A Large-Scale Empirical Analysis of Email Spam Detection Through Network Characteristics in a Stand-Alone Enterprise

Tu Ouyang[a,*], Soumya Ray[a], Mark Allman[b], Michael Rabinovich[a]

[a]*Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, OH, USA*
[b]*International Computer Science Institute, Berkeley, CA, USA*

Abstract

Spam is a never-ending issue that constantly consumes resources to no useful end. In this paper, we envision spam filtering as a pipeline consisting of DNS blacklists, filters based on SYN packet features, filters based on traffic characteristics and filters based on message content. Each stage of the pipeline examines more information in the message but is more computationally expensive. A message is rejected as spam once any layer is sufficiently confident. We analyze this pipeline, focusing on the first three layers, from a single-enterprise perspective. To do this we use a large email dataset collected over two years. We devise a novel ground truth determination system to allow us to label this large dataset accurately. Using two machine learning algorithms, we study (i) how the different pipeline layers interact with each other and the value added by each layer, (ii) the utility of individual features in each layer, (iii) stability of the layers across time and network events and (iv) an operational use case investigating whether this architecture can be practically useful. We find that (i) the pipeline architecture is generally useful in terms of accuracy as well as in an operational setting, (ii) it generally ages gracefully across long time periods and (iii) in some cases, later layers can compensate for poor performance in the earlier layers. Among the caveats we find are that (i) the utility of network features is not as high in the single enterprise viewpoint as reported in other prior work, (ii) major network events can sharply affect the detection rate, and (iii) the operational (computational) benefit of the pipeline may depend on the efficiency of the final content filter.

Keywords: spam, network-level characteristics, longitudinal analysis

1. Introduction

Spam is clearly an ongoing challenge for both network operators and users. Spam imposes many costs ranging from simple network capacity, computation and storage costs to user productivity issues that arise as people manually filter spam that is not automatically caught or worse, lose legitimate messages in the maze of filters our messages now must traverse. While in general, spam filtering in one form or another works reasonably well in terms of keeping spam away from users without excessively dropping legitimate communication, the costs of this filtering are high *behind the scenes*. For instance, computational complexity issues arise with parsing and tokenizing email, database lookups, etc. Further, depending on the setup the storage requirements of retaining a "junk" folder for users to pick through if they suspect a lost message is also high. For instance, one of the authors' "junk" folders consists of 66 MB of messages over the last 30 days. In addition, spam represents an arms race whereby spammers are constantly developing new ways to circumvent filters (by adjusting their content, where they transmit spam from, etc.) while network administrators are constantly updating their tools and databases to keep spam away from their users. Therefore, while a users' viewpoint might well be that spam appears to be a largely "solved" problem the reality is quite different for administrators and operators.

A research direction recently emerged focusing on *non-content* features to build models that can disambiguate spam and legitimate email (commonly referred to as "ham") [35, 9, 20]. For instance, email that comes from bogus IP addresses or does not follow the usual size distribution of ham might be more likely to be spam. The overall thrust of these efforts is to try to reduce the cost of spam filtering by making some decisions without relying on computationally

*Corresponding author

Email addresses: tu.ouyang@case.edu (Tu Ouyang), sray@case.edu (Soumya Ray), mallman@icir.org (Mark Allman), michael.rabinovich@case.edu (Michael Rabinovich)

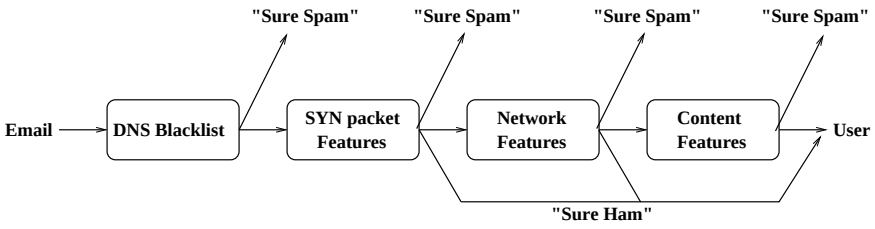


Figure 1: The conceptual spam filtering pipeline. Each layer looks at more information in the message and so is more computationally expensive to use, but also potentially more accurate at filtering spam. We assume the fourth layer, content filters, will always be present. Our work explores the accuracy and cost tradeoffs between the first three layers, and the value added in the presence of the fourth layer.

expensive content filters or by throwing away some spam (when it can be detected with high confidence) and thus not burdening the storage infrastructure. Further motivation of not relying solely on content features is that content of spam email could be manipulated relatively easily by spammers to blur the boundary between ham and spam and confuse the content-based filters. For instance, Lowd and Meek describe a "good word attack", which inserts words indicative of legitimate email into spam and gets up to 50% of currently blocked spam through certain types of content filters [22]. Conversely, Nelson et al. show that by inserting carefully selected words into spam, a content-based filter can be subverted to block some ham messages [27].

Of course, using non-content features to filter email is itself not a new idea—the well established method of using DNS blacklists filter email by simply looking at the IP address of the sender and comparing it to a list of addresses "known" to be sending spam. Recent work has expanded upon this idea, however, by exploiting machine learning methods to train classifiers using features derived from the first SYN packet and features derived from the *network characteristics* (e.g., round-trip time, TCP advertised window sizes) of the email transfer[1] [20, 9]. These "content-blind" techniques are attractive because they leverage properties that are hard to manipulate and can help discard spam quickly and at less computational cost. We find that each of these approaches and their respective feature sets have some utility in separating ham from spam.

Though it was not presented this way, the prior work above motivates us to consider a conceptual "spam filter pipeline" (Figure 1) at the entry point to an enterprise. In this pipeline, each email is processed first by the DNS blacklist, followed by the information in the first SYN packet, followed by more complete network characteristics (considering the first packet features in combination with other characteristics of the email transfer), and finally the content. Each filter looks at more information in the email, and so is more computationally expensive than the previous, but also *potentially* more accurate. Each filter discards what it concludes is "surely" spam (of course, with some error) and sends any remaining messages to the next layer. Moreover, in this paper, we consider two separate classifiers in each stage—one trained to detect and discard "sure spam" as above, and the other to detect (again with some error) any "sure ham" messages, which are transferred to the user's inbox, thereby reducing further the number of uncertain messages passed to the next stage. We note that this pipeline is primarily a *conceptual* framework; in an operational setting, the exact architecture may differ depending on the techniques used and other considerations such as operational efficiency and storage costs. In particular, we consider a setup where the pipeline contains only one stage between the DNS blacklist and the content filter, considering the performance of this stage when it is limited to only the first packet features or utilizes richer network characteristics from the email transfer.

This paper analyzes this pipeline focusing on the stages before the content filter. We assume content filters will always be present; it seems unrealistic to have a highly accurate spam filter that never looks at the content. Specifically, we address the questions from a single enterprise perspective:

1. What is the added value in terms of accuracy by the second and third layers? [2]
2. Does the added value depend on the machine learning algorithm used to process these features?
3. How do the layers interact with each other? Can a later layer "take over" if an earlier one becomes inaccurate?
4. Does the added value stay stable over time and across network events?

[1] These characteristics include both network-layer and TCP-layer features; we refer to them collectively as "network features" to emphasize that they are lower-level features not derived from content.

[2] Other work has evaluated the accuracy of DNS blacklists [38] and content filters [8, 28].

5. What features are primarily responsible for the value addition?
6. What is the computational load of such a pipelined architecture as opposed to just using the DNS blacklist and a single content filter (a currently standard setup)?

To answer these questions, we use a very large dataset collected over a two-year period, from May 2009 to April 2011. Our dataset includes over 1.4 million messages received at the International Computer Science Institute (ICSI). The size of our dataset allows us to not only re-appraise previous results in a new and larger context—which is valuable in its own right given the heterogeneity of the Internet—but also to carefully address the questions above. However, the size of our dataset also creates a problem in terms of determining ground truth, since it is impossible to hand-label all of these messages, not only because of their numbers but also because of privacy concerns. To solve this, we use a novel procedure we developed for automatic email labeling with high accuracy. This is also a contribution of our work. We believe our methodology for ground truth development will be applicable beyond the scope of our current work and become generally useful in evaluating spam classification techniques.

In the following sections, we first describe the data we collected and our novel ground truth determination procedure. Then we describe experiments designed to answer the questions above and discuss the results. Finally, we discuss related work and conclude.

2. Data Collection and Feature Extraction

As part of our general monitoring activities we collect full content packet traces from the border of the International Computer Science Institute (ICSI). Retaining all such traces permanently is prohibitive for a number of reasons and therefore the ICSI traces are retained only temporarily—essentially until their operational relevance for forensics and troubleshooting has diminished. For this study, we retained packets involving ICSI's two main SMTP servers.[3] The dataset we use in this paper covers one week each month (the 11^{th} through the 18^{th}) from May 2009 through April 2011. We only consider mail to ICSI users and not outgoing mail from ICSI users as this latter would introduce significant bias in the network characteristics because ICSI's two SMTP servers and connectivity would be a strong influence.

ICSI's standard email setup involves using the Spamhaus SBL, XBL and PBL DNS blacklists [39] to help mitigate the impact of spam. SMTP transactions with blacklisted hosts are terminated by the SMTP daemon with an SMTP error delivered to the remote host. These transactions do not give us the message body to understand if the message is actually spam nor a chance to observe a number of our flow-level features (described below). We do not include these blacklisted transactions in most of our experiments, so that our setup resembles the typical operational setup at many institutions that includes a DNS blacklist as a first step in the spam filtering process. When we do consider these transactions (see Section 5.3), namely, in the experiments involving only features of the SYN packets[4], we categorize all blacklisted transactions as spam. As we will see, these experiments indicate that filtering emails with the blacklist may make our job more difficult by removing some of the more egregious sources of spam, leaving the rest of the pipeline to contend with more subtle spammers.

The characteristics of the data are given in Table 1. The middle columns denote messages which are excluded because they are outgoing, or for which we do not have network characteristics or content for various reasons. The "Various Bad" column includes the following: messages aimed at an invalid ICSI user—and therefore never delivered to ICSI; SMTP sessions that sent malformed `HELLO` commands thus failed a `BAD_HELLO` check; TLS encrypted SMTP sessions whose content cannot be reconstructed from packet traces; and delivery attempts without any actual message. For unknown reasons, some SMTP sessions did not end up with a whole email message; these are listed in the "Unknown" category. The next two columns in Table 1 show the number of spam and ham email in each month. The specifics of how we derived these numbers is given in the next section.

From our corpus of packet traces we use *Bro* [31] to re-assemble email messages from each of the incoming SMTP connections and *spamflow* [9], *Bro* and *p0f* [41] to generate the network characteristics of each connection. We use the messages themselves to develop ground truth (as described in the next section). Table 2 lists the network- and transport-layer features we distill from the packet traces. Here, "local" indicates one of the two ICSI mail servers

[3] We do record email not involving these two servers, but since in some cases our tools rely on the notion of a well-known mail server to form an understanding of directionality, we exclude this traffic. The excluded traffic is a relatively small fraction of the SMTP traffic recorded. For instance, we exclude 0.3% of the SMTP packets and 0.5% of the SMTP bytes on the first day of our data collection.

[4] SYN packets in the ICSI setup occur before the blacklist query and thus are available.

