2011

Auto-learning of SMTP TCP Transport-Layer Features for Spam and Abusive Message Detection

Kakavelakis, Georgios

Auto-learning of SMTP TCP Transport-Layer Features for Spam and Abusive Message Detection

Georgios Kakavelakis
Naval Postgraduate School

Robert Beverly
Naval Postgraduate School

Joel Young
Naval Postgraduate School

Abstract

Botnets are a significant source of abusive messaging (spam, phishing, etc) and other types of malicious traffic. A promising approach to help mitigate botnet-generated traffic is signal analysis of transport-layer (i.e. TCP/IP) characteristics, e.g. timing, packet reordering, congestion, and flow-control. Prior work [4] shows that machine learning analysis of such traffic features on an SMTP MTA can accurately differentiate between botnet and legitimate sources. We make two contributions toward the *real-world* deployment of such techniques: i) an architecture for real-time on-line operation; and ii) auto-learning of the unsupervised model across different environments without human labeling (i.e. training). We present a "SpamFlow" SpamAssassin plugin and the requisite auxiliary daemons to integrate transport-layer signal analysis with a popular open-source spam filter. Using our system, we detail results from a production deployment where our auto-learning technique achieves better than 95 percent accuracy, precision, and recall after reception of $\approx$ 1,000 emails.
[1]

1 Introduction

"Botnets" are distributed collections of compromised networked machines under common control [7]. Automated methods scan, infect, or socially engineer vulnerable hosts in order to incorporate them into the botnet. Botnets provide a formidable computing and communication platform by harnessing the power of thousands, or even millions, of nodes for a common collective purpose [21]. Unfortunately, that purpose is often malicious and economically or politically motivated.

As one common use scenario, botnets account for more than 85 percent of all abusive electronic mail (including spam, phishing, malware, etc) by one estimate [14]. Botnet-based spamming campaigns are large and long-lived [20], with more than 340,000 botnet hosts involved in nearly 8,000 campaigns in one study [27]. The Messaging Anti-Abuse Working Group (MAAWG) coalition of service providers reported that across 500M monitored mailboxes in one quarter of 2007, 75 percent of all messages (almost 400 billion) were spam [18]. A subsequent 2010 MAAWG study reports the situation has *worsened*: abusive messages accounted for 89 percent of all electronic mail in a representative sample across many providers.

Abusive message traffic abounds on the Internet. This deluge of unwanted traffic is more than a mere nuisance: a broad survey of large service providers finds that abusive messages account for the largest fraction of expended operational resources [1]. Despite extensive research and operational deployments, attackers and attacks have evolved at a rate faster than the Internet's ability to defend. There remains ample room for improvement of in-production botnet attribution and mitigation.

One promising approach for mitigating botnet-generated abusive messaging is statistical traffic analysis. Prior work [4] shows that by using transport-layer traffic features, e.g. TCP retransmits, out-of-order packets, delay, jitter, etc., one can reliably infer whether the source of an email SMTP [16] flow is legitimate or originating from a member of a botnet. Botnets must send large volumes of abusive messages to remain financially viable. Because bots are frequently attached via asymmetric (low upload bandwidth) residential connections, they necessarily congest their local uplink – an effect that is remotely detectable. Perhaps most importantly, transport-layer classifiers are content (e.g. the words of the message itself) and IP reputation (e.g. blocklist) agnostic, facilitating privacy-preserving deployment even within the network core. Deployed on individual Mail Transport Agents (MTAs), such techniques can permit early-rejection of messages before application delivery,

[1] The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes not withstanding any copyright annotations thereon.

significantly reducing system load.

Thus far, research in transport-layer classification has been offline, where experimental data is examined *a posteriori*. In this paper, we present the system engineering efforts required to integrate TCP transport features into the classification decisions of the popular open-source SpamAssassin [17] spam filter. A crucial obstacle to realizing such techniques is the ability to adequately train and build a model of normal and abusive traffic across a variety of operational environments. Rather than requiring human labeling or overly general models to build ground-truth, we exploit the auto-learning functionality of SpamAssassin. Our primary contributions include:

1. *On-line* and *real-time* transport-layer classification of live email messages on a production MTA.
2. Auto-learning of transport features to automatically learn the unsupervised model across different operating environments without human training.

The remainder of this paper describes related work (§2). From this foundation, we describe our system architecture and testing methodology (§3). We present production deployment results in §4 and discuss their implications (§5). We conclude by outlining future work.

2 Related Work

Recent research efforts have shown great promise in understanding the character and behavior of botnets. While these proposed solutions are currently effective, they frequently rely on brittle heuristics and unreliable indicators. For instance, Xie et al. provide a system [27] to identify and characterize botnets using an automatic technique based on discerning spam URLs in email. Other research relies on IP addresses as indicators [29]. However, malicious botnet IP addresses are highly dynamic as new hosts are compromised, existing hosts receive new DHCP leases, or sources are spoofed [3]. Indeed, "fresh" IP addresses, i.e. those not in real-time blocklists, are a valuable commodity. Similarly, DNS is a poor identifier of malicious hosts given the prevalence of botnets employing DNS fast-flux [5] techniques to distribute load among redirectors, survive node failures, and obfuscate back-end hosting infrastructure.

A large body of work examines *network-layer* (IP) properties of botnets. Ramachandran et al. [22] characterize spamming behavior by correlating data collected from three sources: a sinkhole, a large e-mail provider, and the command and control of a Bobax botnet. By focusing on network-level properties including: i) IP address space from which spam originates; ii) the autonomous system (AS) that sent spam messages to their sinkhole; and iii) BGP route announcements, they show that spam and legitimate e-mail originate from the same portion of the IP address space. Thus, IP addresses are not a reliable indicator of malicious or abusive nodes.

