Spam-filtering Framework .* Many network-level (or content-based) spam-filtering algorithms never find deployment because they offer an interface that cannot be easily modified for real-time, practical use. Thus, we designed SpamSpotter, a real-time dynamic blacklisting system that allows clients to make queries using the familiar DNS-based query interface of DNS-BLs. Unlike DNSBLs, however, SpamSpotter uses dynamic algorithms on its back-end to perform classification—a claim we demonstrated by implementing three different algorithms in SpamSpotter: SpamTracker, SNARE [51], and Trinity [14].

- *Vote gaming to Pollute User Feedback in Webmail.* Large Webmail providers rely on user feedback to filter spam on which even their finely-tuned classifiers cannot make decisions on; by soliciting user feedback, the provider can quickly build consensus on whether a particular message is spam, and filter future messages from that sender. We present evidence that spammers use a stealthy attack, which we call *vote gaming*, where spammers vote "Not Spam" on their own spam messages that pollutes the consensus, and prolongs the true classification of spam messages and senders. We create a model and design an efficient system that clusters user accounts that perform vote gaming using *the IP addresses from which they perform voting.* We find that spammers form large clusters while legitimate users do not cluster well.

- *Spam from Hijacked BGP Prefixes.* We present evidence of a stealthy attack where spammers hijack large prefixes (*e.g.*, an entire /8) for short periods of time from rogue ISPs to send spam. This technique, which we call "spectrum agility" allows spammers to hop between unannounced prefixes within smaller, legitimate allocations. We design a system, SpamLoJack, that *joins* a real-time spam feed with a co-located feed of BGP updates to detect spam from potentially hijacked routes (using origin AS number) or those from route announcements that are short-lived. We further integrate SpamLoJack into our real-time dynamic blacklisting tool, SpamSpotter.

- *Spammers performing DNSBL Reconnaissance.* We find that certain spammers continually query DNS-based IP blacklists to discover if / when their own bots become listed in the blacklist. Because spammers use coordinated sets of bots to perform such queries, we design spatial and temporal features to discover coordinated queries, and, potentially, divulge the IP addresses of bots performing such queries.

8.3 Lessons Learned

This dissertation presents new attacks and answers several questions about spammer behavior, but the performance of this research has also provided the author with several non-trivial revelations and takeaways that we hope will be useful for future research in this area. We explore some of these lessons below.

IP addresses are not a reliable indicator of a spammer. We demonstrated in Chapters 3 and 5 IP blacklists fail to list a significant fraction of spam senders, many of which are "fresh" IP addresses that have not been seen before in previous months. Even though DNSBL operators can run their own spam traps and list any IP address from which it receives spam, they do not do so due to fear of false positives (*e.g.*, if the IP is dynamically allocated, it may be re-allocated to a legitimate machine in future). There are multiple reasons why we expect the amount of spammers caught by DNSBLs to further decline.

First, due to the exhaustion of the IPv4 address space, more users and organizations are using NATs [57] to create a private pool of IP addresses behind fewer public IP addresses. If there are a few spam bots in the private network, DNSBLs must either blacklist all of an organization's public addresses (risking false positives), or not list any IPs and risk false negatives. Both scenarios will decrease the utility of IP blacklists.

Second, instead of sending email directly from a bot's IP address—which may be in a dynamic IP address range and blacklisted based on policy alone (*e.g.*, the SpamHaus Policy Blacklist [123]), spammers have begun finding new ways to deliver spam to their recipients. Spammers have begun compromising (or creating accounts en-masse) at Web-based messaging services. Such services include Web-based email providers such as Gmail, Yahoo! Mail and Hotmail (who boast a total of nearly a billion users), as well as instant messaging services and social messaging sites such as Facebook. Spam delivered through such services will carry the IP address of the Web service which would render IP blacklisting or methods such as DKIM [29] and SPF [44] useless.

In the above cases, content-based filters will continue to work provided they are constantly updated, maintained, and scaled. We believe that network-level features will also continue to be useful in filtering out spam. For example, we showed that coordinated voting behavior in Webmail services can be identified using clusters built on network-level properties (Chapter 6). Web-based service providers may also employ spatial and temporal features to identify coordinated activity on their services (Chapter 4). Finally, for large business email service providers such as Gmail which sinks email for thousands of domains, they can deploy a recipient-oriented clustering algorithm such as SpamTracker to identify new spammers (Chapter 5).

