Detecting Spammers with SNARE: Spatio-temporal Network-level Automatic Reputation Engine

Shuang Hao, Nadeem Ahmed Syed, Nick Feamster, Alexander G. Gray, Sven Krasser *

*College of Computing, Georgia Tech   *McAfee, Inc.*

{shao, nadeem, feamster, agray}@cc.gatech.edu, sven_krasser@mcafee.com

Abstract

Users and network administrators need ways to filter email messages based primarily on the reputation of the sender. Unfortunately, conventional mechanisms for sender reputation—notably, IP blacklists—are cumbersome to maintain and evadable. This paper investigates ways to infer the reputation of an email sender based solely on network-level features, without looking at the contents of a message. First, we study first-order properties of network-level features that may help distinguish spammers from legitimate senders. We examine features that can be ascertained without ever looking at a packet's contents, such as the distance in IP space to other email senders or the geographic distance between sender and receiver. We derive features that are *lightweight*, since they do not require seeing a large amount of email from a single IP address and can be gleaned without looking at an email's contents—many such features are apparent from even a single packet. Second, we incorporate these features into a classification algorithm and evaluate the classifier's ability to automatically classify email senders as spammers or legitimate senders. We build an *automated* reputation engine, *SNARE*, based on these features using labeled data from a deployed commercial spam-filtering system. We demonstrate that *SNARE* can achieve comparable accuracy to existing static IP blacklists: about a 70% detection rate for less than a 0.3% false positive rate. Third, we show how *SNARE* can be integrated into existing blacklists, essentially as a first-pass filter.

1 Introduction

Spam filtering systems use two mechanisms to filter spam: content filters, which classify messages based on the contents of a message; and sender reputation, which maintains information about the IP address of a sender as an input to filtering. Content filters (e.g., [22, 23]) can block certain types of unwanted email messages, but they can be brittle and evadable, and they require analyzing the contents of email messages, which can be expensive. Hence, spam filters also rely on *sender reputation* to filter messages; the idea is that a mail server may be able to reject a message purely based on the reputation of the sender, rather than the message contents. DNS-based blacklists (DNSBLs) such as Spamhaus [7] maintain lists of IP addresses that are known to send spam. Unfortunately, these blacklists can be both incomplete and slow-to-respond to new spammers [32]. This unresponsiveness will only become more serious as both botnets and BGP route hijacking make it easier for spammers to dynamically obtain new, unlisted IP addresses [33, 34]. Indeed, network administrators are still searching for spam-filtering mechanisms that are both *lightweight* (i.e., they do not require detailed message or content analysis) and *automated* (i.e., they do not require manual update, inspection, or verification).

Towards this goal, this paper presents *SNARE* (Spatio-temporal Network-level Automatic Reputation Engine), a sender reputation engine that can accurately and automatically classify email senders based on lightweight, network-level features that can be determined early in a sender's history—sometimes even upon seeing only a single packet. *SNARE* relies on the intuition that about 95% of all email is spam, and, of this, $75 - 95\%$ can be attributed to botnets, which often exhibit unusual sending patterns that differ from those of legitimate email senders. *SNARE* classifies senders based on *how* they are sending messages (i.e., traffic patterns), rather than *who* the senders are (i.e., their IP addresses). In other words, *SNARE* rests on the assumption that there are lightweight network-level features that can differentiate spammers from legitimate senders; this paper finds such features and uses them to build a system for automatically determining an email sender's reputation.

SNARE bears some similarity to other approaches that classify senders based on network-level behavior [12, 21, 24, 27, 34], but these approaches rely on inspecting the message contents, gathering information across a large number of recipients, or both. In contrast, *SNARE* is based on *lightweight* network-level features, which could allow it to scale better and also to operate on higher traffic rates. In addition, *SNARE* is *more accurate* than previous reputation systems that use network-level behavioral features to classify senders: for example, *SNARE*'s false positive rate is an order of magnitude less than that in our previous work [34] for a similar detection rate. It is the first reputation system that is both as accurate as existing static IP blacklists and automated to keep up with changing sender behavior.

Despite the advantages of automatically inferring sender reputation based on "network-level" features, a major hurdle remains: We must identify *which features* effectively and efficiently distinguish spammers from legitimate senders. Given the massive space of possible features, finding a collection of features that classifies senders with both low false positive and low false negative rates is challenging. This paper identifies thirteen such network-level features that require varying levels of information about senders' history.

Different features impose different levels of overhead. Thus, we begin by evaluating features that can be computed purely locally at the receiver, with no information from other receivers, no previous sending history, and no inspection of the message itself. We found several features that fall into this category are surprisingly effective for classifying senders, including: The AS of the sender, the geographic distance between the IP address of the sender and that of the receiver, the density of email senders in the surrounding IP address space, and the time of day the message was sent. We also looked at various aggregate statistics across messages and receivers (e.g., the mean and standard deviations of messages sent from a single IP address) and found that, while these features require slightly more computation and message overhead, they do help distinguish spammers from legitimate senders as well. After identifying these features, we analyze the relative importance of these features and incorporate them into an automated reputation engine, based on the *RuleFit* [19] ensemble learning algorithm.

In addition to presenting the first automated classifier based on network-level features, this paper presents several additional contributions. First, we presented a detailed study of various network-level characteristics of both spammers and legitimate senders, a detailed study of how well each feature distinguishes spammers from legitimate senders, and explanations of why these features are likely to exhibit differences between spammers and legitimate senders. Second, we use state-of-the-art ensemble learning techniques to build a classifier using these features. Our results show that *SNARE*'s performance is at least as good as static DNS-based blacklists, achieving a 70% detection rate for about a 0.2% false positive rate. Using features extracted from a single message and aggregates of these features provides slight improvements, and adding an AS "whitelist" of the ASes that host the most commonly misclassified senders reduces the false positive rate to 0.14%. This accuracy is roughly equivalent to that of existing static IP blacklists like SpamHaus [7]; the advantage, however, is that *SNARE* is *automated*, and it characterizes a sender based on its sending *behavior*, rather than its IP address, which may change due to dynamic addressing, newly compromised hosts, or route hijacks. Although *SNARE*'s performance is still not perfect, we believe that the benefits are clear: Unlike other email sender reputation systems, *SNARE* is both automated and lightweight enough to operate solely on network-level information. Third, we provide a deployment scenario for *SNARE*. Even if others do not deploy *SNARE*'s algorithms exactly as we have described, we believe that the collection of network-level features themselves may provide useful inputs to other commercial and open-source spam filtering appliances.

The rest of this paper is organized as follows. Section 2 presents background on existing sender reputation systems and a possible deployment scenario for *SNARE* and introduces the ensemble learning algorithm. Section 3 describes the network-level behavioral properties of email senders and measures first-order statistics related to these features concerning both spammers and legitimate senders. Section 4 evaluates *SNARE*'s performance using different feature subsets, ranging from those that can be determined from a single packet to those that require some amount of history. We investigate the potential to incorporate the classifier into a spam-filtering system in Section 5. Section 6 discusses evasion and other limitations, Section 7 describes related work, and Section 8 concludes.

2 Background

In this section, we provide background on existing sender reputation mechanisms, present motivation for improved sender reputation mechanisms (we survey other related work in Section 7), and describe a classification algorithm called *RuleFit* to build the reputation engine. We also describe McAfee's TrustedSource system, which is both the source of the data used for our analysis and a possible deployment scenario for *SNARE*.

2.1 Email Sender Reputation Systems

Today's spam filters look up IP addresses in DNS-based blacklists (DNSBLs) to determine whether an IP address is a known source of spam at the time of lookup. One commonly used public blacklist is Spamhaus [7]; other blacklist operators include SpamCop [6] and SORBS [5]. Current blacklists have three main shortcomings. First, they only provide reputation at the granularity of IP addresses. Unfortunately, as our earlier work observed [34], IP addresses of senders are dynamic: roughly 10% of spam senders on any given day have not been previously observed. This study also observed that many spamming IP addresses will go inactive for several weeks, presumably until they are removed from IP blacklists. This dynamism makes maintaining responsive IP blacklists a manual, tedious, and inaccurate process; they are also often coarse-grained, blacklisting entire prefixes—sometimes too aggressively—rather than individual senders. Second, IP blacklists are typically incomplete: A previous study has noted that as much as 20% of spam received at spam traps is not listed in any blacklists [33]. Finally, they are sometimes inaccurate: Anecdotal evidence is rife with stories of IP addresses of legitimate mail servers being incorrectly blacklisted (e.g., because they were reflecting spam to mailing lists). To account for these shortcomings, commercial reputation systems typically incorporate additional data such as SMTP metadata or message fingerprints to mitigate these shortcomings [11]. Our previous work introduced "behavioral blacklisting" and developed a spam classifier based on a single behavioral feature: the number of messages that a particular IP address sends to each recipient domain [34]. This paper builds on the main theme of behavioral blacklisting by finding better features that can classify senders earlier and are more resistant to evasion.

Field	*Description*
timestamp	UNIX timestamp
ts_server_name	Name of server that handles the query
score	Score for the message based on a combination of anti-spam filters
source_ip	Source IP in the packet (DNS server relaying the query to us)
query_ip	The IP being queried
body_length	Length of message body
count_taddr	Number of To-addresses

Figure 1: Description of data used from the McAfee dataset.

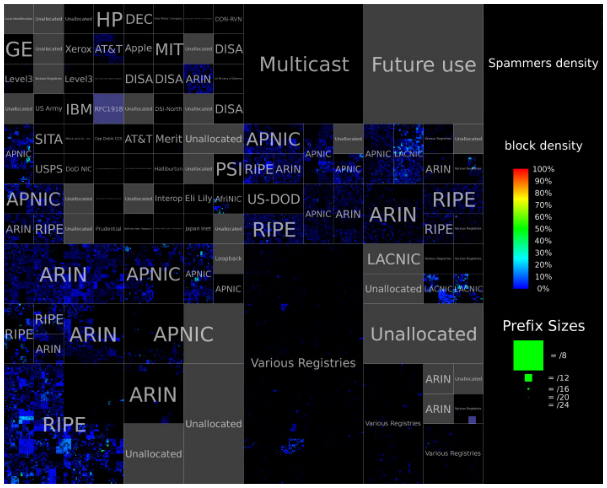


Figure 2: Distribution of senders' IP addresses in Hilbert space for the one-week period (October 22–28, 2007) of our feature study. (The grey blocks are unused IP space.)

2.2 Data and Deployment Scenario

This section describes McAfee's TrustedSource email sender reputation system. We describe how we use the data from this system to study the network-level features of email senders and to evaluate *SNARE*'s classification. We also describe how *SNARE*'s features and classification algorithms could be incorporated into a real-time sender reputation system such as TrustedSource.

Data source TrustedSource is a commercial reputation system that allows lookups on various Internet identifiers such as IP addresses, URLs, domains, or message fingerprints. It receives query feedback from various different device types such as mail gateways, Web gateways, and firewalls. We evaluated *SNARE* using the query logs from McAfee's TrustedSource system over a fourteen-day period from October 22–November 4, 2007. Each received email generates a lookup to the TrustedSource database, so each entry in the query log represents a single email that was sent from some sender to one of McAfee's TrustedSource appliances. Due to the volume of the full set of logs, we focused on logs from a single TrustedSource server, which reflects about 25 million email messages as received from over 1.3 million IP addresses each day. These messages were reported from approximately 2,500 distinct TrustedSource appliances geographically distributed around the world. While there is not a precise one-to-one mapping between domains and appliances, and we do not have a precise count for the number of unique domains, the number of domains is roughly of the same order of magnitude.

The logs contain many fields with *metadata for each email message*; Figure 1 shows a subset of the fields that we ultimately use to develop and evaluate *SNARE*'s classification algorithms. The `timestamp` field reflects the time at which the message was received at a TrustedSource appliance in some domain; the `source_ip` field reflects the source IP of the machine that issued the DNS query (i.e., the recipient of the email). The `query_ip` field is the IP address being queried (i.e., the IP address of the email sender). The IP addresses of the senders are shown in the Hilbert space, as in Figure 2[1], where each pixel represents a /24 network prefix and the intensity indicates the observed IP density in each block. The distribution of the senders' IP addresses shows that the TrustedSource database collocated a representative set of email across the Internet. We use many of the other features in Figure 1 as input to *SNARE*'s classification algorithms.

To help us label senders as either spammers or legitimate senders for both our feature analysis (Section 3) and training (Sections 2.3 and 4), the logs also contain *scores* for each email message that indicate how McAfee scored the email sender based on its current system. The `score` field indicates McAfee's sender reputation score, which we stratify into five labels: certain ham, likely ham, certain spam, likely ham, and uncertain. Although these scores are not perfect ground truth, they do represent the output of both manual classification and continually tuned algorithms that also operate on more heavy-weight features (e.g., packet payloads). Our goal is to develop a fully automated classifier that is as accurate as TrustedSource but (1) classifies senders *automatically* and (2) relies only on lightweight, evasion-resistant network-level features.

Deployment and data aggregation scenario Because it operates only on network-level features of email messages, *SNARE* could be deployed either as part of TrustedSource or as a standalone DNSBL. Some of the features that *SNARE* uses rely on aggregating sender behavior across a wide variety of senders. To aggregate these features, a monitor could collect information about the global behavior of a sender across a wide variety of recipient domains. Aggregating this information is a reasonably lightweight operation: Since the features that *SNARE* uses are based on simple features (i.e., the IP address, plus auxiliary information), they can be piggybacked in small control messages or in DNS messages (as with McAfee's TrustedSource deployment).

2.3 Supervised Learning: RuleFit

Ensemble learning: *RuleFit* Learning ensembles have been among the popular predictive learning methods over the last decade. Their structural model takes the form

$$F(\mathbf{x}) = a_0 + \sum_{m=1}^{M} a_m f_m(\mathbf{x}) \tag{1}$$

Where $\mathbf{x}$ are input variables derived form the training data (spatio-temporal features); $f_m(\mathbf{x})$ are different functions called ensemble members ("base learner") and M is the size of the ensemble; and $F(\mathbf{x})$ is the predictive output (labels for "spam" or "ham"), which takes a linear combination of ensemble members. Given the base learners, the technique determines the parameters for the learners by regularized linear regression with a "lasso" penalty (to penalize large coefficients a_m).

Friedman and Popescu proposed *RuleFit* [19] to construct regression and classification problems as linear combinations of simple rules. Because the number of base learners in this case can be large, the authors propose using the rules in a decision tree as the base learners. Further, to improve the accuracy, the variables themselves are also included as basis functions. Moreover, fast algorithms for minimizing the loss function [18] and the strategy to control the tree size can greatly reduce the computational complexity.

Variable importance Another advantage of *RuleFit* is the interpretation. Because of its simple form, each rule is easy to understand. The relative importance of the respective variables can be assessed after the predictive model is built. Input variables that frequently appear in important rules or basic functions are deemed more relevant. The importance of a variable x_i is given as importance of the basis functions that correspond directly to the variable, plus the average importance of all the other rules that involve x_i. The *RuleFit* paper has more details [19]. In Section 4.3, we show the relative importance of these features.

Comparison to other algorithms There exist two other classic classifier candidates, both of which we tested on our dataset and both of which yielded poorer performance (i.e., higher false positive and lower detection rates) than *RuleFit*. Support Vector Machine (SVM) [15] has been shown empirically to give good generalization performance on a wide variety of problems such as handwriting recognition, face detection, text categorization, etc. On the other hand, they do require significant parameter tuning before the best performance can be obtained. If the training set is large, the classifier itself can take up a lot of storage space and classifying new data points will be correspondingly slower since the classification cost is $O(S)$ for each test point, where S is the number of support vectors. The computational complexity of SVM conflicts with *SNARE*'s goal to make decision quickly (at line rate). Decision trees [30] are another type of popular classification method. The resulting classifier is simple to understand and faster, with the prediction on a new test point taking $O(\log(N))$, where N is the number of nodes in the trained tree. Unfortunately, decision trees compromise accuracy: its high false positive rates make it less than ideal for our purpose.

[1] A larger figure is available at `http://www.gtnoise.net/snare/hilbert-ip.png`.

3 Network-level Features

In this section, we explore various spatio-temporal features of email senders and discuss why these properties are relevant and useful for differentiating spammers from legitimate senders. We categorize the features we analyze by increasing level of overhead:

- *Single-packet features* are those that can be determined with no previous history from the IP address that *SNARE* is trying to classify, and given only a *single packet* from the IP address in question (Section 3.1).
- *Single-header and single-message features* can be gleaned from a single SMTP message header or email message (Section 3.2).
- *Aggregate features* can be computed with varying amounts of history (i.e., aggregates of other features) (Section 3.3).

Each class of features contains those that may be either purely local to a single receiver or aggregated across multiple receivers; the latter implies that the reputation system must have some mechanism for aggregating features in the network. In the following sections, we describe features in each of these classes, explain the intuition behind selecting that feature, and compare the feature in terms of spammers vs. legitimate senders.

No single feature needs to be perfectly discriminative between ham and spam. The analysis below shows that it is unrealistic to have a single perfect feature to make optimal resolution. As we describe in Section 2.3, *SNARE*'s classification algorithm uses a *combination* of these features to build the best classifier. We do, however, evaluate *SNARE*'s classifier using these three different classes of features to see how well it can perform using these different classes. Specifically, we evaluate how well *SNARE*'s classification works using only single-packet features to determine how well such a lightweight classifier would perform; we then see whether using additional features improves classification.

3.1 Single-Packet Features

In this section, we discuss some properties for identifying a spammer that rely only on a single packet from the sender IP address. In some cases, we also rely on auxiliary information, such as routing table information, sending history from neighboring IP addresses, etc., not solely information in the packet itself. We first discuss the features that can be extracted from just a single IP packet: the geodesic distance between the sender and receiver, sender neighborhood density, probability ratio of spam to ham at the time-of-day the IP packet arrives, AS number of the sender and the status of open ports on the machine that sent the email. The analysis is based on the McAfee's data from October 22–28, 2007 inclusive (7 days).[2]

3.1.1 Sender-receiver geodesic distance: Spam travels further

Recent studies suggest that social structure between communicating parties could be used to effectively isolate spammers [13, 20]. Based on the findings in these studies, we hypothesized that legitimate emails tend to travel shorter geographic distances, whereas the distance traveled by spam will be closer to random. In other words, a spam message may be just as likely to travel a short distance as across the world.

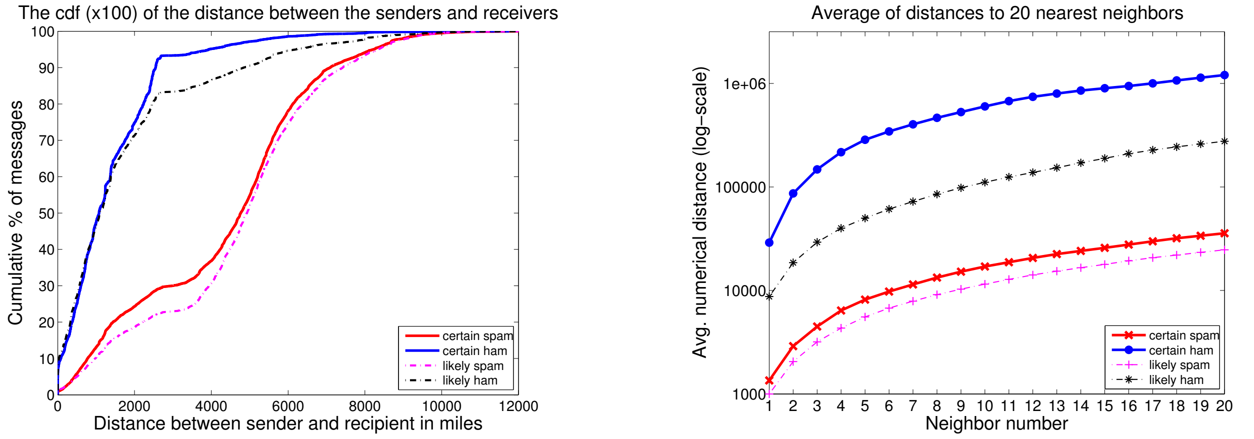
Figure 3(a) shows that our intuition is roughly correct: the distribution of the distance between the sender and the target IP addresses for each of the four categories of messages. The distance used in these plots is the geodesic distance, that is, the distance along the surface of the earth. It is computed by first finding the physical latitude and longitude of the source and target IP using the MaxMind's GeoIP database [8] and then computing the distance between these two points. These distance calculations assume that the earth is a perfect sphere. For *certain ham*, 90% of the messages travel about 2,500 miles or less. On the other hand, for *certain spam*, only 28% of messages stay within this range. In fact, about 10% of spam travels more than 7,000 miles, which is a quarter of the earth's circumference at the equator. These results indicate that geodesic distance is a promising metric for distinguishing spam from ham, which is also encouraging, since it can be computed quickly using just a single IP packet.

3.1.2 Sender IP neighborhood density: Spammers are surrounded by other spammers

Most spam messages today are generated by botnets [33, 37]. For messages originating from the same botnet, the infected IP addresses may all lie close to one another in numerical space, often even within the same subnet. One way to detect whether an IP address belongs to a botnet is to look at the past history and determine if messages have been received from other IPs in the same subnet as the current sender, where the subnet size can be determined experimentally. If many different IPs from the same subnet are sending email, the likelihood that the whole subnet is infested with bots is high.

The problem with simply using subnet density is that the frame of reference does not transcend the subnet

[2] The evaluation in Section 4 uses the data from October 22–November 4, 2007 (14 days), some of which are not included in the data trace used for measurement study.



(a) Geodesic distance between the sender and recipient's geographic location.

(b) Average of numerical distances to the 20 nearest neighbors in the IP space.

Figure 3: Spatial differences between spammers and legitimate senders.

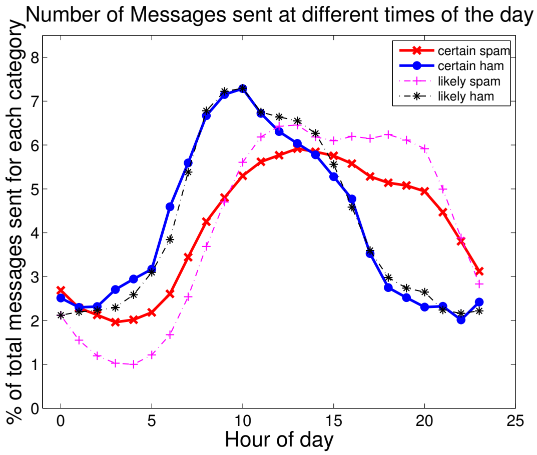


Figure 4: Differences in diurnal sending patterns of spammers and legitimate senders.

boundaries. A more flexible measure of *email sender density* in an IP's neighborhood is the distances to its k nearest neighbors. The distance to the k nearest neighbors can be computed by treating the IPs as set of numbers from 0 to $2^{32} - 1$ (for IPv4) and finding the nearest neighbors in this single dimensional space. We can expect these distances to exhibit different patterns for spam and ham. If the neighborhood is *crowded*, these neighbor distances will be small, indicating the possible presence of a botnet. In normal circumstances, it would be unusual to see a large number of IP addresses sending email in a small IP address space range (one exception might be a cluster of outbound mail servers, so choosing a proper threshold is important, and an operator may need to evaluate which threshold works best on the specific network where *SNARE* is running).

The average distances to the 20 nearest neighbors of the senders are shown in Figure 3(b). The x-axis indicates how many nearest neighbors we consider in IP space, and the y-axis shows the average distance in the sample to that many neighbors. The figure reflects the fact that a large majority of spam originates from hosts have high email sender density in a given IP region. The distance to the k^{th} nearest neighbor for spam tends to be much shorter on average than it is for legitimate senders, indicating that spammers generally reside in areas with higher densities of email senders (in terms of IP address space).

3.1.3 Time-of-day: Spammers send messages according to machine off/on patterns

Another feature that can be extracted using information from a single packet is the time of day when the message was sent. We use the *local* time of day at the sender's physical location, as opposed to Coordinated Universal Time (UTC). The intuition behind this feature is that local legitimate email sending patterns may more closely track "conventional" diurnal patterns, as opposed to spam sending patterns.