	Msgs	Outbound	DNSBL	Various Bad	Unknown	Spam	Ham
May/09	279K	41K	165K	20K	4K	30K	18K
Jun/09	302K	38K	185K	28K	6K	26K	18K
Jul/09	317K	49K	174K	39K	8K	30K	18K
Aug/09	292K	54K	165K	26K	5K	26K	15K
Sep/09	300K	46K	172K	31K	7K	27K	17K
Oct/09	223K	37K	116K	19K	8K	25K	17K
Nov/09	249K	43K	105K	17K	7K	55K	21K
Dec/09	335K	42K	168K	39K	5K	53K	25K
Jan/10	311K	40K	152K	35K	5K	51K	26K
Feb/10	235K	50K	116K	20K	6K	26K	17K
Mar/10	270K	49K	122K	23K	84K	53K	18K
Apr/10	270K	4K	118K	18K	10K	54K	22K
May/10	250K	49K	104K	14K	6K	47K	28K
Jun/10	215K	45K	91K	16K	4K	39K	21K
Jul/10	277K	56K	120K	21K	14K	29K	35K
Aug/10	255K	48K	102K	21K	15K	48K	23K
Sep/10	227K	30K	40K	81K	12K	46K	17K
Oct/10	229K	40K	39K	84K	10K	42K	14K
Nov/10	215K	44K	39K	77K	4K	38K	13K
Dec/10	214K	44K	43K	63K	4K	46K	15K
Jan/11	202K	40K	39K	59K	5K	45K	15K
Feb/11	209K	47K	37K	54K	10K	45K	16K
Mar/11	164K	42K	32K	34K	4K	38K	14K
Apr/11	175K	40K	40K	35K	3K	42K	15K

Table 1: Data overview: The second column shows the total number of email messages, the third column shows outbound messages, the fourth column shows messages blocked by the DNS blacklist, columns 3–6 show messages removed from the analysis and the last two columns give the number of spam and ham.

that are near the packet tracing vantage point, while "remote" indicates the non-ICSI host that is connecting to the ICSI mail server. In other words, the "remote" host is sending mail to the "local" host. Features marked with a "$*$" were derived by *Bro*, those marked with a "¶" were derived by *p0f* and the remaining features were obtained via *spamflow*.[5] Finally, we place a B next to those features that are common between our study and prior work [9]. From this data, we derived two sets of features: *packet features* from the SYN packet and *flow features* from the network traffic characteristics of the whole SMTP session. These features are explained below.

Packet Features. The upper part of Table 2 lists the features we use that are derived just from the SYN packet of the incoming SMTP connection. We call these "packet features." The *geoDistance*, *senderHour*, *AS-Spamminess*, and *neighborDist* features are used in prior work [20] although we derive the last three differently, as follows. We do not translate sender's hour into the ratio of ham to spam that were sent during that hour, because sender's hour itself is a numeric feature directly suitable for inclusion in our models. Prior work uses the AS number directly as a numeric feature. However, AS numbers are individual labels which do not lend themselves to meaningful aggregation in models (e.g., just because ASes 3 and 12 show some common behavior does not mean that ASes 4–11 share that behavior). Further, if treated as discrete values, the number of distinct AS values may be problematic for some classification methods. So we translate sender's AS number into a numerical value that reflects the prevalence of spam originating in the AS. The value for this feature is derived by using all messages in a training sample to develop a database of the "spamminess" of an AS. If a test message came from an AS that did not occur in the training set, we assign the average spamminess over all ASes as the value of this feature for that message.

To calculate the neighbor distance, *neighborDist*, Hao, et al. [20] first split their dataset into 24-hour bins. *NeighborDist* is then the average distance to the 20 nearest IPs among preceding senders in the same bin, or among all

[5] We derived these few features outside *spamflow* purely as a convenience. We believe that all network and transport layer features could be readily derived in a single place—likely the TCP implementation itself in an operational setting.

Feature	Description
geoDistance^H	The geographical distance between the sender and ICSI, based on the MaxMind GeoIP database [24].
senderHour^H	The hour of packet arrival in sender's timezone.
AS-Spamminess^H	Number of spam from AS divided by total messages, from AS in the training set.
neighborDist^H	Average numerical distribution from sender's IP to the nearest 20 IPs of other senders.
OS	OS of remote host as determined by *p0f* tool from SYN packet.
ttl	IP TTL field from SYN received from remote host.
ws	Advertised window size from SYN received from remote host.
3whs^B	Time between the arrival of the SYN from the remote host and arrival of ACK of the SYN/ACK sent by the local host.
fins_local^B	Number of TCP segments with "FIN" bit set sent by the local mail server.
fins_remote^B	Number of TCP segments with "FIN" bit set received from the remote host.
idle^B	Maximum time between two successive packet arrivals from remote host.
jvar^B	The variance of the inter-packet arrival times from the remote host.
pkts_sent / pkts_recvd	Ratio of the number of packets sent by the local host to the number of packets received from the remote host
rsts_local^B	Number of segments with "RST" bit set sent by the local mail server.
rsts_remote^B	Number of segments with "RST" bit set received from remote host.
rttv	Variance of RTT from local mail server to remote host.
rxmt_local^B	Number of retransmissions sent by the local mail server.
rxmt_remote^B	Approximate number of retransmissions sent by the remote host.
bytecount	Number of non-retransmitted bytes received from the remote host.
throughput	*bytecount* divided by the connection duration.

Table 2: Message features. Features marked with H and B are from [20] and [9], respectively.

the neighbors if there are fewer than 20 preceding senders [20, 19]. This procedure is not suitable for our enterprise environment because a one-day bin does not provide enough email to accumulate enough history. Further, since the database is smaller, boundary effects due to insufficient number of neighbors in the beginning of each bin influence the results greatly. This is illustrative of our first contribution (discussed in more detail below): a single edge network's myopic view may thwart development of accurate models in some cases. To mitigate this effect, we build IP databases using an entire training sample—consisting of 9/10 of the data for each month, given we use 10-fold cross validation (described later). We then use this database to produce *neighborDist* values for the training and test data. Note that because of our procedure, each fold of our experiments uses different databases for the *AS-Spamminess* and *neighborDist* features. We refer to these two features as "database features" below.

Hao et al. also use a feature that requires port scanning the sending IP. This is operationally problematic as it is time consuming and may trigger security alarms on remote hosts. Further, while we have historical packet trace data, we do not have historical port scanning data and mixing current port scans with historical packet traces would be a dubious experimental procedure. While we deleted or modified several single packet features we also added several features: senders' OS, IP's residual TTL, and TCP's advertised window.

Flow features. The lower part of Table 2 shows the set of flow features we use to describe messages. This list overlaps with that used in prior work [9] (with common features tagged with a B)). We added several features we believe may help discriminate ham from spam. In addition, we removed three features, as well: *packets* (in each direction), *cwnd0* and *cwndmin*. The number of packets is closely related to the *bytecount* and *pkts_sent/pkts_recvd* features in our list. The two *cwnd* features are actually not about the congestion window, but about the advertised window. The *cwnd0* feature tries to capture the case when the remote host goes into a zero-window probing mode because the local host has exhausted some buffer. Further, the *cwndmin* feature likewise assesses the occupancy of the local TCP buffer. We excluded these because they did not depend on the state of the remote host or the network path. Further, [9] shows these to be not particularly useful in detecting spam. In addition to changing the set of features note that some of the names of the features have been changed to aid clarity.

3. Ground Truth Determination

Assessing any automatic classification technology requires the establishment of ground truth, which is our case means pre-determination of which messages are spam and which are ham. Establishing ground truth presents a difficult challenge in large datasets such as ours. Hand-labeling is the best way to establish ground truth as was done for the 18K messages used in [9]. However, using hand labeling to generate ground truth does not scale in terms of number

of messages, due to the time consuming manual process, or across multiple users, due to ethical concerns relating to reviewing others' email. Therefore, hand labeling more than 1.4 million messages from across ICSI in our dataset is not feasible. Further, using a single spam filter to automatically label messages [17] would potentially bias the detection techniques we develop by making our detection mimic a known detection scheme and fall prey to any blind spots present in the given tool. Note that [17] is not developing new detection methodology, but rather characterizing spam traffic within the context of a working operational setup and therefore the use of a single strategy is less problematic and logistically understandable.

Consequently, to produce *approximate ground truth* we developed an automatic labeling procedure that involves combining the predictions of four open source spam filters, each with (largely) independent methodologies for classifying a message, followed by manual verification of a sample large enough to yield tight confidence intervals, but significantly smaller than the full set, so manual checking of this sample was possible. This sample was collected from January 2009, and the checked messages were not further used in our experiments. We employed the following tools:

SpamAssassin: We used SpamAssassin 3.1.7 (current when we started our investigation) [4, 37]. SpamAssassin includes a series of tests and signatures that it uses to score a message—with a threshold then used to determine if the message is spam or ham. We used the default threshold of 5.0. SpamAssassin can optionally use a Bayesian learning algorithm and several non-local tests whereby remote databases are queried. We did not use either of these features (in an attempt to stay independent of the other tools).

SpamProbe: We use SpamProbe v1.4b [10] which is a naïve Bayesian analysis tool that works on both single and two-word phrases in email. SpamProbe calculates the probability that the given message is spam and classifies messages with a probability of at least 0.6 (the default) as spam.

SpamBayes: We use SpamBayes v0.3 [25] which is another naïve Bayesian analysis tool. As with SpamProbe this tool tokenizes the input and calculates the probability a message is spam. While we use two basic Bayesian tools we note that the details of the implementation matter and these tools disagree for 30% of the messages in our January 2009 dataset (as detailed below).

CRM-114: We use the Orthogonal Sparse Bigrams classifier in CRM-114 v20060704a-BlameRobert [1]. CRM-114 is a Bayesian filter that calculates the probability a message is spam using features corresponding to pairs of words and their inter-word distances. We run CRM-114 using the standard defaults except for the "thick_threshold". CRM-114 classifies messages as spam, ham or unsure. While operationally useful, since we are trying to determine ground truth, we force CRM-114 to eliminate the unsure classification and make a decision by setting the thick_threshold to zero.

The latter three tools above need to be trained before they can classify messages. That is, the tools have no built-in notions of ham and spam, but must be given labeled samples of these two classes to learn their characteristics. In our study we used the TREC email corpus from 2007 [12] (the latest available when we started this project). The corpus includes 25K ham messages and 50K spam messages. The messages that comprise the corpus arrived at one particular mail server over the course of three months. A hybrid of automated techniques and hand labeling was used to categorize the messages. This corpus is far from ideal in that the TREC corpus is dated relative to the data we collected for our study. This is problematic because spam is a constant arms race between spammers developing new techniques to evade filters and filters becoming increasingly savvy about spamming techniques. Further, we train each tool one time, whereas in a more realistic setting the tools could be regularly trained on new types of emails (either automatically or by being corrected by users, depending on the tool). That said, as illustrated below a manual check of our labels indicates that our overall procedure for obtaining approximate ground truth is accurate despite the less than ideal starting point.

An additional note is that the tools chosen do in fact offer differing views on our email corpus. Table 3 shows the pair-wise disagreement between tools used in our study on messages collected from January 2009. Even the best aligned tools (SpamBayes and CRM-114) differ in 19% of the cases. This illustrates that we have chosen heterogeneous tools—which we believe is a key to the following procedure for generating ground truth because the tools can effectively check each other.

Our strategy begins with each tool making a judgment on a given message and then we combine these judgments to derive the overall conclusion. How to use results from multiple classifiers has been the subject of previous work (e.g., [23]), and our initial inclination was to use one of the existing methods, namely, the "majority voting" scheme

Tool 1	Tool 2	Disagreement
SpamAssassin	CRM-114	37%
SpamAssassin	SpamBayes	39%
SpamAssassin	SpamProbe	23%
SpamBayes	CRM-114	19%
SpamBayes	SpamProbe	30%
CRM-114	SpamProbe	31%

Table 3: Pair-wise percentage of disagreement between spam detection tools used in developing ground truth in January 2009.

whereby a message was classified as spam in the ground truth if three or four of the tools judged it as spam (breaking the ties in favor of spam since there is more spam in the corpus). Otherwise, we considered the message ham.