Obtaining ground truth is hard. One of the chief problems we faced in our research was the lack of good-quality labels on data—something that can be used as "ground truth" in developing algorithms and techniques. It is extremely hard to obtain good quality labels for large-scale services, even after relying on IP blacklists: blacklist only stop senders who are almost certainly spammers, and a significant amount of spam gets by these blacklists. Of the email that is let through, only a fraction is truly legitimate. The rest may include true spam and also some *unsolicited bulk email*—which is not technically "spam" but is considered by many users as such. Because mail service providers prefer to be cautious with classifying emails as spam, we expect that the labels provided by mail service providers contain quite a few false negatives (*i.e.*, spam classified as ham); we showed in Chapter 5 that 15% of the email that SpamTracker classified as spam was let through by the provider.

Many spam studies use small datasets (*e.g.*, those collected from a mail server of a single campus) where researchers use a high-accuracy content-based filter to label their email dataset. This approach, while useful to demonstrate the properties of a specific algorithm, is unlikely to work at scale of production mail servers that receive email for many different domains. At such large scale, ground truth can only be obtained by heavily sampling email data and manually validating sampled emails, or alternatively, by using consensus built by users on certain emails. The latter point underscores the importance of ensuring that user votes are not polluted by attacks such as vote gaming (Chapter 6).

Learning spam behavior is hard. Related to the observation that ground truth is not easy to obtain, we also emphasize that implementing spam classifiers, using supervised or unsupervised learning, is hard. Even assuming that we have accurate labels (for supervised learning), it is difficult—if not impossible—to obtain good classification performance using a single technique in a real dataset from a large mail service provider. The chief reason is due to the *massive variety* of both legitimate and spam email that large providers receive: the mail they receive includes internationalization and foreign-language support, varying message formats, varying attachment formats, image formats, audio and video, etc. Large providers deal with such varied spam with variety of their own: they use a complex combination of network-level features, classifiers, per-recipient classifiers, and heuristics to achieve acceptable detection and false positive rates for all types of email.

The above fact offers a cautionary tale for spam-filtering algorithms that use datasets from a single campus and present high detection rates and low false positives. Researchers and peer-reviewers often focus their attention too heavily on these numbers, but these numbers may not mean anything in a real deployment: an algorithm that has good performance on a campus dataset may perform much worse when subject to a broader variety of spam. Fortunately, large email service providers do not expect new spam-filtering algorithms to provide a 99.9% detection rate; they may often be satisfied with a 20% detection rate provided the false positive rates are kept to a low figure (usually 1% or lower). This is why our algorithm for vote gaming—which, on the surface, detects a relatively small fraction of gamed users—is a useful tool for Yahoo! Mail.

We designed and built SpamSpotter with the goal of testing and improving new spam filtering algorithms. By providing an open framework and service for researchers to deploy their algorithms, we expect to attract a broad variety of mail recipients to use and contribute data to our service. We believe that such a service would help researchers design algorithms that perform better on real data, and, in turn, improve the real-world performance of these algorithms.

8.4 Future Work

The research in this dissertation suggests various avenues for future work in spam filtering using network-level features. We present three projects below that may be highly relevant to spam filtering in future.

8.4.1 Collaborative Spam Detection using Network Level Features

Spam is a problem that affects all mail recipients, yet there have been very few efforts for coordinated mitigation of spam using network-level features. As we show in this dissertation, spammers—especially, spam bots—exhibit unique, coordinated behavior that can be used to construct behavioral classifiers that go beyond just the identity of the bot or the format of messages it sends. A collaborative system where multiple mail recipients contribute information about the network-level characteristics (as well as content-based features) would allow all collaborators to improve their spam classifiers and to quickly identify spammers that may be "new" from the perspective of just a single domain.

Some projects, such as Vipul's Razor [92], have attempted to build collaborative systems based on content features, but these remain largely unused due the complexity of installing and integrating the agent with recipient mail servers. Present-day mail server administrators want a third-party hosted service that they can quickly integrate with their systems with a few lines of modifications to configuration files. This ease of deployment was one of the goals in our design of SpamSpotter, where mail server operators could contribute data *and* receive responses using a single DNS query to the remotely-hosted SpamSpotter service.

Because all spam bots in a botnet run the same code, we believe that there may be many more ways to detect the coordinated behavior of these bots. For example, we have found that spammers are often bursty, sending multiple emails in a single shot; if we find this behavior occurring from the same bot at multiple domains, we may proactively reject email from this bot. Another area for research could be the pattern by which bots cycle across domains: because we find that each bot sends small amounts of email to a *set of domains*, collaboration between the domains would allow them to detect new spam campaigns using telltale patterns of spamming, as we demonstrated with SpamTracker. We expect that developing these algorithms and integrating them in a SpamSpotter-like system will improve the current state of collaborative spam filtering.