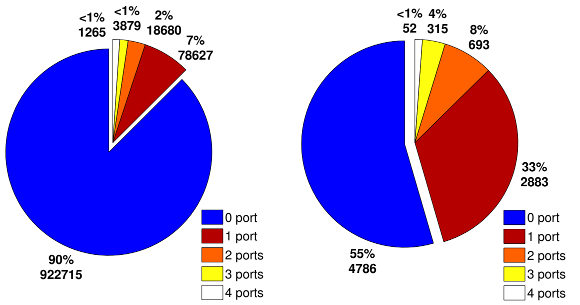
Figure 4 shows the relative percentage of messages of each type at different times of the day. The legitimate senders and the spam senders show different diurnal patterns. Two times of day are particularly striking: the relative amount of ham tends to ramp up quickly at the start of the workday and peaks in the early morning. Volumes decrease relatively quickly as well at the end of the workday. On the other hand spam increases at a slower, steadier pace, probably as machines are switched on in the morning. The spam volume stays steady throughout the day and starts dropping around 9:00 p.m., probably when machines are switched off again. In summary, legitimate senders tend to follow workday cycles, and spammers tend to follow machine power cycles.

To use the timestamp as a feature, we compute the probability ratio of spam to ham at the time of the day when the message is received. First, we compute the *a priori* spam probability $p_{s,t}$ during some hour of the day t, as $p_{s,t} = n_{s,t}/n_s$, where $n_{s,t}$ is the number of spam messages received in hour t, and n_s is the number of spam messages received over the entire day. We can compute the *a priori* ham probability for some hour t, $p_{h,t}$ in a similar fashion. The probability ratio, r_t is then simply $p_{s,t}/p_{h,t}$. When a new message is received, the precomputed spam to ham probability ratio for the corresponding hour of the day at the senders timezone, r_t can be used as a feature; this ratio can be recomputed on a daily basis.

3.1.4 AS number of sender: A small number of ASes send a large fraction of spam

As previously mentioned, using IP addresses to identify spammers has become less effective for several reasons. First, IP addresses of senders are often transient. The compromised machines could be from dial-up users, which depend on dynamic IP assignment. If spam comes form mobile devices (like laptops), the IP addresses will be changed once the people carry the devices to a different place. In addition, spammers have been known to adopt stealthy spamming strategies where each bot only sends several spam to a single target domain, but overall the botnets can launch a huge amount of spam to many domains [33]. The low emission-rate and distributed attack requires to share information across domains for detection.

On the other hand, our previous study revealed that a significant portion of spammers come from a relatively small collection of ASes [33]. More importantly, the ASes responsible for spam differ from those that send legitimate email. As a result, the AS numbers of email senders could be a promising feature for evaluating the senders' reputation. Over the course of the seven days in our trace, more than 10% of unique spamming IPs (those sending certain spam) originated from only 3 ASes; the top 20 ASes host 42% of spamming IPs. Although our previous work noticed that a small number of ASes originated a large fraction of spam [33], we believe that this is the first work to suggest using the AS number of the email sender as input to an automated classifier for sender reputation.



(a) spam

(b) legitimate mail

Figure 5: Distribution of number of open ports on hosts sending spam and legitimate mail.

3.1.5 Status of service ports: Legitimate mail tends to originate from machines with open ports

We hypothesized that legitimate mail senders may also listen on other ports besides the SMTP port, while bots might not; our intuition is that the bots usually send spam directly to the victim domain's mail servers, while the legitimate email is handed over from other domains' MSA (Mail Submission Agent). The techniques of reverse DNS (rDNS) and Forward Confirmed Reverse DNS (FCrDNS) have been widely used to check whether the email is from dial-up users or dynamically assigned addresses, and mail servers will refuse email from such sources [1].

We propose an additional feature that is orthogonal to DNSBL or rDNS checking. Outgoing mail servers open specific ports to accept users' connections, while the bots are compromised hosts, where the well-known service ports are closed (require root privilege to open). When packets reach the mail server, the server issues an active probe sent to the source host to scan the following four ports that are commonly used for outgoing mail service: 25 (SMTP), 465 (SSL SMTP), 80 (HTTP) and 443 (HTTPS), which are associated with outgoing mail services. Because neither the current mail servers nor the McAfee's data offer email senders' port information, we need to probe back sender's IP to check out what service ports might be open. The probe process was performed during both October 2008 and January 2009, well after the time when the email was received. Despite this delay, the status of open ports still exposes a striking difference between legitimate senders and spammers. Figure 5 shows the percentages and the numbers of opening ports for spam and ham categories respectively. The statistics are calculated on the senders' IPs from the evaluation dataset we used in Section 4 (October 22–28, 2007). In the spam case, 90% of spamming IP addresses have *none* of the standard mail service ports open; in contrast,

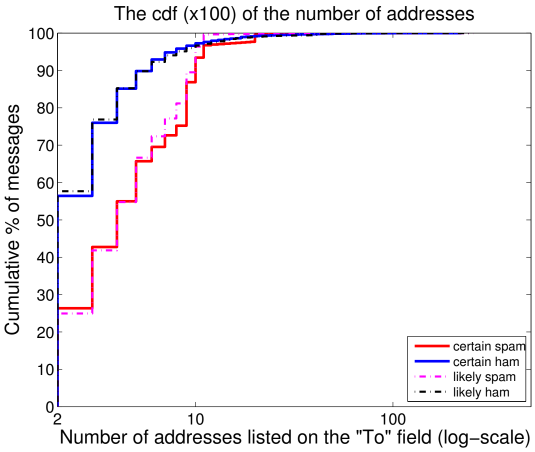


Figure 6: Distribution of number of addresses listed on the “To” field for each category (ignoring single-recipient messages).

half of the legitimate email comes from machines listening on at least one mail service port. Although firewalls might block the probing attempts (which causes the legitimate mail servers show no port listening), the status of the email-related ports still appears highly correlated with the distinction of the senders. When providing this feature as input to a classifier, we represent it as a bitmap (4 bits), where each bit indicates whether the sender IP is listening on a particular port.

3.2 Single-Header and Single-Message Features

In this section, we discuss other features that can be extracted from a single SMTP header or message: the number of recipients in the message, and the length of the message. We distinguish these features from those in the previous section, since extracting these features actually requires opening an SMTP connection, accepting the message, or both. Once a connection is accepted, and the SMTP header and subsequently, the compete message are received. At this point, a spam filter could extract additional non-content features.

3.2.1 Number of recipients: Spam tends to have more recipients

The features discussed so far can be extracted from a single IP packet from any given specific IP address combined with some historical knowledge of messages from other IPs. Another feature available without looking into the content is the number of address in “To” field of the header. This feature can be extracted after receiving the entire SMTP header but before accepting the message body. However, the majority of messages only have one address listed. Over 94% of spam and 96% of legitimate email is sent to a single recipient. Figure 6 shows the distribution of number of addresses in the “To” field for each category of messages for all emails that are sent to more than one recipient. The x-axis is on a log-scale to focus the plot on the smaller values. Based on this plot and looking at the actual values, it appears that if there are very large number of recipients on the “To” field (100 or more), there does not seem to be a significant difference between the different types of senders for this measure. The noticeable differences around 2 to 10 addresses show that, generally, ham has fewer recipients (close to 2) while spam is sent to multiple addresses (close to 10). (We acknowledge that this feature is probably evadable and discuss this in more detail in Section 6.1).

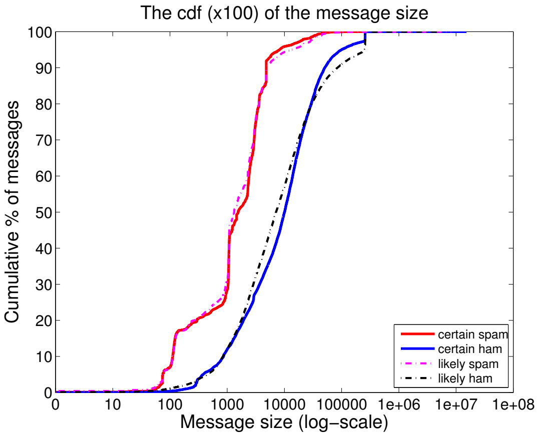


Figure 7: Distribution of message size (in bytes) for the different categories of messages.

3.2.2 Message size: Legitimate mail has variable message size; spam tends to be small

Once an entire message has been received, the email body size in bytes is also known. Because a given spam sender will mostly send the same or similar content in all the messages, it can be expected that the variance in the size of messages sent by a spammer will be lower than among the messages sent by a legitimate sender. To stay effective, the spam bots also need to keep the message size small so that they can maximize the number of messages they can send out. As such the spam messages can be expected to be biased towards the smaller size. Figure 7 shows the distribution of messages for each category. The spam messages are all clustered in the 1–10KB range, whereas the distribution of message size for legitimate senders is more evenly distributed. Thus, the message body size is another property of messages that may help differentiate spammers from legitimate senders.

3.3 Aggregate Features

The behavioral properties discussed so far can all be constructed using a single message (with auxiliary or neighborhood information). If some history from an IP is available, some *aggregate IP-level features* can also be constructed. Given information about multiple messages from a single IP address, the overall *distribution* of the following measures can be captured by using a combination of *mean and variance of*: (1) geodesic distance between the sender and recipient, (2) number of recipients in the "To" field of the SMTP header, and (3) message body length in bytes. By summarizing behavior over multiple messages and over time, these aggregate features may yield a more reliable prediction. On the flip side, computing these features comes at the cost of increased latency as we need to collect a number of messages before we compute these. Sometimes gathering aggregate information even requires cross-domain collaboration. By averaging over multiple messages, these features may also smooth the structure of the feature space, making marginal cases more difficult to classify.

4 Evaluating the Reputation Engine

In this section, we evaluate the performance of *SNARE*'s *RuleFit* classification algorithm using different sets of features: those just from a single packet, those from a single header or message, and aggregate features.

4.1 Setup

For this evaluation, we used fourteen days of data from the traces, from October 22, 2007 to November 4, 2007, part of which are different from the analysis data in Section 3. In other words, the entire data trace is divided into two parts: the first half is used for measurement study, and the latter half is used to evaluate *SNARE*'s performance. The purpose of this setup is both to verify the hypothesis that the feature statistics we discovered would stick to the same distribution over time and to ensure that feature extraction would not interfere with our evaluation of prediction.

Training We first collected the features for each message for a subset of the trace. We then randomly sampled 1 million messages from each day on average, where the volume ratio of spam to ham is the same as the original data (i.e., 5% ham and 95% spam; for now, we consider only messages in the "certain ham" and "certain spam" categories to obtain more accurate ground truth). Only our evaluation is based on this sampled dataset, *not* the feature analysis from Section 3, so the selection of those features should not have been affected by sampling. We then intentionally sampled equal amounts of spam as the ham data (30,000 messages in each categories for each day) to train the classifier because training requires that each class have an equal number of samples. In practice, spam volume is huge, and much spam might be discarded before entering the *SNARE* engine, so sampling on spam for training is reasonable.

Validation We evaluated the classifier using temporal cross-validation, which is done by splitting the dataset into subsets along the time sequence, training on the subset of the data in a time window, testing using the next subset, and moving the time window forward. This process is repeated ten times (testing on October 26, 2007 to November 4, 2007), with each subset accounting for one-day data and the time window set as 3 days (which indicates that long-period history is not required). For each round, we compute the detection rate and false positive rate respectively, where the detection rate (the "true positive" rate) is the ratio of spotted spam to the whole spam corpus, and false positive rate reflects the proportion of misclassified ham to all ham instances. The final evaluation reflects the average computed over all trials.

Summary Due to the high sampling rate that we used for this experiment, we repeated the above experiment for several trials to ensure that the results were consistent across trials. As the results in this section show, detection rates are approximately 70% and false positive rates are approximately 0.4%, even when the classifier is based only on single-packet features. The false positive drops to less 0.2% with the same 70% detection as the classifier incorporates additional features. Although this false positive rate is likely still too high for *SNARE* to subsume all other spam filtering techniques, we believe that the performance may be good enough to be used in conjunction with other methods, perhaps as an early-stage classifier, or as a substitute for conventional IP reputation systems (e.g., SpamHaus).

4.2 Accuracy of Reputation Engine

In this section, we evaluate *SNARE*'s accuracy on three different groups of features. Surprisingly, we find that, even relying on only single-packet features, *SNARE* can automatically distinguish spammers from legitimate senders. Adding additional features based on single-header or single-message, or aggregates of these features based on 24 hours of history, improves the accuracy further.

(a) Single Packet

	Classified as	
	Spam	Ham
Spam	**70%**	30%
Ham	**0.44%**	99.56%

(b) Single Header/Message

	Classified as	
	Spam	Ham
Spam	**70%**	30%
Ham	**0.29%**	99.71%

(c) 24+ Hour History

	Classified as	
	Spam	Ham
Spam	**70%**	30%
Ham	**0.20%**	99.80%

Table 1: *SNARE* performance using *RuleFit* on different sets of features using covariant shift. Detection and false positive rates are shown in bold. (The detection is fixed at 70% for comparison, in accordance with today's DNSBLs [10]).

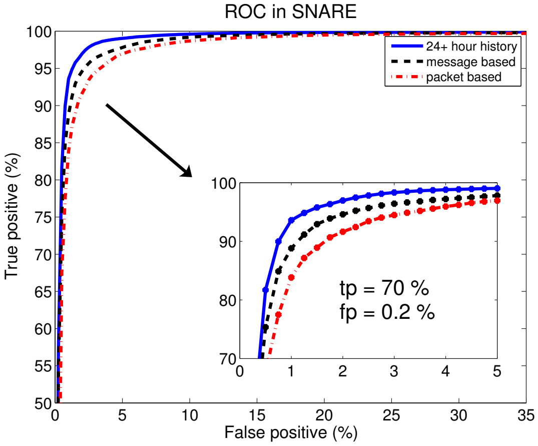


Figure 8: ROC in *SNARE*.

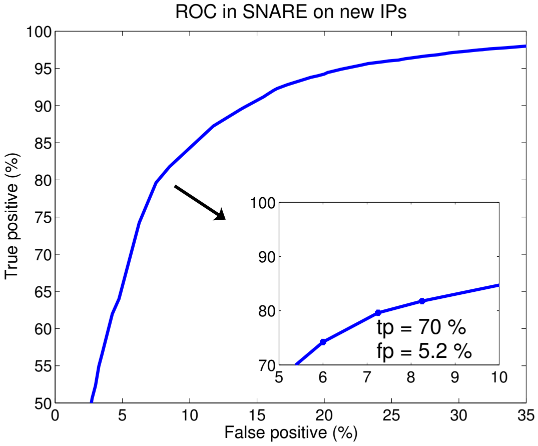


Figure 9: ROC on fresh IPs in *SNARE*.

4.2.1 Single-Packet Features

When a mail server receives a new connection request, the server can provide *SNARE* with the IP addresses of the sender and the recipient and the time-stamp based on the TCP SYN packet alone. Recall from Section 3 even if *SNARE* has never seen this IP address before, it can still combine this information with recent history of behavior of other email servers and construct the following features: (1) geodesic distance between the sender and the recipient, (2) average distance to the 20 nearest neighbors of the sender in the log, (3) probability ratio of spam to ham at the time the connection is requested (4) AS number of the sender's IP, and (5) status of the email-service ports on the sender.

To evaluate the effectiveness of these features, we trained *RuleFit* on these features. The dash-dot curve in Figure 8 demonstrate the ROC curve of *SNARE*'s reputation engine. The fp $= 0.2\%$ and tp $= 70\%$ statistics refer to the curve with 24-hour history (solid line), which will be addresses later. We check the false positive given a fixed true positive, 70%. The confusion matrix is shown in Table 1(a). Just over 0.44% of legitimate email gets labelled as spam. This result is significant because it relies on features constructed from a limited amount of data and just a single IP packet from the candidate IP. Sender reputation system will be deployed in conjunction with a combination of other techniques including content based filtering. As such, as a first line of defense, this system will be very effective in eliminating a lot of undesired senders. In fact, once a sender is identified as a spammer, the mail server does not even need to accept the connection request, saving network bandwidth and computational resources. The features we describe below improve accuracy further.

4.2.2 Single-Header and Single-Message Features

Single-packet features allow *SNARE* to rapidly identify and drop connections from spammers even before looking at the message header. Once a mail server has accepted the connection and examined the entire message, *SNARE* can determine sender reputation with increased confidence by looking at an additional set of features. As described in Section 3.2, these features include the number of recipients and message body length. Table 1(b) shows the prediction accuracy when we combine the single-packet features (i.e., those from the previous section) with these additional features. As the results from Section 3 suggest, adding the *message body length* and *number of recipients* to the set of features further improves *SNARE*'s detection rate and false positive rate.

It is worth mentioning that the number of recipients listed on the "To" field is perhaps somewhat evadable: a sender could list the target email addresses on "Cc" and "Bcc" fields. Besides, if the spammers always place a single recipient address in the "To" field, this value will be the same as the large majority of legitimate messages. Because we did not have logs of additional fields in the SMTP header beyond the count of email addresses on the "To" field, we could not evaluate whether considering number of recipients listed under "Cc" and "Bcc" headers is worthwhile.

4.2.3 Aggregate Features

If multiple messages from a sender are available, the following features can be computed: the mean and variance of geodesic distances, message body lengths and number of recipients. We evaluate a classifier that is trained on *aggregate statistics* from the past 24 hours together with the features from previous sections.

Table 1(c) shows the performance of *RuleFit* with these aggregate features, and the ROC curve is plotted as the solid one in Figure 8. Applying the aggregate features decreases the error rate further: 70% of spam is identified correctly, while the false positive rate is merely 0.20%. The content-based filtering is very efficient to identify spam, but can not satisfy the requirement of processing a huge amount of messages for big mail servers. The prediction phase of *RuleFit* is faster, where the query is traversed from the root of the decision tree to a bottom label. Given the low false positive rate, *SNARE* would be a perfect first line of defense, where suspicious messages are dropped or re-routed to a farm for further analysis.

4.3 Other Considerations

Detection of "fresh" spammers We examined data trace, extracted the IP addresses not showing up in the previous training window, and further investigated the detection accuracy for those 'fresh' spammers with all *SNARE*'s features. If fixing the true positive as 70%, the false positive will increase to 5.2%, as shown in Figure 9. Compared with Figure 8, the decision on the new legitimate users becomes worse, but most of the new spammers can still be identified, which validates that *SNARE* is capable of *automatically* classifying "fresh" spammers.

Relative importance of individual features We use the fact that *RuleFit* can evaluate the *relative importance* of the features we have examined in Sections 3. Table 2 ranks all spatio-temporal features (with the most important feature at top). The top three features—*AS num*, *avg length* and *neig density*—play an important role in separating out spammers from good senders. This result is quite promising, since most of these features are lightweight: Better yet, two of these three can be computed having received only a single packet from the sender. As we will discuss in Section 6, they are also relatively resistant to evasion.

rank	*Feature Description*
1	AS number of the sender's IP
2	average of message length in previous 24 hours
3	average distance to the 20 nearest IP neighbors of the sender in the log
4	standard deviation of message length in previous 24 hours
5	status of email-service ports on the sender
6	geodesic distance between the sender and the recipient
7	number of recipient
8	average geodesic distance in previous 24 hours
9	average recipient number in previous 24 hours
10	probability ratio of spam to ham when getting the message
11	standard deviation of recipient number in previous 24 hours
12	length of message body
13	standard deviation of geodesic distance in previous 24 hours

Table 2: Ranking of feature importance in *SNARE*.

Correlation analysis among features We use mutual information to investigate how tightly the features are coupled, and to what extent they might contain redundant information. Given two random variables, mutual information measures how much uncertainty of one variable is reduced after knowing the other (i.e., the information they share). For discrete variables, the mutual information of X and Y is calculated as: $I(X,Y) = \sum_{x,y} p(x,y)\log(\frac{p(x,y)}{p(x)p(y)})$. When logarithm base-two is used, the quantity reflects how many bits can be removed to encode one variable given the other one. Table 3 shows the mutual information between pairs of features for one day of training data (October 23, 2007). We do not show statistics from other days, but features on those days reflect similar quantities for mutual information. The features with continuous values (e.g., geodesic distance between the sender and the recipient) are transformed into discrete variables by dividing the value range into 4,000 bins (which yields good discrete approximation); we calculate mutual information over the discrete probabilities. The indexes of the features in the table are the same as the ranks in Table 2; the packet-based features are marked with black circles. We also calculate the entropy of every feature and show them next to the indices in Table 3.

The interpretation of mutual information is consistent only within a single column or row, since comparison of mutual information without any common variable is meaningless. The table, of course, begs additional analysis but shows some interesting observations. The top-ranked feature, AS number, shares high mutual information (shown in bold) with several other features, especially with feature 6, geodesic distance between sender and recipient. The aggregate features of first-order statis-

	❶ (8.68)	2 (7.29)	❸ (2.42)	4 (6.92)	❺ (1.20)	❻ (10.5)	7 (0.46)	8 (9.29)	9 (2.98)	❿ (4.45)	11 (3.00)	12 (6.20)
2 (7.29)	**4.04**											
❸ (2.42)	**1.64**	1.18										
4 (6.92)	**3.87**	4.79	1.23									
❺ (1.20)	**0.65**	0.40	0.11	0.43								
❻ (10.5)	**5.20**	3.42	0.88	3.20	0.35							
7 (0.46)	**0.11**	0.08	0.02	0.08	0.004	0.15						
8 (9.29)	**5.27**	5.06	1.20	4.79	0.46	5.16	0.13					
9 (2.98)	**1.54**	1.95	0.53	2.03	0.09	1.17	0.10	2.08				
❿ (4.45)	**0.66**	0.46	0.07	0.49	0.02	0.87	0.006	0.85	0.13			
11 (3.00)	**1.87**	1.87	0.75	2.04	0.16	1.55	0.09	2.06	1.87	0.20		
12 (6.20)	**2.34**	2.53	0.49	2.12	0.20	2.34	0.07	2.30	0.52	0.31	0.73	
13 (8.89)	**4.84**	4.78	1.15	4.69	0.41	4.77	0.11	6.47	1.98	0.69	2.04	2.13

Table 3: Mutual information among features in *SNARE*; packet-based features are shown with numbers in dark circles. (The indices are the feature ranking in Table 2.)

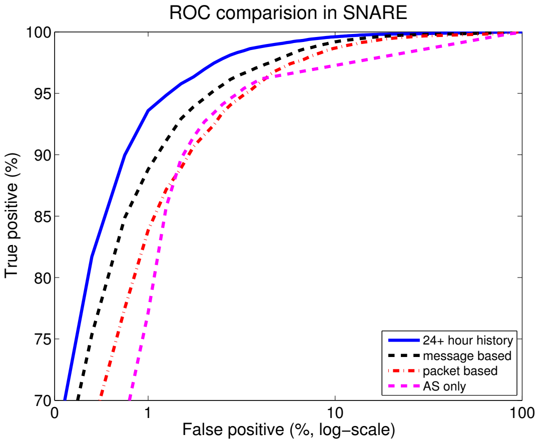


Figure 10: ROC comparison with AS-only case.

tics (e.g., feature 2, 4, 8) also have high values with each other. Because spammers may exhibit one or more of these features across each message, aggregating the features across multiple message over time indicates that, observing a spammer over time will reveal many of these features, though not necessarily on any message or single group of message. For this reason, aggregate features are likely to share high mutual information with other features that are common to spammers.

One possible reason that aggregate features have high mutual information with each other is that aggregating the features across multiple messages over time incorporates history of an IP address that may exhibit many of these characteristics over time.

Performance based on AS number only Since AS number is the most influential feature according to *RuleFit* and shares high mutual information with many other features, we investigated how well this feature alone can distinguish spammers from legitimate senders. We feed the AS feature into the predictive model and plot the ROC as the lower dashed curve in Figure 10. To make a close comparison, the "packet-based", "message-based", and "history-based" ROCs (the same as those in Figure 8) are shown as well, and the false positive is displayed on a log scale. The classifier gets false positive 0.76% under a 70% detection rate. Recall from Table 1 the false positive rate with "packet-based" features is almost a half, 0.44%, and that with "history-based" features will further reduce to 0.20%, which demonstrates that other features help to improve the performance. We also note that using the AS number alone as a distinguishing feature may cause large amounts of legitimate email to be misclassified, and could be evaded if an spammer decides to announce routes with a forged origin AS (which is an easy attack to mount and a somewhat common occurrence) [2, 26, 39].

5 A Spam-Filtering System

This section describes how *SNARE*'s reputation engine could be integrated into an overall spam-filtering system that includes a whitelist and an opportunity to continually retrain the classifier on labeled data (e.g., from spam traps, user inboxes, etc.). Because *SNARE*'s reputation engine still has a non-zero false positive rate, we show how it might be incorporated with mechanisms that could help further improve its accuracy, and also prevent discarding legitimate mail even in the case of some false positives. We propose an overview of the system and evaluate the benefits of these two functions on overall system accuracy.