To determine whether our ground truth is sound we manually spot checked a set of messages from January 2009. Ideally, we would simply have picked a random subset of all messages and verified the classification in our ground truth. However, since we are working with real email the content could be sensitive and therefore we were not able to use the ideal procedure. In particular, we did not manually examine messages that the spam filters universally agreed were ham (with an exception being sketched below) due to concerns over looking through users' email. However, we determined that if at least one tool indicated that a message was spam then the message was likely to not be sensitive and therefore manually looking at a random sample from across the institute was reasonable. Therefore, we chose two kinds of messages for manual checking: (i) those messages that could be—in an automated fashion—clearly determined to involve the third author of this paper (the only author with access to the raw messages) and were marked as ham by all spam filtering tools (this yields 646 of the 35K such "clearly ham" messages in the corpus) and (ii) a uniformly random sample of 2% of the messages[6]) where at least one tool classified the messages as spam (this yields 920 of the 47K messages marked as spam by at least one tool in our corpus). We found that 5 of the 646 "ham" messages to actually be spam. Of the 920 spam-suspected messages selected, we found that 912 were indeed spam. Thus, our spot-check set contains $646 - 5 + 8 = 649$ ham and $920 + 5 - 8 = 917$ spam messages.

Using this spot-checking procedure, we found that ground truth generation based on majority voting is inadequate. Of the 917 spam messages in our spot-check set, 5 were not classified as such by any tool, 218 were classified as spam by only one tool, 362 were classified as spam by only two tools, 250 that were classified as spam by three tools and the rest were classified as spam by all four tools. Thus, majority voting, even after breaking ties in favor of spam, would result in unacceptably high false negative rates (about $(218 + 5)/917 = 24.3\%$).

On the other hand, consider the following strategy: label an email as spam if *at least one tool* judges it as spam, and label it as ham otherwise (i.e., if all tools agree it is ham). This strategy has an $5/917 = 0.55\%$ estimated false negative rate (FNR; spam classified as ham), and a $8/(646 - 5 + 8) = 1.23\%$ estimated false positive rate (FPR; ham classified as spam), with the standard deviation of the estimates being 0.07% for FNR and 0.11% for FPR. The overall estimated error rate is then $13/(917 + 649) = 0.83\%$ with standard deviation 0.09%. To verify that this is not an artifact of the samples being chosen from January 2009, we carried out similar verification using messages from September and May and obtained comparable results: 0.3% FNR, 2.39% FPR in September and 0.28% FNR, 1.56% FPR in May. Therefore, based on our samples, our labeling strategy is accurate (well within 3%).

In retrospect the any-one approach is intuitive in that off-the-shelf spam filters are conservative in classifying a message as spam because the consequence of a false positive (blocking a legitimate mail) is more severe than that of false negative (allowing a spam to reach the user). This causes the errors to go in the direction of identifying less spam. The manual checking shows that being more aggressive in combining the filters' judgments can mitigate this effect and reduce the error significantly, which is important since the cost of inaccuracy is symmetrical when determining ground truth (as opposed to an operational setting).

We note that we do not use the results of the manual inspection from January 2009 to train our classifiers. In fact, the messages chosen for manual inspection were removed from the remainder of the analysis in our experiments. Further, note that we are developing ground truth by observing *content features*. Our experiments in the following sections involve classifying messages based on *network features*. We believe that even if there is a small error in our ground truth that error does not bias our results because the content and network features are largely independent.

It is interesting to speculate whether this procedure could be used more generally. That is, could using existing

[6]Using 2% as our sampling rate is somewhat arbitrary and is based on both trying to keep the potential exposure of sensitive messages low while also choosing a feasible number of messages for manual processing.

tools with limited manual verification allow for large scale automatic labeling of email? This has implications for retraining spam classifiers as well, which has to be done periodically. We defer a systematic study of the general utility of this procedure, considering other tools and other methods of combining predictions, to future work.

4. Methodology

In this section, we describe the algorithms and empirical methodology we use to answer the questions of interest.

4.1. Learning Algorithms

We use two algorithms in our experiments: decision trees [34] (from Weka [40]) and Rulefit [15].

We use *decision trees* [26] as our primary algorithm. These are commonly used in machine learning. They work using the idea of *recursive partitioning*: to build a decision tree, the learning algorithm iteratively chooses a feature (for us, this is a network characteristic) based on some metric and uses the feature to partition the training examples. The metric we use is a common metric called *information gain*, which chooses the feature that gives the most information about the class label, as measured by the reduction in the entropy of the class label distribution. This is repeated until a partition only has examples from one category (for us, until we get a partition consisting only of ham or spam), which becomes a leaf node annotated with that category. To classify an example, it is "threaded" down the tree by checking each test until it reaches a leaf. At this point it is assigned the category associated with this leaf. Decision trees have several advantages for our purposes. First, they are computationally efficient: learning a tree takes on average $O(mn \log m)$, where m is the number of examples and n is the number of features, and classification is just $O(d)$, where d is the depth of the tree. This is important for us because we are working with large data sets. Second, decision trees are easy for people to understand and reason about. Once constructed, we are able to inspect the trees and derive insights applicable to our problem. Third, it is easy to modify the tree construction procedure to incorporate *differential misclassification costs*, that is, to produce trees that take into account that mistaking ham for spam is in general a worse error than mistaking spam for ham (we leverage such models in constructing an operationally useful system in Section 6). Finally, preliminary inspection of our data revealed that some of our features have "threshold effects." Suppose for a feature f, if $f \leq x$, a message is more likely to be spam; if $x \leq f \leq y$ a message is more likely to be ham, if $f \geq y$, a message is more likely to be spam. In such a case, a decision tree with successive tests for $f \leq x$ and $f \geq y$ will easily lead to the desired classification.

To build decision trees from our data, we used the implementation of decision trees in the machine learning toolkit Weka [40]. This implementation is called "J48." We ran this with the default parameter settings of "-C 0.25 -M 2" in all cases. The parameter C denotes a confidence threshold used for pruning the decision tree after construction and the parameter M denotes the minimum number of examples in any leaf of the tree (i.e., once the number of examples reaches this value, the partitioning process stops). These parameters have not been tuned to our application. It is possible that improved results might be obtained if these parameters were tuned further. A part of a decision tree learned on data from May 2009 is shown in Figure 3(a) (we discuss this tree in more detail in Section 5.1).

Rulefit (also used in prior work [20]) constructs a linear classifier that uses the primitive features as well as Boolean tests on feature values as predictors. These tests are similar to the ones used in the internal nodes of decision trees and evaluate to $+1$ or -1 for each example, depending on whether the corresponding test is true or false. To build Rulefit models from our data, we used a previous implementation in R with default parameters [3].

4.2. Empirical Methodology and Metrics

To evaluate the performance of our tree models, we measure several metrics using stratified 10-fold cross validation (implemented in Weka [40]). This procedure is standard machine learning methodology and partitions the data into 10 disjoint "folds" that are *stratified* based on the class label, i.e. the proportion of ham to spam in each fold is the same as the overall proportion. In each of the ten iterations, we build a classifier on nine folds, then evaluate the tree on the held out fold (the "test set"). We report the average value of several metrics on the 10 folds. First, we report predictive accuracy on the test set. This is a general measure of agreement between the learned trees and the ground truth (as established in Section 2). Further, we report the true positive rate (TPR), which is the fraction of spam that was correctly identified as such by our models, and the false positive rate (FPR), which is the fraction of ham that was erroneously classified as spam. Note that we treat spam as the positive class. Finally, we construct Receiver Operating Characteristic (ROC) graphs by thresholding a probability measure reported by the learning algorithm. This probability indicates, for each prediction, the "confidence" of the classifier in the prediction. To plot an ROC graph,

	Packet Features (with AS-Spamminess)			Packet Features (without AS-Spamminess)		
Month	**Acc**	**TPR**	**AROC**	**Acc**	**TPR**	**AROC**
May/09	0.663	0.464	0.783	0.594	0.348	0.768
Jun/09	0.564	0.266	0.785	0.454	0.072	0.652
Jul/09	0.563	0.312	0.805	0.438	0.108	0.666
Aug/09	0.543	0.280	0.773	0.479	0.172	0.660
Sep/09	0.586	0.322	0.783	0.497	0.169	0.663
Oct/09	0.640	0.406	0.779	0.488	0.145	0.683
Nov/09	0.507	0.319	0.660	0.436	0.218	0.826

Table 4: Results for packet features with and without AS-spamminess. The TPR is reported at 1% FPR for the "with AS-spamminess" case and at 0.2% FPR for the "without AS-spamminess" case. Results include months from May/09 to Nov/09.

we threshold this confidence measure and plot the TPR against the FPR at each threshold value. This graph gives us a detailed view of the behavior of the classifier and allows us to choose a suitable operating point that trades off an acceptable false positive rate against the true positive rate. Following prior work [20], we select operating points so that FPR would not exceed 0.2%: since misclassifying ham as spam is likely to be much more expensive than the reverse, a low FPR is required for an effective spam filter. As a summary statistic for the ROC graph, we report the area under ROC (AROC), which has been shown to correct several weaknesses of accuracy as a quality metric [32].

5. Utility of Packet and Flow Features

In this section, we empirically answer the questions we posed in the introduction, repeated here for convenience:

1. What is the added value in terms of accuracy by classifiers based on packet features and network features?
2. Does the added value depend on the machine learning algorithm used to process these features?
3. How do the layers interact with each other? Can a later layer "take over" if an earlier one becomes inaccurate?
4. Does the added value stay stable over time and across network events?
5. What features are primarily responsible for the value addition?

The last question we posed, that of computational cost of the pipeline vis-a-vis the state of the art, is addressed in the next section.

5.1. Value Added by Packet Features at the Enterprise

Our first question is whether the second layer in our spam pipeline adds value, i.e. is effective at identifying spam accurately in a stand-alone enterprise mail service using only its own vantage point. To study this, we run *cost-sensitive* decision trees on each month's data from May 2009 to April 2011, using only the single packet features (in the upper part of Table 2) to describe each message that passed through the DNS blacklist.

The choice of cost sensitive trees is because when running these experiments, we observed that the unmodified, cost-insensitive trees had an average FPR of 18.74% across the months from May/2009 to Nov/2009, and our attempts to reduce it by thresholding the confidence drastically degraded the TPR. This is clearly not operationally usable. To remedy this, we produce cost sensitive trees that penalize FP errors more than FN errors. Cost sensitive trees need a *cost ratio* as input. We fix a ratio of 175:1 in our experiments, which allowed us to reduce FPR to below 1% while maintaining significant TPR. In principle, it is possible to search for the best cost ratio when retraining the classifier instead of using a fixed cost ratio value, and the performance figures we report here could possibly be improved further if such a search was performed. We chose not do this in the current study since we compare many classifiers over multiple months; if each classifier had a different cost ratio, this would complicate the comparison of results across different classifiers and across time.

Once a tree has been produced with this cost ratio, we classify new messages as follows. We use the ROC graph of the learned classifier on the training data to pick an operating point, where the FPR is at the desired value. Effectively, this determines a confidence threshold, so that messages classified as spam with probability above this threshold are highly likely to be spam. In this case, messages classified as spam with probability below this threshold are effectively treated as ham; they cannot be discarded since that might raise the FPR beyond the desired rate, so they will be sent to the user's inbox (or the next step in our pipeline).

