Automatically Inferring the Evolution of Malicious Activity on the Internet

Shobha Venkataraman
AT&T Labs – Research
shvenk@research.att.com

David Brumley
Carnegie Mellon University
dbrumley@cmu.edu

Subhabrata Sen
AT&T Labs – Research
sen@research.att.com

Oliver Spatscheck
AT&T Labs – Research
spatsch@research.att.com

Abstract

Internet-based services routinely contend with a range of malicious activity (e.g., spam, scans, botnets) that can potentially arise from virtually any part of the global Internet infrastructure and that can shift longitudinally over time. In this paper, we develop the first algorithmic techniques to automatically infer regions of the Internet with shifting security characteristics in an online fashion. Conceptually, our key idea is to model the malicious activity on the Internet as a decision tree over the IP address space, and identify the dynamics of the malicious activity by inferring the dynamics of the decision tree. Our evaluations on large corpuses of mail data and botnet data indicate that our algorithms are fast, can keep up with Internet-scale traffic data, and can extract changes in sources of malicious activity substantially better (a factor of 2.5) than approaches based on using predetermined levels of aggregation such as BGP-based network-aware clusters. Our case studies demonstrate our algorithm's ability to summarize large shifts in malicious activity to a small number of IP regions (by as much as two orders of magnitude), and thus help focus limited operator resources. Using our algorithms, we find that some regions of the Internet are prone to much faster changes than others, such as a set of small and medium-sized hosting providers that are of particular interest to mail operators.

1 Introduction

Business-critical Internet-based services have to routinely contend with and mitigate a range of malicious activity (e.g. spam, scans, botnets) that can arise from virtually any part of the global Internet infrastructure. Identifying the regions of malicious activity on the Internet is valuable for enhancing the security of networks, applications, and end-users along multiple dimensions and timescales. However, static snapshots showing malicious activity at a particular point of time are of limited use because evil is constantly on the move. Administrators often eventually discover and clean up infected hosts, which causes attackers to target new vulnerabilities and attack new hosts elsewhere. Indeed, operators care far more about the evolution of malicious activity than static snapshots, as the evolution provides warning signs of emerging threats from regions previously-considered benign.

However, there has been little work on developing algorithms that can automatically infer how *aggregations* of malicious IPs evolve over time. Previous work has either created static snapshots [28, 29], or has explored the feasibility of using various a priori fixed IP clustering schemes for spam-filtering over longer periods [9, 13, 22, 27, 30], among which BGP-based prefix clustering schemes, such as network-aware clusters [19] have been especially popular. One challenge is it is not obvious a priori what level of aggregation granularity to use. While we know malicious IP addresses tend to be clustered, e.g., to ISPs with poorly-managed networks [5, 21, 23, 30], many natural options for a particular granularity provide inaccurate results. For instance, the individual IP address is too fine-grained to provide useful results [16, 23, 30, 31], e.g., DHCP can cause a single attacker to appear and disappear quickly from specific IP addresses. On the other hand, predetermined aggregations of IP addresses such as by AS or BGP prefix also does not afford the correct granularity. For example, network-aware clustering using BGP routing prefixes are likely to cluster the well-managed infrastructure hosts of an ISP together with its poorly-managed broadband access customers. This is highlighted in several recent results [9, 13, 22, 27], which have illustrated that BGP-based IP aggregations allow only for a coarse classification of malicious activity.

Since there are no obvious natural a priori aggregations to use, we need to be able to automatically infer the appropriate aggregation levels for detecting changes in dif-

ferent parts of the Internet, based on current observations. The appropriate aggregation varies drastically from region to region: some regions, such as small or mid-sized hosting providers, likely need to be monitored at very fine granularities (such a /24 or smaller prefix size), while other regions (e.g., entire countries that appear to be spam havens) need to be monitored at much coarser granularities. The problem becomes even more critical as IPv6 starts to get widely deployed – it is infeasible to even enumerate every IP address in the IPv6 address space. A practical algorithm therefore needs to scale as a function of the number of distinct prefix aggregations needed, not as a function of the size of the address space. A further complication is that not every change in malicious activity is useful to find, e.g., newly spamming IPs are of little interest if they belong to a well-known spam haven, but of substantial interest if they belong to a well-managed business network. Previous work has not addressed this problem.