5.1 System Overview

Figure 11 shows the overall system framework. The system needs not reside on a single server. Large public email providers might run their own instance of *SNARE*, since they have plenty of email data and processing resources. Smaller mail servers might query a remote

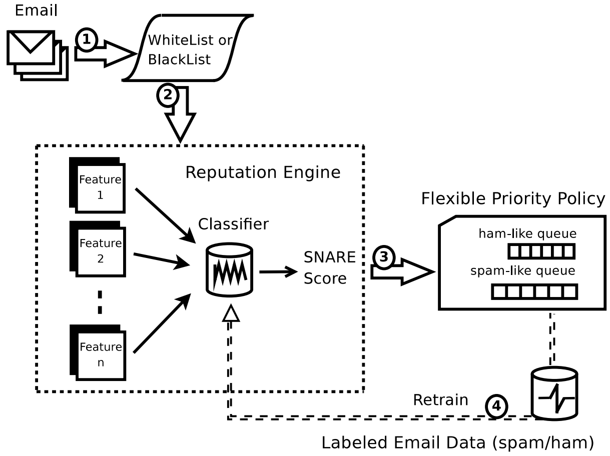


Figure 11: *SNARE* framework.

SNARE server. We envision that *SNARE* might be integrated into the workflow in the following way:

1. **Email arrival.** After getting the first packet, the mail server submits a query to the *SNARE* server (only the source and destination IP). Mail servers can choose to send more information to *SNARE* after getting the SMTP header or the whole message. Sending queries on a single packet or on a message is a tradeoff between detection accuracy and processing time for the email (i.e., sending the request early will make mail server get the response early). The statistics of messages in the received queries will be used to build up the *SNARE* classifier.
2. **Whitelisting.** The queries not listed in the whitelist will be passed to *SNARE*'s reputation engine (presented in Section 2.3) *before* any spam-filtering checks or content-based analysis. The output is a score, where, by default, positive value means likely spam and negative value means likely ham; and the absolute values represent the confidence of the classification. Administrators can set a different score threshold to make tradeoff between the false positive and the detection rate. We evaluate the benefits of whitelisting in Section 5.2.1.
3. **Greylisting and content-based detection.** Once the reputation engine calculates a score, the email will be delivered into different queues. More resource-sensitive and time-consuming detection methods (e.g., content-based detection) can be applied at this point. When the mail server has the capability to receive email, the messages in ham-like queue have higher priority to be processed, whereas the messages in spam-like queue will be offered less resources. This policy allows the server to speed up processing the messages that *SNARE* classifies as spam. The advantage of this hierarchical detecting scheme is that the legitimate email will be delivered to users' inbox sooner. Messages in the spam-like queue could be shunted to more resource-intensive spam filters before they are ultimately dropped.[3]
4. **Retraining** Whether the IP address sends spam or legitimate mail in that connection is not known at the time of the request, but is known after mail is processed by the spam filter. *SNARE* depends on accurately labelled training data. The email will eventually receive more careful checks (shown as "Retrain" in Figure 11). The results from those filters are considered as ground truth and can be used as feedback to dynamically adjust the *SNARE* threshold. For example, when the mail server has spare resource or much email in the spam-like queue is considered as legitimate later, *SNARE* system will be asked to act more generous to score email as likely ham; on the other hand, if the mail server is overwhelmed or the ham-like queue has too many incorrect labels, *SNARE* will be less likely to put email into ham-like queue. Section 5.2.2 evaluates the benefits of retraining for different intervals.

5.2 Evaluation

In this section, we evaluate how the two additional functions (whitelisting and retraining) improve *SNARE*'s overall accuracy.

5.2.1 Benefits of Whitelisting

We believe that a whitelist can help reduce *SNARE*'s overall false positive rate. To evaluate the effects of such a whitelist, we examined the features associated with the false positives, and determine that, 43% of all of *SNARE*'s false positives for a single day originate from just 10 ASes. We examined this characteristic for different days and found that 30% to 40% of false positives from any given day originate from the top 10 ASes. Unfortunately, however, these top 10 ASes do not remain the same from day-to-day, so the whitelist may need to be retrained periodically. It may also be the case that other features besides AS number of the source provide an even better opportunity for whitelisting. We leave the details of refining the whitelist for future work.

Figure 12 shows the average ROC curve when we whitelist the top 50 ASes responsible for most misclassified ham in each day. This whitelisting reduces the best

[3] Although *SNARE*'s false positive rates are quite low, some operators may feel that any non-zero chance that legitimate mail or sender might be misclassified warrants at least a second-pass through a more rigorous filter.

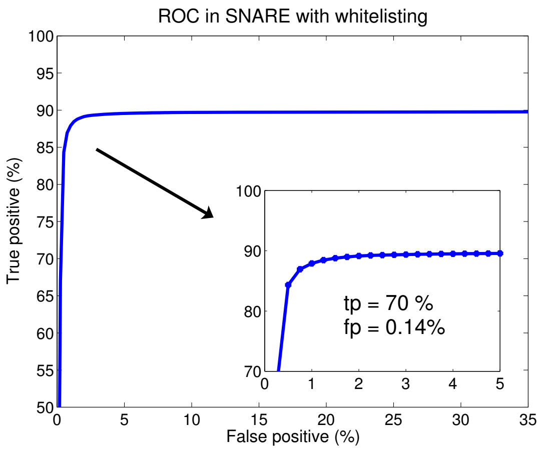


Figure 12: ROC in *SNARE* with whitelisting on ASes.

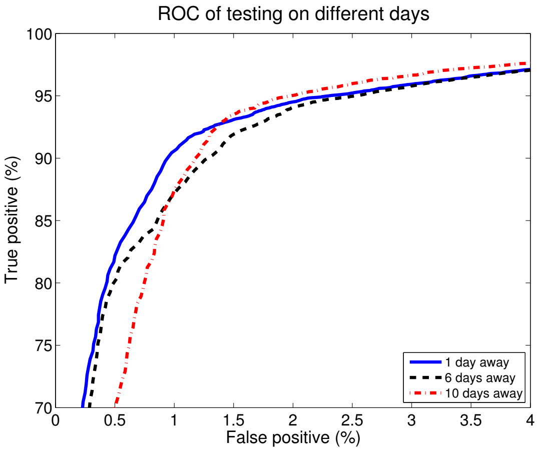


Figure 13: ROC using previous training rules to classify future messages.

possible detection rate considerably (effectively because about 11% of spam originates from those ASes). However, this whitelisting also reduces the false positive rate to about 0.14% for a 70% detection rate. More aggressive whitelisting, or whitelisting of other features, could result in even lower false positives.

5.2.2 Benefits of Retraining

Setup Because email sender behavior is dynamic, training *SNARE* on data from an earlier time period may eventually grow stale. To examine the requirements for periodically retraining the classifier, we train *SNARE* based on the first 3 days' data (through October 23–25, 2007) and test on the following 10 days. As before, we use 1 million randomly sampled spam and ham messages to test the classifier for each day.

Results Figure 13 shows the false positive and true positive on 3 future days, October 26, October 31, and November 4, 2007, respectively. The prediction on future days will become more inaccurate with time passage. For example, on November 4 (ten days after training), the false positive rate has dropped given the same true positive on the ROC curve. This result suggests that, for the spammer behavior in this trace, retraining *SNARE*'s classification algorithms daily should be sufficient to maintain accuracy. (We expect that the need to retrain may vary across different datasets.)

6 Discussion and Limitations

In this section, we address various aspects of *SNARE* that may present practical concerns. We first discuss the extent to which an attacker might be able to evade various features, as well as the extent to which these features might vary across time and datasets. We then discuss scalability concerns that a production deployment of *SNARE* may present, as well as various possible workarounds.

6.1 Evasion-Resistance and Robustness

In this section, we discuss the evasion resistance of the various network-level features that form the inputs to *SNARE*'s classification algorithm. Each of these features is, to some degree, evadable. Nevertheless, *SNARE* raises the bar by making it more difficult for spammers to evade detection without altering the techniques that they use to send spam. Although spammers might adapt to evade some of the features below, we believe that it will be difficult for a spammer to adjust all features to pass through *SNARE*, particularly without somewhat reducing the effectiveness of the spamming botnet. We survey each of the features from Table 2 in turn.

AS number AS numbers are more persistently associated with a sender's identity than the IP address, for two reasons: (1) The spamming mail server might be set up within specific ASes without the network administrator shutting it down. (2) Bots tend to aggregate within ASes, since the machines in the same ASes are likely to have the same vulnerability. It is not easy for spammers to move mail servers or the bot armies to a different AS; therefore, AS numbers are robust to indicate malicious hosts.

Message length In our analysis, we discovered that the size of legitimate email messages tends to be much more variable than that of spam (perhaps because spammers often use templates to sent out large quantities of mail [25]). With knowledge of this feature, a spammer might start to randomize the lengths of their email messages; this attack would not be difficult to mount, but it might restrict the types of messages that a spammer could send or make it slightly more difficult to coordinate a massive spam campaign with similar messages.

Nearest neighbor distances Nearest neighbor distance is another feature that will be hard to modify. Distances to k nearest neighbors effectively isolate existence of unusually large number of email servers within a small sequence of IP addresses. If the spammers try to alter their neighborhood density, they will not be able to use too many machines within a compromised subnet to send spam to the same set of destinations. Although it is possible for a botnet controller to direct bots on the same subnet to target different sets of destinations, such evasion does require more coordination and, in some cases, may restrict the agility that each spamming bot has in selecting its target destinations.

Status of email service ports Some limitation might fail the active probes, e.g., the outgoing mail servers use own protocol to mitigate messages (such as Google mail) or a firewall blocks the connections from out of the domain. But the bots do not open such ports with high probability, and the attackers need to get root privilege to enable those ports (which requires more sophisticated methods and resources). The basic idea is to find out whether the sender is a legitimate mail server. Although we used active probes in *SNARE*, other methods could facilitate the test, such as domain name checking or mail server authentication.

Sender-receiver geodesic distance The distribution of geodesic distances between the spammers' physical location and their target IP's location is a result of the spammers' requirement to reach as many target mail boxes as possible and in the shortest possible time. Even in a large, geographically distributed botnet, requiring each bot to bias recipient domains to evade this feature may limit the flexibility of how the botnet is used to send spam. Although this feature can also be evaded by tuning the recipient domains for each bot, if bots only sent spam to nearby recipients, the flexibility of the botnet is also somewhat restricted: it would be impossible, for example, to mount a coordinate spam campaign against a particular region from a fully distributed spamming botnet.

Number of recipients We found that spam messages tend to have more recipients than legitimate messages; a spammer could likely evade this feature by reducing the number of recipients on each message, but this might make sending the messages less efficient, and it might alter the sender behavior in other ways that might make a spammer more conspicuous (e.g., forcing the spammer to open up more connections).

Time of day This feature may be less resistant to evasion than others. Having said that, spamming botnets' diurnal pattern results from when the infected machines are switched on. For botnets to modify their diurnal message volumes over the day to match the legitimate message patterns, they will have to lower their spam volume in the evenings, especially between 3:00 p.m. and 9:00 p.m. and also reduce email volumes in the afternoon. This will again reduce the ability of botnets to send large amounts of email.

6.2 Other Limitations

We briefly discuss other current limitations of *SNARE*, including its ability to scale to a large number of recipients and its ability to classify IP addresses that send both spam and legitimate mail.

Scale *SNARE* must ultimately scale to thousands of domains and process hundreds of millions of email addresses per day. Unfortunately, even state-of-the-art machine learning algorithms are not well equipped to process datasets this large; additionally, sending data to a central coordinator for training could potentially consume considerably bandwidth. Although our evaluation suggests that *SNARE*'s classification is relatively robust to sampling of training data, we intend to study further the best ways to sample the training data, or perhaps even perform in-network classification.

Dual-purpose IP addresses Our conversations with large mail providers suggest that one of the biggest emerging threats are "web bots" that send spam from Web-based email accounts [35]. As these types of attacks develop, an increasing fraction of spam may be sent from IP addresses that also send significant amounts of legitimate mail. These cases, where an IP address is neither good nor bad, will need more sophisticated classifiers and features, perhaps involving timeseries-based features.

7 Related Work

We survey previous work on characterizing the network-level properties and behavior of email senders, email sender reputation systems, and other email filtering systems that are not based on content.

Characterization studies Recent characterization studies have provided increasing evidence that spammers have distinct network-level behavioral patterns. Ramachandran *et al.* [34] showed that spammers utilize transient botnets to spam at low rate from any specific IP to any domain. Xie *et al.* [38] discovered that a vast majority of mail servers running on dynamic IP address were used solely to send spam. In their recently published study [37], they demonstrate a technique to identify bots by using signatures constructed from URLs in spam messages. Unlike *SNARE*, their signature-based botnet identification differs heavily on analyzing message content. Others have also examined correlated behavior of botnets, primarily for characterization as opposed to detection [25, 31]. Pathak *et al.* [29] deployed a relay sinkhole to gather data from multiple spam senders destined for multiple domains. They used this data to demonstrate how spammers utilize compromised relay servers to evade detection; this study looked at spammers from multiple vantage points, but focused mostly on characterizing spammers rather than developing new detection mechanisms. Niu *et al.* analyzed network-level behavior of Web spammers (e.g., URL redirections and "doorway" pages) and proposed using context-based analysis to defend against Web spam [28].

Sender reputation based on network-level behavior SpamTracker [34] is most closely related to *SNARE*; it uses network-level behavioral features from data aggregated across multiple domains to infer sender reputation. While that work initiated the idea of *behavioral blacklisting*, we have discovered many other features that are more lightweight and more evasion-resistant than the single feature used in that paper. Beverly and Sollins built a similar classifier based on transport-level characteristics (e.g., round-trip times, congestion windows) [12], but their classifier is both heavyweight, as it relies on SVM, and it also requires accepting the messages to gather the features. Tang *et al.* explored the detection of spam senders by analyzing the behavior of IP addresses as observed by query patterns [36]. Their work focuses on the breadth and the periodicity of message volumes in relation to sources of queries. Various previous work has also attempted to cluster email senders according to groups of recipients, often with an eye towards spam filtering [21, 24, 27], which is similar in spirit to *SNARE*'s geodesic distance feature; however, these previous techniques typically require analysis of message contents, across a large number of recipients, or both, whereas *SNARE* can operate on more lightweight features. McAfee's TrustedSource [4] and Cisco IronPort [3] deploy spam filtering appliances to hundreds or thousands of domains which then query the central server for sender reputation and also provide meta-data about messages they receive; we are working with McAfee to deploy *SNARE* as part of TrustedSource.

Non-content spam filtering Trinity [14] is a distributed, content-free spam detection system for messages originating from botnets that relies on message volumes. The SpamHINTS project [9] also has the stated goal of building a spam filter using analysis of network traffic patterns instead of the message content. Clayton's earlier work on extrusion detection involves monitoring of server logs at both the local ISP [16] as well as the remote ISP [17] to detect spammers. This work has similar objectives as ours, but the proposed methods focus more on properties related to SMTP sessions from only a single sender.

8 Conclusion

Although there has been much progress in content-based spam filtering, state-of-the-art systems for *sender reputation* (e.g., DNSBLs) are relatively unresponsive, incomplete, and coarse-grained. Towards improving this state of affairs, this paper has presented *SNARE*, a sender reputation system that can accurately and automatically classify email senders based on features that can be determined early in a sender's history—sometimes after seeing only a single IP packet.

Several areas of future work remain. Perhaps the most uncharted territory is that of using temporal features to improve accuracy. All of *SNARE*'s features are essentially discrete variables, but we know from experience that spammers and legitimate senders also exhibit different temporal patterns. In a future version of *SNARE*, we aim to incorporate such temporal features into the classification engine. Another area for improvement is making *SNARE* more evasion-resistant. Although we believe that it will be difficult for a spammer to evade *SNARE*'s features and still remain effective, designing classifiers that are more robust in the face of active attempts to evade and mis-train the classifier may be a promising area for future work.

Acknowledgments

We thank our shepherd, Vern Paxson, for many helpful suggestions, including the suggestions to look at mutual information between features and several other improvements to the analysis and presentation. We also thank Wenke Lee, Anirudh Ramachandran, and Mukarram bin Tariq for helpful comments on the paper. This work was funded by NSF CAREER Award CNS-0643974 and NSF Awards CNS-0716278 and CNS-0721581.

References

[1] FCrDNS Lookup Testing. `http://ipadmin.junkemailfilter.com/rdns.php`.

[2] Internet Alert Registry. `http://iar.cs.unm.edu/`.

[3] IronPort. `http://www.ironport.com`.

[4] McAfee Secure Computing. `http://www.securecomputing.com`.

[5] SORBS: Spam and Open Relay Blocking System. `http://www.au.sorbs.net/`.

[6] SpamCop. `http://www.spamcop.net/bl.shtml`.

[7] SpamHaus IP Blocklist. `http://www.spamhaus.org`.

[8] GeoIP API. MaxMind, LLC. `http://www.maxmind.com/app/api`, 2007.

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

[10] DNSBL Resource: Statistics Center. `http://stats.dnsbl.com/`, 2008.

[11] Alperovitch, D., Judge, P., and Krasser, S. Taxonomy of email reputation systems. In *Proc. of the First International Workshop on Trust and Reputation Management in Massively Distributed Computing Systems (TRAM)* (2007).

[12] Beverly, R., and Sollins, K. Exploiting the transport-level characteristics of spam. In *5th Conference on Email and Anti-Spam (CEAS)* (2008).

[13] Boykin, P., and Roychowdhury, V. Personal email networks: An effective anti-spam tool. *IEEE Computer 38*, 4 (2005), 61–68.

[14] Brodsky, A., and Brodsky, D. A distributed content independent method for spam detection. In *First Workshop on Hot Topics in Understanding Botnets (HotBots)* (2007).

[15] Burges, C. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery 2*, 2 (1998), 121–167.

[16] Clayton, R. Stopping spam by extrusion detection. In *First Conference of Email and Anti-Spam (CEAS)* (2004).

[17] Clayton, R. Stopping outgoing spam by examining incoming server logs. In *Second Conference on Email and Anti-Spam (CEAS)* (2005).

[18] Friedman, J., and Popescu, B. Gradient directed regularization. *Stanford University, Technical Report* (2003).

[19] Friedman, J., and Popescu, B. Predictive learning via rule ensembles. *Annals of Applied Statistics (to appear)* (2008).

[20] Golbeck, J., and Hendler, J. Reputation network analysis for email filtering. In *First Conference on Email and Anti-Spam (CEAS)* (2004).

[21] Gomes, L. H., Castro, F. D. O., Almeida, R. B., Bettencourt, L. M. A., Almeida, V. A. F., and Almeida, J. M. Improving spam detection based on structural similarity. In *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)* (2005).

[22] Goodman, J., Cormack, G., and Heckerman, D. Spam and the ongoing battle for the inbox. *Communications of the ACM 50*, 2 (2007), 24–33.

[23] Hulton, E., and Goodman, J. Tutorial on junk email filtering. *Tutorial in the 21st International Conference on Machine Learning (ICML)* (2004).

[24] Johansen, L., Rowell, M., Butler, K., and McDaniel, P. Email communities of interest. In *4th Conference on Email and Anti-Spam (CEAS)* (2007).

[25] Kanich, C., Kreibich, C., Levchenko, K., Enright, B., Paxson, V., Voelker, G. M., and Savage, S. Spamalytics: An empirical analysis of spam marketing conversion. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)* (2008).

[26] Karlin, J., Forrest, S., and Rexford, J. Autonomous security for autonomous systems. *Computer Networks 52*, 15 (2008), 2908–2923.

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

[28] Niu, Y., Wang, Y.-M., Chen, H., Ma, M., and Hsu, F. A quantitative study of forum spamming using context-based analysis. In *Proceedings of the 14th Annual Network and Distributed System Security Symposium (NDSS)* (2007).

[29] Pathak, A., Hu, C., Y., and Mao, Z., M. Peeking into spammer behavior from a unique vantage point. In *First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)* (2008).

[30] Quinlan, J. Induction of decision trees. *Machine Learning 1*, 1 (1986), 81–106.

[31] Rajab, M., Zarfoss, J., Monrose, F., and Terzis, A. A multifaceted approach to understanding the botnet phenomenon. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (IMC)* (2006).

[32] Ramachandran, A., Dagon, D., and Feamster, N. Can DNSBLs keep up with bots? In *3rd Conference on Email and Anti-Spam (CEAS)* (2006).

[33] Ramachandran, A., and Feamster, N. Understanding the network-level behavior of spammers. In *Proceedings of the ACM SIGCOMM* (2006).

[34] Ramachandran, A., Feamster, N., and Vempala, S. Filtering spam with behavioral blacklisting. In *ACM Conference on Computer and Communications Security (CCS)* (2007).

[35] Private conversation with Mark Risher, Yahoo Mail., 2008.

[36] Tang, Y. C., Krasser, S., Judge, P., and Zhang, Y.-Q. Fast and effective spam IP detection with granular SVM for spam filtering on highly imbalanced spectral mail server behavior data. In *2nd International Conference on Collaborative Computing (CollaborateCom)* (2006).

[37] Xie, Y., Yu, F., , Achan, K., Panigrahy, R., Hulten, G., and Osipkov, I. Spamming bots: Signatures and characteristics. In *Proceedings of ACM SIGCOMM* (2008).

[38] Xie, Y., Yu, F., Achan, K., Gilum, E., Goldszmidt, M., and Wobber, T. How dynamic are IP addresses. In *Proceedings of ACM SIGCOMM* (2007).

[39] 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 (IMW)* (2001).

Improving Tor using a TCP-over-DTLS Tunnel

Joel Reardon[*]
Google Switzerland GmbH
Brandschenkestrasse 110
Zürich, Switzerland
reardon@google.com

Ian Goldberg
University of Waterloo
200 University Ave W.
Waterloo, ON, Canada
iang@cs.uwaterloo.ca

Abstract

The Tor network gives anonymity to Internet users by relaying their traffic through the world over a variety of routers. All traffic between any pair of routers, even if they represent circuits for different clients, are multiplexed over a single TCP connection. This results in interference across circuits during congestion control, packet dropping and packet reordering. This interference greatly contributes to Tor's notorious latency problems.

Our solution is to use a TCP-over-DTLS (Datagram Transport Layer Security) transport between routers. We give each stream of data its own TCP connection, and protect the TCP headers—which would otherwise give stream identification information to an attacker—with DTLS. We perform experiments on our implemented version to illustrate that our proposal has indeed resolved the cross-circuit interference.

1 Introduction

Tor [2] is a tool to enable Internet privacy that has seen widespread use and popularity throughout the world. Tor consists of a network of thousands of nodes—known as Onion Routers (ORs)—whose operators have volunteered to relay Internet traffic around the world. Clients—known as Onion Proxies (OPs)—build circuits through ORs in the network to dispatch their traffic. Tor's goal is to frustrate an attacker who aims to match up the identities of the clients with the actions they are performing. Despite its popularity, Tor has a problem that dissuades its ubiquitous application—it imposes greater latency on its users than they would experience without Tor.

While some increased latency is inevitable due to the increased network path length, our experiments show that this effect is not sufficient to explain the increased cost. In Section 2 we look deeper, and find a component of the transport layer that can be changed to improve Tor's performance. Specifically, each pair of routers maintains a single TCP connection for all traffic that is sent between them. This includes multiplexed traffic for different circuits, and results in cross-circuit interference that degrades performance. We find that congestion control mechanisms are being unfairly applied to all circuits when they are intended to throttle only the noisy senders. We also show how packet dropping on one circuit causes interference on other circuits.

Section 3 presents our solution to this problem—a new transport layer that is backwards compatible with the existing Tor network. Routers in Tor can gradually and independently upgrade, and our system provides immediate benefit to any pair of routers that choose to use our improvements. It uses a separate TCP connection for each circuit, but secures the TCP header to avoid the disclosure of per-circuit data transfer statistics. Moreover, it uses a user-level TCP implementation to address the issue of socket proliferation that prevents some operating systems from being able to volunteer as ORs.

Section 4 presents experiments to compare the existing Tor with our new implementation. We compare latency and throughput, and perform timing analysis of our changes to ensure that they do not incur non-negligible computational latency. Our results are favourable: the computational overhead remains negligible and our solution is successful in addressing the improper use of congestion control.

Section 5 compares our enhanced Tor to other anonymity systems, and Section 6 concludes with a description of future work.

1.1 Apparatus

Our experiments were performed on a commodity Thinkpad R60—1.66 GHz dual core with 1 GB of RAM. Care was taken during experimentation to ensure that the system was never under load significant enough to influ-

[*] Work done while at the University of Waterloo

ence the results. Our experiments used a modified version of the Tor 0.2.0.x stable branch code.