A partial set of the results we obtain are shown in the first three columns of Table 4. While the FPR drops significantly compared to the non-cost-sensitive case, it does not reach 0.2% —even when increasing the cost ratio

	Packet Features			Flow Features			All Features		
Month	**Acc**	**TPR**	**AROC**	**Acc**	**TPR**	**AROC**	**Acc**	**TPR**	**AROC**
May/09	0.594	0.348	0.768	0.583	0.330	0.709	0.632	0.410	0.759
Jun/09	0.454	0.072	0.652	0.555	0.244	0.689	0.546	0.230	0.686
Jul/09	0.438	0.108	0.666	0.547	0.281	0.701	0.564	0.308	0.748
Aug/09	0.479	0.172	0.660	0.538	0.266	0.691	0.576	0.327	0.754
Sep/09	0.497	0.169	0.663	0.558	0.270	0.681	0.582	0.309	0.711
Oct/09	0.488	0.145	0.683	0.488	0.145	0.633	0.529	0.215	0.675
Nov/09	0.436	0.218	0.826	0.417	0.191	0.823	0.449	0.236	0.788
Dec/09	0.443	0.180	0.687	0.431	0.162	0.593	0.468	0.217	0.702
Jan/10	0.467	0.197	0.713	0.442	0.158	0.625	0.475	0.208	0.617
Feb/10	0.608	0.358	0.806	0.549	0.261	0.675	0.623	0.382	0.806
Mar/10	0.346	0.126	0.843	0.371	0.159	0.840	0.385	0.178	0.881
Apr/10	0.405	0.163	0.837	0.388	0.138	0.799	0.444	0.218	0.858
May/10	0.376	0.003	0.501	0.448	0.119	0.599	0.452	0.125	0.595
Jun/10	0.488	0.210	0.786	0.445	0.143	0.755	0.496	0.223	0.787
Jul/10	0.579	0.070	0.564	0.649	0.226	0.652	0.668	0.267	0.712
Aug/10	0.336	0.021	0.520	0.407	0.127	0.638	0.424	0.152	0.616
Sep/10	0.797	0.725	0.882	0.794	0.721	0.861	0.806	0.737	0.871
Oct/10	0.779	0.706	0.880	0.774	0.700	0.851	0.802	0.736	0.892
Nov/10	0.818	0.757	0.896	0.804	0.739	0.870	0.806	0.741	0.878
Dec/10	0.801	0.738	0.888	0.791	0.725	0.863	0.798	0.734	0.881
Jan/11	0.392	0.191	0.889	0.773	0.698	0.866	0.791	0.722	0.901
Feb/11	0.785	0.708	0.894	0.800	0.729	0.871	0.804	0.734	0.894
Mar/11	0.803	0.732	0.886	0.790	0.715	0.856	0.795	0.722	0.866
Apr/11	0.834	0.775	0.904	0.805	0.735	0.867	0.832	0.773	0.904

Table 5: Results for decision trees using packet features (left), flow features (middle), all features (right). Packet features exclude AS-spamminess in all cases. TPR is reported at 0.2% FPR.

significantly. Analysis of the classifiers reveals that the AS-spamminess feature—from the training data—is chosen as highly predictive in each case. However, it appears to be less predictive on the test data, probably because even using 90% of our data from a month does not result in a comprehensive database. Thus, even though our results confirm previous findings (e.g., [35]) that AS origin differs for spam and ham, we find the predictive of this feature in an enterprise environment limited.

Next, we remove AS-spamminess from the analysis, forcing our classifiers to use other features. These results are shown in the last three columns of Table 4 and the whole set of results from May/09 to Apr/11 is shown in the Table 5. In this case, we are able to obtain an FPR of 0.2%. We then observe that though the classifiers produced achieve a low FPR, their TPR varies significantly over time—occasionally less than 1% and reaching to over 70% in some months. A lot of the TPR results, though, are not high, between 10% to 40%.

We investigate the months with exceptionally high detection rate later in Section 5.3. With respect to the months with exceptionally low detection rate, one possibility is that the chosen cost ratio of 175:1 is too high in some months and it is preventing messages from being classified as spam. We tested this hypothesis on May, July and August 2010, three months where performance was exceptionally poor. Indeed, lowering the cost ratio from 175:1 to 100:1 resulted in the significant TPR improvements in some cases at the same target 0.2% FPR. For July 2010, TPR improved to 26% from 7%, bringing it generally in line with prevailing results for other months. In the cases of May and August 2010, adjusting the cost ratio increased the TPR from 0.3% to 8% in May and from 2.1% to 9.8% in August. Decreasing the cost ratio further to 75:1 brought no additional overall improvements: July's detection rate slightly improved to 27.4% but May's and August's detection rate degraded to, respectively, 7.6% and 8.2%. In other cases, such as January 2011, *increasing* the cost ratio further to 225:1 improves TPR to 34.5% at 0.2% FPR, while decreasing it to 100:1 reduces TPR even further to 12.3%.

The above result provides support to the idea mentioned earlier of searching for the optimal cost ratio during model retraining. However, even though the TPR increases, it is still not high. These results indicate that, even if cost ratio is varied, *on their own,* single-packet features may not be very good detectors of spam in the single enterprise context. While this finding may appear to contradict the previous study by Hao et al. [20], that study was performed from the perspective of a large aggregation point and without DNS blacklist pre-filtering.

While TPR for packet features is generally low, for the period from September 2010 to April 2011, it reaches above 70% (except January 2011). The results for these months seem to agree with prior work [20], where a TPR of

	Packet Features		
Month	**Acc**	**TPR**	**AROC**
May/09	0.600	0.359	0.909
Jun/09	0.492	0.139	0.882
Jul/09	0.478	0.171	0.906
Aug/09	0.506	0.216	0.905
Sep/09	0.509	0.189	0.892
Oct/09	0.515	0.191	0.901
Nov/09	0.421	0.197	0.884

Table 6: Results for packet features *without* AS-spamminess using Rulefit algorithm, The TPR is reported at 0.2% FPR.

70% was achieved at FPR=0.44% (without a blacklist). [7] In Section 5.3 we investigate the "good" months (with TPR of 70% or more) to understand why those months appear to be outliers in our data.

In Figure 3(a), we show a fragment of a decision tree of packet features, obtained using the data from May 2009. This is not the full tree; in this fragment, each leaf is labeled with "majority class", i.e., the category associated with the majority of messages in that leaf.

- The most important discriminant in this tree is *geoDistance*, the roughly measures the geographical location of the message's origin. Utility experiments (presented in section 5.6) also indicate the importance of this feature. This feature remains in the top of the tree for other months also, although the exact threshold value changes with the change of the geographical origin of spam over time.

- The next feature in the tree is the *ttl* feature, which is derived from both the operating system of the remote host and how many router hops a message has gone through. This may seem like a counter-intuitive feature to use near the root of the tree. However, note that the starting IP TTL value, *maxTTL*, varies across operating systems: Windows XP and related operating systems use an initial TTL of 128, while Linux uses 64. Accordingly, in our data, the *ttl* feature is strongly correlated with the operating system. The actual number of hops traversed by the traffic is seemingly dominated by the difference in initial TTL values. Therefore, the test for $ttl < 98$ is effectively a determination between Windows or Unix-based remote hosts. If this test is False, then the originating machine was likely a Windows-based machine and the message is likely spam [8].

- The *neighborDist* feature calculates the average network distance (in IP address space) from the remote sending host to its closest 20 neighbors. Legitimate SMTP servers usually do not have other 20 SMTP servers nearby, thus a large value of neighbor distance suggests a legitimate remote host.

5.2. Effect of the Learning Algorithm

We next address the question: what if we used a different learning algorithm? Perhaps the low detection rate achieved by packet features in some months is not due to the features themselves, but merely to the inability of the algorithm to exploit the information in those features. To understand this, we used a second algorithm, Rulefit [15], to learn a classifier from our data. Rulefit produces a linear classifier that is augmented by Boolean threshold features. The results with Rulefit are shown in Table 6. From these results, we observe that both decision trees and Rulefit exhibit quite similar results in our experiments. Thus the lack of utility of the single packet features in certain months appears to be not a function of the learning algorithm they are used with, but purely a consequence of the limited information they convey about the message in this setting.

To get a better understanding of why changing the algorithm might not make a significant difference, notice that it is intuitively plausible that looking at a single packet reveals limited information about the message, and one can only construct few features from this information (we use seven). This set of features generally describes the characteristics of a group of hosts sending email. But unless the granularity is extremely fine, such a group will sometimes send ham and sometimes spam. We therefore found many instances where messages were described by identical packet features but had opposing labels, i.e., some were labeled as spam and some as ham. For example, in May 2009, from 47,653 messages in that month, we identified 21,638 patterns of packet features. Of these, 485 patterns have both

[7] In later sections we present results without the blacklist and also using the Rulefit algorithm that was used in [20].

[8] It is important to remember that the presented tree is just a part of the full tree, which contains over 100 nodes. The full tree does not judge *all* email messages with $ttl > 98$ to be spam, although the *majority* of such messages are spam.

	Packet Features(Pre-Blacklist)		
Month	**Acc**	**TPR**	**AROC**
May/09	0.824	0.808	0.967
Jun/09	0.346	0.290	0.964
Jul/09	0.467	0.421	0.962
Aug/09	0.714	0.691	0.969
Sep/09	0.644	0.612	0.967
Oct/09	0.520	0.462	0.957
Nov/09	0.709	0.670	0.943

Table 7: Results for packet features *without* AS-spamminess using Rulefit. The TPR is reported at 0.44% for this experiment for a direct comparison with prior work [20].

spam and ham messages and the ratio of ham over spam+ham in each of these 485 pattern varies between 0.25 and 0.75, suggesting ham or spam is not just few outliers. Moreover, these 485 feature patterns appear in 7,385 (15% of 47,653) messages. Similarly, this phenomenon seems to be at least partly responsible for the sudden dip in TPR for January 2011, because the fraction of such confusing patterns in that month was 16.1%, whereas in December 2010 it was 9% and in February 2011 it was 13.9%. This clearly is problematic for any classifier and may increase the FPR and lower the TPR. On the other hand, if finer feature granularity is used (e.g., imagine using the IP address—with a range of 4 billion values—as a feature), then significantly more data will need to be collected to train a useful classifier. This appears to be a fundamental limit of single packet features for spam detection at the enterprise, that cannot be overcome by simply changing the machine learning algorithm being used.

5.3. Interaction between Packet Features and the DNS Blacklist

We next turn to the question: how do the DNS blacklist and the classifier based on packet features interact? One issue here is that it is possible that the DNS blacklist is somehow filtering the "easy" cases for the packet features. In this case, using the DNS blacklist as the first layer may be masking the true discriminatory power of the packet features. Further, if this is the case, it may suggest that the packet features may be able to "take over" if the DNS blacklist somehow stops working. Partial evidence for such a scenario is found in prior work [20], which evaluated packet features without a blacklist and reported a TPR of 70% at FPR of 0.44% [9], which according to the authors is similar accuracy to DNS blacklists. Finally, if this is true, might it in part explain the accuracy jump we see from September 2010 to April 2011 (Table 5)?

To understand this interaction better, we first train a (cost-sensitive) Rulefit classifier and evaluate it on *all* messages, including those blocked by ICSI's operational DNS blacklist setup ("DNSBL"+"Ham"+"Spam" in Table 1). We can do this since the first SYN packet is still available for analysis even for the DNS-blacklist-blocked messages. Of course, the message bodies are not available in this case, so our method for determining ground truth does not apply. Instead we trust the DNS blacklist to be accurate and label all messages filtered by it as spam. We use Rulefit in this experiment, rather than trees, in order to directly compare our results to that of prior work [20], which also used this algorithm. We further pick an operating point with the same FPR of 0.44% reported in [20]. The results are shown in Table 7.

From these results, we observe that as before, there is significant variation across months. However, the TPR of packet features increases significantly in all months in this case over the post-blacklist data. For some months, our results are comparable to, or even exceed, the 70% TPR reported by Hao et al. , though for other months, we find the TPR is far lower (e.g., 29% in June). One possible reason for this variability is that it is due to the enterprise's vantage point. Another possibility is that it is a fundamental property of packet features, and was not observed by Hao et al. since their data was collected over a period of only 14 days.

These results suggest that at least in some cases, the absence of the DNS blacklist may lead to high detection rates for packet features. Could this in any way explain the period from September 2010 to April 2011, where we found packet features to have high detection rates? To understand this, we analyze the period July to October 2010, during which packet features first show poor performance followed by a sudden sharp jump in TPR. Table 8 shows, for this period, the total spam received at ICSI, the spam blocked by the blacklists, the spam blocked by the combined blacklist and packet feature pipeline, followed by the predicted performance if the packet features had been used on their own without the blacklist.