In this paper, we develop the first algorithmic techniques to automatically infer regions of the internet with *shifting* security characteristics in an online fashion. We call an IP prefix that turns from good to evil a Δ-*bad prefix*, and a bad prefix that sees the light and becomes good a Δ-*good prefix*. Our key idea is that shifts in malicious activity will trigger errors in an accurate classifier of the IP address space's malicious activity. We model the IP address space as a decision tree, which when given a particular prefix, outputs a label "good" or "bad". We also periodicially measure the error in the decision tree, i.e., measure when it labels an IP prefix as good when it is, in fact, originating malicious traffic. The intuition is that when the decision tree has such errors on prefixes that it used to label accurately, it is not indicative of a problem with the decision tree, but instead indicative of a Δ-good or Δ-bad change. An additional challenge is that not all prefixes need to be modeled at the same granularity, e.g., AT&T's prefix should not be modeled at the same granularity as MIT, even though both own a /8. A key component of our algorithm is it automatically infers the right granularity to minimize error in labeling IP prefixes Δ-good or Δ-bad.

More specifically, we present two algorithms to answer two main questions. First, can we identify the specific regions on the Internet that have changed their malicious activity? Second, are there regions on the Internet that change their malicious activity much more frequently than others? The first question helps operators quickly focus their attention on the region of importance, e.g., if one of their networks is suddenly compromised. The second question explores structural properties about the nature of changes in malicious activity, highlighting the prefixes that need to be "under watch", as they are among the most likely to be future sources of attacks.

We present two algorithms, Δ-*Change* and Δ-*Motion* respectively, that address the above two questions. At a high-level, Δ-Change answers the first question by analyzing how well the different prefix aggregations in the static snapshots model input data. By design, it ensures that every prefix identified by our algorithms has indeed undergone a change, i.e., our list of Δ-bad and Δ-good prefixes has no false positives. Δ-Motion answers the second question by using previously-accurate snapshots to identify individual IP addresses that have changed their behaviour, and then partitions the address space into regions that have a high volume of changes and regions that have few changes. Our algorithms work without assuming a fixed distribution of IP addresses (a common assumption in many learning algorithms, which allows for easier learning and inference). Indeed, part of the data comes from malicious adversaries who have an incentive to mislead our algorithms and evade detection.

We evaluate our algorithms experimentally on two different sources of malicious activity from a tier-1 ISP – four months of mail data labeled with spamming activity, and three months of network traces labeled with botnet activity, and we demonstrate that our algorithmic techniques can find changes in spam and botnet activity. In particular, our experiments show *we can find more shifts in malicious activity by a factor of 2.5 than by applying extensions of existing static algorithms such as network aware clusters*. Through case studies, we demonstrate how our algorithms can provide operators with a network-wide understanding of malicious activity (both internal as well as external), and help them prioritize scarce manual effort to the most affected regions. For example, in one case study, our algorithm summarized a large shifts in botnet activity into a very small number of Δ-change prefixes (22,000-36,000 new IPs from DNSChanger and Sality botnets into 19-66 prefixes – a drop of over two orders of magnitude). In another case study, our algorithm discovered a large number of regional ISPs whose spamming activity dropped during the takedown of the Grum botnet. Finally, we find that there are certain regions of the IP address space that are much more prone to changes in spamming activity. For example, we found that a set of small and mid-sized hosting providers (which do not appear as distinct entities in BGP prefixes) are extremely prone to changes in spam activity – this is an intuitive result which network operators can easily validate (and then begin to monitor), and which our algorithm discovered automatically from noisy decision tree snapshots with nearly 100,000 nodes each.

Our algorithms are also scalable: our current (unoptimized) implementation is able to process a day's worth of data (30-35 million IPs) in around 20-22 minutes, on a 2.4GHz processor with only a single pass over the data and uses only 2-3 MB of memory. Further, a switch to IPv6 will have relatively little impact on our algorithm, as the re-

quired size of the decision trees is only a function of the distinct administrative entities in terms of malicious behaviour, rather than the size of the address space.

More broadly, our results show that while there is plenty of change in the malicious (both spamming and botnet) activity on the Internet, there is also significant structure and predictability in these changing regions, which may be useful for enhancing mitigation strategies.