2 Problems with Tor's Transport Layer

We begin by briefly describing the important aspects of Tor's current transport layer. For more details, see [2]. An end user of Tor runs an Onion Proxy on her machine, which presents a SOCKS proxy interface [7] to local applications, such as web browsers. When an application makes a TCP connection to the OP, the OP splits it into fixed-size *cells* which are encrypted and forwarded over a *circuit* composed of (usually 3) Onion Routers. The last OR creates a TCP connection to the intended destination host, and passes the data between the host and the circuit.

The circuit is constructed with hop-by-hop TCP connections, each protected with TLS [1], which provides confidentiality and data integrity. The OP picks a first OR (OR_1), makes a TCP connection to it, and starts TLS on that connection. It then instructs OR_1 to connect to a particular second OR (OR_2) of the OP's choosing. If OR_1 and OR_2 are not already in contact, a TCP connection is established between them, again with TLS. If OR_1 and OR_2 *are* already in contact (because other users, for example, have chosen those ORs for their circuits), the existing TCP connection is used for all traffic between those ORs. The OP then instructs OR_2 to contact a third OR, OR_3, and so on. Note that there is *not* an end-to-end TCP connection from the OP to the destination host, nor to any OR except OR_1.

This multi-hop transport obviously adds additional unavoidable latency. However, the observed latency of Tor is larger than accounted for simply by the additional transport time. In [12], the first author of this paper closely examined the sources of latency in a live Tor node. He found that processing time and input buffer queueing times were negligible, but that output buffer queueing times were significant. For example, on an instrumented Tor node running on the live Tor network, 40% of output buffers had data waiting in them from 100 ms to over 1 s more than 20% of the time. The data was waiting in these buffers because the the operating system's output buffer for the corresponding socket was itself full, and so the OS was reporting the socket as unwritable. This was due to TCP's congestion control mechanism, which we discuss next.

Socket output buffers contain two kinds of data: packet data that has been sent over the network but is unacknowledged[1], and packet data that has not been sent due to TCP's congestion control. Figure 1 shows the size of the socket output buffer over time for a particular connection. First, unwritable sockets occur when the remaining capacity in an output buffer is too small to accept new data. This in turn occurs because there is already too much data in the buffer, which is because there is too much unacknowledged data in flight and throttled data waiting to be sent. The congestion window (CWND) is a variable that stores the number of packets that TCP is currently willing to send to the peer. When the number of packets in flight exceeds the congestion window then the sending of more data is throttled until acknowledgments are received. Once congestion throttles sending, the data queues up until either packets are acknowledged or the buffer is full.

In addition to congestion control, TCP also has a flow control mechanism. Receivers advertise the amount of data they are willing to accept; if more data arrives at the receiver before the receiving application has a chance to read from the OS's receive buffers, this advertised receiver window will shrink, and the sender will stop transmitting when it reaches zero. In none of our experiments did we ever observe Tor throttling its transmissions due to this mechanism; the advertised receiver window sizes never dropped to zero, or indeed below 50 KB. Congestion control, rather than flow control, was the reason for the throttling.

While data is delayed because of congestion control, it is foolhardy to attempt to circumvent congestion control as a means of improving Tor's latency. However, we observe that Tor's transport between ORs results in an *unfair* application of congestion control. In particular, Tor's circuits are multiplexed over TCP connections; i.e., a single TCP connection between two ORs is used for multiple circuits. When a circuit is built through a pair of unconnected routers, a new TCP connection is established. When a circuit is built through an already-connected pair of ORs, the existing TCP stream will carry both the existing circuits and the new circuit. This is true for all circuits built in either direction between the ORs.

In this section we explore how congestion control affects multiplexed circuits and how packet dropping and reordering can cause interference across circuits. We show that TCP does not behave optimally when circuits are multiplexed in this manner.

2.1 Unfair Congestion Control

We believe that multiplexing TCP streams over a single TCP connection is unwise and results in the unfair application of TCP's congestion control mechanism. It results in multiple data streams competing to send data over a TCP stream that gives more bandwidth to circuits that send more data; i.e., it gives each byte of data the same priority regardless of its source. A busy circuit that

[1] Recall that TCP achieves reliability by buffering all data locally until it has been acknowledged, and uses this to generate retransmission messages when necessary

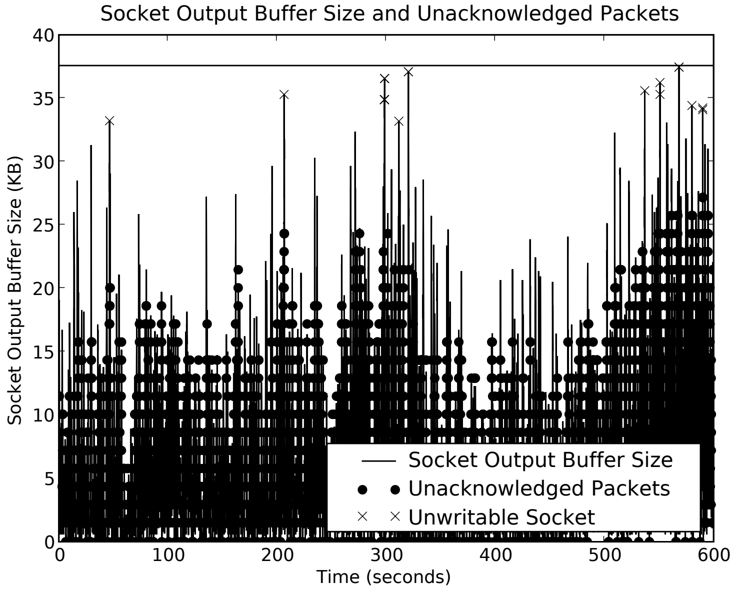


Figure 1: TCP socket output buffer size, writability, and unacknowledged packets over time.

triggers congestion control will cause low-bandwidth circuits to struggle to have their data sent. Figure 2 illustrates data transmission for distinct circuits entering and exiting a single output buffer in Tor. Time increases along the X-axis, and data increases along the Y-axis. The main part of the figure shows two increasing line shapes, each corresponding to the data along a different circuit over time. When the shapes swell, that indicates that Tor's internal output buffer has swelled: the left edge grows when data enters the buffer, and the right edge grows when data leaves the buffer. This results in the appearance of a line when the buffer is well-functioning, and a triangular or parallelogram shape when data arrives too rapidly or the connection is troubled. Additionally, we strike a vertical line across the graph whenever a packet is dropped.

What we learn from this graph is that the buffer serves two circuits. One circuit serves one MB over ten minutes, and sends cells evenly. The other circuit is inactive for the most part, but three times over the execution it suddenly serves 200 KB of cells. We can see that each time the buffer swells with data it causes a significant delay. Importantly, the other circuit is affected despite the fact that it did not change its behaviour. Congestion control mechanisms that throttle the TCP connection will give preference to the burst of writes because it simply provides more data, while the latency for a low-bandwidth application such as `ssh` increases unfairly.

2.2 Cross-Circuit Interference

Tor multiplexes the data for a number of circuits over a single TCP stream, and this ensures that the received data will appear in the precise order in which the component streams were multiplexed—a guarantee that goes beyond what is strictly necessary. When packets are dropped or reordered, the TCP stack will buffer available data on input buffers until the missing in-order component is available. We hypothesize that when active circuits are multiplexed over a single TCP connection, Tor suffers an unreasonable performance reduction when either packet dropping or packet reordering occur. Cells may be available in-order for one particular circuit but are being delayed due to missing cells for another circuit. In-order guarantees are only necessary for data sent within a single circuit, but the network layer ensures that data is only readable in the order it was dispatched. Packet loss or reordering will cause the socket to indicate that no data is available to read even if other circuits have their sequential cells available in buffers.

Figure 3 illustrates the classic head-of-line blocking behaviour of Tor during a packet drop; cells for distinct circuits are represented by shades and a missing packet is represented with a cross. We see that the white, light grey, and black circuits have had all of their data successfully received, yet the kernel will not pass that data to the Tor application until the dropped dark grey packet is retransmitted and successfully received.

We verify our cross-circuit interference hypothesis in two parts. In this section we show that packet drops on a

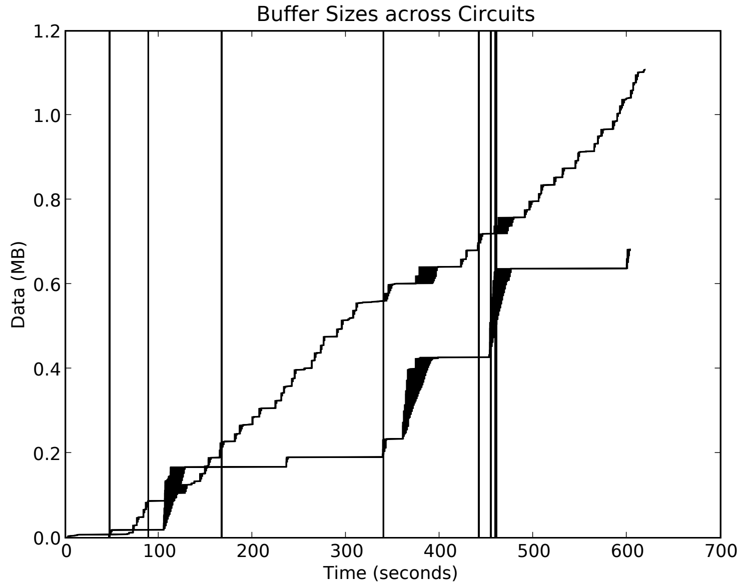


Figure 2: Example of congestion on multiple streams.

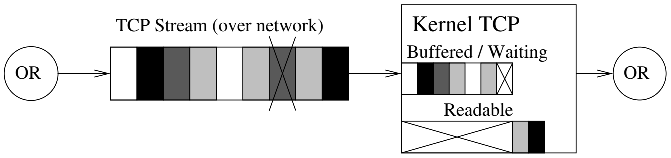


Figure 3: TCP correlated streams. Shades correspond to cells for different circuits.

Experiment 1 Determining the effect of packet dropping on circuit multiplexing.

1: A Tor network of six ORs on a single host was configured to have a latency of 50 milliseconds and a variable packet drop rate.
2: Eight OP built circuits that were fixed so that the second and third ORs were the same for each client, but the first hop was evenly distributed among the remaining ORs. Figure 4 illustrates this setup.
3: There were three runs of the experiment. The first did not drop any packets. The second dropped 0.1% of packets on the shared link, and the third dropped 0.1% of packets on the remaining links.
4: The ORs were initialized and then the clients were run until circuits were established.
5: Each OP had a client connect, which would tunnel a connection to a timestamp server through Tor. The server sends a continuous stream of timestamps. The volume of timestamps measures throughput, and the difference in time measures latency.
6: Data was collected for one minute.

shared link degrade throughput much more severely than drops over unshared links. Then in Section 4 we show that this effect disappears with our proposed solution.

To begin, we performed Experiment 1 to investigate the effect of packet dropping on circuit multiplexing. The layout of circuits in the experiment, as shown in Figure 4, is chosen so that there is one shared link that carries data for all circuits, while the remaining links do not.

In the two runs of our experiments that drop packets, they are dropped according to a target drop rate, either on the heavily shared connection or the remaining connections. Our packet dropping tool takes a packet, decides if it is eligible to be dropped in this experiment, and if so then it drops it with the appropriate probability. However, this model means the two runs that drop packets will see different rates of packet dropping systemwide, since we observe greater traffic on the remaining connections. This is foremost because it spans two hops along the circuit instead of one, and also because traffic from multiple circuits can be amalgamated into one packet for transmission along the shared connection. As a result, a fixed drop rate affecting the remaining connec-

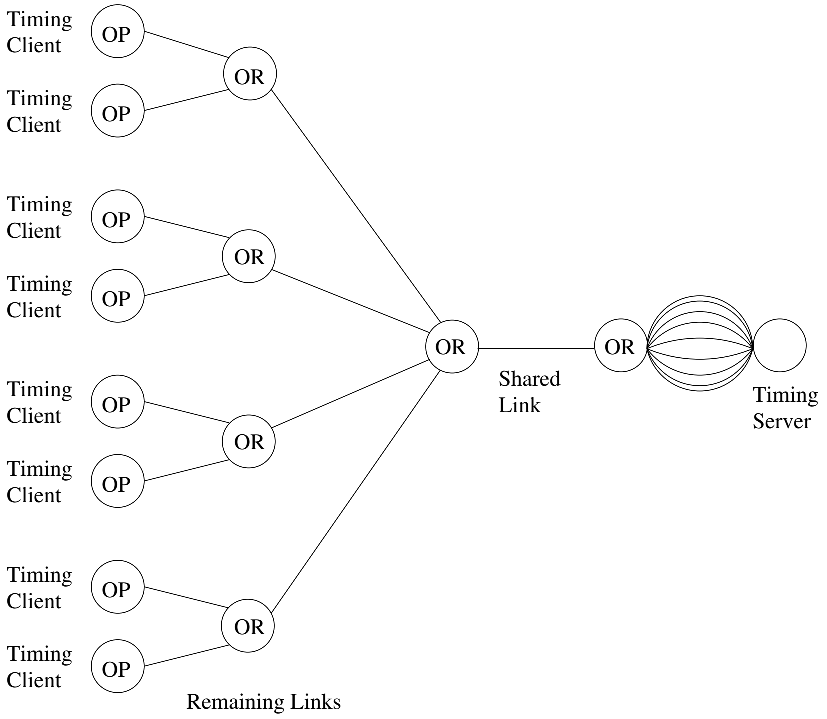


Figure 4: Setup for Experiment 1. The shared link multiplexes all circuits from the various OPs to the final OR; the remaining links carry just one or two circuits each. The splay of links between the final OR and the timing server reflect the fact that a separate TCP connection is made from the final OR to the timing server for each timing client.

Configuration	Network Throughput (KB/s)	Circuit Throughput (KB/s)	Throughput Degradation	Effective Drop Rate
No dropping	221 ± 6.6	36.9 ± 1.1	0 %	0 %
0.1 % (remaining)	208 ± 14	34.7 ± 2.3	6 %	0.08 %
0.1 % (shared)	184 ± 17	30.8 ± 2.8	17 %	0.03 %

Table 1: Throughput for different dropping configurations. Network throughput is the total data sent along all the circuits.

Configuration	Average Latency	Latency Increase	Effective Drop Rate
No dropping	933 ± 260 ms	0 %	0 %
0.1 % (remaining)	983 ± 666 ms	5.4 %	0.08 %
0.1 % (shared)	1053 ± 409 ms	12.9 %	0.03 %

Table 2: Latency for different dropping configurations.

tions will result in more frequent packet drops than one dropping only along the shared connection. This disparity is presented explicitly in our results as the effective drop rate; i.e., the ratio of packets dropped to the total number of packets we observed (including those ineligible to be dropped) in the experiment.

The results of Experiment 1 are shown in Tables 1 and 2. They show the results for three configurations: when no packet dropping is done, when 0.1% of packets are dropped on all connections except the heavily shared one, and when 0.1% of packets are dropped only on the shared connection. The degradation column refers to the loss in performance as a result of introducing packet drops. The average results for throughput and delay were accumulated over half a dozen executions of the experiment, and the mean intervals for the variates are computed using Student's T distribution to 95% confidence.

These results confirm our hypothesis. The throughput degrades nearly threefold when packets are dropped on the shared link instead of the remaining links. This is despite a significantly lower overall drop rate. The behaviour of one TCP connection can adversely affect all correlated circuits, even if those circuits are used to transport less data.

Table 2 suggests that latency increases when packet dropping occurs. Latency is measured by the time required for a single cell to travel alongside a congested circuit, and we average a few dozen such probes. Again we see that dropping on the shared link more adversely affects the observed delay despite a reduced drop rate. However, we note that the delay sees wide variance, and the 95% confidence intervals are quite large.

2.3 Summary

Multiplexing circuits over a single connection is a potential source of unnecessary latency since it causes TCP's congestion control mechanism to operate unfairly towards connections with smaller demands on throughput. High-bandwidth streams that trigger congestion control result in low-bandwidth streams having their congestion window unfairly reduced. Packet dropping and reordering also cause available data for multiplexed circuits to wait needlessly in socket buffers. These effects degrade both latency and throughput, which we have shown in experiments.

To estimate the magnitude of this effect in the real Tor network, we note that 10% of Tor routers supply 87% of the total network bandwidth [8]. A straightforward calculation shows that links between top routers—while only comprising 1% of the possible network links—transport over 75% of the data. At the time of writing, the number of OPs is estimated in the hundreds of thousands and there are only about one thousand active ORs [14]. Therefore, even while most users are idle, the most popular 1% of links will be frequently multiplexing circuits.

Ideally, we would open a separate TCP connection for every circuit, as this would be a more appropriate use of TCP between ORs; packet drops on one circuit, for example, would not hold up packets in other circuits. However, there is a problem with this naive approach. An adversary observing the network could easily distinguish packets for each TCP connection just by looking at the port numbers, which are exposed in the TCP headers. This would allow him to determine which packets were part of which circuits, affording him greater opportunity for traffic analysis. Our solution is to tunnel packets from multiple TCP streams over DTLS, a UDP protocol that provides for the confidentiality of the traffic it transports. By tunnelling TCP over a secure protocol, we can protect both the TCP payload and the TCP headers.

3 Proposed Transport Layer

This section proposes a TCP-over-DTLS tunnelling transport layer for Tor. This tunnel transports TCP packets between peers using DTLS—a secure datagram (UDP-based) transport [9]. A user-level TCP stack running inside Tor generates and parses TCP packets that are sent over DTLS between ORs. Our solution will use a single unconnected UDP socket to communicate with all other ORs at the network level. Internally, it uses a separate user-level TCP connection for each circuit. This decorrelates circuits from TCP streams, which we have shown to be a source of unnecessary latency. The use of DTLS also provides the necessary security and confidentiality of the transported cells, including the TCP header. This prevents an observer from learning per-circuit metadata such data how much data is being sent in each direction for individual circuits. Additionally, it reduces the number of sockets needed in kernel space, which is known to be a problem that prevents some Windows computers from volunteering as ORs. Figure 5 shows the design of our proposed transport layer, including how only a single circuit is affected by a dropped packet.

The interference that multiplexed circuits can have on each other during congestion, dropping, and reordering is a consequence of using a single TCP connection to transport data between each pair of ORs. This proposal uses a separate TCP connection for each circuit, ensuring that congestion or drops in one circuit will not affect other circuits.

3.1 A TCP-over-DTLS Tunnel

DTLS [9] is the datagram equivalent to the ubiquitous TLS protocol [1] that secures much traffic on the Inter-

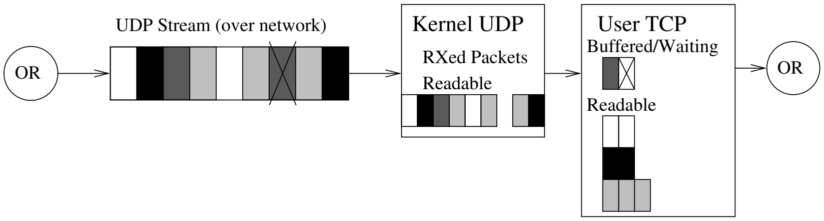


Figure 5: Proposed TCP-over-DTLS Transport showing decorrelated streams. Shades correspond to cells for different circuits (cf. Figure 3).

net today, including https web traffic, and indeed Tor. DTLS provides confidentiality and authenticity for Internet datagrams, and provides other security properties such as replay prevention. IPsec [6] would have been another possible choice of protocol to use here; however, we chose DTLS for our application due to its acceptance as a standard, its ease of use without kernel or superuser privileges, and its existing implementation in the OpenSSL library (a library already in use by Tor). The TLS and DTLS APIs in OpenSSL are also unified; after setup, the same OpenSSL calls are used to send and receive data over either TLS or DTLS. This made supporting backwards compatibility easier: the Tor code will send packets either over TCP (with TLS) or UDP (with DTLS), as appropriate, with minimal changes.

Our new transport layer employs a user-level TCP stack to generate TCP packets, which are encapsulated inside a DTLS packet that is then sent by the system in a UDP/IP datagram. The receiving system will remove the UDP/IP header when receiving data from the socket, decrypt the DTLS payload to obtain a TCP packet, and translate it into a TCP/IP packet, which is then forwarded to the user-level TCP stack that processes the packet. A subsequent read from the user-level TCP stack will provide the packet data to our system.

In our system, the TCP sockets reside in user space, and the UDP sockets reside in kernel space. The use of TCP-over-DTLS affords us the great utility of TCP: guaranteed in-order delivery and congestion control. The user-level TCP stack provides the functionality of TCP, and the kernel-level UDP stack is used simply to transmit packets. The secured DTLS transport allows us to protect the TCP header from snooping and forgery and effect a reduced number of kernel-level sockets.

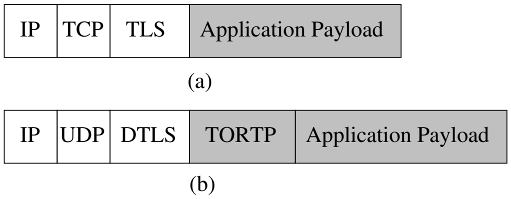
ORs require opening many sockets, and so our user-level TCP stack must be able to handle many concurrent sockets, instead of relying on the operating system's TCP implementation that varies from system to system. In particular, some discount versions of Windows artificially limit the number of sockets the user can open, and so we use Linux's free, open-source, and high-performance TCP implementation inside user space. Even Windows users will be able to benefit from an improved TCP implementation, and thus any user of an operating system supported by Tor will be able to volunteer their computer as an OR if they so choose.

UDP allows sockets to operate in an unconnected state. Each time a datagram is to be sent over the Internet, the destination for the packet is also provided. Only one socket is needed to send data to every OR in the Tor network. Similarly, when data is read from the socket, the sender's address is also provided alongside the data. This allows a single socket to be used for reading from all ORs; all connections and circuits will be multiplexed over the same socket. When reading, the sender's address can be used to demultiplex the packet to determine the appropriate connection for which it is bound. What follows is that a single UDP socket can be used to communicate with as many ORs as necessary; the number of kernel-level sockets is constant for arbitrarily many ORs with which a connection may be established. This will become especially important for scalability as the number of nodes in the Tor network grows over time. From a configuration perspective, the only new requirement is that the OR operator must ensure that a UDP port is externally accessible; since they must already ensure this for a TCP port we feel that this is a reasonable configuration demand.

Figure 6(a) shows the packet format for TCP Tor, and Figure 6(b) shows the packet format for our TCP-over-DTLS Tor, which has expanded the encrypted payload to include the TCP/IP headers generated by the user-level TCP stack. The remainder of this section will discuss how we designed, implemented, and integrated these changes into Tor.

3.2 Backwards Compatibility

Our goal is to improve Tor to allow TCP communication using UDP in the transport layer. While the original ORs transported cells between themselves, our proposal is to transport, using UDP, both TCP headers and cells between ORs. The ORs will provide the TCP/IP packets to a TCP stack that will generate both the appropriate stream of cells to the Tor application, as well



(a)

(b)

Figure 6: Packets for TCP Tor and our TCP-over-DTLS improved Tor. Encrypted and authenticated components of the packet are shaded in grey. (a) shows the packet format for TCP Tor. (b) shows the packet format for our TCP-over-DTLS Tor. TORTP is a compressed form of the IP and TCP headers, and is discussed in Section 3.4.

as TCP/IP packets containing TCP acknowledgements to be returned.

The integration of this transport layer into Tor has two main objectives. The first is that of interoperability; it is essential that the improved Tor is backwards compatible with the TCP version of Tor so as to be easily accepted into the existing codebase. Recall that Tor has thousands of ORs, a client population estimated in the hundreds of thousands, and has not experienced any downtime since it launched in 2003. It is cumbersome to arrange a synchronized update of an unknown number of anonymous Tor users. A subset of nodes that upgrade and can take advantage of TCP-over-DTLS can provide evidence of the transport's improvement for the user experience—this incremental upgrade is our preferred path to acceptance. Our second objective is to minimize the changes required to the Tor codebase. We must add UDP connections into the existing datapath by reusing as much existing code as possible. This permits future developers to continue to improve Tor's datapath without having to consider two classes of communication. Moreover, it encourages the changes to quickly be incorporated into the main branch of the source code. While it will come with a performance cost for doing unnecessary operations, we perform timing analyses below to ensure that the resulting datapath latency remains negligible.