Subsequent work from Hao et al. [11] demonstrates that AS alone as a feature may cause a large rate of false positives. They achieve better results by extracting lightweight features from network-level properties such as geodesic distance between sender and receiver, sender IP neighborhood density, probability ratio of spam to ham at the time of day the message arrives, the AS number of the sender, and the status of open ports on the sender machine. Further studies [15, 28] have shown that a spammer can evade such techniques by advertising routes from a forged AS number [11].

Schatzmann et al. [24] similarly focus on network-level characteristics of spammers, but from the perspective of an AS or service provider. Their idea is to passively collect the aggregate decisions of a large number of e-mail servers that perform some level of prefiltering (e.g. blocklisting). Using passive flow collection to gather byte, packet, and packet size counts, this aggregated knowledge can enhance spam mitigation.

Commercial vendors expend considerable effort dividing the Internet IP address space into regions, with particular attention given to identifying residential broadband addresses. By discriminating against residential hosts, the hope is to block traffic from nodes that should not be sourcing email in the first place. This approach is both brittle and raises architectural misgivings in the form of arbitrarily discriminating against classes of users without prior provocation. Such residential blocking may have implications on notions of network neutrality as neutrality legislation catches up with technology.

In contrast to these spam detection and mitigation techniques, Beverly and Sollins [4] present a content and IP reputation agnostic scheme based on statistical signal analysis of the *transport* (TCP) traffic stream. The premise is that spammers must send large volumes of e-mail to be effective, causing constituent network links to experience contention and congestion. Such congestion effects are particularly prominent for many botnet hosts which reside on residential broadband connections where there are large gateway buffers [12] and asymmetric bandwidth. Transport-layer properties such as the number of lost segments and round trip time (RTT) therefore exhibit different distributions, permitting discrimination between spam and legitimate behavior. Among many TCP features, their analysis found that RTT and minimum-congestion window are the most discriminatory. This transport-only classifier exhibits more than 90 percent accuracy and precision on their data.

Follow-on work to [4] explore similar ideas, including the use of lightweight single-TCP/SYN passive operating system signatures at the router-level [10]. Ouyang

et al. [19] conduct a large-scale empirical analysis of transport-layer characteristics on over 600,000 messages. Among tested features, their analysis similarly finds the three-way-handshake latency, time-to-live (TTL), and inter-packet idle time and variance most discriminating for ham versus spam. These features remain stable over time, yielding 85-92 percent classification accuracy.

Based on the encouraging results of this body of prior work, we endeavor to take a step toward the *real-world deployment* of transport-classifier based botnet detection and abusive traffic mitigation techniques.

3 System Architecture

The TCP/IP network stack logically divides functionality between layers. As a result, applications do not normally have access to lower-layer features. For example, TCP (implemented in the kernel or lower) provides an abstraction of a reliable and in-order data stream to the application via a socket interface. Applications are removed from the details of packet arrival timing, ordering, TTL, etc. Thus, our design must collect, on a per-message basis, transport-layer traffic characteristics and expose them up the stack to the SpamFlow (SF) plugin. This section describes our system architecture and the interaction between various components: SpamAssassin, SpamFlow, and the SpamFlow plugin.

3.1 Overview

We start with an overview of our SpamFlow system architecture, shown in Figure 1. For clarity of exposition, we describe all functionality as being co-located with the Mail Transport Agent (MTA); however, the components can easily be distributed across different machines. The system is comprised of four main components: SpamAssassin, the SpamFlow traffic feature extraction engine, the SpamFlow plugin, and the classification software – referred to as SpamAssassin, SpamFlow, SF plugin, and classifier respectively.

Every message received by the MTA is processed by SpamAssassin and then piped to the plugin. Simultaneously, SpamFlow continuously and promiscuously listens on the network interface, capturing SMTP packets via the pcap API [13], aggregating packets into flows, and computing the relevant traffic statistics (e.g. TCP retransmits, out-of-order packets, delay, jitter, etc.). The plugin queries SpamFlow with the message's identifier in order to retrieve the flow-level transport features corresponding to that message. Next, the plugin sends the message's transport feature vector to the classifier. In response, the classifier returns a binary or probabilistic prediction (depending on the classifier employed) that then influences the final score of the message, and hence the final disposition. We describe each component in more detail in the following subsections.

3.2 SpamAssassin

SpamAssassin [17] is an open-source, rule and content learning-based spam filter. Each rule is assigned a weight by a perceptron algorithm [25] and then the weighted scores are summed to produce an overall score for each message. The classification process involves comparing the overall score with a user-defined threshold t (which defaults to a value that maximized performance on a broadly representative training sample). If the score is above t, then the message is classified as spam; otherwise, as legitimate. SpamAssassin is modular and extensible for adding other filtering techniques. Popular plugins include real-time block lists (RBLs), domain-keys, permit lists, collaborative filtering, learning-based techniques (e.g. naïve Bayes), and others.

Furthermore, SpamAssassin features a threshold-based mode in which new exemplar emails trigger an automatic retraining process. While SpamAssassin refers to this retraining as "auto-learning," this is typically known as "online" or "iterative" learning in machine learning. The primary difference is that advanced iterative learning approaches modify the classification model to account for new emails, whereas in auto-learning the entire model is rebuilt each time. In SpamAssassin auto-learning, a previously unseen message is used to retrain the model if it receives a score greater than τ^+ (assumed spam) or less than τ^- (assumed non-spam). For example, when a message exceeds these threshold values, SpamAssassin rebuilds the model of the built-in naïve Bayes classifier, and classifies subsequent messages with the newly updated model.