2 Definitions and Preliminaries

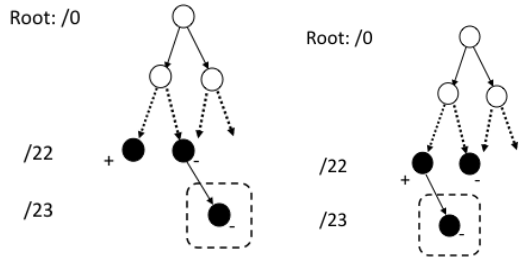
Our high-level goal is to design an algorithm that takes as input a stream of IP addresses flagged malicious or non-malicious (e.g., spam logs, labeled with spam-filtering software), and finds a set of IP prefixes whose IP addresses have changed from malicious to non-malicious, or vice-versa, across the stream. In this section, we describe how important changes can be naturally modeled by monitoring a decision tree on the IP address space.

Background. We first introduce some standard machine learning terminology. A *classification function* (or a *classifier*) is a function that takes as input a given IP address, and outputs a *label* denoting whether the IP address is malicious (also denoted by a "-") or non-malicious (also denoted by a "+"). The classification function makes a *mistake* whenever it labels a malicious IP address as non-malicious, or a non-malicious IP address as malicious. The *classification error* of a classification function is the fraction of the input IP addresses on which it makes a mistake.

We also introduce some networking background. An *IP address prefix* (also called *IP prefix*) i/d denotes the part of the IP address space that is covered by the first d bits of i, e.g., the prefix $10.0.0.0/8$ indicates the part of the IP address space whose first octet is 10, i.e., all IP addresses in the set $10.*.*.*$. Note that the prefix $i/d+1$, (i.e., $10.0.0.0/9$ in our example) denotes a subset of the address denoted that i/d (i.e., $10.0.0.0/8$). The IP address hierarchy can be naturally interpreted as a binary tree: the leaves of the tree correspond to individual IP addresses, the internal nodes correspond to the IP prefixes, and IP prefix i/d is the parent of the prefix $i/d+1$ in this representation. We say that IP prefix x belongs to prefix y if x is a parent of y in this tree, e.g., $10.0.0.0/9$ belongs to $10.0.0.0/8$.

2.1 Modeling the Problem

We begin with a motivating example. Consider a /23 prefix owned by an access provider, and suppose that a number of hosts with IP addresses in this /23 get compromised and start spamming. Now if this /23 prefix belongs to a larger prefix (say, a parent /22) that is already a known spam-haven, this new spamming activity of the /23 is not very interesting to an operator, since the larger region is known to spam (i.e., it is not surprising that a smaller region within a known spam-haven also starts to spam). If, on the other hand, the larger prefix (e.g., the parent /22) has originated only legitimate traffic so far, the new spamming activity becomes much more interesting to network operators, because they previously assumed that the region did not spam. By notifying operators of the change, they can control or block the spam from the /23 to their networks. We illustrate this example in Figure 1.



(a) No change (b) Change

Figure 1. Example of Δ-bad Changes. (a) shows a prefix that is not Δ-bad, because /23 starts originating malicious traffic when its parent /22 is already known to originate malicious traffic (b) shows a prefix that is defined as Δ-bad, because the /23 starts originating malicious traffic when its parent /22 is not known to originate malicious traffic.

A key part of this example is having an accurate classifier for the the type of traffic originated by the two /22 prefixes – we need to know what kind of traffic a particular region is expected to originate, before we can understand when the region has changed its malicious activity. However, we do not have such a classifier given to us as input, and we need to infer it dynamically from a stream of IP addresses and their associated labels. Thus, to infer change, we first have to infer such a classifier for the prefixes from the labeled IP addresses, and then use this classifier to infer changes. Moreover, the appropriate prefix granularity for such a classifier is different in different parts of the Internet, we need to also infer the required prefix granularities. Because it is likely impossible to infer a classifier with zero error, we instead will look for changes relative to any classifier that makes no more than τ error on the data, for small (input) $\tau > 0$. By definition, all such classifiers must classify most of the data identically. In particular, let s_t denote the stream of input $< IP, label >$ pairs appearing in epoch t; our goal is to detect prefixes that have changed in s_{t+1} relative to a classifier that makes no more than an input τ error on s_t.