Interoperability between existing ORs and those using our improved transport is achieved by fully maintaining the original TCP transport in Tor—improved ORs continue to advertise a TCP OR port and multiplexed TCP connections can continue to be made. In addition, improved nodes will also advertise a UDP port for making TCP-over-DTLS connections. Older nodes will ignore this superfluous value, while newer nodes will always choose to make a TCP-over-DTLS connection whenever such a port is advertised. Thus, two UDP nodes will automatically communicate using UDP without disrupting the existing nodes; their use of TCP-over-DTLS is inconsequential to the other nodes. As more nodes support TCP-over-DTLS, more users will obtain its benefits, but we do not require a synchronized update to support our improvements.

Clients of Tor are not required to upgrade their software to obtain the benefits of UDP transport. If two nodes on their circuit use TCP-over-DTLS to communicate then this will happen transparently to the user. In fact, it is important that the user continue to choose their circuit randomly among the ORs: intentionally choosing circuits consisting of UDP nodes when there are only a few such nodes decreases the privacy afforded to the client by rendering their circuit choices predictable.

3.3 User-level TCP Stack

If we simply replaced the TCP transport layer in Tor with a UDP transport layer, our inter-OR communication would then lack the critical features of TCP: guaranteed in-order transmission of streams, and the most well-studied congestion control mechanism ever devised. We wish to remove some of the unnecessary guarantees of TCP for the sake of latency; i.e., we do not need cells from separate circuits over the same connection to arrive in the order they were dispatched. However, we must still be able to reconstruct the streams of each individual circuit at both ends of the connection. We use a TCP implementation in user space (instead of inside the operating system) to accommodate us; a user-level TCP stack provides the implementation of the TCP protocols [4] as part of our program. User-level socket file descriptors and their associated data structures and buffers are accessible only in user space and so are visible and relevant only to Tor. We use the UDP transport layer and DTLS to transport TCP packets between the UDP peers. Only part of the TCP packet is transmitted; the details will be discussed in section 3.4, but it serves our purposes now to conceptualize the two nodes as transporting full TCP packets as the UDP datagram's payload. Upon receiving a UDP datagram, the kernel will remove the UDP header and provide Tor with the enclosed DTLS packet; Tor will decrypt the DTLS payload and present the result (a TCP packet) to its user-level TCP stack. Similarly, when the user-level TCP stack presents a packet for transmission, the node will encrypt it with DTLS and forward the resulting packet to the kernel which then sends it to the intended destination over UDP. The stack also performs retransmission and acknowledgement of TCP data that are integral to TCP's reliability; these are protected with DTLS and forwarded over UDP in the same manner.

A user-level TCP stack provides an implementation of the suite of socket function calls, such as *socket()*, *send()*, and *recv()*. These reimplementations exist in harmony with the proper set of operating system commands, allowing both a user-level and kernel-level network layer. Thus, data structures and file descriptors created by calls to the user-level stack are visible and relevant only to the parent process; the operating system manages its sockets separately. The user-level stack responds to socket calls by generating packets internally for dispatching as dictated by TCP.

It may seem cumbersome to include an entire TCP implementation as a core component of Tor. In particular, patching the kernel's implementation of TCP to support our features would take significantly less effort. However, Tor relies on volunteers to route traffic; complicated installation procedures are an immediate roadblock towards the ubiquitous use of Tor. The diverse operating systems Tor aims to support and the diverse skill level of its users prevent its installation from requiring external procedures, or even superuser privileges.

Daytona [11] is a user-level TCP stack that we chose for our purposes. It was created by researchers studying network analysis, and consists of the implementation of Linux's TCP stack and the reimplementations of user-level socket functions. It uses `libpcap` to capture packets straight from the Ethernet device and a raw socket to write generated packets, including headers, onto the network. Daytona was designed to operate over actual networks while still giving user-level access to the network implementation. In particular, it allowed the researchers to tune the implementation while performing intrusive measurements. A caveat—there are licensing issues for Daytona's use in Tor. As a result, the deployment of this transport layer into the real Tor network may use a different user-level TCP stack. Our design uses Daytona as a replaceable component and its selection as a user-level TCP stack was out of availability for our proof-of-concept.

3.4 UTCP: Our Tor-Daytona Interface

Our requirements for a user-level TCP stack are to create properly formatted packets, including TCP retransmissions, and to sort incoming TCP/IP packets into data streams: a black box that converts between streams and packets. For our purpose, all notions of routing, Ethernet devices, and interactions with a live network are unnecessary. To access the receiving and transmitting of packets, we commandeer the `rx()` (receive) and `tx()` (transmit) methods of Daytona to instead interface directly with reading and writing to connections in Tor.

UTCP is an abstraction layer for the Daytona TCP stack used as an interface for the stack by Tor. Each UDP connection between ORs has a UTCP-connection object that maintains information needed by our stack, such as the set of circuits between those peers and the socket that listens for new connections. Each circuit has a UTCP-circuit object for similar purposes, such as the local and remote port numbers that we have assigned for this connection.

As mentioned earlier, only part of the TCP header is transmitted using Tor—we call this header the TORTP header; we do this simply to optimize network traffic. The source and destination addresses and ports are replaced with a numerical identifier that uniquely identifies the circuit for the connection. Since a UDP/IP header is transmitted over the actual network, Tor is capable of performing a connection lookup based on the address of the packet sender. With the appropriate connection, and a circuit identifier, the interface to Daytona is capable of translating the TORTP header into the corresponding TCP header.

When the UTCP interface receives a new packet, it uses local data and the TORTP headers to create the corresponding TCP header. The resulting packet is then injected into the TCP stack. When Daytona's TCP stack emits a new packet, a generic *tx()* method is invoked, passing only the packet and its length. We look up the corresponding UTCP circuit using the addresses and ports of the emitted TCP header, and translate the TCP header to our TORTP header and copy the TCP payload. This prepared TORTP packet is then sent to Tor, along with a reference to the appropriate circuit, and Tor sends the packet to the destination OR over the appropriate DTLS connection.

3.5 Congestion Control

The congestion control properties of the new scheme will inherit directly from those of TCP, since TCP is the protocol being used internally. While it is considered an abuse of TCP's congestion control to open multiple streams between two peers simply to send more data, in this case we are legitimately opening one stream for each circuit carrying independent data. When packets are dropped, causing congestion control to activate, it will only apply to the single stream whose packet was dropped. Congestion control variables are not shared between circuits; we discuss the possibility of using the message-oriented Stream Control Transport Protocol (SCTP), which shares congestion control information, in Section 5.2.

If a packet is dropped between two ORs communicating with multiple streams of varying bandwidth, then the drop will be randomly distributed over all circuits with a probability proportional to their volume of traffic over the link. High-bandwidth streams will see their packets dropped more often and so will back off appropriately. Multiple streams will back off in turn until the congestion is resolved. Streams such as ssh connections that send data interactively will always be allowed to have at least one packet in flight regardless of the congestion on other circuits.

Another interesting benefit of this design is that it gives Tor direct access to TCP parameters at runtime. The lack of sophistication in Tor's own congestion control mechanism is partially attributable to the lack of direct access to networking parameters at the kernel level. With the TCP stack in user space Tor's congestion control can be further tuned and optimized. In particular, end-to-end congestion control could be gained by extending our work to have each node propagate its TCP rate backwards along the circuit: each node's rate will be the minimum of TCP's desired rate and the value reported by the subsequent node. This will address congestion imbalance issues where high-bandwidth connections send traffic faster than it can be dispatched at the next node, resulting in data being buffered upon arrival. When TCP rates are propagated backwards, then the bandwidth between two ORs will be prioritized for data whose next hop has the ability to immediately send the data. Currently there is no consideration for available bandwidth further along the circuit when selecting data to send.

4 Experimental Results

In this section we perform experiments to compare the existing Tor transport layer with an implementation of our proposed TCP-over-DTLS transport. We begin by timing the new sections of code to ensure that we have not significantly increased the computational latency. Then we perform experiments on a local Tor network of routers, determining that our transport has indeed addressed the cross-circuit interference issues previously discussed.

4.1 Timing Analysis

Our UDP implementation expands the datapath of Tor by adding new methods for managing user-level TCP streams and UDP connections. We profile our modified Tor and perform static timing analysis to ensure that our new methods do not degrade the datapath unnecessarily. Experiment 2 was performed to profile our new version of Tor.

The eightieth percentile of measurements for Experiment 2 are given in Table 3. Our results indicate that no new datapath component results in a significant source of computational latency.

Experiment 2 Timing analysis of our modified TCP-over-DTLS datapath.

1: TCP-over-DTLS Tor was modified to time the duration of the aspects of the datapath:

- injection of a new packet (DTLS decryption, preprocessing, injecting into TCP stack, possibly sending an acknowledgment),
- emission of a new packet (header translation, DTLS encryption, sending packet),
- the TCP timer function (increments counters and checks for work such as retransmissions and sending delayed acknowledgements), and
- the entire datapath from reading a packet on a UDP socket, demultiplexing the result, injecting the packet, reading the stream, processing the cell, writing the result, and transmitting the generated packet.

2: The local Tor network was configured to use 50 ms of latency between connections.

3: A client connected through Tor to request a data stream.

4: Data travelled through the network for several minutes.

We have increased the datapath latency to an expected value of 250 microseconds per OR, or 1.5 milliseconds for a round trip along a circuit of length three. This is still an order of magnitude briefer than the round-trip times between ORs on a circuit (assuming geopolitically diverse circuit selection). Assuming each packet is 512 bytes (the size of a cell—a conservative estimate as our experiments have packets that carry full dataframes), we have an upper bound on throughput of 4000 cells per second or 2 MB/s. While this is a reasonable speed that will likely not form a bottleneck, Tor ORs that are willing to devote more than 2 MB/s of bandwidth may require better hardware than the Thinkpad R60 used in our experiments.

4.2 Basic Throughput

We perform Experiment 3 to compare the basic throughput and latency of our modification to Tor, the results of which are shown in Table 4. We can see that the UDP version of Tor has noticeably lower throughput. Originally it was much lower, and increasing the throughput up to this value took TCP tuning and debugging the user-level TCP stack. In particular, errors were uncovered in Daytona's congestion control implementation, and it is

Datapath Component	Duration
Injecting Packet	100 microseconds
Transmitting Packet	100 microseconds
TCP Timer	85 microseconds
Datapath	250 microseconds

Table 3: Time durations for new datapath components. The results provided are the 80th percentile measurement.

Configuration	Network Throughput	Circuit Delay	Base Delay
TCP Tor	176 ± 24.9 KB/s	1026 ± 418 ms	281 ± 12 ms
TCP-over-DTLS Tor	111 ± 10.4 KB/s	273 ± 31 ms	260 ± 1 ms

Table 4: Throughput and delay for different reordering configurations. The configuration column shows which row correspond to which version of Tor we used for our ORs in the experiment. Network throughput is the average data transfer rate we achieved in our experiment. Circuit delay is the latency of the circuit while the large bulk data transfer was occurring, whereas the base delay is the latency of the circuit taken in the absence of any other traffic.

suspected that more bugs remain to account for this disparity. While there may be slight degradation in performance when executing TCP operations in user space instead of kernel space, both implementations of TCP are based on the same Linux TCP implementation operating over in the same network conditions, so we would expect comparable throughputs as a result. With more effort to resolve outstanding bugs, or the integration of a user-level TCP stack better optimized for Tor's needs, we expect the disparity in throughputs will vanish. We discuss this further in the future work section.

More important is that the circuit delay for a second stream over the same circuit indicates that our UDP version of Tor vastly improves latency in the presence of a high-bandwidth circuit. When one stream triggers the congestion control mechanism, it does not cause the low-bandwidth client to suffer great latency as a consequence. In fact, the latency observed for TCP-over-DTLS is largely attributable to the base latency imposed on connections by our experimental setup. TCP Tor, in contrast, shows a three-and-a-half fold increase in latency when the circuit that it multiplexes with the bulk stream is burdened with traffic.

The disparity in latency for the TCP version means that information is leaked: the link between the last two nodes is witnessing bulk transfer. This can be used as a reconnaissance technique; an entry node, witnessing a bulk transfer from an client and knowing its next hop, can probe potential exit nodes with small data requests to learn congestion information. Tor rotates its circuits every ten minutes. Suppose the entry node notices a bulk transfer when it begins, and probes various ORs to determine the set of possible third ORs. It could further reduce this set by re-probing after nine minutes, after which time most of the confounding circuits would have rotated to new links.

We conclude that our TCP-over-DTLS, while currently suffering lower throughput, has successfully addressed the latency introduced by the improper use of the congestion control mechanism. We expect that once perfected, the user-level TCP stack will have nearly the same throughput as the equivalent TCP implementation in the kernel. The response latency for circuits in our improved Tor is nearly independent of throughput on existing Tor circuits travelling over the same connections; this improves Tor's usability and decreases the ability for one circuit to leak information about another circuit using the same connection through interference.

4.3 Multiplexed Circuits with Packet Dropping

Packet dropping occurs when a packet is lost while being routed through the Internet. Packet dropping, along with packet reordering, are consequences of the implementation of packet switching networks and are the prime reason for the invention of the TCP protocol. In this section, we perform an experiment to contrast the effect of packet dropping on the original version of Tor and our improved version.

We reperformed Experiment 1—using our TCP-over-DTLS implementation of Tor instead of the standard implementation—to investigate the effect of packet dropping. The results are presented in Tables 5 and 6. We reproduce our results from Tables 1 and 2 to contrast the old (TCP) and new (TCP-over-DTLS) transports.

We find that throughput is much superior for the TCP-over-DTLS version of Tor. This is likely because the

Version	Configuration	Network Throughput (KB/s)	Circuit Throughput (KB/s)	Throughput Degradation	Effective Drop Rate
TCP-over-DTLS	No dropping	284 ± 35	47.3 ± 5.8	0 %	0 %
	0.1 % (remain.)	261 ± 42	43.5 ± 7.0	8 %	0.08 %
	0.1 % (shared)	270 ± 34	45.2 ± 5.6	4 %	0.03 %
TCP	No dropping	221 ± 6.6	36.9 ± 1.1	0 %	0 %
	0.1 % (remain.)	208 ± 14	34.7 ± 2.3	6 %	0.08 %
	0.1 % (shared)	184 ± 17	30.8 ± 2.8	17 %	0.03 %

Table 5: Throughput for different dropping configurations.

Version	Configuration	Average Latency	Latency Degradation	Effective Drop Rate
TCP-over-DTLS	No dropping	428 ± 221 ms	0 %	0 %
	0.1 % (remaining)	510 ± 377 ms	20 %	0.08 %
	0.1 % (shared)	461 ± 356 ms	7 %	0.03 %
TCP	No dropping	933 ± 260 ms	0 %	0 %
	0.1 % (remaining)	983 ± 666 ms	5.4 %	0.08 %
	0.1 % (shared)	1053 ± 409 ms	12.9 %	0.03 %

Table 6: Latency for different dropping configurations.

TCP congestion control mechanism has less impact on throttling when each TCP stream is separated. One stream may back off, but the others will continue sending, which results in a greater throughput over the bottleneck connection. This is reasonable behaviour since TCP was designed for separate streams to function over the same route. If congestion is a serious problem then multiple streams will be forced to back off and find the appropriate congestion window. Importantly, the streams that send a small amount of data are much less likely to need to back off, so their small traffic will not have to compete unfairly for room inside a small congestion window intended to throttle a noisy connection. The benefits of this are clearly visible in the latency as well: cells can travel through the network considerably faster in the TCP-over-DTLS version of Tor. Despite the large confidence intervals for latency mentioned earlier, we see now that TCP-over-DTLS consistently has significantly lower latency than the original TCP Tor.

The TCP-over-DTLS version has its observed throughput and latency affected proportionally to packet drop rate. It did not matter if the drop was happening on the shared link or the remaining link, since the shared link is not a single TCP connection that multiplexes all traffic. Missing cells for different circuits no longer cause unnecessary waiting, and so the only effect on latency and throughput is the effect of actually dropping cells along circuits.

5 Alternative Approaches

There are other means to improve Tor's observed latency than the one presented in this paper. For comparison, in this section we outline two significant ones: UDP-OR, and SCTP-over-DTLS.

5.1 UDP-OR

Another similar transport mechanism for Tor has been proposed by Viecco [15] that encapsulates TCP packets from the OP and sends them over UDP until they reach the exit node. Reliability guarantees and congestion control are handled by the TCP stacks on the client and the exit nodes, and the middle nodes only forward traffic. A key design difference between UDP-OR and our proposal is that ours intended on providing backwards compatibility with the existing Tor network while Viecco's proposal requires a synchronized update of the Tor software for all users. This update may be cumbersome given that Tor has thousands of routers and an unknown number of clients estimated in the hundreds of thousands.

This strategy proposes benefits in computational complexity and network behaviour. Computationally, the middle nodes must no longer perform unnecessary operations: packet injection, stream read, stream write, packet generation, and packet emission. It also removes the responsibility of the middle node to handle retrans-

Experiment 3 Basic throughput and delay for TCP and TCP-over-DTLS versions of Tor.

1: To compare TCP and TCP-over-DTLS we run the experiment twice: one where all ORs use the original TCP version of time, and one where they all use our modified TCP-over-DTLS version of Tor.
2: A local Tor network running six routers on a local host was configured to have a latency of 50 milliseconds.
3: Two OPs are configured to connect to our local Tor network. They use distinct circuits, but each OR along both circuit is the same. The latency-OP will be used to measure the circuit's latency by sending periodic timestamp probes over Tor to a timing server. The throughput-OP will be used to measure the circuit's throughput by requesting a large bulk transfer and recording the rate at which it arrives.
4: We start the latency-OP's timestamp probes and measure the latency of the circuit. Since we have not begun the throughput-OP, we record the time as the base latency of the circuit.
5: We begin the throughput-OP's bulk transfer and measure throughput of the circuit. We continue to measure latency using the latency-OP in the presence of other traffic. The latency results that are collected are recorded separately from those of step 4.
6: Data was collected for over a minute, and each configuration was run a half dozen times to obtain confidence intervals.

missions, which means a reduction in its memory requirements. The initial endpoint of communication will be responsible for retransmitting the message if necessary. We have shown that computational latency is insignificant in Tor, so this is simply an incidental benefit.

The tangible benefit of UDP-OR is to improve the network by allowing the ORs to function more exactly like routers. When cells arrive out of order at the middle node, they will be forwarded regardless, instead of waiting in input buffers until the missing cell arrives. Moreover, by having the sender's TCP stack view both hops as a single network, we alleviate problems introduced by disparity in network performance. Currently, congestion control mechanisms are applied along each hop, meaning that an OR in the middle of two connections with different performance metrics will need to buffer data to send over the slower connection. Tor provides its own congestion control mechanism, but it does not have the sophistication of TCP's congestion control.

We require experimentation to determine if this proposal is actually beneficial. While it is clear that memory requirements for middle nodes are reduced [15], the endpoints will see increased delay for acknowledgements. We expect an equilibrium for total system memory requirements since data will be buffered for a longer time. Worse, the approach shifts memory requirements from being evenly distributed to occurring only on exit nodes—and these nodes are already burdened with extra responsibilities. Since a significant fraction of Tor nodes volunteer only to forward traffic, it is reasonable to use their memory to ease the burden of exit nodes.

Circuits with long delays will also suffer reduced throughput, and so using congestion control on as short a path as possible will optimize performance. If a packet is dropped along the circuit, the endpoint must now generate the retransmission message, possibly duplicating previous routing efforts and wasting valuable volunteered bandwidth. It may be more efficient to have nodes along a circuit return their CWND for the next hop, and have each node use the minimum of their CWND and the next hop's CWND. Each node then optimizes their sending while throttling their receiving.

5.1.1 Low-cost Privacy Attack

UDP-OR may introduce an attack that permits a hostile entry node to determine the final node in a circuit. Previously each OR could only compute TCP metrics for ORs with whom they were directly communicating. Viecco's system would have the sender's TCP stack communicate indirectly with an anonymous OR. Connection attributes, such as congestion and delay, are now known for the longer connection between the first and last nodes in a circuit. The first node can determine the RTT for traffic to the final node. It can also reliably compute the RTT for its connection to the middle node. The difference in latency reflects the RTT between the second node and the anonymous final node. An adversary can use a simple technique to estimate the RTT between the second node and every other UDP Tor node in the network [3], possibly allowing them to eliminate many ORs from the final node's anonymity set. If it can reduce the set of possible final hops, other reconnaissance techniques can be applied, such as selectively flooding each OR outside of Tor and attempting to observe an increased latency inside Tor [10]. Other TCP metrics may be amalgamated to further aid this attack: congestion window, slow-start threshold, occurrence of congestion over time, standard deviation in round-trip times, etc. The feasibility of this attack should be examined before allowing nodes who do not already know each other's identities to share a TCP conversation.

5.2 Stream Control Transmission Protocol

The Stream Control Transmission Protocol (SCTP) [13] is a message-based transport protocol. It provides similar features to TCP: connection-oriented reliable delivery with congestion control. However, it adds the ability to automatically delimit messages instead of requiring the receiving application to manage its own delimiters. The interface is based on sending and receiving messages rather than bytes, which is appropriate for Tor's cell-based transport.

More importantly, SCTP also adds a feature well-suited to our purposes—multiple streams can be transported over the same connection. SCTP allows multiple independent ordered streams to be sent over the same socket; we can use this feature to assign each circuit a different stream. Cells from each circuit will arrive in the order they were sent, but the order cells arrive across all circuits may vary from they dispatch order. This is exactly the behaviour we want for cells from different circuits being sent between the same pair of ORs.

While SCTP is not as widely deployed as TCP, the concept of using a user-level SCTP implementation [5] inside Tor remains feasible. This suggests a SCTP-over-DTLS transport similar in design to our TCP-over-DTLS design. This means that the extra benefits of TCP-over-DTLS will also extend to SCTP-over-DTLS: backwards compatibility with the existing Tor network, a constant number of kernel-level sockets required, and a secured transport header.

What is most interesting about the potential of SCTP-over-DTLS is SCTP's congestion control mechanism. Instead of each TCP stream storing its own congestion control metrics, SCTP will share metrics and computations across all streams. An important question in the development of such a scheme is whether SCTP will act fairly towards streams that send little data when other streams invoke congestion control, and whether the sharing of congestion control metrics results in a privacy-degrading attack by leaking information.

6 Future Work

6.1 Live Experiments

The most pressing future work is to perform these experiments on live networks of geographically distributed machines running TCP-over-DTLS Tor, using computers from the PlanetLab network, or indeed on the live Tor network. Once running, we could measure latency and throughput as we have already in our experiments, comparing against results for regular Tor. Moreover, we can also compare other approaches, such as SCTP-over-DTLS and UDP-OR, using the same experiments. Note that UDP-OR could of course not be tested on the live Tor network, but it could be in a PlanetLab setup. A key metric will be the distribution of throughput and latency for high- and low-volume circuits before and after our improvements, and an analysis of the cause of the change. Additionally, once most ORs use UDP, we can determine if the reduced demand on open sockets solves the problem of socket proliferation on some operating systems.

6.2 TCP Stack Memory Management

Tor requires thousands of sockets to buffer fixed-size cells of data, but data is only buffered when it arrives out-of-order or has not been acknowledged. We envision dynamic memory management such as a shared cell pool to handle memory in Tor. Instead of repeatedly copying data cells from various buffers, each cell that enters Tor can be given a unique block of memory from the cell pool until it is no longer needed. A state indicates where this cell currently exists: input TCP buffer, input Tor buffer, in processing, output Tor buffer, output TCP buffer. This ensures that buffers are not allocated to store empty data, which reduces the overall memory requirements. Each cell also keeps track of its socket number, and its position in the linked list of cells for that socket. While each socket must still manage data such as its state and metrics for congestion control, this is insignificant as compared to the current memory requirements. This permits an arbitrary number of sockets, for all operating systems, and helps Tor's scalability if the number of ORs increases by orders of magnitude.

This approach results in the memory requirements of Tor being a function of the number of cells it must manage at any time, independent of the number of open sockets. Since the memory requirements are inextricably tied to the throughput Tor offers, the user can parameterize memory requirements in Tor's configuration just as they parameterize throughput. A client willing to denote more throughput than its associated memory requirements will have its contribution throttled as a result. If network conditions result in a surge of memory required for Tor, then it can simply stop reading from the UDP multiplexing socket. The TCP stacks that sent this unread data will assume there exists network congestion and consequently throttle their sending—precisely the behaviour we want—while minimizing leaked information about the size of our cell pool.

7 Summary

Anonymous web browsing is an important step in the development of the Internet, particularly as it grows ever more inextricable from daily life. Tor is a privacy-enhancing technology that provides Internet anonymity using volunteers to relay traffic, and uses multiple relays in series to ensure that no entity (other than the client) in the system is aware of both the source and destination of messages.