3.3 SpamFlow

SpamFlow [4] is our network analyzer. Using libpcap [13], SpamFlow promiscuously listens on the network interface and builds source host/port flows (the destination MTA address is constant and known and thus not part of the flow tuple). As SMTP flows complete, either via an explicit TCP termination handshake or via timeout, SpamFlow extracts transport-layer features for each as detailed extensively in [4]. SpamFlow listens for XML queries for a particular flow's IP and port, responding in kind with the features for that flow.

We explored two options for uniquely identifying messages to correlate between messages and their constituent flow data. First, every message contains a unique message string ("`Message-ID`" in the header) [23] to facilitate replies, threading, etc. Using deep packet inspection, SpamFlow could reassemble email messages from

```
--- src/smtpd/smtpd.c.orig
+++ src/smtpd/smtpd.c
@@ -2807,9 +2807,9 @@
 */
 if (!proxy || state->xforward.flags == 0) {
   out_fprintf(out_stream, REC_TYPE_NORM,
-    "Received: from %s (%s [%s])",
+    "Received: from %s (%s [%s:%s])",
     state->helo_name ? state->helo_name : state->name,
-    state->name, state->rfc_addr);
+    state->name, state->rfc_addr, state->port);
```

Figure 2: Postfix modification to support traffic identifiers

```
--- received.c.orig
+++ received.c
@@ -44,2 +44,3 @@
 char *remoteip;
+char *remoteport;
 char *remotehost;
@@ -63,2 +64,5 @@
   safeput(qqt,remoteip);
+  remoteport = getenv("TCPREMOTEPORT");
+  qmail_puts(qqt,":");
+  safeput(qqt,remoteport);
   qmail_puts(qqt,")\n  by ");
```

Figure 3: qmail modification to support traffic identifiers

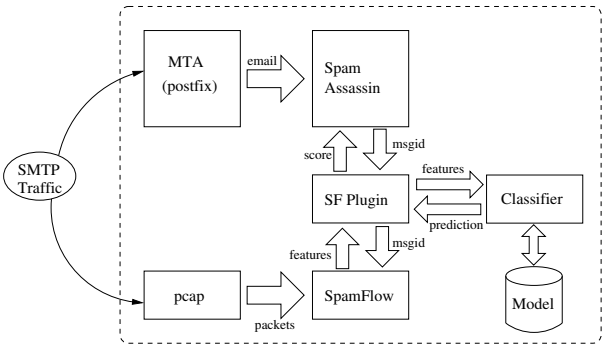


Figure 1: SpamFlow system architecture: transport-layer features are aggregated on a per-flow basis. The SpamFlow SpamAssassin plugin uses XML-RPC to obtain each message's feature vector which is then sent to the classifier. Predictions are relayed to the plugin and integrated into the final SpamAssassin message score.

the packet payloads to uniquely identify each flow by `Message-ID`. The immediate downside to using the message identification field is that doing so removes the benefit of only examining packet header statistics: namely privacy and efficiency.

Instead, we opt to follow a simpler approach and use remote host IP address and ephemeral port number as the message identifier. These fields are readily available without any transport reassembly and are, in general, unique. Naturally, IP address and port tuples are reused (there is a maximum of only 2^{16} unique TCP client-side ephemeral ports). For a tuple collision to occur in SpamFlow, two identical flows must arrive within less time than the messages can be delivered to the MTA and processed by SpamAssassin, i.e. on the order of a few seconds. Not only is this in violation of the TCP time wait procedure, we do not observe any duplicate flows within such short time periods in our empirical data.

The final detail is how to expose the message identifier to the plugin so it can query SpamFlow. We modify our MTA server to add the $(IP\_address, TCP\_port)$ identification tuple of the remote MTA to the header of each incoming e-mail. The actual MTA code modifications are small and straightforward. For reference we provide the code changes for the popular Postfix and qmail MTAs in Figures 2 and 3.

3.4 SpamFlow Plugin

SpamFlow does not operate as a standalone MTA or spam classifier. Therefore, we integrate it with an existing one. We select SpamAssassin [17] because it is open source and widely used; for instance, the commercial Barracuda [2] network appliance is based on SpamAssassin. Importantly, SpamAssassin employs a modular architecture that allows developers to extend its functionality through plugins. As SpamAssassin is written in Perl, we develop a small, lightweight SpamAssassin Perl plugin tying the various components of Figure 1 together. In real-time, as e-mail messages are routed through the SpamFlow plugin, it scores them using a previously learned model of transport features. This score, in combination with the scores from other rules, provides a final message disposition.

The plugin controls the system binding the traffic analysis engine and the classifier together. First, the plugin provides SpamFlow with the 2-tuple identifier of the message under inspection and receives in return the corresponding message's transport-layer features. After obtaining the features, the plugin passes them to a logically distinct machine learning classifier and retrieves the corresponding prediction. Figure 4 shows an example where the MTA added the message identifier (here, `77.239.18.226:37689`) and the plugin attached SpamFlow's transport feature vector to the message's headers.