8.4.2 Exploiting Coordinated Bot Behavior for Supervised Classifiers

It is well-understood that bots behave in a coordinated fashion, and that unsupervised learning techniques such as clustering (*e.g.*, SpamTracker, canopy clustering, etc.), or the graph-based technique demonstrated in BotGraph [149] can be used to identify bots or create real-time classifiers. These classifiers are based on the global behavior of all bots in a botnet, as opposed to the specific behavior of a single bot.

Domain-specific classifiers are complementary to the clustering-based techniques mentioned above. Domain-specific classifiers are typically supervised learning algorithms that use domain-specific features to generate a classifier. For example, a classifier such as SpamAssassin may use content-based features of each message, and a classifier like SNARE uses network-level features of each sender. These methods learn classifiers based on the *individual attributes* of a particular sender (in the case of SNARE) or a particular message (in the case of SpamAssassin). In practice, both domain-specific classifiers and unsupervised classifiers are used—separately—in production spam filters to ensure better overall classification.

A potential area for future work is to combine these types of classifiers instead of using each independent of the other—*i.e.*, use the data generated from a clustering approach to improve domain-specific classifiers. For example, a domain-specific classifier used to filter spam in Webmail accounts may involve features such as historical attributes of the user account, the features of the user's profile, etc. Because unsupervised classifiers are generated from the dynamics between sender IP addresses and higher-layer identifiers (*e.g.*, the domains to which an IP sends email, or the Webmail accounts from which an IP address logs in), they form an orthogonal feature set to the features used to train the domain-specific classifier. We expect that techniques such as co-training [11] and regularization can be used to combine these orthogonal classifiers into a potentially stronger overall classifier.

8.4.3 Limits on Evasion of Behavioral Classifiers

A question posed at any new spam filtering technique is the following: once spammers learn of the technique, what stops them from evading it? A benefit of behavioral algorithms based on network-level features is that they do not use features of spam that spammers can easily modify—for example, a bot cannot act independently of other bots in its botnet. However, knowing the specifics of an algorithm, spammers can still attempt to evade the classifier. Our research question is: what is the relation between the fraction of spammers that can successfully evade a network-level spam classifier, and the impact it has on the spam botnet's output?

To illustrate this question, consider SpamTracker. Suppose we form a large cluster of spammers each of whom send spam to at least 10 common recipient domains. If the botmaster that owns these bots wishes to evade SpamTracker, he might attempt to have each bot send spam to no more than 9 domains in common with another bot, which implies that each bot would now be sending at least 10% less spam than before (assuming 10 total domains and that each bot sends the same amount of spam to each domain). However, if SpamTracker clusters spammers who send spam to at least 3 common domains and the spammer restricts each bot to at most 2 domains in common with another bot, each bot now sends at least *33%* less spam than before.

What we seek, for each network-level spam filtering algorithm, is the relationship between the ease of evasion and the loss of "productivity" of the spammer. Our hypothesis is that, because bots rely so much on the low-and-slow sending pattern of sending small amounts of spam to many recipients, they will suffer a massive drop in spam output if they attempt to evade behavioral detection techniques such as SpamTracker.

REFERENCES

[1] Moot wins, Time Inc. Loses. http://musicmachinery.com/2009/04/27/moot-wins-time-inc-loses/.

[2] Bobax trojan analysis. http://www.lurhq.com/bobax.html, March 2005.

[3] Top 100 Digg Users Control 56 percent of Digg's Homepage Content. http://www.seomoz.org/blog/top-100-digg-users-control-56-of-diggs-homepage-content, 2006.

[4] Spamhaus delisting policy, 2007. http://www.spamhaus.org/sbl/policy.html.

[5] Email Spam Record Activity. http://www.guardian.co.uk/technology/2011/jan/10/email-spam-record-activity, 2011.

[6] Hijacked: 159.223.0.0/16. http://seclists.org/nanog/2011/Mar/979, 2011. North American Network Operator's Group Mailing list.

[7] E. Allman, J. Callas, M. Delany, M. Libbey, J. Fenton, and M. Thomas. *DomainKeys Identified Mail (DKIM) Signatures*. Internet Engineering Task Force, May 2007. http://www.ietf.org/rfc/rfc4871.txt.