Relaying messages increases latency since traffic must travel a longer distance before it is delivered. However, the observed latency of Tor is much larger than just this effect would suggest. To improve the usability of Tor, we examined where this latency occurs, and found that it happens when data sat idly in buffers due to congestion control. Since multiple Tor circuits are multiplexed over a single TCP connection between routers, we observed cross-circuit interference due to the nature of TCP's in-order, reliable delivery and its congestion control mechanisms.

Our solution was the design and implementation of a TCP-over-DTLS transport between ORs. Each circuit was given a unique TCP connection, but the TCP packets themselves were sent over the DTLS protocol, which provides confidentiality and security to the TCP header. The TCP implementation is provided in user space, where it acts as a black box that translates between data streams and TCP/IP packets. We performed experiments on our implemented version using a local experimentation network and showed that we were successful in removing the observed cross-circuit interference and decreasing the observed latency.

Acknowledgements

We would like to thank Steven Bellovin, Vern Paxson, Urs Hengartner, S. Keshav, and the anonymous reviewiers for their helpful comments on improving this paper. We also gratefully acknowledge the financial support of The Tor Project, MITACS, and NSERC.

References

[1] Tim Dierks and Eric Rescorla. RFC 5246—The Transport Layer Security (TLS) Protocol Version 1.2. http://www.ietf.org/rfc/rfc5246.txt, August 2008.

[2] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. *Proceedings of the 13th USENIX Security Symposium*, 2004.

[3] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating Latency between Arbitrary Internet End Hosts. *ACM SIGCOMM Computer Communication Review*, 2002.

[4] Information Sciences Institute. RFC 793—Transmission Control Protocol. http://www.ietf.org/rfcs/rfc793.txt, September 1981.

[5] Andreas Jungmaier, Herbert Hölzlwimmer, Michael Tüxen, and Thomas Dreibholz. The SCTP library (sctplib). http://www.sctp.de/sctp-download.html, 2007. Accessed February 2009.

[6] Stephen Kent and Randall Atkinson. RFC 2401—Security Architecture for the Internet Protocol. http://www.ietf.org/rfcs/rfc2401.txt, November 1998.

[7] Marcus Leech et al. RFC 1928—SOCKS Protocol Version 5. http://www.ietf.org/rfc/rfc1928.txt, March 1996.

[8] Damon McCoy, Kevin Bauer, Dirk Grunwald, Parisa Tabriz, and Douglas Sicker. Shining Light in Dark Places: A Study of Anonymous Network Usage. University of Colorado Technical Report CU-CS-1032-07, August 2007.

[9] Nagendra Modadugu and Eric Rescorla. The Design and Implementation of Datagram TLS. *Network and Distributed System Security Symposium*, 2004.

[10] Steven J. Murdoch and George Danezis. Low-Cost Traffic Analysis of Tor. In *IEEE Symposium on Security and Privacy*, pages 183–195, 2005.

[11] Prashant Pradhan, Srikanth Kandula, Wen Xu, Anees Shaikh, and Erich Nahum. Daytona: A User-Level TCP Stack. http://nms.lcs.mit.edu/˜kandula/data/daytona.pdf, 2002.

[12] Joel Reardon. Improving Tor using a TCP-over-DTLS Tunnel. Master's thesis, University of Waterloo, Waterloo, ON, September 2008.

[13] Randall Stewart, Qiaobing Xie, Ken Morneualt, Chip Sharp, Hanns Juergen Schwarzbauer, Tom Taylor, Ian Rytina, Malleswar Kalla, Lixia Zhang, and Vern Paxson. RFC 2960—Stream Control Transmission Protocol. http://www.ietf.org/rfc/rfc2960.txt, October 2000.

[14] TorStatus. Tor Network Status. http://torstatus.kgprog.com/. Accessed February 2009.

[15] Camilo Viecco. UDP-OR: A Fair Onion Transport Design. http://www.petsymposium.org/2008/hotpets/udp-tor.pdf, 2008.

Locating Prefix Hijackers using LOCK

Tongqing Qiu
Georgia Tech
tongqqiu@cc.gatech.edu

Lusheng Ji
AT&T Labs – Research
lji@research.att.com

Dan Pei
AT&T Labs – Research
peidan@research.att.com

Jia Wang
AT&T Labs – Research
jiawang@research.att.com

Jun (Jim) Xu
Georgia Tech
jx@cc.gatech.edu

Hitesh Ballani
Cornell University
hitesh@cs.cornell.edu

Abstract

Prefix hijacking is one of the top known threats on today's Internet. A number of measurement based solutions have been proposed to detect prefix hijacking events. In this paper we take these solutions one step further by addressing the problem of locating the attacker in each of the detected hijacking event. Being able to locate the attacker is critical for conducting necessary mitigation mechanisms at the earliest possible time to limit the impact of the attack, successfully stopping the attack and restoring the service.

We propose a robust scheme named LOCK, for LOCating the prefix hijacKer ASes based on distributed Internet measurements. LOCK locates each attacker AS by actively monitoring paths (either in the control-plane or in the data-plane) to the victim prefix from a small number of carefully selected monitors distributed on the Internet. Moreover, LOCK is robust against various countermeasures that the hijackers may employ. This is achieved by taking advantage of two observations: that the hijacker cannot manipulate AS path before the path reaches the hijacker, and that the paths to victim prefix "converge" around the hijacker AS. We have deployed LOCK on a number of PlanetLab nodes and conducted several large scale measurements and experiments to evaluate the performance. Our results show that LOCK is able to pinpoint the prefix hijacker AS with an accuracy up to 94.3%.

1 Introduction

The Internet consists of tens of thousands of Autonomous Systems (ASes), each of which is an independently administrated domain. Inter-AS routing information is maintained and exchanged by the Border Gateway Protocol (BGP). The lack of adequate authentication schemes in BGP leaves an opportunity for misbehaving routers to advertise and spread fabricated AS paths for targeted prefixes. Originating such a false AS path announcement is referred to as "prefix hijacking". Once a BGP router accepts such a false route and replaces a legitimate route with it, the traffic destined for the target prefix can be redirected as the hijacker wishes. The victim prefix network of a successful hijacking will experience performance degradation, service outage, and security breach. The incident of the prefix of YouTube being hijacked by an AS in Pakistan for more than 2 hours [1] is just a recent and better known reminder of the possibility of real prefix hijacking attacks.

Recently proposed solutions for combating prefix hijacking either monitor the state of Internet and detect ongoing hijacking events [12, 21, 22, 26, 34, 45], or attempt to restore service for victim prefix networks [42]. Both approaches are compatible with existing routing infrastructures and generally considered more deployable than another family of proposals (e.g., [4, 8, 11, 13, 18, 19, 27, 32, 35–37, 44]) which aim at prefix hijacking prevention, because the latter usually require changes to current routing infrastructures (e.g., router software, network operations), and some also require public key infrastructures.

However, the aforementioned detection and service restoration solutions only solve parts of the problem and a critical step is still missing towards a complete and automated detection-recovery system. That is how to locate the hijackers. More importantly, the location of hijackers is one of the key information that enables existing mitigation methods against prefix hijacking (e.g., [42]). One may consider this step trivial. Indeed in current practice this step is actually accomplished by human interactions and manual inspections of router logs. However, we would argue that the success of the current practice is due to the fact that discovered attacks so far are still primitive. Many of them are simply not attacks but rather the results of router mis-configurations. As we will elaborate, locating sophisticated hijackers is far from a trivial problem and the current practice will have great difficulties in locating them.

In this paper, we present a scheme called LOCK to LOCate prefix hijacKers. It is a light-weight and incrementally deployable scheme for locating hijacker ASes. The main idea behind LOCK are based the following two observations: that the hijacker cannot manipulate AS path before the path reaches the hijacker, and that the paths to victim prefix "converge" around the hijacker AS. Our contributions are four-fold. First, to the best of our knowledge, it is the first work studying the attacker locating problem for prefix hijacking, even when countermeasures are engaged by the hijacker. Second, our locating scheme can use either data-plane or control-plane information, making the deployment more flexible in practice. Third, we propose an algorithm for selecting locations where data-plane or control-plane data are collected such that the hijackers can be more efficiently located. Finally, we have deployed LOCK on a number of PlanetLab nodes and conducted several large scale measurements and experiments to evaluate the performance of LOCK against three groups of hijacking scenarios: synthetic attacks simulated using real path and topology information collected on the Internet, reconstructed previously known attacks, and controlled attack experiments conducted on the Internet. We show that the proposed approach can effectively locate the attacker AS with up to 94.3% accuracy.

The rest of the paper is organized as follows. Section 2 provides background information on prefix hijacking. Section 3 provides an overview of the framework of our LOCK scheme. Then we describe detailed monitoring and locating methodologies in Section 4 and Section 5 respectively. Section 6 evaluates the performance of the LOCK scheme. Section 7 briefly surveys related works before Section 8 concludes the paper.

2 Background

As mentioned before, IP prefix hijacking occurs when a mis-configured or malicious BGP router either originates or announces an AS path for an IP prefix not owned by the router's AS. In these BGP updates the misbehaving router's AS appears very attractive as a next hop for forwarding traffic towards that IP prefix. ASes that receive such ill-formed BGP updates may accept and further propagate the false route. As a result the route entry for the IP prefix in these ASes may be *polluted* and traffic from certain part of the Internet destined for the victim prefix is redirected to the attacker AS.

Such weakness of the inter-domain routing infrastructure has great danger of being exploited for malicious purposes. For instance the aforementioned misbehaving AS can either drop all traffic addressed to the victim prefix that it receives and effectively perform a denial-of-service attack against the prefix owner, or redirect traffic to an incorrect destination and use this for a phishing attack [28]. It can also use this technique to spread spams [33].

We refer to this kind of route manipulation as *IP prefix hijack attacks* and the party conducting the attack *hijacker* or *attacker*. Correspondingly the misbehaving router's AS becomes the *hijacker AS*, and the part of the Internet whose traffic towards the victim prefix is redirected to the hijacker AS is *hijacked*. So do we call the data forwarding paths that are now altered to go through the hijacker AS *hijacked*. We also refer to the victim prefix as the *target prefix*.

Following the convention in [45], we classify prefix hijacks into the following three categories:

- **Blackholing**: the attacker simply drops the hijacked packets.
- **Imposture**: the attacker responds to senders of the hijacked traffic, mimicking the true destination's (the target prefix's) behavior.
- **Interception**: the attacker forwards the hijacked traffic to the target prefix after eavesdropping/recording the information in the packets.

While the conventional view of the damage of prefix hijacking has been focused on blackholing, the other two types of hijacking are equally important, if not more damaging [6]. In addition, the characteristics of different hijack types are different, which often affect how different types of attacks are detected. In this paper, we use the term *hijack* to refer to all three kinds of prefix hijack attacks unless otherwise specified.

There have been a number of approaches proposed for detecting prefix hijacks. They utilize either information in BGP updates collected from control plane [21, 22, 26, 34], or end-to-end probing information collected from data plane [43, 45], or both [6, 12, 36]. We will not get into the details of most of these approaches here because LOCK is a *hijacker-locating scheme*, not a hijack detection scheme. The difference between these two will be explained later in section 3.1. To locate a hijacker, the LOCK scheme only needs to know whether a given prefix is hijacked. Therefore LOCK can be used together with any detection method to further locate the hijacker AS. Moreover, LOCK can locate the hijacker using either data-plane or control-plane information.

3 Framework

In this section, we present an overview of key ideas of the hijacker locating algorithm in LOCK. Similar to detecting prefix hijacking, locating hijacker AS can be done in either control-plane or data-plane. Either way, the goal is, to use the AS path information to the hijacked prefix observed at multiple and diverse vantage points (or monitors) to figure out who the hijacker is. In control-plane approach, the AS path information is obtained from BGP routing tables or update messages. In data-plane approach, the AS path is obtained via AS-level traceroute (mapping the IP addresses in traceroute to AS numbers).

Both methods have pros and cons. Realtime data-plane information from multiple diverse vantage points is easier to be obtained than realtime BGP information(e.g. the BGP updates from [3] are typically delayed for a few hours). On the other hand, it is relatively easier for the attacker to manipulate the data-plane AS path to countermeasure the locating algorithm than the control-plane AS path. LOCK can use either data-plane or control-plane AS paths to locate the hijackers.

3.1 Challenges

Currently, the most commonly used hijacker-locating approach (called *simple locating approach*) is to look at the origin ASes of the target prefix. For example, Figure 1 (a) shows the control-plane AS path information to target prefix p at vantage points $M1$, $M2$, and $M3$, respectively, before hijacker H launches the hijack. All three vantage points observe the origin AS is T. In Figure 1 (b), Hijacker AS H announces a path H to target prefix p, which ASes A, B, $M1$, and $M2$ accept since the paths via H are better than their previous ones via CDT. In this case, the simple locating approach can easily identify the newly-appearing origin AS H as the hijacker.

However, this simple locating approach can fail even without any countermeasures by the hijackers. For example, in Figure 1(c), hijacker H pretends there is a link between H and the target AS T, and announces an AS path HT, again accepted by A,B,$M1$, and $M2$. The simple locating approach does not work here since the origin AS in all the AS paths are still T.

One might try to look beyond just origin AS and check other ASes in the path, but the hijacker AS might counter this such that the hijacker AS might not even appear in any of the AS paths. For example, in Figure 1(d) H simply announces an AS path T without prepending its own AS number H [1].

Above challenges in control-plane locating approaches also exist in data-plane approaches. Almost all data-plane path probing mechanisms are derived from the well known *traceroute* program. In *traceroute*, triggering messages with different initial TTL values are sent towards the same destination. As these messages are forwarded along the path to this destination, as soon as a message's TTL reduces to zero after reaching a router, the router needs to send back an ICMP Timeout message to notify the probing source. If the triggering messages go through the hijacker, this happens when the triggering messages' initial TTL values are greater than the hop distance from the probing source to the hijacker, the hijacker can do many things to interfere the path probing as a countermeasure to the locating algorithm.

In Figure 2(a), the hijacker AS's border router responds to traceroute honestly in blackholing (in which for example the border router responds with a *no route* ICMP message with its own IP address) and imposture (in which for example a router in H responds "destination reached" message with its own IP address). In either case, the router address belongs to H and maps to AS H, and the simple locating approach can identify H as the newly appearing origin AS hence the hijacker AS.

However, in the interception attack shown in Figure 2(b), the hijacker further propagates the traceroute probe packets to the target via $XYZT$, thus the origin AS is still H. Hence the simple locating approach fails in this case.

Furthermore, the hijacker can use various countermeasures. For instance, the hijacker may simply drop the triggering messages without replying to interrupt *traceroute* probing from proceeding further. Or it may send back ICMP Timeout messages with arbitrary source IP addresses to trick the probing source into thinking routers of those addresses are en route to the destination. The hijacker may even respond with ICMP Timeout messages before the triggering messages' TTL values reach zero. In Figure 2 (c), hijacker H manipulates the traceroute response such that after the IP-to-AS mapping, the AS path appears to $M1$ to be $ACDT$, and appears to $M2$ to be BDT, neither of which contains hijacker AS H in it, making the hijacker locating difficult. We refer to above manipulation of traceroute response as *countermeasure* for data-plane locating approach, and call such hijackers *countermeasure-capable* or *malicious*.

In summary, sophisticated hijackers that are capable of engaging countermeasures can inject false path information into measurements collected in both control plane and data plane, easily evading simple hijacker-locating mechanisms. We therefore design a more effective algorithm for locating these hijackers in the next section.

3.2 Locating Hijackers

The basic idea of LOCK is based on two key observations, which apply to both data-plane and control-plane approaches, different types of hijacks (blackholing, imposture, and interception), and with or without countermeasures by the attackers.

The first observation is that *the hijacker cannot manipulate the portion of the AS path from a polluted vantage point to the upstream (i.e., closer to the vantage point) neighbor AS of the hijacker AS*. For example, in

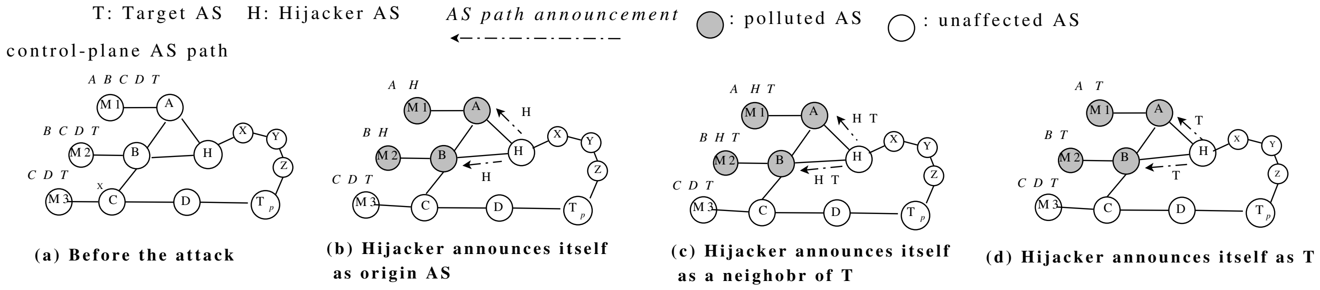


Figure 1: Control plane examples

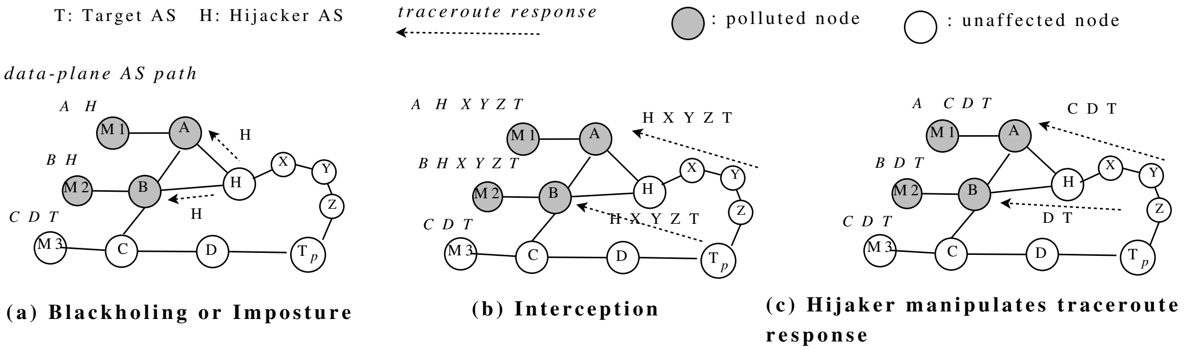


Figure 2: Data plane examples

Figures 1(c) and (d) and Figures 2 (b) and (c), for the polluted vantage points $M1$ and $M2$, the upstream ASes for hijacker AS H are A and B, and the portion of AS path $M1A$ and $M2B$ are trustworthy. This is easy to understand since the routers from the vantage points to hijacker upstream ASes are all well-behaving ones thus conform to BGP protocol in control-plane and ICMP protocol used in traceroute in data-plane.

The second observation is that *the trustworthy portion of polluted AS paths from multiple vantage points to a hijacked victim prefix "converge" "around" the hijacker AS.* This is also intuitive since, if the set of monitors are topologically diverse enough, the trustworthy portion of AS paths from all the polluted monitors to the target prefix must include the upstream AS neighbors of the hijacker AS (e.g. in Figure 1(d), and Figure 2 (c)) thus converge "around" the hijacker AS, or directly converge at hijacker AS (e.g. in Figure 1(b) and (c) and Figure 2(a) and (b)).

Since we do not know beforehand the hijack scenarios and whether there is any countermeasure, we focus on identifying these upstream neighbors of the hijacker AS, and then intuitively hijacker should be within the intersection of the 1-hop neighbor sets of the hijacker's neighbors. And chances are that the size of the intersection set is very small if the monitors have diversified locations. The neighbor sets of a given AS can be obtained from a daily snapshot of the state-of-arts AS level topology repository such as [16].

For example, in both Figures 1 and 2, ideally suppose we know that ASes A, B (which are on the polluted paths from vantage points $M1$ and $M2$, respectively) are the upstream neighbors of the hijacker. We can then infer that the hijacker AS should be within the intersection of $neighorset(A) = \{M1, B, H\}$ and $neighborse(B) = \{M2, A, H\}$, which is H. Of course in reality LOCK does not know beforehand which ASes are the upstream neighbors of the hijackers, thus each AS in a polluted path can potentially be such a neighbor of the hijacker AS. And hence the hijacker could be a neighbor of any of these nodes. We therefore put all the neighbors of *each* AS on a polluted path together with the path nodes themselves to form a *neighborhood set* of the polluted path. The hijacker should be included in this neighborhood set.

For reasons that we will explain in the Section 5, instead of using the neighborhood set of an arbitrary path, LOCK conservatively starts from the union of all the neighborhood sets of all polluted paths, $\mathcal{H}$. Then given that all polluted paths go through a neighbor AS of the hijacker, an AS which appears in more neighborhood sets is more likely to be the hijacker. We thus "rank" the ASes within $\mathcal{H}$ based on how many neighborhood sets an AS is in to narrow down to the handful of top ranked ASes.

Also when there are multiple convergence points, the earliest convergence point is more likely to be the hijacker than the later ones. More detailed ranking algorithm will be presented in Section 5.

As shown in this section, LOCK can utilize either control-plane or data-plane information. However, for the ease of presentation and due to space limitation, in the rest of paper we focus on data-plane approach unless otherwise specified.

4 Monitor Selection

LOCK operates in a distributed fashion from a number of monitors on the Internet. Both the number of monitors and locations of these monitors affect the accuracy in locating prefix hijackers. In general, the more monitors used by LOCK, the higher accuracy LOCK can achieve in locating prefix hijackers, and the more measurement overhead are incurred by LOCK. More importantly, the measurement overhead increase linearly as the number of monitors increases, while at the same time the improved accuracy gained by each additional monitor can gradually diminish. Therefore, it is hopeful to achieve very good accuracy with a limited number of carefully selected monitors.

In this section, we present a novel algorithm for selecting a number of monitors from a candidate set. In particular, we model the monitor selection problem as follows. Initially, we have M candidate monitors around the world. For each target prefix, we select a subset m monitors among the M candidates. In order to achieve the highest possible hijacker-locating accuracy with a limited number of monitors, the selection of monitors should be guided by two objectives: (i) maximize the likelihood of observing hijacking events on the target prefix; and (ii) maximize the diversity of paths from monitors to the target prefix so that a hijacking event can be observed from multiple distinct vantage points.

Our monitor selection algorithm consists of three steps:

1. **Clustering**: The M candidate monitors are grouped into m clusters. Monitors in the same cluster have more similar paths to the target prefix than those in different clusters.

2. **Ranking**: Candidate monitors in each cluster are ranked based on probability of their paths to the target prefix being polluted when the prefix is hijacked. The monitors with higher ranks are more likely to observe the hijacking event.

3. **Selecting**: The monitor which ranks the highest in each cluster is chosen to monitor the target prefix. Thus, a total of m monitors are selected for each target prefix.

4.1 Clustering

For a given target prefix, the candidate monitors are clustered based on similarity of their AS-level paths to the prefix. We measure the *similarity* between a pair of paths as the number of common ASes between these two paths over the length of the shorter path. If there is no common AS, the similarity score is 0. On the other hand, if the two paths are identical or one path is a sub-path of the other, the similarity score is 1. We also define the *similarity* between two clusters of paths as the maximum similarity between any two paths, one from each cluster.

We model the clustering part as a hierarchical clustering problem. Such problems have well-known algorithms, such as [17], that are polynomial-time complex. In this paper, we adopt the following simple clustering algorithm[2]. First, we start from M clusters, with one candidate site in each cluster, and compute similarity score for each pair of clusters. Second, we identify the pair of clusters with the largest similarity score among all pairs of clusters, and merge these two clusters into a single cluster. Third, we recompute the similarity score between this newly-formed cluster with each of the other clusters. We repeat steps two and three until only m clusters remain.

4.2 Ranking

We rank candidate monitors in each cluster based on their likelihood of observing hijacking events on the target prefix t (i.e., the path from monitor to target prefix is polluted by hijacking). For a given candidate site s, whether or not the route from s to t is polluted by hijacker h depends on the original best route (before the hijacking happens) from s to t and the fake route announced by h. This has been demonstrated by previous analysis in [6].