Between components, we use XML-RPC [26] to communicate. XML-RPC is a simple protocol that allows communication between procedures running in different applications or machines. Specifically, the client uses the HTTP-POST request to pass data to the server; the server in return sends an HTTP response. In our implementation, we register the classifier with a `classify` procedure that takes as input the features. Thus, the plugin

```
From Josephine@rsi.com Tue Feb 01 23:21:58 2011
Return-Path: <Josephine@rsi.com>
X-Spam-Checker-Version: SpamAssassin 3.3.1 (2010-03-16) on ralph.rbeverly.net
X-Spam-Level: ******
X-Spam-Status: Yes, score=6.9 required=5.0 tests=BAYES_50,RCVD_IN_XBL,HTML_MESSAGE,
          SPAMFLOW, UNPARSEABLE_RELAY autolearn=no version=3.3.1
X-Spam-Spamflow-Tag: 3792891725:37689,12,10,0,0,0,0,1,1,0,53248,34.464852,0.162818,
          120.441156,148.297699,51.891697,5840,48,1,64
Received: (qmail 30920 invoked from network); 1 Feb 2011 23:21:57 -0000
Received: from cm-static-18-226.telekabel.ba (77.239.18.226:37689)
Received: from vdhvjcvivjvbwyhscvfwq (192.168.1.185) by bluebellgroup.com (77.239.18.226) with Microsoft SMTP
Message-ID: <4D489025.504060@etisbew.com>
Date: Wed, 2 Feb 2011 00:20:48 +0100
From: Essie <Essie@hermes.com>
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12)
```

Figure 4: Example email message headers with transport features added by the SpamFlow system

sends the HTTP-POST request with the name of the classify procedure along with the features, as comma separated values forming a string[2], and receives via HTTP response the classification prediction from the classifier.

Not only is XML-RPC simple and standardized, it allows the classifier to potentially operate on a different machine from SpamFlow, which in the future could allow the XML-RPC classifier to serve many SpamFlow instances in a multi-threaded fashion and distribute load. Further, all popular programming languages provide XML-RPC APIs, notably allowing us to use our language of choice for the various tasks. In our specific implementation, we develop SpamFlow in C++ while the classifier is a Python daemon.

3.5 Classification Engine

The final component of the system architecture is the traffic classification engine which we implement using the open source Orange [9] machine learning and data mining Python package. While the details of the machine learning algorithms are out of scope for this paper, we note that Orange includes a variety of algorithms and statistical modules for performance evaluation.

Our classifier implementation experiments with three machine-learning algorithms: naïve Bayes, decision trees (specifically, the C4.5 algorithm), and support vector machines (SVM). These three algorithms are broadly representative of different classes of learning strategies and allow us to evaluate both system classification performance, generality, and system speed.

4 Results

This section first describes results from load testing the SpamFlow system in a controlled laboratory environment in order to understand its practical feasibility. We then detail performance results using auto-learning of transport features in a live production environment.

4.1 Load Testing

To understand the system-level performance of our SpamFlow design as outlined in §3, we create the controlled testing environment depicted in Figure 5. One host runs the SpamFlow system and is physically connected to a second traffic sourcing host. The traffic sourcing host implements our custom e-mail "replayer" application and a modified Dummynet [6] network emulator.

The replayer reads from the TREC public email corpus [8] of 92,187 messages, of which 52,788 are spam and 39,399 are legitimate. For each message, the replayer: 1) extracts the headers and adds as recipient a valid user of our virtual-network domain; 2) establishes an SMTP session with the MTA (Postfix) of the SpamFlow system under test; 3) sets the differentiated services code point (DSCP) in the IP header of each message according to the ground truth label (spam or ham); 4) uses the standard SMTP protocol to transmit the message.

We set the DSCP differently for spam and non-spam messages in order to influence the emulated network behavior. Our goal is to coarsely simulate the characteristics that botnet-generated spam traffic exhibits, such as TCP timeouts, retransmissions, resets, and highly variable RTT estimates. For our evaluation, we select Dummynet [6], a publicly-available tool that enables introduction of delay, loss, bandwidth and queuing constraints, etc. for packets passing through virtual network links. In our testing setup, Dummynet applies different queuing, scheduling, bandwidth, delay, loss, etc. depending on the DSCP bits which correspond to email type. Dummynet emulates a only fixed propagation delay. We therefore modify it to generate random delays drawn from a normal distribution with a mean delay of $\mu = 150$ms with $\sigma = 50$ms standard deviation for spam

[2]The CSV string is used for expediency; in the future, we plan to use individual XML identifiers for each feature.

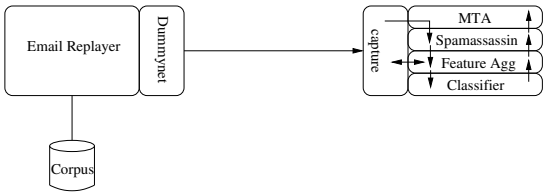


Figure 5: Laboratory testing environment: enabling tightly controlled and easily configurable *repeatable* experiments. The replayer application replays an email corpus while Dummynet emulates different network behavior to mimic botnet and legitimate traffic. Using the testbed, we load test and debug SpamFlow.

traffic that originates from the replayer, and a $\mu = 40$ms delay with $\sigma = 25$ms for legitimate traffic in both directions. We introduce delay in legitimate traffic in order to avoid overfitting our learned statistical model. These delays need not be precise as they are intended to merely mimic a congested environment. To emulate timeouts, retransmissions, and resets, we apply a random-packet-drop policy on the Dummynet pipe.

Note that we disable all SpamAssassin rules requiring network access, e.g. real-time blocklists, as such rules are dynamic and thus sensitive to dates and time.

While we recognize that our modifications to Dummynet only partially emulate a congested network (for example, loss events are independent – an assumption that does not hold true in a real queue), our goal in the emulation environment is to enable reproducible testing. Thus, we use the environment to emulate high-rate traffic and evaluate performance, throughput, system load, etc. on representative traffic. Section 4.3 goes on to detail real-world performance on live production traffic.

Table 1 shows the performance of the three classifiers with respect to training time. C4.5 has the smallest training time. SVM, on the other hand, has the largest training time, due to the more complex decision model.