[8] D. S. Anderson, C. Fleizach, S. Savage, and G. M. Voelker. Spamscatter: Characterizing Internet Scam Hosting Infrastructure. In *Proc. 16th USENIX Security Symposium*, Boston, MA, Aug. 2007.

[9] D. Bank and R. Richmond. Where the Dangers Are. *The Wall Street Journal*, July 2005. http://online.wsj.com/public/article/SB112128442038984802-w4qR772hjUeqGT2W0FIcA3_FNjE_20060717.html.

[10] R. Beverly and K. Sollins. Exploiting the Transport-Level Characteristics of Spam. In *5th Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, July 2008.

[11] A. Blum and T. Mitchell. Combining Labeled and Unlabeled Data with Co-training. In *11th Conference on Computational Learning Theory (COLT)*, 1998.

[12] Symantec Security Alert–W32.Bobax.D worm. `http://www.sarc.com/avcenter/venc/data/w32.bobax.d.html`.

[13] A. Brodsky and D. Brodsky. A Distributed Content-Independent Method for Spam Detection. In *Workshop on Hot Topics in Understanding Botnets*, Apr. 2007.

[14] A. Brodsky and D. Brodsky. A Distributed Content Independent Method for Spam Detection. In *First USENIX Workshop on Hot Topics in Understanding Botnets (HotBots)*, Cambridge, MA, Apr. 2007.

[15] D. Brumley. Tracking hackers on IRC. `http://www.doomdead.com/texts/ircmirc/TrackingHackersonIRC.htm`, 2003.

[16] D. Cheng, R. Kannan, S. Vempala, and G. Wang. A Divide-and-Merge Methodology for Clustering. *ACM Transactions on Database Systems*, 31(4):1499–1525, 2006.

[17] R. Clayton. spamHINTS: Happily It's Not The Same. `http://www.spamhints.org/`, 2007.

[18] Cloudmark Authority Anti-Spam. `http://www.cloudmark.com/serviceproviders/authority/spam/`.

[19] CNN Technology News. Expert: Botnets No. 1 emerging Internet threat. `http://www.cnn.com/2006/TECH/internet/01/31/furst/`, Jan. 2006.

[20] M. P. Collins, T. J. Shimeall, S. Faber, J. Janies, R. Weaver, and M. D. Shon. Using uncleanliness to predict future botnet addresses. In *Proc. ACM SIGCOMM Internet Measurement Conference*, San Diego, CA, USA, Oct. 2007.

[21] Commtouch Inc. 2006 Spam Trends Report: Year of the Zombies. `http://www.commtouch.com/downloads/Commtouch_2006_Spam_Trends_Year_of_the_Zombies.pdf`.

[22] E. Cooke, F. Jahanian, and D. McPherson. The Zombie Roundup: Understanding, Detecting and Disrupting Botnets. In *Usenix Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, June 2005.

[23] Description of coordinated spamming, Feb. 2005. `http://www.waltdnes.org/spam`.

[24] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *LATIN 2004: Theoretical Informatics*, pages 29–38, 2004.

[25] D. Dagon, C. Zou, and W. Lee. Modeling botnet propagation using time zones. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS '06)*, 2006.

[26] E. Damiani, S. de Vimercati, and P. Samarati. P2P-Based Collaborative Spam Detection and Filtering. In *4th IEEE Conference on P2P*, 2004.

[27] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proc. 6th USENIX OSDI*, San Francisco, CA, Dec. 2004.

[28] S. Dietrich, N. Long, and D. Dittrich. Analyzing distributed denial of service attack tools: The shaft case. In *Proceedings of the LISA 2000 System Administration Conference*, December 2000.

[29] DomainKeys Identified Mail (DKIM). http://www.dkim.org/.

[30] R. Droms. *Dynamic Host Configuration Protocol*. Internet Engineering Task Force, Mar. 1997. RFC 2131.

[31] N. Feamster. Open problems in BGP anomaly detection. In *CAIDA Workshop on Internet Signal Processing*, San Diego, CA, Nov. 2004.

[32] N. Feamster, D. Andersen, H. Balakrishnan, and M. F. Kaashoek. Measuring the effects of Internet path faults on reactive routing. In *Proc. ACM SIGMETRICS*, San Diego, CA, June 2003.

[33] F. C. Freiling, T. Holz, and G. Wicherski. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. Technical Report ISSN-0935-3232, RWTH Aachen, April 2005.

[34] Y. Freund and R. E. Schapire. Experiments with a new Boosting Algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, Bari, Italy, 1996.