We assume that “prefer customer route” and “valley-free routing” are commonly adopted interdomain routing policies on today’s Internet. We denote the original best route from s to t as a “customer-route”, a “peer-route”, or a “provider-route” if the next-hop AS on the route from s to t is a customer, a peer, or a provider of the AS to which s belongs, respectively. According to the interdomain routing policies, a customer-route would be the most preferable and a provider-route would be the least preferable by each router; similarly, when policy preferences are equal, the route with shorter AS path is more preferable [10]. Therefore, when hijacker h announces a fake path, the monitor whose original best route is provider-route is more likely to be polluted than a original route of peer-route, which in turn is more likely to be polluted

Algorithm 1: Ranking monitors in each cluster

```
foreach monitor $i$ in the cluster
    if provider-route $R[i] = 300$; /* Assign the ranking. The larger the number is, the higher the rank is. */
    elseif peer-route $R[i] = 200$;
    else $R[i] = 100$;
    $R[i] += D(i, t)$; /* Add the AS-level distance */
```

than a original route of customer-route; when the policy preferences are equal, the monitor whose original best route has a longer AS path to t is more likely to be polluted than the one whose original best route has a shorter AS path (Please refer to Table 1 of [6] for detailed analysis). Our ranking algorithm is shown in Algorithm 1. Note that establishing AS topology itself is a challenging problem. We use most advanced techniques [30] to infer the AS relationship. Admittedly, inferred results could be incomplete. However, the evaluation part will show that the ranking algorithm based on such data can still achieve high location accuracy.

5 Hijacker-Locating Algorithm

LOCK locates hijacker AS based on AS paths from a set of monitors to the victim prefix. The AS path from a monitor to the victim prefix can be either obtained from the control plane (e.g., BGP AS path) or from the data plane (e.g., traceroute path). In the latter case, LOCK will need to pre-process the path and compute the corresponding AS path (described in Section 5.1).

5.1 Pre-Processing

When a prefix is hijacked, a portion of the Internet will experience the hijack. Traffic originated from this portion of the Internet and destined for the hijacked prefix will be altered to go through the hijacker. Monitors deployed in this affected portion of the Internet can observe that their monitor-to-prefix paths being altered. These monitor-to-prefix paths are the foundation of our hijacker-locating algorithm. Only paths changed by the hijack event should be supplied to the hijacker-locating algorithm. Methods such as the one outlined in [45] help separate real hijack induced path changes from changes caused by other non-hijack reasons.

If the monitor-to-prefix path is obtained from the data plane, then LOCK pre-processes the path in the following way. The most common tool for acquiring IP forwarding path in the data plane is the well known *traceroute* program. This program sends out a series of triggering packets with different initial TTL values to trigger the routers en route to the destination to return ICMP Timeout messages as soon as they observe a triggering message's TTL value reaching 0, hence revealing these routers' identities. These *traceroute* results are router-level paths and they need to be converted to AS-level paths. During this conversion, NULL entries in *traceroute* results are simply discarded. This simplification rarely has any effect on the resulted AS path because as *traceroute* proceeds within a particular AS, only if all routers in this AS failed to show up in *traceroute* results our results may be affected, which we have found this to be very rare. These resulting AS paths are known as the "reported paths" by the monitors in the rest of the section.

We use publicly available IP to AS mapping data provided by the iPlane services [15] to convert router IP addresses to their corresponding AS numbers. It is known that accurately mapping IP addresses to AS numbers is difficult due to problems such as Internet Exchange Points (IXPs) and sibling ASes [6, 25]. We argue that the impact of these mapping errors on the results of our hijacker-locating algorithm is minor. Firstly the distribution of the nodes, either routers or ASes, that may cause any mapping error in their corresponding Internet topologies, either router level or AS level, is sparse. If our paths do not contain these problematic nodes, our results are not affected by mapping errors. Secondly, it will become apparent, as more of the details of the hijacker-locating algorithm are described, that our algorithm is rather robust against such mapping errors. As long as these errors do not occur when mapping nodes near the hijacker, they will not affect the result of our algorithm. It is also worthwhile noting that the IP to AS mapping data do not need to be obtained from realtime control plane data. That is, the IP to AS mapping can be pre-computed and stored since it usually does not change over short period of time.

It is also helpful to perform sanity checks on the AS paths before we begin the hijacker-locating algorithm. The hijacker may forge *traceroute* results if a *traceroute* triggering message actually passes through the hijacker. Since the prefix has been hijacked, triggering messages with large enough initial TTL values, at least larger than the hop distance between the probing monitor and the hijacker, will inevitably pass through the hijacker. For a sophisticated hijacker, this is a good opportunity to fabricate responses to these triggering messages to conceal its own identity. As a result, the AS paths mapped from such a fake *traceroute* results may contain erroneous ASes as well. It is easy to see that these "noises" only appear in the later portion of a path because the portion that is before the hijacker cannot be altered by the hijacker, – the ICMP triggering messages do not reach the hijacker. Hence if a node in a path is determined to be a fake node, we really do not need to consider any nodes beyond that point because this point must be already beyond the hacker's position in the path.

In the pre-processing part, we consider the duplicated appearances of AS nodes. If a node appears more than once in a path, any appearance beyond the first is considered fake. This is because real *traceroute* results should not contain loops.

5.2 Basic Algorithm

We denote the set of monitors that have detected the hijacking and reported their altered monitor-to-prefix paths by $\mathcal{M}$. For each monitor m_i within $\mathcal{M}$, there is an AS level monitor-to-prefix path P_i, either computed by pre-processing *traceroute* path or obtained directly from BGP routes. We define the neighborhood set of a specific path P_i, denoted as $\mathcal{N}(P_i)$, as the union of all path nodes and their one-hop neighbors. The target prefix' AS should be removed from all $\mathcal{N}(P_i)$. The reason is simple, – it is not the hijacker AS. Note that LOCK computes the neighborhood set based on AS topology inferred from RouteView [3] before the hijacking is detected, rather than real-time BGP data when the hijacking is ongoing. Though the hijacker can try to pollute the AS topology information before launching real hijacking attack on the victim prefix, the impact of such evasion is minimal on the neighborhood set computation because it is difficult for hijacker to "remove" an observed true link from the AS topology by announcing fake routes.

We are interested in the neighborhood sets of the AS paths instead of just the AS paths themselves because the hijacker may actually not show up in any of the AS paths if it manipulates *traceroute* results. However, even under this condition the ASes which are immediately before the hijacker along the paths are real. Thus, the union of all neighborhood sets of all reported AS paths, $\mathcal{H} = \bigcup_i \mathcal{N}(P_i)$, form our search space for the hijacker. We denote each node in this search space as a_k. The hijacker-locating algorithm is essentially a ranking algorithm which assigns each node in $\mathcal{H}$ a rank based on their suspicious level of being the hijacker.

The LOCK algorithm ranks each AS node $a_k \in \mathcal{H}$ based on two values, *covered count* $\mathcal{C}(a_k)$ and *total distance to monitors* $\mathcal{D}(a_k)$. The *covered count* is simply computed by counting a_k appearing in how many path neighborhood sets. For each neighborhood set $\mathcal{N}(P_i)$ that a_k is a member, we compute the distance between a_k and the monitor of the path m_i, $d(m_i, a_k)$. This distance equals to the AS-level hop count from m_i to a_k along the path P_i if a_k is on the path P_i. Otherwise, $d(m_i, a_k)$ equals to the distance from m_i to a_k's neighbor, who is both on P_i and the closest to m_i, plus 1. If a_k is not a member of a path neighborhood set $\mathcal{N}(P_i)$, the distance $d(m_i, a_k)$ is set to 0. The *total distance to monitors* equals to the summation of all $d(m_i, a_k)$.

Algorithm 2: The pseudo-code of locating algorithm

```
Initializing
  set $\mathcal{H}$, $\mathcal{C}$, $\mathcal{D}$ empty;
Updating
  foreach $m_i$ in the monitor set $\mathcal{M}$
    foreach $a_k \in \mathcal{N}(P_i)$
      if $a_k \in \mathcal{H}$
        $\mathcal{D}(a_k)$ += $d(m_i, a_k)$;
        $\mathcal{C}(a_k)$ += 1;
      else
        insert $a_k$ in $\mathcal{H}$ ;
        $\mathcal{C}(a_k) = 0$;
        $\mathcal{D}(a_k)$ = $d(m_i, a_k)$;
Ranking
  sort $a_k \in \mathcal{H}$ by $\mathcal{C}(a_k)$;
  for $a_k$ with the same value of $C(a_k)$;
    sort $a_k$ by $\mathcal{D}(a_k)$;
```

After for each a_k in $\mathcal{H}$ both *covered count* $\mathcal{C}(a_k)$ and *total distance to monitors* $\mathcal{D}(a_k)$ are computed, we rank all nodes in $\mathcal{H}$ firstly based on their *covered count*. The greater the *covered count* a node a_k has, the higher it is ranked. Then for nodes having the same *covered count*, ties are broken by ranking them based on their *total distance to monitors*, –the lower the total distance, the higher the rank. If there are still ties, node ranks are determined randomly.

Hence, the final result of the locating algorithm is a list of nodes a_k, ordered based on how suspicious each node is being the hijacker. The most suspicious AS appears on the top of the list. The pseudo-code of the locating algorithm is shown in Algorithm 2.

The ranking algorithm described here may seem overly complicated for finding where the reported paths converge. However it is designed specifically to be robust against various measurement errors and possible hijacker countermeasures. One particular reason for this design is to reduce the effect of individual false paths. If a monitor-to-prefix path is changed due to reasons other than being hijacked and the monitor falsely assesses the situation as hijack, the path reported by this monitor may cause confusion on where the paths converge. Since it is difficult to distinguish this kind of paths beforehand, our algorithm has adopted the approach as described above to discredit the effect of these individual erroneous paths. For similar reasons, our ranking algorithm is robust against the IP-to-AS mapping errors if any.

Another reason for outputting an ordered list is that there are cases that hijacked paths converge before these paths reach the hijacker (*early converge*). This is more likely to happen when the hijacker is located far away from the Internet core where the connectivity is rich. In this case the hijacked paths may converge at an upstream provider of the hijacker in stead of the hijacker itself. Although as we will show later these hijacking scenarios typically have small impacts, in other words the portion of the Internet that is affected by such hijacks is small; still we wish to locate the hijacker. A list of suspects ranked by level of suspicion is well suited for addressing this issue.

5.3 Improvements

After the suspect list is computed, we can apply additional post-processing methods to further improve our results. The basic algorithm is very conservative in the way that $\mathcal{H}$ includes all possible candidates. Now we look into ways that $\mathcal{H}$ may be reduced. The hope is that with a trimmed suspect set to begin with, the locating algorithm can get more focused on the hijacker by increasing the rate that the most suspicious node on the list is the hijacker. Both improvements are designed to alleviate the early converge problem we mentioned before. Note that the improvements may exclude the real hijacker from the suspect set, but the evaluation (in Section 6.3.5) shows that chance is very small.

5.3.1 Improvement One: AS Relationship

In the basic algorithm, we have only taken AS topology into account. In other words, all topological neighbors of nodes on a reported AS path are added to the path's neighborhood set. In reality, not all physical connections between ASes are actively used for carrying traffic. In particular, some connections may be used only for traffic of one direction but not the other. This is largely due to profit-driven routing policies between different ISPs. Internet paths have been found to follow the "valley-free" property [10], i.e. after traversing a provider-to-customer edge or a peer edge, a path will not traverse another customer-to-provider path or another peer edge. If we constrain our suspect set using this AS relationship based property by removing the neighbors that do not follow the "Valley-free" property from the neighborhood set of each reported path, we are able to reduce the size of the neighborhood set and further on the suspect set $\mathcal{H}$.

One matter needs to be pointed out is that not all links on the reported paths are necessarily real due to the hijacker's countermeasures. Since we do not know what links are fabricated we should not trim the neighborhood sets too aggressively. We only perform this improvement on path links that we are reasonably certain that they are real. In particular, as we know that an attacker cannot forge path links that are before itself, thus we can reasonably consider that on each reported path the links that are before the node immediately before the most suspicious node are real, and the trimming is only done on neighbors of these links.

This AS relationship based improvement is incorporated into the basic algorithm in an iterative fashion. We first pre-compute AS relationship information using method proposed in [10]. Note that this is done offline and does not require any real time access to the control plane information because AS relationship rarely change over time. After each execution of the basic algorithm produces a ranked suspect list, we can assume that on each path from the path's reporting monitor to the node immediately before the most suspicious node, all AS paths are valid. Based on these valid links, we can further infer the valid link in each neighborhood set. When there is any change of neighborhood set, we run the locating algorithm again to update the suspicious list. The iteration will stop if there is no change of suspicious list.

5.3.2 Improvement Two: Excluding Innocent ASes

The second improvement focuses on removing nodes from the suspect set $\mathcal{H}$ of whose innocence we are reasonably certain. One group of these nodes are the ones that are on the reported paths that actually pass through the most suspicious node and before the most suspicious node. The reason for this exclusion is again that the attacker cannot forge the identity of these nodes.

The second group of the innocent nodes are selected based on the path disagreement test described in [45]. In path disagreement test, a reference point that is outside of the target prefix but topologically very close to the prefix is selected and the path from a monitor to this reference point and the path from same monitor to the target prefix are compared. If they differ significantly it is highly likely that the prefix has been hijacked. The high accuracy of this test leads us to believe that nodes on monitor-to-reference point paths are not likely to be the hijacker. They can be excluded from the suspect set.

The second improvement is again incorporated into the basic algorithm in an iterative fashion. After each execution of the basic algorithm, the suspect set is reduced by removing nodes of the two aforementioned innocent groups. Then basic algorithm is executed again using the reduced suspect set. The iteration is repeated until the basic suspect set is stable.

6 Evaluation

We implemented and deployed LOCK on PlanetLab [31]. This is a necessary step to show that LOCK is deployable in real world system. Also using the PlanetLab testbed, we evaluated the performance of LOCK based on measurements of the deployed LOCK system. In this section, we first present our measurement setup and evaluation methodology. Then we evaluate the performance of the monitor selection algorithm in LOCK, and the effectiveness of LOCK against against synthetic hijacks, reconstructed previously-known hijacking events, and real hijacking attacks launched by us.

6.1 Measurement Setup

6.1.1 Candidate Monitors

In our experiments, we first chose a number of geographically diversified PlanetLab [31] nodes as candidate network location monitors. We manually selected 73 PlanetLab nodes in 36 distinct ASes in different geographical regions. More specifically, relying on their DNS names, half of the nodes are in the U.S., covering both coasts and the middle. The other half were selected from other countries across multiple continents. Among these candidate monitors, a set of monitors were selected using the algorithm presented in Section 4 to monitor each target prefix.

6.1.2 Target Prefixes

We selected target prefixes from four different sources: (i) Multiple Origin ASes (MOAS) prefixes, (ii) Single Origin AS (SOAS) prefixes with large traffic volume, (iii) prefixes of popular Web sites, and (vi) prefixes of popular online social networks. To get prefixes from sources (i) and (ii), we first use BGP tables obtained from RouteViews [3] and RIPE [2] to identify the initial candidates of MOAS and SOAS prefixes. Then for each candidate prefix, we tried to identify a small number (up to 4) of live (*i.e.* responsive to *ping*) IP addresses. To avoid scanning the entire candidate prefixes for live IP addresses, we mainly used the prefixes' local DNS server IP addresses to represent the prefix. If we failed to verify any live IP address for a particular prefix, we discarded this prefix from our experiments. Using this method, we selected 253 MOAS prefixes. We also ranked all SOAS prefixes based on "popularity" (*i.e.* traffic volume observed at a Tier-1 ISP based on Netflow) of the prefix and selected top 200 prefixes with live local DNS server IP addresses.

We also selected prefixes that correspond to popular applications on the Internet: Web and online social networks. In particular, we selected the top 100 popular Web sites based on the Alex [5] ranking and obtain their IP addresses and corresponding prefixes. We also obtained IP addresses and prefixes of YouTube and 50 popular online social networks. Each of the selected online social networks has at least 1 million registered users in multiple countries. Combining prefixes from all above four sources, we have a total of 451 target prefixes.

6.1.3 Measurement Data Gathering

In our experiments, each monitor measures its paths to all selected IP addresses in all target prefixes via *traceroute*. We also measured paths from each monitor to reference points of target prefixes [45]. In addition, each monitor also measures its paths to other monitors. We obtain AS-level paths of above measured paths by mapping IP addresses to their ASes based on the IP-to-AS mapping published at iPlane [15].

The results presented here are based on monitoring data collected from March $20th$, 2008 to April $20th$, 2008. In particular, we measured each path (from a monitor to a target prefix) every 5 minutes.

In addition, we obtained the AS topology data during the same time period from [16]. We also used the AS relationship information captured for customer-to-provider and peer links over 6 month (from June 2007 to December 2007) using the inferring technique described in [24].

6.2 Evaluation Methodology

We evaluated LOCK based on three sets of prefix hijacking experiments: (i) synthetic prefix hijacking events based on Internet measurement data; (ii) reconstructed previously-known prefix hijacking events based on Internet measurement data; and (iii) prefix hijacking events launched by us on the Internet.

6.2.1 Simulating Synthetic Prefix Hijacking Events

We consider commonly used interdomain routing policies: "prefer customer routes" and "valley-free routing". In particular, an AS prefers routes announced from its customer ASes over those announced from its peer ASes, further over those announced from its provider ASes. These policies are driven by financial profit of ASes. If two routes have the same profit-based preference, then the shorter route (i.e., fewer AS hop count) is preferred. When the hijacker announces a fake prefix, we assume that it does this to all its neighbors (i.e. providers, peers, and customers) to maximize hijacking impact.

For each attack scenario, we simulated all three types of hijacking scenarios, namely imposture, interception, malicious, as shown in Figure 2 in Section 3. Each attack scenario is simulated as follows. In each attack scenario, we selected one PlanetLab node as the hijacker h and another PlanetLab node as the target prefix t. The hijacking is then observed from the monitors.

In the imposture scenario, the path from s to t will become the path from s to h if s is polluted by h's attack. Otherwise, the path from s to t remains the same as before the attack. This was repeated for all possible selections of h, t, and s, except for cases where t's AS is on the AS path from s to h because the hijack will never succeed in these cases. In addition, since some paths were not traceroute-able, we had to discard combinations that require these paths.

The setup for simulating interceptions and malicious scenarios is similar to that of the imposture scenario. In the interception scenario, the path from s to t will be the concatenation of paths from s to h and from h to t if s is polluted by h's attack. However, we exclude the cases that there is one or more common ASes between these two paths. This is because the hijacker h cannot successfully redirect the traffic back to the target prefix t, i.e., the interception hijack fails.

In the malicious scenario, the hijacker h has countermeasure against LOCK. The path from s to t will be the path from s to h (the AS of h will not show up) with a few random AS hops appended after h. The generation of these random AS hops is kind of tricky. If h generates different noisy tails for different monitors, these tails may not converge at all. In this case, it is easier for our locating algorithm to locate the hijacker. In our simulations, in anticipating that the hijacker may fill its replies to traceroute probes with fake identities, we replaced the node identities with random entries for all nodes that are farther than the hijacker (inclusive) in the paths resulted from running traceroute from different monitors.

6.2.2 Reconstructing Previously-Known Prefix Hijacking Events

We obtained the list of previously-known prefix hijacking events from the Internet Alert Registry [14]. IAR provides the network operator community with the up to date BGP (Border Gateway Protocol) routing security information. Its discussion forum [3] posts suspicious hijacking events. We chose 7 that had been verified and confirmed to be prefix hijacking events, including some famous victims such as YouTube and eBay, during a time period from 2006 to 2008.

We reconstructed these 7 hijacking events using the following method. First, we selected a traceroutable IP in each victim AS as the probing target t, and a traceroutable IP in each hijacker AS as the hijacker h. Then we collected the traceroute information from each monitoring site s to these targets t and hijackers h. The routing policy is based again on the profit driven model. Since we don't know what kind of behavior each hijacker took (imposture, interception or malicious), We conservatively assume that the hijacker will try to evade our measurement. So it follows the malicious scenario we mentioned before.

6.2.3 Launching Controlled Prefix Hijacking Events

We conducted controlled prefix hijacking experiments on the Internet using hosts under our control at four different sites, namely Cornell, Berkeley, Seattle, and Pittsburgh. Each host ran the Quagga software router and established eBGP sessions with different ISPs. Effectively, this allowed us to advertise our dedicated prefix (204.9.168.0/22) into the Internet through the BGP sessions. The idea behind the experiments was to use our prefix as the target prefix with one of the sites serving as the owner of the prefix and the other three sites (separately) serving as the geographically distributed attackers trying to hijack the prefix. More implementation details can be found in [6]. In our experiment, we focused on the imposture scenario. There were 12 hijacking cases by switching the role of each site. These attacks were launched according to a pre-configured schedule during period from May 2, 2008 to May 4, 2008.

6.2.4 Performance Metrics

LOCK identifies suspicious hijackers and ranks them based on their likelihood of being the true hijacker. The hijacker ranked at top one is most suspicious. We thus define the *top-n accuracy* of LOCK as the percentage of hijacking events that the true hijacker ranks as top n on the suspect list, where n is a parameter. We use this parameterized definition because different operators might have different preference. Some might prefer knowing just the most suspicious hijacker, in which top-1 accuracy is most important. Others might not mind learning a longer suspect list to increase the likelihood that the hijacker is included in the suspect list. We will later show that the top-2 accuracy is already very high.

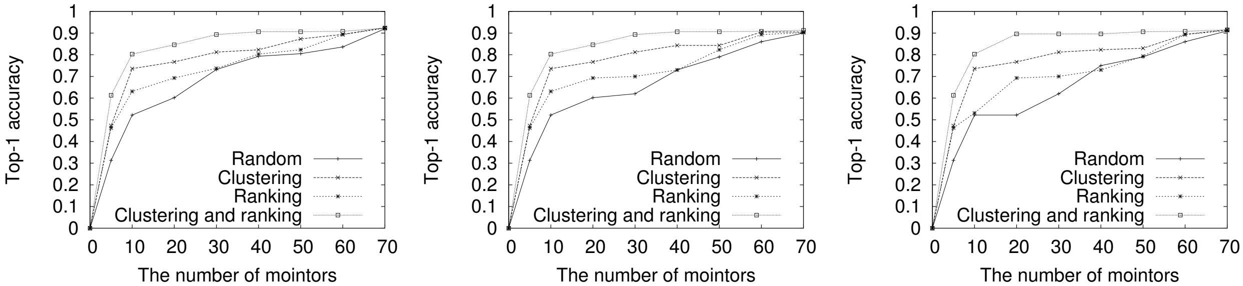
In addition, we define *impact* of a hijacker h as the fraction of the ASes from which the traffic to the target prefix t is hijacked to h, similar to what is done in [23]. We will then study the correlation between LOCK's locating accuracy of a given hijacker and the impact of its attack.

6.3 Evaluation on Synthetic Prefix Hijacking Events

In this section, we use the results of LOCK based on the data plane measurement to illustrate our findings.

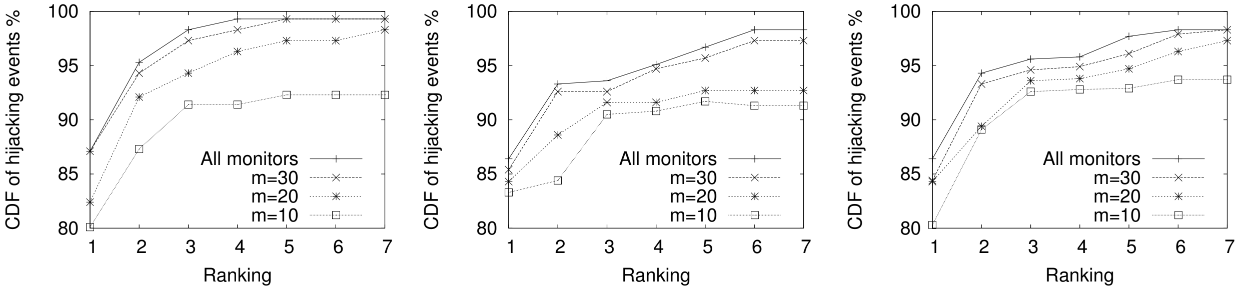
6.3.1 Monitor Selection

We compare the performance of the monitor selection algorithm (referred as *clustering and ranking*) proposed in Section 4 with the following three monitor selection algorithms: (i) *random:* randomly selecting m monitors



(a) Imposture (b) Interception (c) Malicious

Figure 3: Performance of monitor selection algorithms



(a) Imposture (b) Interception (c) Malicious

Figure 4: The CDF of the rank of hijackers in synthetic attacks

from all M candidates. (ii) *clustering:* dividing M monitors into clusters based on the clustering algorithm proposed in Section 4.1, then randomly selecting one monitor from each cluster; and (iii) *ranking:* ranking M monitors based on the ranking algorithm proposed in Section 4.2, then selecting the first m candidates.