[9] This was not done for a single enterprise as in our study, but in an aggregate setting where data was collected from thousands of organizations.

Month	Total Spam	DNSBL	Packet Features(Post-Blacklist)	DNSBL + Packet Features	Packet Features(Pre-Blacklist)
Jul/10	149K	120K(80%)	7.0%	81.9%	41.5%
Aug/10	146K	102K(67%)	2.0%	67.9%	15.9%
Sep/10	86K	40K(47%)	72.5%	85.3%	37.1%
Oct/10	81K	39K(48%)	70.6%	84.8%	27.2%

Table 8: The detection rate of DNS-Blacklist and Packet Features, and their combined effect, presented as TPR which are reported at 0.2% FPR.

Month	Top Countries Sending Spam
Jul/10	US (41.2%), Germany (15.8%), China (7.1%), Vietnam (3.8%)
Aug/10	Germany (54.4%), US (20.9%), China (3.9%), Vietnam (2.7%)
Sep/10	Germany (74.3%), US (18%), China (1.1%), France (0.8%)
Oct/10	Germany (72.0%) ,US (18.9%), China (1.4%), Canada (0.6%)

Table 9: Top-4 countries that sent spam to ICSI SMTP servers and their respective spam percentage

This table reveals that there is less spam received in each month beginning from September 2010. From that period on, the amount of spam is usually on the order of 100K or less, as compared to around 150K before this period. Further, while pre-September, the blacklist is able to block around 75% of the incoming spam, from September onwards this drops to 50%. A possible explanation for these observations is that around this time, law enforcement activities resulted in the shutdown of various botnets in the US. In an article [2] published by the Kaspersky lab, it is noted:

> The events that took place during the last six months of 2010 included the shutdown of the command centers for the Pushdo, Cutwail and Bredolab botnets and criminal proceedings started against partnership programs that operated mass mailings of pharmaceutical-themed spam. These developments led to the amount of unwanted correspondence being halved if the average volume of global spam in the last week of October is compared with those levels recorded in mid-August, 2010.

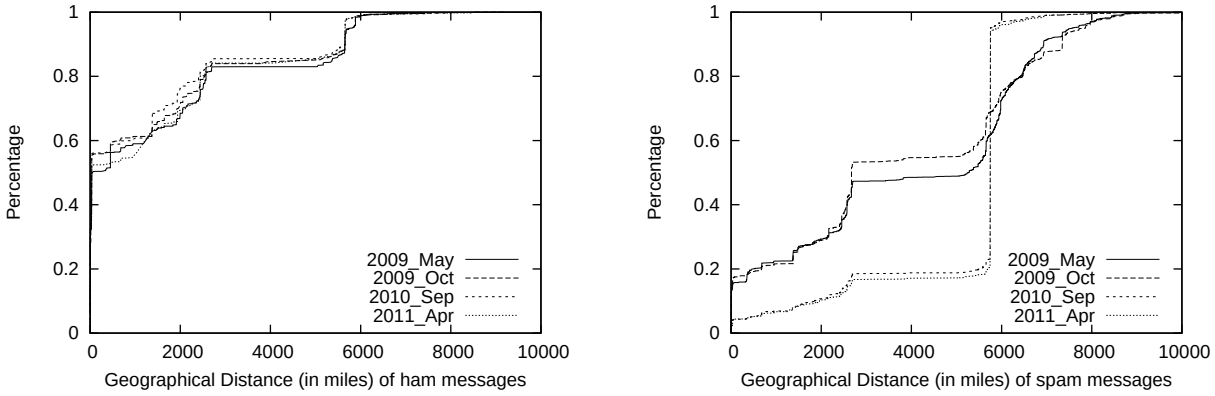
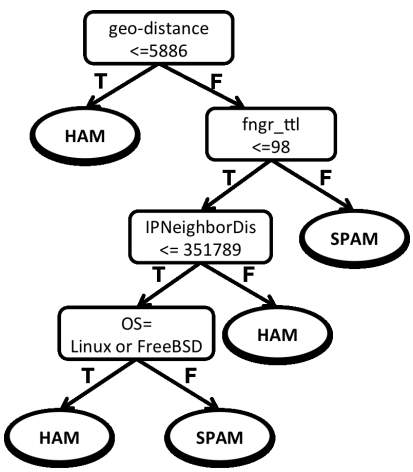
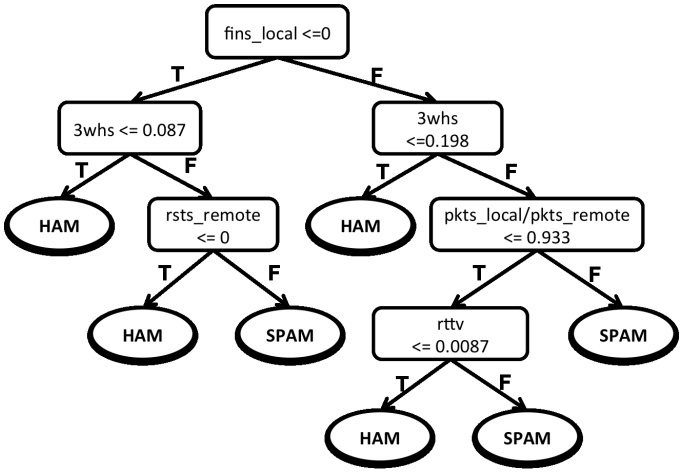


Figure 2: CDF plots for values of *g*eoDistance of ham and spam. Left: Ham in May/09, Oct/09, Sep/10 and Apr/11. Distributions of the four are similar. Right: Spam messages from the same four months. Distributions of last two months are very different from the previous two.

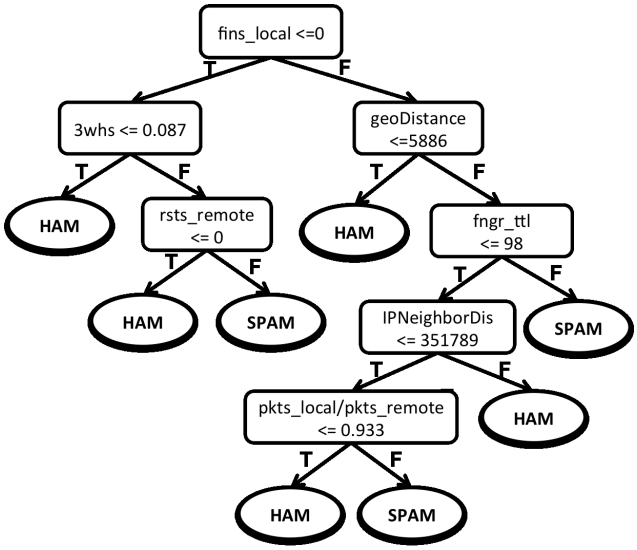
As we see, however, this activity not only reduced the volume of spam (which is good), but also reduced the effectiveness of the DNS blacklists (which is bad). Our data indicates that at this point, there is a change in the distribution of the *geoDistance* feature for spam messages (Figure 2 right), while the distribution of the geoDistance feature for ham messages remains stable across months (Figure 2 left). This indicates that when these botnets were "shut down," some major sources of spam shifted away from USA to Europe and Asia. Table 9 shows the distribution of countries of origin of the spam messages (estimated using GeoIP database [24] snapshots in corresponding months) during this period that passed through the DNS blacklist. As a result of this shift, the DNS blacklists are no longer as effective. However, *geoDistance* is a packet feature in our classifiers. By analyzing the learned trees, we observe that after September 2010, the top feature in the trees is *geoDistance*, and more than 70% percent of spam originates from a distance of 5730 to 5880 miles to the ICSI servers (this correlates well with Germany), while only a small percentage of ham messages are from that distance region (this pattern does not hold before this point in time, and



(a) Fragment of a tree using packet features.



(b) Fragment of a tree using flow features.



(c) Fragment of a tree using packet + flow features.

Figure 3: Three decision trees learned from the May 2009 data. The leaves in the figure are not the actual leaves in the full tree, and thus their labels reflect the majority class among all messages at that point.

geoDistance is not as effective then). Thus in this case the packet features are able to "take up the slack" when the DNS blacklist becomes less effective. To further support this point, the combined TPR of the blacklist together with the packet feature based classifier remains fairly stable and high across this period, though each by itself is fluctuating rather sharply ("DNSBL + Packet Features" column, Table 8).

As a final interesting point, we note that a weak blacklist is still better than no blacklist in this pipeline. When we produce packet-level classifiers using *all* messages in this period, we observe that their accuracy (last column of Table 8) is always significantly lower than the accuracy of the combined DNS blacklist/packet feature pipeline. Thus, unlike the conclusion made in [20] for a large aggregation point, in a single enterprise environment, it seems unlikely that a packet feature based classifier can simply replace a DNS blacklist. Each has something to contribute and supplements the other, so that the combined system is more accurate than either on its own, and the accuracy remains roughly stable over time.

5.4. Value Added by Flow Features at the Enterprise

We next turn our attention to *flow features* of messages. These features are computed from the entire packet trace comprising the message. [10] This is obviously more computationally expensive than computing features just from one packet. Prior work [9] indicates that such features could be useful in detecting spam. We reappraise this result with our much larger dataset, and further ask the question: what *added* utility do such features offer beyond the DNS blacklist and packet features?

To answer these questions, we consider two experimental scenarios. We first produce decision trees from only the flow features in the lower half of Table 2 and use these to classify messages that have passed through the DNS blacklist. Then we add the packet features (without AS-spamminess) to this list and use the full set in the same way. Thus, rather than two separate layers producing two classifiers, this produces a single classifier using both sets of features. This allows us to directly compare the informativeness of the different feature sets, since the tree will pick the more informative features to use. For these experiments, we use decision trees as the learning algorithm. Again this is because our prior results show that any patterns we see generalize to Rulefit as well, yet Rulefit is more expensive to run while trees are easier to interpret. The results are shown in Table 5, and fragments of trees found using only flow features and using all features are shown in Figures 3(b) and 3(c).

Comparing results for packet features and flow features in Table 5, we first observe that in most months, the flow features by themselves are at least as accurate as packet features. This is intuitively plausible because looking at the complete packet trace should yield at least as much information as just the first packet. However, we also see that large, significant differences between packet and flow feature accuracies are rare. This may indicate that while there is extra information, that information may not generally provide significant additional detection power, but may serve to "correct" packet features when those features are not accurate (e.g. January 2011), just as packet features serve to correct the DNS blacklist when that performs poorly. To further support this, we note that when using all features together to produce a single tree, the resulting performance is generally better than using either the packet or flow features by themselves, though generally not by much. The combined feature set is effective at removing between 20% to 77% spam in absolute terms. A question is whether an effective pre-filter can be constructed using such a classifier, so that only messages that cannot be classified with high probability are sent to computationally expensive content filters. We consider this question in more detail section 6.

In Figure 3(b), we show a fragment of a decision tree of flow features, obtained using the data from May 2009.