[35] J. H. Friedman and B. E. Popescu. Predictive Learning via Rule Ensembles. Technical report, Dept. of Statistics, Stanford University, Feb. 2005.

[36] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. 25th VLDB*, pages 518–529, 1999.

[37] 'Gmail Killer' From Facebook on Its Way? http://www.cbsnews.com/8301-501465_162-20022793-501465.html, 2010.

[38] Google's reCAPTCHA busted by new attack. http://www.theregister.co.uk/2009/12/14/google_recaptcha_busted/, 2009.

[39] Gmail spam filtering. http://www.google.com/mail/help/fightspam/spamexplained.html, 2010.

[40] L. H. Gomes, F. D. O. Castro, R. B. Almeida, L. M. A. Bettencourt, V. A. F. Almeida, and J. M. Almeida. Improving Spam Detection Based on Structural Similarity. In *Proc. SRUTI Workshop*, Cambridge, MA, July 2005.

[41] L. H. Gomes, C. Cazita, J. Almeida, V. Almeida, and W. Meira. Characterizing a Spam Traffic. In *Proc. Internet Measurement Conference*, Taormina, Italy, Oct. 2004.

[42] Goodmail Systems, 2006. `http://www.goodmailsystems.com/`.

[43] Protocol Buffers. `http://code.google.com/apis/protocolbuffers`.

[44] S. Gorling. An overview of the Sender Policy Framework(SPF) as an anti-phishing mechanism. *Internet Research*, 17(2):169–179, 2007.

[45] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *Proc. 17th USENIX Security Symposium*, Vancouver, BC, Canada, Aug. 2008.

[46] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *Proc. 16th USENIX Security Symposium*, Boston, MA, Aug. 2007.

[47] G. Gu, J. Zhang, , and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *The 15th Annual Networks and Distributed Systems Secuirty Symposium*, San Diego, CA, 2008.

[48] Hadoop. `http://hadoop.apache.org/core/`, 2008.

[49] C. Hanna. Using snort to detect rogue IRC bot programs. Technical report, October 2004.

[50] S. Hansell. Postage is due for companies sending email, February 5, 2006. `http://www.nytimes.com/2006/02/05/technology/05AOL.html`.

[51] S. Hao, N. Syed, N. Feamster, A. Gray, and S. Krasser. Detecting Spammers with SNARE: Spatio-temporal Network-level Automatic Reputation Engine. In *Proceedings of the Usenix Security Symposium*, Vancouver, BC, Aug. 2009.

[52] S. Hao, N. Syed, N. Feamster, A. Gray, and S. Krasser. Detecting Spammers with SNARE: Spatio-temporal Network-level Automatic Reputation Engine. `http://www.cc.gatech.edu/~feamster/papers/snare-tr.pdf`, Feb. 2009. In submission; Earlier version appeared as Georgia Tech Technical Report GT-CSE-08-02.

[53] J. Hartigan and M. Wong. Algorithm AS 136: A K-means clustering algorithm. *Applied Statistics*, 28(1):100–108, 1979.

[54] S. Hershkop. *Behavior-based Email Analysis with Application to Spam Detection*. PhD thesis, Columbia University, 2006.

[55] D. Hilbert. ber die stetige Abbildung einer Linie auf ein Flchenstck. *Annals of Mathematics*, 38, 1891.

[56] Honeynet Project. Know Your Enemy: Tracking Botnets. `http://www.honeynet.org/papers/bots/botnet-commands.html`, 2006.

[57] Internet Engineering Task Force. *IP Network Address Translator (NAT) Terminology and Considerations*, Aug. 1999. RFC 2663.

[58] Interview with Yahoo! Mail Employee. `http://blog.unica.com/full-metal-email-confessions-of-an-anti-spam-zealot/`, 2010.

[59] IronPort Antispam. `http://www.ironport.com/technology/ironport_antispam.html`, 2006.

[60] IronPort Carrier Grade Email Security Appliance. `http://www.ironport.com/products/ironport_x1000.html`, 2007.

[61] L. Johansen, M. Rowell, K. Butler, and P. McDaniel. Email Communities of Interest. In *4th Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, July 2007.

[62] J. P. John, A. Moschuk, S. D. Gribble, and A. Krishnamurthy. Studying Spamming Botnets Using Botlab. In *Proc. 6th USENIX NSDI*, Boston, MA, Apr. 2009.

[63] Joris Evers. Most spam still coming from the U.S. `http://news.com.com/Most+spam+still+coming+from+the+U.S./2100-1029_3-6030758.html`, Jan. 2006.