We then examine throughput: the rate at which the system is able to classify and process emails from the replayer. Naïve Bayes, C4.5, and SVM achieve 1,300, 1,000, and 700 messages per second throughput respectively in our environment. Naïve Bayes provides the highest throughput, likely due to its simple decision rule.

Many factors impact throughput; our intent is to understand the *relative* performance of each classifier and to establish real-world feasibility. The takeaway from these measurements is that, taking into account the relative independence of our system from the classification method, we can select the classification model that fits our needs. For example, the low training time of C4.5 makes it a good candidate when we need to retrain often.

Table 1: SpamFlow training time (sec) as a function of classifier type and sample size

	Training Samples			
Classifier	**10**	**100**	**1000**	**10,000**
Naive Bayes	0.88	15.02	105.45	104.84
C4.5	0.15	0.96	16.02	29.80
SVM	0.72	12.69	224.25	260.02

4.2 Production Environment

Live testing is important because it reveals how the system interacts with possibly unknown features of the external environment. We deployed our system in a live environment at our university for a small domain from January 25, 2011 to March 2, 2011 and collected a trace of 5,926 e-mail messages.

Ground truth was first established using an unmodified SpamAssassin version 3.3.1 instance without transport-layer traffic features, i.e. with only the default built-in rules and content analysis. We then manually examined all the legitimate ham messages and relabeled those that were false negatives. We manually sampled the spam messages to eliminate false positives and establish reasonable ground truth. While the volume of traffic captured is small, our intent in this experiment is to establish the ability to auto-learn the transport-layer features in a production environment and ascertain the resulting classification performance. We envision larger-scale, higher-volume live testing in the future.

Auto-learning is the incremental process of building the classification model based on exemplar e-mail messages whose scores exceed certain threshold values. In our case, we use features of e-mail messages otherwise classified via orthogonal methods as having very high or very low scores (for instance, those emails whose content triggers many of SpamAssassin's rule-based indicators). Specifically, we explicitly retrain the classifier's model each time a new message obtains an especially high or low score from the other SpamAssassin methods (rule- and Bayesian-word based); i.e. a score above or below set thresholds. After retraining is complete, we evaluate performance iteratively on subsequent messages until a new message arrives with a score above or below the threshold, triggering retraining again.

Our thresholds selection is based on empirical spam and ham SpamAssassin score distributions. Spam message scores follow a normal distribution with $\mu = 16.3$ and $\sigma = 7.7$, whereas scores of legitimate messages have a mean of $\mu = 1.3$, but are skewed left. Therefore, for the legitimate messages we first experiment with a threshold $\tau^+ = 16$ and $\tau^- = 1$, which allows the classifiers to be trained on an approximately even fraction of training and test examples: a total of 2,683/5,590 (48.0%) spam and

296/436 (67.9%) ham messages.

We canonically call spam a "positive" and ham a "negative" to indicate disposition. Correct predictions result in either a true positive (tp) or true negative (tn). A spam message that is mispredicted as ham produces a false negative (fn), while a ham message misclassified as spam produces a false positive (fp). Note that false positives in email filtering are particularly expensive for users as there is a high cost to missing or discarding legitimate messages. As performance metrics, we consider accuracy, precision, recall, specificity, and F-score:

$$accuracy = \frac{tp+tn}{tp+fp+tn+fn} \quad (1)$$

$$precision = \frac{tp}{tp+fp} \quad (2)$$

$$recall = \frac{tp}{tp+fn} \quad (3)$$

$$specificity = \frac{tn}{fp+tn} \quad (4)$$

$$F-score = 2\left(\frac{precision*recall}{precision+recall}\right) \quad (5)$$

All of these metrics are important to consider to properly understand system performance. For instance, accuracy is misleading if the underlying class prior is heavily skewed: if 95% of the messages are in fact spam, then a deterministic classifier that always predicts "spam" will achieve seemingly high 95% accuracy without any learning. Precision therefore measures, among messages predicted to be spam, the fraction that are truly spam. Recall measures the influence of misclassified spam messages, i.e. is a metric of the classifier's ability to detect spam. Specificity, or true negative rate, determines how well the classifier is differentiating between false positives and true negatives. Finally, because there is a natural tension between achieving high precision and high recall, a common metric is F-Score which is simply the harmonic mean of precision and recall.

4.3 Production Testing

Figure 6 shows the classification performance metrics of the three classifiers we implement in SpamFlow as a function of cumulative training samples received. Figure 6 therefore depicts the classifiers' auto-learning over time as new exemplar training messages are received.

Figure 6(a) displays cumulative accuracy for each classifier over time and includes the spam prior. The spam prior is simply the fraction of all training emails that are spam. A naïve classifier could simply predict the prior, so values above the prior indicate true learning. We observe both decision trees and SVMs providing greater than 95 percent accuracy. Figure 6(c) similarly shows decision tree and SVM providing high F-scores, indicative of very good performance using only transport-layer features. Of note is that this level of performance is achieved after receiving only 100-200 messages. The weakness in SpamFlow only using traffic characteristics appears in the specificity, Figure 6(e), where false positives drive our best specificity down to approximately 75 percent.

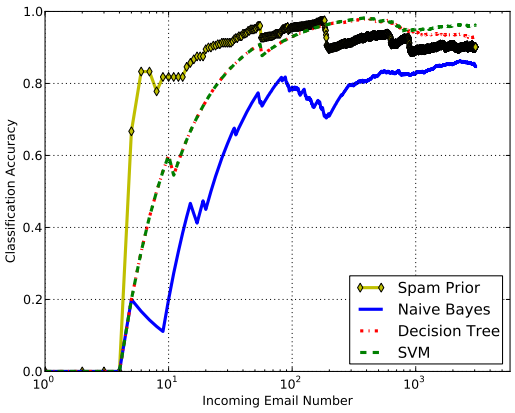
To better understand the sensitivity of our auto-learning results to the imposed thresholds τ, we experiment with a spam threshold two deviations above the mean: $\tau^+ = 30$. By increasing the spam threshold, the SpamFlow auto-learning uses fewer spam-training examples. However, we expect to have higher confidence in their true disposition of spam with the higher threshold. Important to our evaluation, $\tau^+ = 30$ has the effect of balancing the training complexion so that there is not a strong class prior: 227 exemplar spam messages and 296 exemplar ham messages.