- The top feature used in this tree is number of fin packets sent by local monitored server. The absence of a fin packet in the trace likely means the connection was ended abruptly.
- Another important discriminant in this tree is the length of the initial 3-way handshake, importance of this feature has also been noted previously [9]. We have also observed that for connections with a fin, threshold values near 0.198 seconds associated with this test, as shown in the right subtree, seems quite stable across time in our dataset. This may indicate a correlation with the geographical origin of spam relative to the monitored servers—observe that this test is "replaced" by a geoDistance test in the tree that uses both flow and packet features (Figure 3(c)).
- For connections without a fin, a very small 3whs means the messages are likely to be ham. This again shows correlation between the email label (spam/ham) and the geographical origin of the message and/or congestion on the path. For the higher 3whs values on the left subtree, the next test is to check the reset feature. If there was a reset from the remote host to terminate the connection we find it likely this message is spam; if not, it is more likely to be ham.
- The next test on the right subtree following the 3-way handshake is the ratio of outgoing to incoming packets. The expected case is for this to be around 0.5 due to delayed acknowledgments [7] which calls for one ACK to be sent for every second data segment received (unless a second data segment does not arrive within a small amount of time). However, additional factors can move this ratio towards 1, including (i) some operating systems' acknowledgment strategies that transmit more ACKs early in the connection to help open TCP's congestion window, (ii) the delayed ACK algorithm that sends ACKs without waiting for second data packet when this packet does not arrive within certain time (commonly 200ms), and (iii) because smaller transfers do not

[10] An interesting possibility we do not pursue in this work is to compute these features from partial traces. In this way network features form a continuum between single packet features and "full" flow features computed from the entire trace.

have many opportunities to aggregate ACKs. Indeed, delayed ACKs are generally not used in the connection management phase, hence the corresponding traffic has the ratio of 1. The SMTP control traffic is symmetric (with requests and responses) and therefore also has the ratio of 1. When the message is small there are generally few data packets and therefore coalescing ACKs is either non-existent or done in small enough numbers that the overall ratio is still closer to 1 than to 0.5. In our January 2009 data we find that when the ratio is closer to 1, 85% of the messages are below 6K in size and all but 1 are less than 11K in size. On the other hand, if the ratio is closer to 0.5, there is a wide distribution of message sizes, with several messages over 100KB. Therefore, the ratio turns out to be largely a proxy for message size. The tree considers "small" messages more likely to be spam given all the preceding tests of other features.

- When the ratio tends more towards 0.5, the tree checks the *rttv* feature, which is the variance of round trip time, A small *rttv* indicates the message has gone through a stable route with low variance, it is more likely to be ham, otherwise it is more likely to be spam.

In Figure 3(c), we show a fragment of a decision tree using both packet and flow features, obtained from May 2009. This tree uses both flow and packet features, indicating that neither set completely supplants the other. The number of fin packets from local SMTP server, a flow feature, is chosen as the top feature. The left subtree only uses flow features. The right subtree starts with *geoDistance*, a packet feature, instead of the flow feature *3whs*. This suggests that in this case, the length of the handshake was used as a proxy for location in the flow-features tree. *geoDistance* is however a more appropriate test in this case, so it is used here. Notice that *geoDistance* does not completely supplant *3whs*, which is used in the left subtree. Following two more packet feature tests, this tree uses the ratio of out/in packets. This feature is thus revealed to have less discriminating ability than the packet features preceding it. Finally, we observe that the tree using both packet and flow features does not use any other features (near the root) than the trees using the individual feature sets—this has implications for feature utility, as we describe in Section 5.6.

5.5. Stability of Packet and Flow Features

Next, we address the question: does the accuracy of these classifiers stay stable over periods of time? We test this by training classifiers and then evaluating them with data *from a later month*. Stability over time has implications for the frequency of retraining spam classifiers: a stable set of characteristics might result in a lower retraining burden. Further, these experiments shed some light on the behavior of network characteristics when there is a major network event.

For this longitudinal study, we build a classifier on one month's data and evaluate it on subsequent months. In this case, we only need a single classifier, so we use all the data for that month to train a classifier, and all the data for each subsequent month to test it, instead of 10-fold cross-validation as used in the earlier results.

As mentioned before, beginning from September 2010, the observed network characteristics changed, possibly due to law enforcement activities that shut down botnets sending spam in the USA. To prevent this event from confounding our analysis, we separate our data into three groups: (i) the "pre-change" set when the training and testing were both conducted before this event (May 2009–August 2010), (ii) the "post-change" set when the training and testing were both conducted after the event (September 2010–April 2011) and (iii) the "across-change" set when training happens before change and testing happens after. We then analyze the stability of network characteristics in each part by producing a decision tree using all of each month's data for that part. We evaluate these trees on each successive month, for example, the tree produced for May 2009 is evaluated on each month from June 2009 through August 2010 for the pre-change experiment and from June 2009 though April 2011 for the across-change experiment. In each case, we plot a graph where the x-axis represents the difference in months between when the tree was constructed and when it was used. For each x, the y-axis then shows the average performance over all month pairs that differ by x. Thus for an x value of 2, we average the performance for the trees built in May and tested in July, built in June and tested in August, and so forth. The results presented at $x = 0$ represent the average cross-validated performance over all months.

Three graphs in the left column of figure 4 shows the results of our experiments using packet features from all three subsets of the data. We first observe that before the change occurred ("pre-Sep/10"), the overall accuracy and true positive rates of the models are quite stable and do not degrade very much. The relative stability of our models is important as it means that the re-training process that will inevitably be needed will likely not be required to run constantly but could instead be performed periodically during off hours. However, we also observe that the FPR degrades rather rapidly, reaching 1.4% after only a month. This indicates that, while the model may not need to be

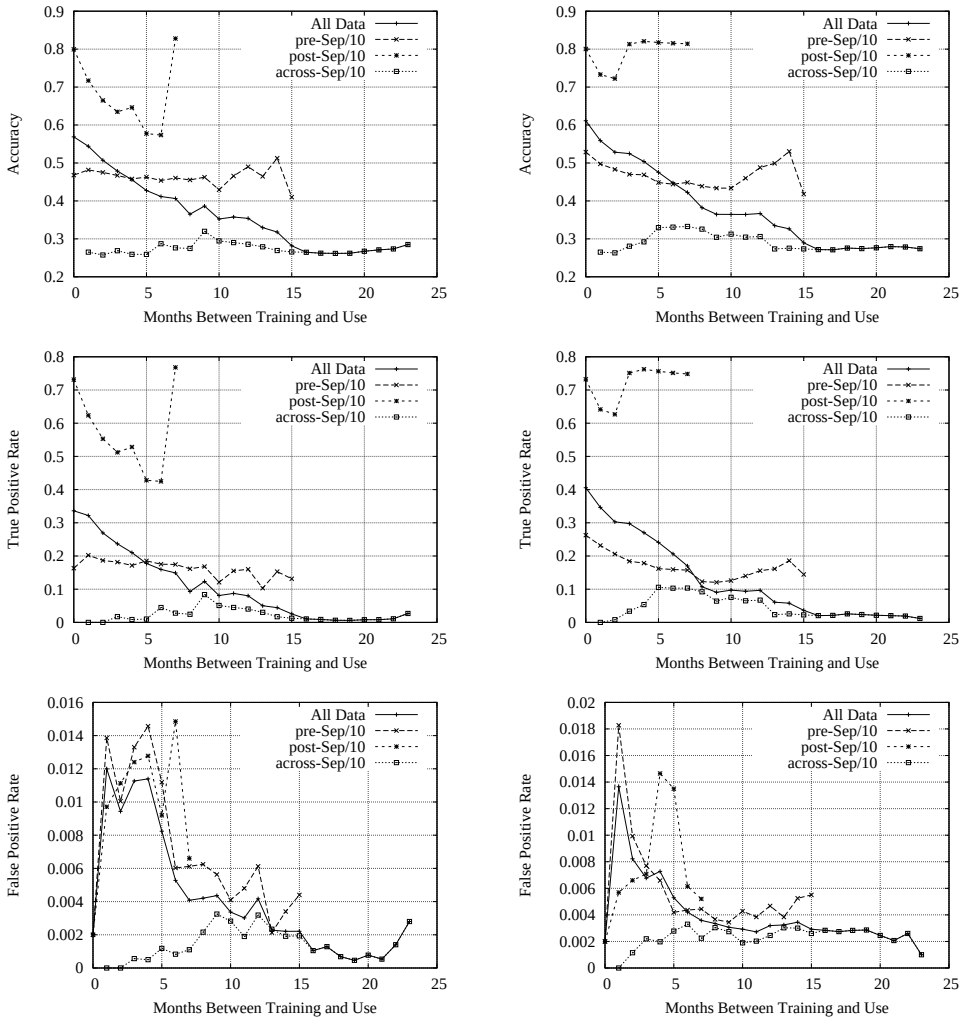


Figure 4: Change in accuracy (top row), TPR (middle row) and FPR (bottom row) over the course of our study as a function of the number of months between training and testing. 3 Figures on left column consist of results from packet features, the other 3 figures on right column are from experiment using (packet + flow) features

rebuilt frequently, the confidence threshold may not carry over from one month to another, but will need to be re-tuned using some data from the new month. There is more variability in TPR after the event ("post-Sep/10"), and classifiers trained at this time, though they have high detection rates, are not as stable as before. This seems intuitively plausible; after a significant shake-up of the Internet underworld, the system may require some time to re-stabilize. Further it is likely that the DNS blacklist is also changing at this point, creating another source of difference in message characteristics being classified by the models. Finally, applying the pre-change classifiers to post-change data produces very poor results. Analyzing the trees indicates that this is mainly because of the important *geoDistance* feature that has changed. The trees before the change also use *geoDistance*, *but with the wrong threshold,* so that once the source of spam shifted, these trees became inaccurate. On the whole, accuracy and TPR degrade slowly over time, as may be expected due to simple feature drift, though the fine structure of the degradation is complicated in this case due to the network event in the middle.

Finally, as shown in the three graphs in the right column of Figure 4 , using all features shows a similar pattern as the packet features: gradual decline in TPR overall, with stable TPR before the change and variable TPRs after, with FPR varying more significantly.

	Use one feature			Use all but one feature		
Feature	**Accuracy**	**TPR**	**AROC**	**Accuracy**	**TPR**	**AROC**
All single-packet	0.594	0.348	0.768	0.594	0.348	0.768
geoDistance	0.473	0.153	0.667	0.546	0.271	0.746
senderHour	0.378	0(FPR=0)	0.500	0.595	0.35	0.769
neighborDist	0.378	0(FPR=0)	0.519	0.522	0.233	0.704
OS	0.378	0(FPR=0)	0.500	0.597	0.353	0.770
ws	0.38	0.004	0.622	0.596	0.352	0.768
ttl	0.378	0(FPR=0)	0.500	0.564	0.30	0.721

Table 10: Left: Average accuracy, TPR at 0.2% FPR and AROC when only one single-packet feature is used. Right: Average accuracy, TPR and FPR when one single-packet feature is left out.

Feature	**Accuracy**	**TPR**	**AROC**	**Accuracy**	**TPR**	**AROC**
All single-packet features	0.594	0.348	0.768	0.594	0.348	0.768
All packet and flow features	0.632	0.410	0.759	0.632	0.410	0.759
	Add one flow feature			**Add all but one flow features**		
3whs	0.586	0.336	0.720	0.643	0.427	0.784
fins_local	0.611	0.376	0.772	0.615	0.382	0.729
fins_remote	0.598	0.354	0.770	0.628	0.403	0.759
idle	0.597	0.353	0.770	0.633	0.411	0.759
jvar	0.594	0.348	0.771	0.632	0.410	0.759
pkts_sent/pkts_received	0.599	0.356	0.770	0.618	0.386	0.759
rsts_local	0.595	0.349	0.768	0.632	0.410	0.759
rsts_remote	0.595	0.349	0.770	0.623	0.395	0.788
rttv	0.606	0.367	0.770	0.618	0.386	0.751
rxmt_local	0.596	0.351	0.768	0.629	0.404	0.759
rxmt_remote	0.595	0.350	0.769	0.624	0.396	0.751
bytecount	0.598	0.354	0.768	0.632	0.409	0.759
throughput	0.607	0.369	0.793	0.625	0.398	0.729

Table 11: Left: Average accuracy, TPR at 0.2% FPR and AROC when all single-packet features and only one flow feature are used. Right: Results when one flow feature is left out, and all other flow and single-packet features are used.