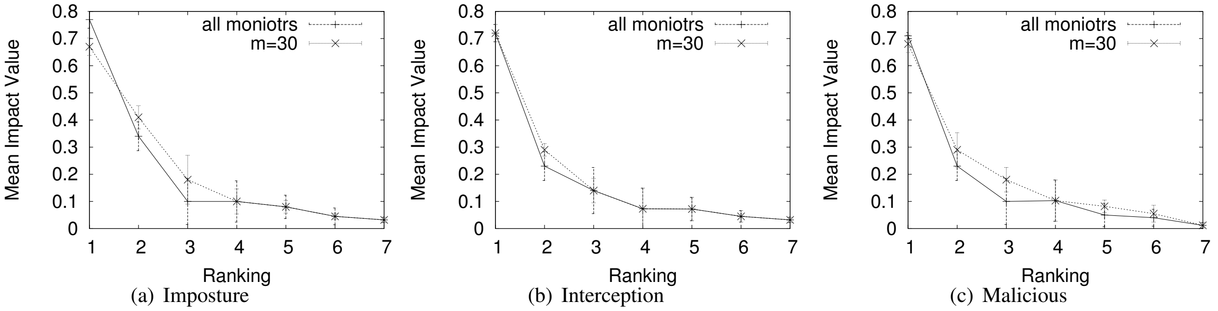
Figure 3 shows the top-1 accuracy of different monitor selection algorithms when varying the subsets of monitors. We focused on synthetic attacks since the dataset is much larger than previously-known hijacks and controlled real hijacks. We find that: (i) There is always a trade-off between the number of monitors selected and hijacker-locating accuracy. Note that even using all 73 monitors, the accuracy is less than 92%. It is not surprising because it is hard to detect the hijacking events which have small impact [23]. (ii) The *clustering and ranking* algorithm outperforms the rest. For example, for imposture attacks, selecting 10 monitors based on the ranking and clustering algorithm is enough for achieving 80% top-1 accuracy. This is only 1/3 of number of monitors needed to reach the same top-1 accuracy with either *ranking* or the *clustering* algorithm, or 1/6 if monitors are selected randomly. Hence in our experiments in the rest of the section, whenever we need to select monitors, we use the *clustering and ranking* algorithm, unless otherwise specified.

Moreover, we want to make sure that the monitor selection algorithm does not overload any monitors by assigning too many target prefixes to it for monitoring. For each target prefix we select $m = 30$ monitors from the total pool of $M = 73$ candidate monitors using the monitor selection algorithm described in Section 4. Individual monitor's work load is computed as the number of target prefixes assigned to it divided by the total number of target prefixes. Ideally, the average work load, which is the load each monitor gets if the monitoring tasks are evenly across all monitors equally instead of assigning prefixes to monitors that can monitor most effectively, is $m/M \approx 0.4$. As as comparison, we observe the real workload ranges from 0.3 to 0.55. In addition, only 4 monitors out of 73 have load above 0.5, which means that they monitor more than half of prefix targets.

6.3.2 Effectiveness of Basic Algorithm

The evaluations of two different aspects of the effectiveness of the hijacker-locating algorithm are presented in this section. We show how well the ranked list captures the hijacker identity, as well as how well the ranked list reflects the impact of the hijack events.

Figure 4 illustrates where the hijacker is ranked in the suspect list produced by the basic algorithm, for different number of monitors selected. Obviously, the higher the hijacker is ranked, the better the basic algorithm is. From this figure, we can see that:



(a) Imposture (b) Interception (c) Malicious

Figure 5: Correlating the impact with the ranking value

- More than 80% of the time, our basic algorithm pinpoints the hijacker by ranking it as *top 1* on the suspect list, regardless what kind of attack and with how many monitors, as long as more than the minimum number of 10 monitors.

- Because of the early convergence problem described in Section 5.2, the hijacker may not be ranked the first. Therefore as we look into not only the highest ranked node but even more nodes, the chance that the hijacker is included in this selective set increases. For example with 10 monitors, the chance that an imposture hijacker is found among the top three nodes is more than 94%, a 14% increase from only looking at the highest ranked suspect.

- The hijacker-locating algorithm performs best in imposture scenarios. The reason is that imposture paths are more likely to be straight without detouring.

- Obviously the more monitors we employ, the better the algorithm works. What is interesting is that seemingly by having $m = 30$ we have reached the point of diminishing return: having more than 30 monitors no longer improves the performance much.

Next, we study the relationship between the impact of a hijack event and where the hijacker is ranked in the suspect list. This shows another aspect of the quality of our hijacker-locating algorithm. That is, not only we want to locate hijackers, we especially want to locate the hijackers causing great damages. Figure 5 shows the ranking (x-axis) vs the median impact of all hijackers with the same ranking (Y-axis). All three plots in Figure 5 show that there is a positive relationship between the hijacker's rank and the impact of its hijack attack. In other words, the larger the impact caused by a hijacker, the more likely our locating algorithm will rank the hijacker high in the suspect list. This is mostly due to the fact that the early converge problems occur mostly at where hijacks have small impacts, near Internet edge.

6.3.3 Effectiveness of Improvements

Finally, we evaluate the quality of two improvements (I1 and I2) proposed in Section 5.3. In particular, we are not only interested in the increase in top-1 accuracy these improvements may bring, but also the false negative rate (FNR), which is the ratio that the improvements mistakenly exclude a hijacker from the suspect list.

Table 1 shows both sets of numbers for different kinds of attacks and different number of monitors. Different combinations of the basic algorithm and the improvements are shown in different rows of the table.

- I2 helps more. The reason is that for I1 we can only trust the path before converges. But for I2, we have more information provided by the reference point traceroute.

- When combining I1 and I2, the accuracy can be further improved. This is because the nodes that I1 and I2 remove from the suspect list are typically not the same.

- In general, LOCK (i.e., B+I1+I2) is able to pinpoint the prefix hijacker AS with an accuracy of over 91%, up to 94.3%.

- The false negative ratio introduced by improvements is relatively low. For example, when using all monitors we can improve the accuracy by more than 5% by applying both I1 and I2, while the false negative ratio resulted from applying the improvements is only 0.09%

6.3.4 Effectiveness on different AS-levels

We study the locating accuracy when the hijacker located in different level in the AS hierarchy. We classify AS nodes into three tiers: Tier-1 nodes, transit nodes, and

Table 1: The effectiveness of improvement

Algorithms	All monitors						m=30					
	Imposture		Interception		Malicious		Imposture		Interception		Malicious	
	Accuracy	FNR	Accuracy	FNR	Accuracy	FNR	Accuracy	FNR	Accuracy	FNR	Accuracy	FNR
B	88.7%	0.00%	86.3%	0.00%	85.4%	0.00%	86.2%	0.00%	84.7%	0.00%	83.5%	0.00%
B+I1	89.8%	0.03%	90.3%	0.17%	88.6%	0.14%	86.4%	0.05%	85.3%	0.14%	84.6%	0.11%
B+I2	91.3%	0.09%	93.1%	0.16%	90.4%	0.10%	90.7%	0.14%	90.6%	0.18%	88.3%	0.20%
B+I1+I2	94.2%	0.09%	94.3%	0.24%	93.1%	0.18%	92.4%	0.20%	91.4%	0.17%	91.8%	0.26%

Table 2: The effectiveness on different AS-levels

Category	Imposture		Interception		Malicious	
	Accuracy	FNR	Accuracy	FNR	Accuracy	FNR
All	92.4%	0.20%	91.4%	0.17%	91.8%	0.26%
Transit	97.6%	0.04%	96.3%	0.07%	94.8%	0.14%
Stub	90.2%	0.18%	90.1%	0.21%	90.4%	0.35%

Table 3: The effectiveness on prevention after locating

Methods	Initial	Stop the origin	Stop in Tier1
LOCK	23.43%	0.10%	2.31%
Simple Locating	23.43%	13.13%	21.90%

stub nodes like in [23]. [4] Our hijackers in planetlab belongs to transit nodes, or stub nodes. When using two improvements and 30 monitors, we compare the accuracy and false negative ration for these two classes, in Table 2. The hijackers on the higher level could be located more easily. The hijackers on the edge is relatively hard to locate. We can still achieve more than 90% accuracy.

6.3.5 Effectiveness of filtering after locating the hijacker

After locating the AS, the next step is to filter the fake AS announcement from it. We compare the average percentage of impacted (polluted) AS, before and after the locating and filtering either stop on the origin or on the Tier1 AS. As a comparison, we also select the last hop of AS of the observed paths as a hijacker (simple locating approach) then do the same filtering. They are under malicious case. Table 3 shows that Lock is more helpful than simple locating method to prevent hijacks.

6.3.6 Remarks

We have shown that LOCK performs well using monitor-to-prefix paths measured in the data plane. Similar observation would hold if control plane paths are used in LOCK. In the non-malicious cases, the monitor-to-prefix paths that observed in the control plane are the same as those observed in the data plane. However, in the malicious case, we have shown that the hijacker can employ more sophisticated evasion technique in the data plane than in the control plane. Therefore, our results shown in this section provide a lower bound of LOCK performance against malicious hijackers.

Table 4: Previously-Known prefix hijacking events

Victim AS	Hijacker AS	Date	#monitors
3691	6461	March 15, 2008	16
36561 (YouTube)	17557	February 24, 2008	9
11643 (eBay)	10139	November 30, 2007	7
4678	17606	January 15, 2007	8
7018	31604	January 13, 2007	13
1299	9930	September 7, 2006	5
701, 1239	23520	June 7, 2006	12

6.4 Evaluation on Previous-Known Attacks

We reconstructed 7 previously known prefix hijacking events. Table 4 shows the dates and ASes of the hijacker and the target prefix (i.e., the victim) of these events. By using all 73 monitors deployed on PlanetLab, LOCK is able to accurately locate the hijacker ASes as the top-1 suspects for all these hijacking events, i.e., the true hijackers are ranked first on the suspect lists. Using the monitor selection algorithm (clustering and ranking) presented in Section 4, we also identified the minimum set of monitors that were required by LOCK to accurately locate the hijacker in each of these previously-known events. The last column of Table 4 shows that all hijackers could be correctly located as top-1 suspects by using 16 or fewer monitors. A detailed investigation shows that these hijacks polluted majority of the monitors, resulting in LOCK's high locating accuracy.

6.5 Evaluation on Controlled Real Attacks

In this set of experiments, we launched real imposture attacks using four sites under our control. The schedule is shown in Table 5. During the experiments each LOCK

Table 5: Locating hijackers in real Internet attacks

Victim Site	Hijacker Site	Launch Time (EST)	Response Time (minutes)	Required monitors
Cornell	Berkeley	May 2 12:01:31	13	12
	Seattle	May 2 16:12:47	7	10
	Pittsburgh	May 2 17:34:39	9	9
Pittsburgh	Cornell	May 2 19:32:09	13	14
	Berkeley	May 2 22:50:25	11	15
	Seattle	May 3 02:26:26	12	15
Seattle	Cornell	May 3 11:20:42	9	8
	Pittsburgh	May 3 13:03:10	12	12
	Berkeley	May 3 19:16:16	8	18
Berkeley	Seattle	May 3 22:35:07	13	14
	Pittsburgh	May 4 00:01:01	12	16
	Cornell	May 4 11:19:20	11	10

monitor probed the target prefix 204.9.168.0/22 once every 5 minutes. For the purpose of this experiment, we used the detection scheme proposed in [45], which was able to detect all the attacks launched from the controlled sites. The hijackers in these experiments were "honest", i.e., no countermeasure was done by the hijackers. Thus we observed that LOCK locates the hijackers as top-1 suspects in all the real imposture attacks.

In this real Internet experiment, we were able to evaluate the response time of LOCK in addition to its accuracy. The response time is defined as the latency from the time the the attack is launched by the hijacker to the time that LOCK locates the hijacker. The response time highly depends on two major factors: the speed of propagation of invalid route advertisement and the probing rate employed by LOCK monitors. It usually takes up to a few minutes for a route advertisement to spread across the Internet. This is the latency that an attack takes before making full impact on the Internet. After a LOCK monitor is impacted by an attack, it may also take a few minutes for the monitor to detect and locate the hijacker because the monitor probes target prefixes periodically. There are also few minor factors that may affect the response time. For example, there can be a few seconds latency for LOCK monitors to get replies for each probe. However, they are neglected in our evaluation because they are orders of magnitude smaller than the above two major factors.

We record the timestamp each attack is launched from a control site and the timestamp LOCK locates the hijacker (i.e., that controlled site). Both of which are synchronized with a common reference time server. The response time is computed by taking the difference between the above two timestamps. If alternative detection scheme is used, the observed response time serves as a conservative upper bound of the latency that LOCK takes to locate the hijacker.

Table 5 shows the response time and minimum number of required monitors for locating these real prefix hijacking events. We observe that LOCK is able to locate the hijacker within 7 $\sim$ 13 minutes. Given that the probe frequency of LOCK monitors is 5 minutes, the results implies that it takes LOCK at most 2 $\sim$ 3 rounds of probes to detect and locate the hijacker. Moreover, all hijackers are correctly located as top-1 suspects by using 18 or fewer monitors.

7 Related Work

A number of solutions have been proposed to proactively defend against prefix hijacking. They can be categorized into two broad categories: crypto based and non-crypto based. Crypto based solutions, such as [4,8,13,19,27,35,36], require BGP routers to sign and verify the origin AS and/or the AS path to detect and reject false routing messages. However, such solutions often require signature generation and verification which have significant impact on router performance. Non-crypto based proposals such as [11,18,32,37,44] require changing router softwares so that inter-AS queries are supported [11,32], stable paths are more preferred [18,37], or additional attributes are added into BGP updates to facilitate detection [44]. All the above proposals are not easily deployable because they all require changes in router software, router configuration, or network operations, and some also require public key infrastructures.

Recently, there has been increasing interest in solutions for reactive detection of prefix hijacking [6, 12, 21, 22, 26, 34, 36, 45] because such solutions use passive monitoring and thus are highly deployable. For example, [43,45] monitor the data plane, [21,22,26,34] monitor the control plane, and [6, 12, 36] monitor both control and data planes. LOCK is different from all these approaches because LOCK locates the hijacker AS for each prefix hijacking event, while the above approaches only focus on detecting a hijacking event without further revealing the location of the hijacker. In fact, LOCK can be used together with any of the above hijacking detection algorithm for identifying hijacker AS because the flexibility of LOCK on using either control plane or data plane information in locating hijacker.

Measurement-based solutions often require careful selection of monitors. In particular, LOCK selects monitors based on their likelihood of observing hijacking events, while [45] proposed an initial monitor selection algorithm to detect hijacks without further evaluation, and [23] tries to understand the impact of hijackers in different locations. In addition, there have been a number of studies [7,9,40] on the limitations of existing BGP monitoring systems (e.g. RouteView) and the impacts of monitor placement algorithms [29] for collecting BGP data for a boarder range of applications such as topology discovery, dynamic routing behavior discovery and network black hole discovery [20,41].

Finally, existing works [38, 39, 42] proposed to mitigating prefix hijacking by using an alternative routing path [38, 39], or by modifying AS_SET [42]. Though LOCK does not directly handle the mitigation of prefix hijacking events, LOCK can provide the hijacker location information required by these mitigation schemes.

8 Conclusion

In this paper, we propose a robust scheme named LOCK for locating the prefix hijacker ASes based on distributed AS path measurements. LOCK has several advantages: 1) LOCK is an unified scheme that locates hijackers in the same fashion across different types of prefix hijacking attacks; 2) LOCK is a distributed scheme with workload distributed among multiple monitors; 3) LOCK is a robust scheme because multiple monitors help improving locating accuracy and discounting individual errors; and 4) LOCK is a flexible scheme because it can use AS path measurement data obtained either from data-plane or from control-plane to locate the hijacker AS.

The performance of the LOCK scheme has been evaluated extensively through experiments in three kinds of settings: test topology constructed based on real Internet measurements, reconstructed known prefix hijack attacks, and controlled prefix hijack attacks conducted on the Internet. We have shown that the LOCK scheme is very accurate, highly effective, and rapid reacting.

Acknowledgement

Tongqing Qiu and Jun Xu are supported in part by NSF grants CNS-0519745, CNS-0626979, CNS-0716423, and CAREER Award ANI-023831.

References

[1] http://www.ripe.net/news/study-youtube-hijacking.html.

[2] RIPE RIS Raw Data. http://www.ripe.net/projects/ris/rawdata.html.

[3] University of Oregon Route Views Archive Project. http://www.routeview.org.

[4] AIELLO, W., IOANNIDIS, J., AND MCDANIEL, P. Origin Authentication in Interdomain Routing. In *Proc. of ACM CCS* (Oct. 2003).

[5] Alexa. http://www.alexa.com/.

[6] BALLANI, H., FRANCIS, P., AND ZHANG, X. A Study of Prefix Hijacking and Interception in the Internet. In *Proc. ACM SIGCOMM* (Aug. 2007).

[7] BARFORD, P., BESTAVROS, A., BYERS, J., AND CROVELLA, M. On the marginal utility of network topology measurements. In *IMW '01* (New York, NY, USA, 2001), ACM, pp. 5–17.

[8] BUTLER, K., MCDANIEL, P., AND AIELLO, W. Optimizing BGP Security by Exploiting Path Stability. In *Proc. ACM CCS* (Nov. 2006).

[9] COHEN, R., AND RAZ, D. The Internet Dark Matter - on the Missing Links in the AS Connectivity Map. In *INFOCOM* (2006).

[10] GAO, L. On Inferring Autonomous System Relationships in the Internet. *IEEE/ACM Transactions on Networking* (2001).

[11] GOODELL, G., AIELLO, W., GRIFFIN, T., IOANNIDIS, J., MCDANIEL, P., AND RUBIN, A. Working Around BGP: An Incremental Approach to Improving Security and Accuracy of Interdomain Routing. In *Proc. NDSS* (Feb. 2003).

[12] HU, X., AND MAO, Z. M. Accurate Real-time Identification of IP Prefix Hijacking. In *Proc. IEEE Security and Privacy* (May 2007).

[13] HU, Y.-C., PERRIG, A., AND SIRBU, M. SPV: Secure Path Vector Routing for Securing BGP. In *Proc. ACM SIGCOMM* (Aug. 2004).

[14] IAR. http://iar.cs.unm.edu/.

[15] iPlane. http://iplane.cs.washington.edu/.

[16] Internet topology collection. http://irl.cs.ucla.edu/topology/.

[17] JOHNSON, S. Hierarchical Clustering Schemes. In *Psychometrika* (1967).

[18] KARLIN, J., FORREST, S., AND REXFORD, J. Pretty Good BGP: Protecting BGP by Cautiously Selecting Routes. In *Proc. IEEE ICNP* (Nov. 2006).

[19] KENT, S., LYNN, C., AND SEO, K. Secure Border Gateway Protocol (S-BGP). *IEEE JSAC Special Issue on Network Security* (Apr. 2000).

[20] KOMPELLA, R. R., YATES, J., GREENBERG, A., AND SNOEREN, A. C. Detection and Localization of Network Black Holes. In *Proc. IEEE INFOCOM* (2007).

[21] KRUEGEL, C., MUTZ, D., ROBERTSON, W., AND VALEUR, F. Topology-based Detection of Anomalous BGP Messages. In *Proc. RAID* (Sept. 2003).

[22] LAD, M., MASSEY, D., PEI, D., WU, Y., ZHANG, B., AND ZHANG, L. PHAS: A Prefix Hijack Alert System. In *Proc. USENIX Security Symposium* (Aug. 2006).

[23] LAD, M., OLIVEIRA, R., ZHANG, B., AND ZHANG, L. Understanding Resiliency of Internet Topology Against Prefix Hijack Attacks. In *Proc. IEEE/IFIP DSN* (June 2007).

[24] MAO, Z. M., QIU, L., WANG, J., AND ZHANG, Y. On AS-Level Path Inference. In *Proc. ACM SIGMETRICS* (2005).

[25] MAO, Z. M., REXFORD, J., WANG, J., AND KATZ, R. Towards an Accurate AS-level Traceroute Tool. In *Proc. ACM SIGCOMM* (2003).

[26] RIPE myASn System. http://www.ris.ripe.net/myasn.html.

[27] NG, J. Extensions to BGP to Support Secure Origin BGP. ftp://ftp-eng.cisco.com/sobgp/drafts/draft-ng-sobgp-bgp-extensions-02.txt, April 2004.

[28] NORDSTROM, O., AND DOVROLIS, C. Beware of BGP Attacks. *ACM SIGCOMM Computer Communications Review (CCR)* (Apr. 2004).

[29] OLIVEIRA, R., LAD, M., ZHANG, B., PEI, D., MASSEY, D., AND ZHANG, L. Placing BGP Monitors in the Internet. UW Technical Report, 2006.

[30] OLIVEIRA, R., PEI, D., WILLINGER, W., ZHANG, B., AND ZHANG, L. In Search of the elusive Ground Truth: The Internet's AS-level Connectivity Structure. In *Proc. ACM SIGMETRICS* (2008).

[31] PlanetLab. `http://www.planet-lab.org`.

[32] QIU, S. Y., MONROSE, F., TERZIS, A., AND MCDANIEL, P. D. Efficient Techniques for Detecting False Origin Advertisements in Inter-domain Routing. In *Proc. IEEE NPsec* (Nov. 2006).

[33] RAMACHANDRAN, A., AND FEAMSTER, N. Understanding the Network-Level Behavior of Spammers. In *Proceedings of ACM SIGCOMM* (2006).

[34] SIGANOS, G., AND FALOUTSOS, M. Neighborhood Watch for Internet Routing: Can We Improve the Robustness of Internet Routing Today? In *Proc. IEEE INFOCOM* (May 2007).

[35] SMITH, B. R., AND GARCIA-LUNA-ACEVES, J. J. Securing the Border Gateway Routing Protocol. In *Proc. Global Internet* (Nov. 1996).

[36] SUBRAMANIAN, L., ROTH, V., STOICA, I., SHENKER, S., AND KATZ, R. H. Listen and Whisper: Security Mechanisms for BGP. In *Proc. USENIX NSDI* (Mar. 2004).

[37] WANG, L., ZHAO, X., PEI, D., BUSH, R., MASSEY, D., MANKIN, A., WU, S., AND ZHANG, L. Protecting BGP Routes to Top Level DNS Servers. In *Proc. IEEE ICDCS* (2003).

[38] XU, W., AND REXFORD., J. Don't Secure Routing Protocols, Secure Data Delivery. In *Proc. ACM HotNets* (2006).

[39] XU, W., AND REXFORD., J. MIRO: multi-path interdomain routing. In *Proc. ACM SIGCOMM* (2006).

[40] ZHANG, B., LIU, R. A., MASSEY, D., AND ZHANG, L. Collecting the Internet AS-level Topology. *Computer Communication Review 35*, 1 (2004), 53–61.

[41] ZHANG, Y., ZHANG, Z., MAO, Z. M., HU, Y. C., , AND MAGGS, B. On the Impact of Route Monitor Selection. In *Proceedings of ACM IMC* (2007).

[42] ZHANG, Z., YANG, Y., HU, Y. C., AND MAO, Z. M. Practical Defenses Against BGP Prefix Hijacking. In *Proc. of CoNext* (Dec. 2007).

[43] ZHANG, Z., ZHANG, Y., HU, Y., MAO, Z., AND BUSH, R. iSPY: Detecting IP Prefix Hijacking on My Own. In *Proc. ACM SIGCOMM* (Aug. 2008).

[44] ZHAO, X., PEI, D., WANG, L., MASSEY, D., MANKIN, A., WU, S., AND ZHANG, L. Dection of Invalid Routing Announcement in the Internet. In *Proc. IEEE/IFIP DSN* (June 2002).

[45] ZHENG, C., JI, L., PEI, D., WANG, J., AND FRANCIS, P. A Light-Weight Distributed Scheme for Detecting IP Prefix Hijacks in Real-Time. In *Proc. ACM SIGCOMM* (Aug. 2007).

Notes

[1] Note that some vendor implementation does not check whether the neighbor has appended its own AS in the announcement , while some vendor implementation does check (in which this hijack does not succeed).

[2] The complexity is not a concern here because the number of clusters is relatively small comparing to traditional clustering problem.

[3] Disscussion form: `http://iar.cs.unm.edu/phpBB2/viewforum.php?f=2`

[4] To choose the set of Tier-1 nodes, we started with a well known list, and added a few high degree nodes that form a clique with the existing set. Nodes other than Tier-1s but provide transit service to other AS nodes, are classified as transit nodes, and the remainder of nodes are classified as stub nodes.