With the spam score threshold raised to $\tau^+ = 30$, Figure 6(b) shows that the spam prior is now close to 50 percent, removing any training class bias. SVM and naïve Bayes still achieve greater than 90 percent accuracy. Again, clearly the auto-learning behavior is working with performance steadily increasing over time and greatly outperforming the spam prior. As with the lower threshold, Figure 6(d) demonstrates very high F-Scores for all of the classifiers.

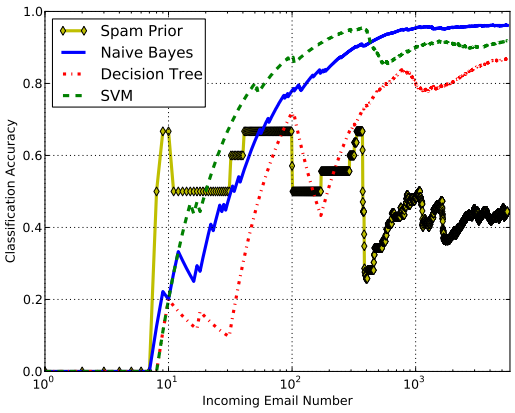
Figure 6(f) highlights the challenge in false positives. However, the most specific classifier, the decision tree algorithm, is also highly accurate and precise. With machine learning there is an inherent trade off between achieving very high true positive rates and keeping false positive rates low. Our results demonstrate the best compromise with the higher auto-learning threshold and the use of decision trees.

Finally, we perform an initial investigation into whether the *combined* votes of SpamAssassin and SpamFlow lead to overall improved performance. We experiment with adding 0.2 (experiment 1) and with adding 1.0 (experiment 2) to the final score if SpamFlow predicts a spam message on the basis of transport traffic characteristics. Otherwise, we subtract 1.0 from the final score. This crude weighting does not leverage SpamFlow's confidence in the prediction, and does not properly weight the vote in accordance with SpamAssassin's other rules. We leave complete integration of SpamFlow's predictions with SpamAssassin's voting as future work.

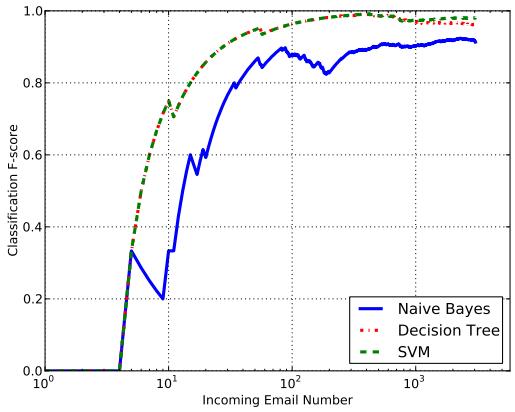
Table 2 shows the confusion data for SpamAssassin alone, SpamFlow alone, and the combination. In the first combined vote, we achieve better performance with the same number of false positives. In the second combined vote, we achieve even better performance, but at the cost of false positives. In all cases, the combination increases the overall F-score.



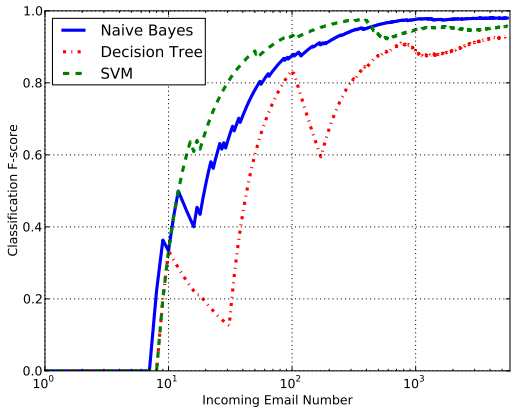
(a) Accuracy ($\tau^+ = 16, \tau^- = 1$)



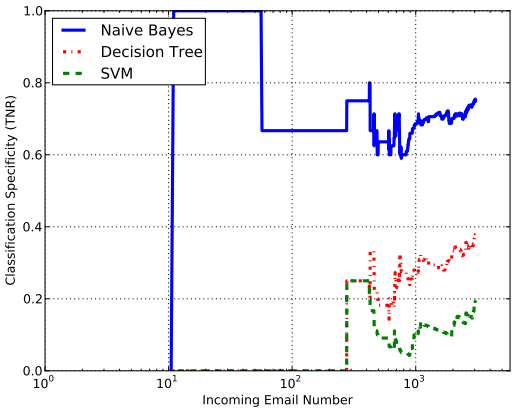
(b) Accuracy ($\tau^+ = 30, \tau^- = 1$)



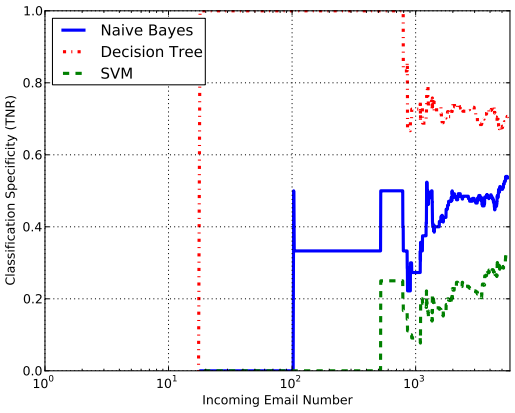
(c) F-Score ($\tau^+ = 16, \tau^- = 1$)



(d) F-Score ($\tau^+ = 30, \tau^- = 1$)



(e) Specificity ($\tau^+ = 16, \tau^- = 1$)



(f) Specificity ($\tau^+ = 30, \tau^- = 1$)

Figure 6: Auto-learning classification results for three SpamFlow classifiers on live production traffic as a function of cumulative exemplar training messages received.