5.6. Utility of Individual Packet and Flow Features

Next, we consider the question: which packet and flow features are most useful in discriminating between ham and spam? For all the utility experiments, we use all the messages from May 2009. We consider three situations. First, we look at the accuracy of our classifiers when only a single packet feature is used, and when we leave a single packet feature out from all the packet features (results in Table 10). Next, we start with the full set of packet features and add flow features one at a time to determine the value added by each flow feature. Finally, we look at what happens if we start with the full feature set and leave one flow feature out (results in Table 11).

Table 10 shows *geoDistance* to be the most useful packet feature. The other packet features, when used in isolation, result in zero or near zero TPR. This is because they produce an empty (or nearly empty in the case of *ws*) tree that always predicts "ham" in order to minimize the cost with a high FP cost. Further, only *geoDistance* and *neighborDist* result in large drops in TPR when they are left out of the feature set. This indicates that though *neighborDist* is not useful by itself, it has some discriminating power when used in conjunction with other packet features. For the other packet features, the TPR stays the same or increases slightly when they are dropped, indicating that they do not provide much added value beyond the remaining features. In particular, our results do not show *senderHour* to be useful for spam detection despite previous findings that spammers and non-spammers display different timing properties [17].

From the results in Table 11, we observe that adding any one flow feature to the set of single packet features improves the performance of the classifier, though by a small margin. In particular, *fins_local*, *rttv* and *throughput* provide the greatest additional discriminating power beyond the single packet features. Further, *fins_local*, *rttv* and *pkts_local/pkts_remote* result in the largest drops in performance when they are dropped, indicating that they are also useful in conjunction with the other features (i.e. no other feature captures their information). Some flow features such as *rsts_local* and *3whs* either do not change the TPR or increase it when they are dropped, indicating that they do not provide much added value beyond the remaining features. Prior results [9, 29] found that *3whs* was the most useful in discriminating ham from spam, if *only* flow features were used. However, it appears in our data that the information in *3whs* is subsumed by the other features, perhaps by packet features such as *geoDistance*.

6. Operational Use

As a final experiment we turn to assessing the methodology developed in this paper in an operational setting. We have not actually implemented and deployed the following strategy, but roughly assess the components to illustrate that the approach may be promising in real-world terms.

Given the results in the previous sections, a filter based on network characteristics is not accurate enough to use as the sole arbiter of email messages. However, we can use a filter based on our developed model as a first pass at winnowing the stream of email by removing messages classified with high confidence (as spam or ham). Our overall goal is to reduce the number of messages that must be analyzed by computationally expensive content filters and hence reduce the overall cost of the process. In this experiment, we evaluate the magnitude of savings obtained by this approach. In other words, referring back to Figure 1, we consider the computational benefits of adding a network-level classifier between the DNS blacklists and content filters in the processing pipeline.

We employ a two-stage filtering process at the network-level processing step. We train two decision tree models with differential misclassification costs. The "FP model" is trained with a high FP:FN cost ratio, while the "FN model" is trained with a high FN:FP cost ratio. Therefore, when the FP model classifies a message as spam, it is likely to be spam, because of the high false positive (ham predicted as spam) cost used during training. Similarly, when the FN model predicts a message to be ham, it is likely to be ham, because of the high false negative (spam predicted as ham) cost used during training. Notice that just setting both FP and FN costs high for the same model would not achieve our desired effect as it would simply remove the bias from the model and be equivalent to the standard cost setting. Each arriving message is assessed using our FP model and if it is judged as spam we discard the message with no further processing (e.g., to the user's "Junk" folder). All messages not removed by the FP filter are assessed using the FN model. Messages that are judged ham by the FN model are placed in the user's "Inbox" with no further processing. All remaining messages are sent to a content filter. It is possible to reverse the order of the FP and FN models in the filtering process, however our experiments showed this to be roughly equivalent in terms of computational cost.

We explore two configurations, with the target error rate of the either stage of the network-layer decisions at 0.2%. One configuration uses packet features only, and the other uses both packet and flow features. For each configuration, we generate the FP and FN models using the following procedure. We use the July 2009 data with ten-fold cross validation to train our models and generate their ROC graphs. We use the FP:FN cost ratio of 175:1 (this cost ratio is also used for the previous experiments) and it produces a model whose ROC graph includes an operating point with the predicted FP rate not exceeding the target error rate. We then use this model with the confidence level producing that operating point as our FP model. Similarly, we generate a model with FN:FP cost ratio of 30:1; its ROC graph includes an operating point with the predicted FN not exceeding the target error rate, and we use this model with the corresponding confidence level as our FN model.

For the packet features configuration, the above procedure produced an FP model using the FP:FN cost ratio of 175:1, with the predicted FPR of 0.2% and predicted TPR is 10.8% (as we have seen already in Table 5). The resulting FN model used FN:FP cost ratio of 30:1, with the predicted false negative rate (FNR) of 0.2% and true negative rate (TNR) of 25.1%. For the packet and flow features configuration, the FP model was trained with the FP:FN cost ratio of 175:1 (predicted FPR=0.2%, TPR=30.8%) and the FN model was obtained with FN:FP cost ratio of 30:1 (FNR=0.2%, TNR=30.7%).

We use these July 2009 models to classify messages from August 2009. On this data, the packet-features FP model had an actual FP rate of 0.10% with a TPR of 9.8%, while the packet-features FN model had an actual FNR of 0.22% with a TNR of 24.9%. On the other hand, the packet+flow features FP model had an actual FPR of 0.30% with a TPR of 28.67% and the packet+flow features FN model had an actual FNR of 0.36% with a TNR of 20.44%. In all cases, the rates of erroneous decisions validate our choice of operating point. Both stages of the packet-features pipeline station identified 6.3K of the 41K messages as not needing further processing (2.5K spam, 3.8K ham), while the packet+flow features pipeline station identified 10.5K of 41K messages as such (7.4K spam, 3.1K ham).

For all the time cost evaluation experiments described below, we run each experiment 30 times and take the median as the final time result for that experiment and report here.

Network Feature-Based Classification: The process of classifying messages based on network characteristics has two components that we consider in turn: (i) deriving the network features and (ii) evaluating those features within a particular model. To determine the effort required to calculate network features we consider the length of time *spamflow* and *p0f* requires to process our packet traces. [11] This process requires 94 seconds for *s*pamflow and

[11] All the times given in this section were computed on the same machine. The absolute magnitude of the numbers is not important and would be

Spam Filter	Content-Only (sec)	Packet + Content (sec)	Packet + Flow + Content (sec)
SpamAssassin	4,216	3,830(10.9%)	3,434 (18.5%)
SpamProbe	1,215	1,321 (-9.1%)	1,244 (-2.4%)
SpamBayes	10,430	9,112 (12.7%)	8,067, (22.8%)
CRM	5,630	4,988 (11.4%)	4,489 (20.4%)

Table 12: Real-world simulation results on August 2009 data.

11 seconds for *p*0f, in all 105 seconds, for the August 2009 dataset. These numbers are calculated by subtracting time to read in the trace from disk as structured data from the running time of *spamflow* and from *p0f*. In our envisioned pipeline, the network features could be supplied from TCP stack directly, so no disk-reading would be involved. A TCP implementation could likely provide network features at much less cost (e.g., using a *getsockopt()* call). TCP already has to do much of the work involved in deriving these features (e.g., de-multiplex the packet stream into connections, track RTTs, understand the advertised window, etc.).

In addition, we use one python script to generate GeoIP related network features, that is, *geoDistance* and *senderHour*, This script needs to perform lookups in GeoIP database for IP address to geo-location mappings. Another python script is used to calculate the *neighborDist* feature. The total running time of these two python scripts is 19 seconds. Overall, to produce all of packet and flow features costs (105 + 19), 124 seconds.

Next we assess the cost of executing the decision tree model across each of the messages in our corpus. We translated the decision tree produced by Weka into a Python function. We then built (i) a small XML-RPC-based server around the decision function and (ii) a client that invokes the server's decision function for each of the 41K messages in our corpus (using the network features as arguments). As discussed above, the decision function is actually a pair of trees for determining sure ham and sure spam. We run the sure spam tree first and then sure ham tree for both configurations: Trees using packet features take 151 seconds to process messages, Trees using packet+flow features take 164 seconds. While we believe that our prototype XML-RPC implementation is likely easy to improve for added performance in a production-ready system, we do not have an estimate of how much additional performance we could expect from such a system. Our expectation is that a working system with kernel-level tracking of network characteristics and more efficient communication to the network feature assessment engine would be less expensive.

Content-Based Classification: Next, we consider the time required to process the messages with the four content classifiers previously used to develop ground truth. We assess both the time required to process the entire 41K message corpus with a particular tool as an estimate of the current spam filtering cost. In addition, we assess the time required to analyze only the 34.7K and 30.5K messages the network classifiers could not judge using packet features and packet+flow features respectively.

Table 12 shows the results of our analysis. The second column gives the total amount of time required to use each of the content filters to process all 41K messages in our corpus. The third and fourth columns show the time required by using network characteristics to filter out sure ham and sure spam and then using the given content filter on the remainder of the messages (with the relative improvement over the content-only classification also given). The third column is the results for packet features, the fourth column is for packet+flow features. With the exception of SpamProbe we find at least a 10.9% improvement when using the packet features and at least 18.5% improvement when using both packet and flow features except compared to SpamProbe (SpamProbe wins a small margin by 2.3%). Note that we purposefully selected a mediocre, "pre-change" (Section 5.3), month for our assessment. A post-change month – with high DNS blacklist pass-through – would see higher savings as the network-level classifier would remove many of the messages missed by the blacklists before these messages hit the content filters.

We used the SpamAssassin daemon in measuring the processing time of SpamAssassin filter. The running time without the daemon is approximately an order of magnitude longer. In addition, as when using SpamAssassin to produce ground truth, we did not employ non-local nor statistical tests. Therefore, SpamAssassin's running time should be taken as a lower bound. SpamProbe is the most efficient filter, by far. We find that removing messages for SpamProbe to consider largely trades costs rather than reducing the overall cost (i.e., the costs associated with filtering based on network features are largely similar to those of using this content filter).[12]

different on another host. Our goal is to explore the relationships between different processing techniques.

[12]Note, if generating the network features within a TCP implementation can save a significant fraction of the cost then we could realize additional savings over using SpamProbe alone. E.g., if the cost of generating network features were reduced by an order of magnitude then the overall system would see an 8–13% performance improvement. This is, however, small enough that the engineering logistics of dealing with multiple filters might

We add a *possible* additional small error rate to the process with the inclusion of our two pre-filters. Therefore, the overall accuracy could be degraded by the chosen error rates used to develop the models when compared to content filtering. On the other hand, mistakes made by the network feature filters could overlap with those already being made by the content filters. Choosing an error rate that is operationally comfortable for any given situation is of course a matter of policy.

Our evaluation only considers the system performance in the one month after the model has been developed. In reality, while we observe that the network feature model ages slowly, it still needs to be either continuously updated or periodically retrained. This can be done based on the user-identified information about misclassified messages that is already commonly used for updating content-based filter models, along with the network features of each message, which could be retained, e.g., in optional email header fields. Because retraining itself can be efficient (e.g., our decision trees take less than a second to be trained over a set of four weeks of messages in our trace, after feature extraction) it should not appreciably affect the processing overhead of the overall operation reported above. We also note that based on our development of ground truth there may be opportunities to further automate the process of re-training without relying as heavily on user labels. We leave this for future work.

Using the network-feature model to discard messages from further processing is not the only way to effectively utilize network-feature classification. The classification is cheap enough that it could be used to prioritize messages, i.e., messages the network characterization identifies as spam could be run through content filters during slow times when the server has plenty of spare capacity, whereas non-spam could be further checked as soon as possible. Another possibility is that network-feature judgments could be used to "backup" a content-based judgment and if both identify a message as spam then it could be discarded completely, whereas if only one classified a message as spam the message could be retained in the users' "junk" folders so that users can catch mistakes. Additional possibilities no doubt exist, as well.