[64] J. Jung and E. Sit. An Empirical Study of Spam Traffic and the Use of DNS Black Lists. In *Proc. Internet Measurement Conference*, pages 370–375, Taormina, Italy, Oct. 2004.

[65] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. Spamalytics: An Empirical Analysis of Spam Marketing Conversion. *Communications of the ACM (CACM)*, 2009.

[66] R. Kannan, S. Vempala, and A. Vetta. On clusterings: good, bad and spectral. *J. of the ACM*, 51(3):497–515, 2004.

[67] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. *Proc. 20th SODA*, 2010.

[68] J. Klensin. *The Simple Mail Transfer Protocol.* Internet Engineering Task Force, Apr. 2008. RFC 5321.

[69] D. Knuth. *The Art of Computer Programming Volume III: Sorting and Searching.* Addison Wesley, 1998.

[70] J. Kong et al. Scalable and Reliabile Collaborative Spam Filters: Harnessing the Global Socail Email Networks. In *3rd Annual Workshop on the Weblogging Ecosystem*, 2006.

[71] S. Krasser, G. Conti, J. Grizzard, J. Gribschaw, and H. Owen. Real-time and forensic network data analysis using animated and coordinated visualization. In *Proceedings of the 6th IEEE Information Assurance Workshop*, 2005.

[72] B. Krebs. Bringing botnets outof the shadows. `http://www.washingtonpost.com/wp-dyn/content/article/2006/03/21/AR2006032100279.html`, 2006.

[73] C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. On the Spam Campaign Trail. In *USENIX Workshop on Large-scale Exploits and Emergent Threats*, San Francisco, CA, 2008.

[74] C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. Spamcraft: An Inside Look at Spam Campaign Orchestration. In *USENIX Workshop on Large-scale Exploits and Emergent Threats*, Boston, MA, 2009.

[75] R. Kumar, A. Tomkins, and E. Vee. Connectivity structure of bipartite graphs via the knc-plot. In *ACM International Conference on Web Search and Data Mining (WSDM)*, 2008.

[76] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang. PHAS: A prefix hijack alert system. In *Proc. 15th USENIX Security Symposium*, Vancouver, BC, Canada, Aug. 2006.

[77] H. Lam and D. Yeung. A learning approach to spam detection based on social networks. In *4th Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, July 2007.

[78] F. Li and M.-H. Hseih. An Empirical Study of Clustering Behavior of Spammers and Group-based Anti-Spam Strategies. In *3rd Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, July 2006.

[79] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In *Proc. ACM SIGCOMM*, pages 3–17, Pittsburgh, PA, Aug. 2002.

[80] MailAvenger, 2006. `http://www.mailavenger.org/`.

[81] Mail Abuse Prevention System (MAPS). `http://www.mail-abuse.com/`.

[82] A. McCallum, K. Nigam, and L. H. Ungar. Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching. In *Proceedings of KDD*, 2000.

[83] Microsoft security bulletin ms04-011. `http://www.microsoft.com/technet/security/bulletin/ms04-011.mspx`, Apr. 2004.

[84] Mongodb database. `http://www.mongodb.org/`.

[85] D. Moore, C. Shannon, and J. Brown. Code-red: A case study on the spread and victims of an internet worm. In *Proc. ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, Nov. 2002.

[86] D. Moore, G. M. Voelker, and S. Savage. Inferring internet denial-of-service activity. In *Proceedings of the 2001 USENIX Security Symposium*, 2001.

[87] Operating System Market Shares. `http://marketshare.hitslink.com/report.aspx?qprid=2`, Jan. 2006.

[88] The Open Relay Database, 2006. `http://ordb.org/`.

[89] PandaLabs Blog. Zunker Spamming Bot Front-end Analysis. `http://blogs.pandasoftware.com/blogs/pandalabs/archive/2007/05/08/Zunker.aspx`, May 2007.

[90] A. Pathak, Y. C. Hu, and Z. M. Mao. Peeking into Spammer Behavior from a Unique Vantage Point. In *Proceedings of the First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, San Francisco, CA, Apr. 2008.

[91] A. Pitsillidis, K. Levchenko, C. Kreibich, C. Kanich, G. M. Voelker, V. Paxson, N. Weaver, and S. Savage. Botnet Judo: Fighting Spam with Itself. In *Proceedings of NDSS, 2010*, San Diego, CA, 2010.

[92] V. Prakash. Vipul's Razor. `http://razor.sourceforge.net/`, 2007.