Table 2: Confusion data comparing SpamAssassin performance with and without SpamFlow auto-learning

	tp	fp	tn	fn	F-Score
SpamAssassin	5288	3	137	87	0.991
SpamFlow	5224	65	75	151	0.980
SA+SpamFlow(1)	5299	3	137	76	0.992
SA+SpamFlow(2)	5335	19	121	40	0.995

5 Discussion

Can spammers adapt and avoid a transport-based classification scheme? By utilizing one of the fundamental weaknesses of spammers, their need to send large volumes of spam on bandwidth constrained links, we believe SpamFlow is difficult for spammers to evade. A spammer might send spam at a lower rate or upgrade their infrastructure in order to remove congestion effects from their flows. However, either strategy is likely to impose monetary and time costs on the spammer.

Of note is that our techniques work equally well in IPv6 as the TCP transport-layer characteristics SpamFlow relies on in IPv4 are the same in IPv6. The fact that SpamFlow is IP address agnostic suggests that it may be an even more important technique in an IPv6 world where the large address space is difficult to reliably map.

One possible limitation of SpamFlow is that it may be unable to distinguish between a botnet host sending large volumes of spam and traffic from a host that is simply busy, or on a congested subnetwork. However, other transport-layer features are decoupled from congestion, for instance a CPU-bound bot host will perform TCP flow control and advertise a small receiver window – an effect that SpamFlow uses as part of its decision process.

Further, SpamFlow detects hosts that send volumes of email that exceed the local uplink and processing capacity. Personal, home or small business servers do not have the same volume requirement as spammers and thus are unlikely to induce the same TCP congestion effects we observe. In reality, there is a value judgment that makes SpamFlow practical and reasonable. Specifically, users who wish to ensure that their emails are delivered typically invest in suitable infrastructure, contract with an outside provider or use their service provider's email systems. Companies are not sourcing large amounts of crucial email from hosts attached by consumer-grade connections. The vast majority of home users utilize their provider's email infrastructure or employ popular web-based services. Thus, SpamFlow only discriminates against sources that are both poorly connected *and* injecting large volumes of mail.

However, in future work, we plan to experiment with the sensitivity of SpamFlow to false positive originating from congestion induced by other nodes and other applications. We believe there will remain adequate discriminatory signal to discern botnet hosts. Even when SpamFlow does mispredict, our results show that combining SpamFlow with other classifiers leads to improved performance and can overcome instances of false positives by individual classifiers.

6 Conclusions and Future Work

This research implemented the necessary infrastructure to perform real-time, on-line transport-layer classification of email messages. We plan to distribute our system as part of the third-party SpamAssassin plugin library in order to facilitate widespread deployment, impart impact on abusive messaging traffic, and to refine the system.

We detail the system architecture to integrate network transport features with SpamAssassin, an MTA, and a classification engine. Our testing reveals that the system can handle realistic traffic loads. Of note, we tackle the bootstrapping problem of obtaining representative network traffic on a per-network basis by leveraging auto-learning to automatically train on exemplar messages.

Using our techniques, we achieve accuracy, precision, and recall performance greater than 95 percent after receiving only $\approx 2^{10}$ messages during live, real-world production testing. We emphasize that these results come from observing *only* network traffic features; in actual deployment, the SpamFlow plugin will, as with other parts of the SpamAssassin system, place a weighted vote. Overall performance will likely improve using traditional features in addition to network traffic features.

We note, however, that our live-testing corpus is small. Our intent in this work was to demonstrate the practical feasibility of using transport network traffic features. In future work, we plan to investigate SpamFlow's performance and scalability in large, production systems against much larger volumes of traffic. Our hope is to enable the practical deployment of transport-layer based abusive traffic detection and mitigation techniques to system administrators.

Finally, we observe that the distributed computing platform offered by botnets enables a wide variety of attacks and scams beyond abusive email. Beyond messaging abuse, botnets are employed in phishing attacks, scam infrastructure hosting, distributed denial-of-service (DDoS) attacks, and more. For example, some botnets effectively provide a Content Distribution Network (CDN) for hosting scam infrastructure. Botnet CDNs are used to host web sites (e.g. landing sites for ordering prescription pharmaceuticals or redirection servers), distribute malicious code, and a variety of other nefarious purposes. Still other botnets are employed to perform dictionary attacks against servers, brute force or other-

wise solve CAPTCHAs [30], etc. in order to create accounts on social network sites and further spread abusive traffic via multiple distribution channels.

We believe transport-layer techniques generalize to any botnet generated traffic, including phishing attacks, scam infrastructure hosting, DDoS, dictionary attacks, CAPTCHA solvers, etc. In future research, we wish to investigate using transport-level traffic analysis to identify a variety of botnet attacks and bots themselves.

Acknowledgments

The authors would like to thank Geoffrey Xie, Le Nolan, Ryan Craven, and the anonymous reviewers. Special thanks to our shepherd Avleen Vig for invaluable feedback. This research was partially supported by a Cisco University Research Grant and by the NSF under grant OCI-1127506.