7. Related Work

Spam detection and mitigation has attracted significant attention within the industrial, operational, open source, and research communities. Spam filtering plays an important role in these efforts. There have been a vast range of filtering techniques proposed and implemented including (with examples cited rather than an exhaustive list): DNS blacklists [39]; email address whitelists based on a user's address book in many popular email clients including GMail, Thunderbird and Outlook [5]; domain signed messages [6], analyzing transport [9] and network layer [35, 20] characteristics and signature [4, 37] and statistical [18] analyses of a message's content.

Content filters are quite popular—both within an institution's mail processing apparatus and in users' client software—and classify a message using the information in the message itself. Content filters have been shown to be effective in combating spam, but are computationally expensive and require constant re-training as spam content adapts in an attempt to evade detection [11, 13]. Content filters are also less effective with non-textual spam, although efforts have recently emerged to address this limitation [16].

Given the high cost of content-based filtering (as illustrated in Section 6), several efforts have attempted to use network-layer information to detect spam flows. Schatzmann et al. propose a collaborative content-blind approach that an ISP can deploy to facilitate spam blocking, based on size distribution of the incoming SMTP flows [36]. The idea is that SMTP servers that perform pre-filtering terminate spam flows quickly after processing the envelope; hence flow size from these servers can be used to rate client MTAs for the benefit other SMTP servers that might not use pre-filtering. The idea of using transport features was originated by Beverly and Sollins [9], follow-on work explored using lightweight single passive operating system signatures for the router-level spam filtering [14]. Ramachandran and Feamster analyze network-level characteristics of spamming hosts [35], and Hao et. al. exploit network-level features, such as the sender's geographical distance from the receiver, autonomous system (AS), etc. for spam detection [20]. While Hao et al. target the environment that aggregates data from thousands of institutions and focuses on a possibility of replacing a DNS blacklist, we concentrate on a stand-alone organization environment and evaluate network-level features when used in a pipeline between the DNS blacklists and content filters. We contrast our findings with those of Hao et el. throughout this paper.

This paper incorporates our preliminary work [30], but expands it significantly by using a 2-year long dataset for evaluation, quantifying the computational benefits from adding network-level stage to the pipeline and a much

not make augmenting an existing SpamProbe setup worthwhile.

deeper analysis. Several new observations are discussed, one of them is the interactions of different filtering layers. We further present longitudinal experiment results to shed light on feature stability that concerns a real-world system using network-level features.

Pujara et. al present a method to learn a policy to combine a series of base classifiers given trade-off between cost and accuracy [33], this technique for dynamic composition of multiple filters into a system could also be used in our proposed filtering pipeline. Kakavelakis et. al. [21] develop a real-time, online system that integrates transport layer characteristics, which were previously proposed in [9], into the existing SpamAssassin tool for spam detection. In this system, a spamminess score is produced by transport-characteristic classifier and combined with scores from other methods, e.g., content filters, static rules, etc., to produce a final spamminess score that is used to decide whether a message is spam or not. Our pipeline differs in that the network-feature filter checks whether a message is spam or not independently from the content filter, allowing us to realize potential computational savings.

8. Conclusions and Future Work

In this work, we have conducted a large scale empirical study into the effectiveness of using packet and flow features to detect email spam at an enterprise. Though the idea has been presented in prior work, to the best of our knowledge, our work is the first thorough exploration of the various issues involved. We conduct our study using data collected over the course of two years. Because hand labeling such a large corpus is impractical, we develop a methodology for constructing sound ground truth of large email corpus that is both too huge and too sensitive to hand-label. Our method depends on four spam tools that use different assessment strategies. Using manual validation of a sample of email we find the error in the ground truth to be roughly 2%. Using this labeled data, we explore several hypotheses. First, we validate prior results on our corpus and find that packet features, computed from the SYN packet, are limited in a stand-alone enterprise, while adding flow features, computed from entire flow, into packet features generally improves the accuracy of the classifier. We found that both packet and flow features aged slowly alone time however a major network event could cause big impact on their accuracy. We observed that when experiencing a major network event, e.g, botnet shutdown or migration, DNS-blacklists and network-feature filter combined together provide a fairly stable high detection rate through the network event. Finally, we evaluate a prototype of the pipeline filtering system that places network level-based filter as a layer in the middle of the pipeline to claim a fraction of the messages as confirmed spam or ham and doing so eliminates the need to pass them to a more computationally expensive content filter, therefore saves computational time: For all content filters we use except SpamProbe, interposing our network-features classifier into the processing pipeline resulted in significant reduction in overall processing time, by 18.5–22.8%, compared to using the corresponding content filter alone. While there are certainly additional ways to leverage network layer features for classification, our initial use-case shows the efficacy of the additional information when used in conjunction with several popular content filters.

There are a number of directions we intend to investigate in future. As mentioned in the previous section, understanding how our development of ground truth might be used to largely automate re-training of network-feature models is intriguing.[13] In addition, we intend to investigate the degree to which we can more tightly integrate network layer and content layer filtering. That is, rather than considering these as two distinct steps in a process we hope to leverage key features in both dimensions, possibly adding network-level features, to produce stronger and more efficient email classifiers, one possible way to do that is using naive Bayes method. Finally, we plan to implement the developed approach and deploy it in a real setting.

A final aspect of future work is that we may be able to leverage the work in this paper to identify additional forms of "badness" from malicious hosts. Since spamming is often only one aspect of generally dubious behavior from compromised hosts the models of network characteristics may in fact hold more generally across connections made by bots since non-spam connections will be sharing host and network resources with spam connections. In this case, an intrusion detection system may be able to better understand questionable behavior without directly observing spam from a particular host.

[13] While our results are suggestive that it may be possible to largely automate re-training it seems unlikely that a system will ever get away from manual classification or at least spot checking to ensure accuracy of the process.

Acknowledgments

We thank Rob Beverly for providing the *spamflow* tool used in his previous work, Shuang Hao and Nick Feamster for clarifying the methodology used in their study [20], and Vern Paxson for discussions about generating and validating ground truth. This work is supported in part by NSF awards CNS-0831821, CNS-0916407 and CNS-0433702.

References

[1] Crm114 - the controllable regex mutilator. http://crm114.sourceforge.net/.

[2] Law enforcement agencies help halve spam in 2010. http://www.kaspersky.com/about/news/virus/2011/Law_Enforcement_Agencies_Help_Halve_Spam_in_2010.

[3] The r implementation of rulefit leanring algorithm. http://www-stat.stanford.edu/~jhf/R-RuleFit.html.

[4] SpamAssassin. http://spamassassin.apache.org/.

[5] Whitelist Instructions. http://sci.scientific-direct.net/wl.html.

[6] E. Allman, J. Callas, M. Delany, M. Libbey, J. Fenton, and M. Thomas. Domain-Based Email Authentication Using Public Keys Advertised in the DNS (DomainKeys), May 2007. RFC 4871.

[7] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control, Sept. 2009. RFC 5681.

[8] I. Androutsopoulos, J. Koutsias, K. Chandrinos, and C. Spyropoulos. An experimental comparison of naive bayesian and keyword-based anti-spam filtering with personal e-mail messages. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 160–167, 2000.

[9] R. Beverly and K. Sollins. Exploiting Transport-Level Characteristics of Spam. In *Proceedings of the Fifth Conference on Email and Anti-Spam (CEAS 2008)*, Aug. 2008.

[10] B. Burton. SpamProbe. http://spamprobe.sourceforge.net/.

[11] G. Cormack and A. Bratko. Batch and online spam filter comparison. In *The Third Conference on Email and Anti-Spam*, 2006.

[12] G. Cormack and T. Lynam. 2007 TREC Public Spam Corpus. http://plg.uwaterloo.ca/~gvcormac/treccorpus07/.

[13] G. V. Cormack and J. M. da Cruz. Using old spam and ham samples to train email filters. Available at http://www.j-chkmail.org/ceas-09/gvcjm-ceas09-full.pdf; a short version appeared at CEAS'2009, 2009.

[14] H. Esquivel, T. Mori, and A. Akella. Router-level spam filtering using tcp fingerprints: Architecture and measurement-based evaluation. In *Sixth Conference on Email and Anti-Spam*, 2009.

[15] J. H. Friedman and B. E. Popescu. Predictive learning via rule ensembles. *Annals of Applied Statistics*, 2(3):916–954, 2008.

[16] G. Fumera, I. Pillai, and F. Roli. Spam filtering based on the analysis of text information embedded into images. *Journal of Machine Learning Research*, 7, 12 2006.

[17] L. H. Gomes, C. Cazita, J. M. Almeida, V. A. F. Almeida, and W. M. Jr. Characterizing a spam traffic. In *Internet Measurement Conference*, pages 356–369, 2004.

[18] P. Graham. A Plan for Spam, 2002. http://www.paulgraham.com/spam.html.

[19] S. Hao and N. Feamster. Personal Communication, 2010.

[20] S. Hao, N. A. Syed, N. Feamster, A. G. Gray, and S. Krasser. Detecting spammers with SNARE: Spatio-temporal network-level automatic reputation engine. In *Usenix Security Symp.*, 2009.

[21] G. Kakavelakis, R. Beverly, and J. Young. Auto-learning of SMTP TCP transport-layer features for spam and abusive message detection. In *large installation system administration conference*, 2011.

[22] D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *Proceedings of the second conference on email and anti-spam (CEAS)*, pages 125–132, 2005.

[23] T. R. Lynam, G. V. Cormack, and D. R. Cheriton. On-line spam filter fusion. In *SIGIR*, pages 123–130, 2006.

[24] MaxMind. http://www.maxmind.com.

[25] T. Meyer and B. Whateley. SpamBayes: Effective Open-Source, Bayesian Based, Email classification System. In *Proc. First Conference on Email and Anti-Spam (CEAS)*, June 2004.

[26] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[27] B. Nelson, M. Barreno, F. Chi, A. Joseph, B. Rubinstein, U. Saini, C. Sutton, J. Tygar, and K. Xia. Exploiting machine learning to subvert your spam filter. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 1–9. USENIX Association, 2008.

[28] C. O'Brien and C. Vogel. Spam filters: bayes vs. chi-squared; letters vs. words. In *Proceedings of the 1st International Symposium on Information and Communication Technologies*, pages 291–296, 2003.

[29] T. Ouyang, S. Ray, M. Allman, and M. Rabinovich. A Large-Scale Empirical Analysis of Email Spam Detection Through Transport-Level Characteristics. Technical Report 10-001, International Computer Science Institute, Jan. 2010.

[30] T. Ouyang, S. Ray, M. Rabinovich, and M. Allman. Can network characteristics detect spam effectively in a stand-alone enterprise? In *Passive and Active Measurement*, pages 92–101. Springer, 2011.

[31] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. In *Proceedings of the 7th USENIX Security Symposium*, Jan. 1998.

[32] F. Provost, T. Fawcett, and R. Kohavi. The case against accuracy estimation for comparing induction algorithms. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 445–453, Madison, WI, 1998. Morgan Kaufmann.

[33] J. Pujara, H. Daumé III, and L. Getoor. Using classifier cascades for scalable e-mail classification. In *CEAS*, 2011.

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

[35] A. Ramachandran and N. Feamster. Understanding the Network-Level Behavior of Spammers. In *ACM SIGCOMM*, 2006.

[36] D. Schatzmann, M. Burkhart, and T. Spyropoulos. Inferring spammers in the network core. In *PAM*, pages 229–238, 2009.

[37] A. Schwartz. *SpamAssassin: The Open Source Solution to SPAM*. O'Reilly, 2004.

[38] S. Sinha, M. Bailey, and F. Jahanian. Shades of grey: On the effectiveness of reputation-based blacklists. In *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, pages 57–64. IEEE, 2008.

[39] Spamhaus DNS Blacklists. `http://www.spamhaus.org/`.

[40] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, second edition, 2005.

[41] M. Zalewski. p0f: Passive OS Fingerprinting tool. http://lcamtuf.coredump.cx/p0f.shtml.