[93] M. Prince, B. Dahl, L. Holloway, A. Keller, and E. Langheinrich. Understanding How Spammers Steal Your E-Mail Address: An Analysis of the First Six Months of Data from Project Honey Pot. In *Second Conference on Email and Anti-Spam*, Stanford, CA, July 2005.

[94] Project Honey Pot. `http://www.projecthoneypot.org/`.

[95] Pyzor. `http://pyzor.sourceforge.net/`.

[96] F. Qian, A. Pathak, Y. C. Hu, Z. M. Mao, and Y. Xie. A Case for Unsupervised Learning Based Spam Filtering. Technical Report CSE-TR-561-10. University of Michigan Technical Report.

[97] Z. Qian, Z. M. Mao, Y. Xie, and F. Yu. On Network-level Clusters for Spam Detection. In *Proceedings of NDSS, 2010*, San Diego, CA, 2010.

[98] Z. Qian, Z. M. Mao, Y. Xie, and F. Yu. Triangular Spamming: A Stealthy and Efficient Spamming Technique. In *Proceedings of the IEEE Symposium on Security and Privacy, 2010*, 2010.

[99] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan-Kaufmann, 1993.

[100] S. Racine. Analysis of internet relay chat usage by ddos zombies. `ftp://www.tik.ee.ethz.ch/pub/students/2003-2004-Wi/MA-2004-01.pdf`, 2004.

[101] A. Ramachandran, D. Dagon, and N. Feamster. Can DNSBLs Keep Up with Bots? In *3rd Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, July 2006.

[102] A. Ramachandran, A. Dasgupta, N. Feamster, and K. Weinberger. Spam or Ham? Characterizing and Detecting Fraudulent "Not Spam" Reports in Web Mail Systems. Technical Report GT-CS-11-06. Georgia Tech School of Computer Science Technical Report.

[103] A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. In *Proc. ACM SIGCOMM*, Pisa, Italy, Aug. 2006.

[104] A. Ramachandran and N. Feamster. Understanding the Network-Level Behavior of Spammers. Technical Report GT-CSS-2006-001, Georgia Tech, Feb. 2006.

[105] A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. In *Proc. ACM SIGCOMM*, Pisa, Italy, Aug. 2006.

[106] A. Ramachandran and N. Feamster. Understanding the Network-Level Behavior of Spammers. In *Proc. ACM SIGCOMM*, Pisa, Italy, Aug. 2006. An earlier version appeared as Georgia Tech TR GT-CSS-2006-001.

[107] A. Ramachandran, N. Feamster, and D. Dagon. Revealing Botnet Membership with DNSBL Counter-Intelligence. In *2nd USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, San Jose, CA, July 2006.

[108] A. Ramachandran, N. Feamster, and S. Vempala. Filtering spam with behavioral blacklisting. In *Proc. 14th ACM Conference on Computer and Communications Security*, Alexandria, VA, Oct. 2007.

[109] A. Ramachandran, H. Khandelwal, S. Hao, and N. Feamster. Spotting Spammers with a Dynamic Reputation System. Technical Report GT-CS-08-09, Georgia Tech School of Computer Science Technical Report, Aug. 2009.

[110] S. Ramasubramanian. Port 25 filters - how many here deploy them bidirectionally? `http://www.merit.edu/mail.archives/nanog/2005-01/msg00127.html`, Jan. 2005.

[111] P. Ramneek. Bots & Botnets: An Overview. `http://www.giac.com/practical/GSEC/Ramneek_Puri_GSEC.pdf`, 2003.

[112] rbldnsd: Small Daemon for DNSBLs. `http://www.corpit.ru/mjt/rbldnsd.html`.

[113] Redis. `http://redis.io/`.

[114] Y. Rekhter, T. Li, and S. Hares. *A Border Gateway Protocol 4 (BGP-4)*. Internet Engineering Task Force, Jan. 2006. RFC 4271.

[115] Renesys Blog. Pakistan Hijacks YouTube. `http://www.renesys.com/blog/2008/02/pakistan_hijacks_youtube_1.shtml`, 2008.

[116] Secure Computing. `http://www.securecomputing.com/`, 2007.

[117] Secure Computing IronMail. `http://www.securecomputing.com/index.cfm?skey=1612`, 2007.

[118] A. Sinclair and M. Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. Information and Computation, 82:93–133, 1989.

[119] The Spam and Open Relay Blocking System (SORBS), 2006. `http://www.sorbs.net/`.

[120] SpamAssassin, 2006. `http://www.spamassassin.org/`.