References

[1] ARBOR NETWORKS. Worldwide infrastructure security report, 2010. `http://www.arbornetworks.com/report`.

[2] BARACUDA NETWORKS. Baracuda spam and virus firewall, 2011. `http://www.barracudanetworks.com/`.

[3] BEVERLY, R., BERGER, A., HYUN, Y., AND CLAFFY, K. Understanding the efficacy of deployed internet source address validation filtering. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference* (2009), IMC '09.

[4] BEVERLY, R., AND SOLLINS, K. Exploiting transport-level characteristics of spam. In *Proceedings of the Fifth Conference on Email and Anti-Spam (CEAS)* (Aug. 2008).

[5] CAGLAYAN, A., TOOTHAKER, M., DRAPAEAU, D., BURKE, D., AND EATON, G. Behavioral analysis of fast flux service networks. In *CSIIRW '09: Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research* (2009).

[6] CARBONE, M., AND RIZZO, L. Dummynet revisited. *SIGCOMM Comput. Commun. Rev. 40* (April 2010), 12–20.

[7] COOKE, E., JAHANIAN, F., AND MCPHERSON, D. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet (SRUTI) Workshop* (July 2005).

[8] CORMACK, G., AND LYNAM, T. TREC public email corpus, 2007. `http://trec.nist.gov/data/spam.html`.

[9] DEMSAR, J., ZUPAN, B., LEBAN, G., AND CURK, T. Orange: From experimental machine learning to interactive data mining. In *Principles of Data Mining and Knowledge Discovery* (2004).

[10] ESQUIVEL, H., MORI, T., AND AKELLA, A. Router-level spam filtering using tcp fingerprints: Architecture and measurement-based evaluation. In *Proceedings of the Sixth Conference on Email and Anti-Spam (CEAS)* (2009).

[11] HAO, S., SYED, N. A., FEAMSTER, N., GRAY, A. G., AND KRASSER, S. Detecting spammers with snare: spatio-temporal network-level automatic reputation engine. In *Proceedings of the 18th conference on USENIX security symposium* (2009).

[12] HÄTÖNEN, S., NYRHINEN, A., EGGERT, L., STROWES, S., SAROLAHTI, P., AND KOJO, M. An experimental study of home gateway characteristics. In *Proceedings of the 10th annual conference on Internet measurement*, pp. 260–266.

[13] JACOBSON, V., LERES, C., AND MCCANNE, S. Tcpdump, 1989. `ftp://ftp.ee.lbl.gov`.

[14] JOHN, J. P., MOSHCHUK, A., GRIBBLE, S. D., AND KRISHNAMURTHY, A. Studying spamming botnets using botlab. In *Proceedings of USENIX NSDI* (Apr. 2009).

[15] KARLIN, J., FORREST, S., AND REXFORD, J. Autonomous security for autonomous systems. *Computer Networks 52*, 15 (2008). Complex Computer and Communication Networks.

[16] KLENSIN, J. Simple Mail Transfer Protocol. RFC 5321 (Draft Standard), Oct. 2008.

[17] MASON, J. Filtering spam with spamassassin. In *Proceedings of SAGE-IE* (Oct. 2002).

[18] MESSAGING ANTI-ABUSE WORKING GROUP. Email metrics report, 2011. `http://www.maawg.org/about/EMR`.

[19] OUYANG, T., RAY, S., RABINOVICH, M., AND ALLMAN, M. Can network characteristics detect spam effectively in a stand-alone enterprise? In *Passive and Active Measurement* (2011).

[20] PATHAK, A., QIAN, F., HU, Y. C., MAO, Z. M., AND RANJAN, S. Botnet spam campaigns can be long lasting: evidence, implications, and analysis. In *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems* (2009), ACM, pp. 13–24.

[21] RAJAB, M. A., ZARFOSS, J., MONROSE, F., AND TERZIS, A. My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets* (2007), USENIX Association, pp. 5–5.

[22] RAMACHANDRAN, A., AND FEAMSTER, N. Understanding the network-level behavior of spammers. In *Proceedings of ACM SIGCOMM* (Sept. 2006).

[23] RESNICK, P. Internet Message Format. RFC 2822 (Proposed Standard), Apr. 2001.

[24] SCHATZMANN, D., BURKHART, M., AND SPYROPOULOS, T. Inferring spammers in the network core. In *Proceedings of the 10th International Conference on Passive and Active Network Measurement* (2009), pp. 229–238.

[25] STERN, H. Fast spamassassin score learning tool, Jan. 2004. `http://svn.apache.org/repos/asf/spamassassin/trunk/masses/README.perceptron`.

[26] WINER, D. XML-RPC specification, Apr. 1998. `http://www.xmlrpc.com/spec`.

[27] XIE, Y., YU, F., ACHAN, K., PANIGRAHY, R., HULTEN, G., AND OSIPKOV, I. Spamming botnets: signatures and characteristics. *SIGCOMM Comput. Commun. Rev. 38*, 4 (2008), 171–182.

[28] ZHAO, X., PEI, D., WANG, L., MASSEY, D., MANKIN, A., WU, S. F., AND ZHANG, L. An analysis of bgp multiple origin as (moas) conflicts. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement* (2001), pp. 31–35.

[29] ZHAO, Y., XIE, Y., YU, F., KE, Q., YU, Y., CHEN, Y., AND GILLUM, E. Botgraph: large scale spamming botnet detection. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation* (2009), pp. 321–334.

[30] ZHU, B. B., YAN, J., LI, Q., YANG, C., LIU, J., XU, N., YI, M., AND CAI, K. Attacks and design of image recognition captchas. In *Proceedings of the 17th ACM conference on Computer and communications security* (2010), CCS '10.