[121] SpamCop. `http://www.spamcop.net/`.

[122] Spamhaus, 2006. `http://www.spamhaus.org/`.

[123] Spamhaus policy block list, 2007. `http://www.spamhaus.org/pbl`.

[124] Spammer-X. *Inside the SPAM Cartel*. Syngress, 2004. Page 40.

[125] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in Your Spare Time. In *Proc. 11th USENIX Security Symposium*, San Francisco, CA, Aug. 2002.

[126] S. J. Stolfo, S. Hershkop, C.-W. Hu, W.-J. Li, O. Nimeskern, and K. Wang. Behavior-based modeling and its application to Email analysis. 6(2):187–221, May 2006.

[127] SwatIt. Bots, drones, zombies, worms and other things that go bump in the night. `http://swatit.org/bots/`, 2004.

[128] Symantec. Storm Worm (Trojan.Peacomm). `http://www.symantec.com/security_response/writeup.jsp?docid=2007-011917-1403-99`.

[129] Symantec. W32.Waledac. `http://www.symantec.com/security_response/writeup.jsp?docid=2008-122308-1429-99`.

[130] Symantec Inc. Botnets now produce 95of spam. `http://www.bizjournals.com/sanjose/stories/2010/08/23/daily29.html`, 2010.

[131] Symantec Inc. Symantec MessageLabs Intelligence Report. http://www.messagelabs.co.uk/mlireport/MLI_2011_03_March_Final-EN.pdf, 2011.

[132] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and Evaluation of a Real-time URL Spam Filtering Service. In *Proc. IEEE Symposium on Security and Privacy*, Oakland, CA, May 2011.

[133] J. Todd. AS number inconsistencies, July 2002. http://www.merit.edu/mail.archives/nanog/2002-07/msg00259.html.

[134] Realtime uri blacklist. http://www.uribl.com/.

[135] V. Valancius, N. Feamster, J. Rexford, and A. Nakao. Wide-Area Route Control for Distributed Services. In *Proc. USENIX Annual Technical Conference*, Boston, MA, June 2010.

[136] Virus Bulletin 2005 Paper on 'Bots and Botnets'. http://arachnid.homeip.net/papers/VB2005-Bots_and_Botnets-1.0.2.pdf.

[137] S. Venkataraman, S. Sen, O. Spatscheck, P. Haffner, and D. Song. Exploiting Network Structure for Proactive Spam Mitigation . In *Proc. 16th USENIX Security Symposium*, Boston, MA, Aug. 2007.

[138] P. Vixie. Distributed Checksum Clearinghouse. http://www.rhyolite.com/anti-spam/dcc/.

[139] Windows Live Hotmail Fact Sheet. http://www.microsoft.com/presspass/presskits/windowslive/docs/WindowsLiveHotmailFS.doc, 2010.

[140] M. Wong and W. Schlitt. *Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail.* Internet Engineering Task Force, Apr. 2006. RFC 4408.

[141] Blitzed Open Proxy Monitor (BOPM). http://wiki.blitzed.org/opm.

[142] Y. Xie, F. Yu, , K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming bots: Signatures and characteristics. In *Proc. ACM SIGCOMM*, Seattle, WA, Aug. 2008.

[143] Y. Xie, F. Yu, K. Achan, E. Gilum, M. Goldszmidt, and T. Wobber. How dynamic are IP addresses. In *Proc. ACM SIGCOMM*, Kyoto, Japan, Aug. 2007.

[144] Yahoo! Research. Sparta Project – Yahoo! Research. `http://research.yahoo.com/node/2446`.

[145] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. K. Gunda, and J. Currey. DryadLINQ: A system for genera-purpose data-parallel computing using a high-level language. In *Proc. 8th USENIX OSDI*, San Diego, CA, Dec. 2008.

[146] ZDNet Security News. Most spam genrated by botnets, expert says. `http://news.zdnet.co.uk/internet/security/0,39020375,39167561,00.htm`, Sept. 2004.

[147] J. Zhang, P. Porras, and J. Ulrich. Highly Predictive Blacklisting. In *Proc. 17th USENIX Security Symposium*, Vancouver, BC, Canada, Aug. 2008.

[148] Y. Zhang and V. Paxson. Detecting stepping stones. In *Proceedings of the 9th USENIX Security Symposium*, August 2000.

[149] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum. BotGraph: Large Scale Spamming Botnet Detection. In *Proc. 6th USENIX NSDI*, Boston, MA, Apr. 2009.