Algorithmic constraints and Adversarial Model The scale of network traffic makes it infeasible to use computationally expensive algorithms. In particular, a solution should have constant processing time per IP, make only a single pass over the input streams, and have memory requirements that are sublinear in the input data size. Such al-

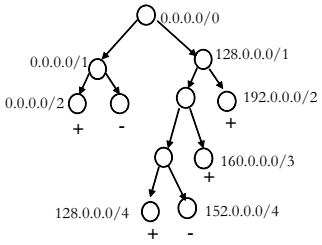


Figure 2. Example IPtree of size 6, since it has 6 leaves. Each leaf has a "+" or a "-", denoting whether the associated prefix originates non-malicious or malicious traffic. Section 3 describes how we learn such a tree from data.

gorithms are called *online*, and are among the most desired (and difficult to create). In addition, our data may have to have some *noise* – e.g., an IP may be labeled as producing spam incorrectly. For example, if our labels are coming from SpamAssassin, and SpamAssassin mislabels legitimate mail from an IP as spam, then our algorithm receives an inaccuracy label for this IP, and must be able to cope with this inaccuracy. Our algorithm's performance thus needs to scale gracefully as the noise increases, and be able to produce accurate output when the noise in the data is tiny. Finally, we cannot assume that input IPs are drawn from a fixed probability distribution over $\mathcal{I}$. Although assuming a fixed distribution would be easier, it would make the algorithm easier to evade. In particular, we assume an adversary can pick the addresses from which malicious activity originates, and therefore, could mislead any algorithm assuming that all IPs originate from a priori fixed distribution.

Practical considerations There are additional constraints that make the algorithm more useful by directing attention towards changes that are most actionable by operators. First, we aim to detect prefixes with at least θ traffic since (1) data may be occasionally mislabeled, and (2) changes in prefixes with very little traffic may not be noteworthy.

In addition, operators only care about prefixes where the change is directly evident: i.e., if the prefix changes from originating mostly non-malicious activity to mostly malicious activity, or vice versa. [1] To formalize this concept, we introduce the concept of a *state* to reflect level of malicious activity of a prefix. Formally, a *state* is defined by an interval in $[0, 1]$; the set of all states D input to the algorithm is given by a collection of non-overlapping intervals in $[0, 1]$. A prefix is assigned a state based on the fraction of traffic it originates that is non-malicious. Thus, for example, the state defined by the interval $[0, 0.2]$ is assigned to prefixes sending between $0 - 20\%$ spam. Conceptually, the state of a prefix can be thought of measuring the level of "badness" of the prefix. We define a *localized* change in a prefix to be one where the prefix has changed its state, and our goal is to find only localized changes. For example, suppose the set D consists of two intervals $[0, 0.2)$ and $[0.2, 1)$. A prefix that used to send less than 20% spam, but now sends between $20-100\%$ has undergone a localized change (in effect, the prefix is considered non-malicious if it sends less than 20% spam, and malicious if it sends at least 20% spam, and we are only interested in finding when the prefix changes from malicious to non-malicious, or vice-versa.) The set D is input to the algorithm. Continuous intervals provide finer-grained data to operators than just malicious and non-malicious. Of course, in reports to the operators, we can always reduce to just malicious and non-malicious if desired. [2]

Modeling Malicious Activity of Prefixes as Decision Tree. We take advantage of the structural properties of malicious activity in order to design an efficient and accurate algorithm for detecting changes. Prior work has demonstrated that malicious traffic tends to be concentrated in some parts of the address space [5, 21, 23, 30] – that is, the IP address space can be partitioned into distinct prefix-based regions, some of which mostly originate malicious traffic and some that mostly originate legitimate traffic. We observe that the IP address space can be represented as a tree of prefixes. Thus, we can model the structure of malicious activity as a decision tree over the IP address space hierarchy rooted at the /0 prefix: the leaves of this tree are prefix-based partitions that send mostly malicious or mostly non-malicious traffic; this is a decision tree since each leaf in the tree can be considered as having a "label" that indicates the kind of traffic that the corresponding prefix-based region originates (e.g., the label might be "bad" when the region originates mostly malicious traffic, "good" when the region originates mostly legitimate traffic). The changes in prefix behaviour can then be precisely captured by changes in this decision tree. In Sec. 3, we describe how we learn this decision tree to model the malicious activity from the data.

More formally: let $\mathcal{I}$ denote the set of all IP addresses, and $\mathcal{P}$ denote the set of all IP prefixes. An *IPTree* T_P over the IP address hierarchy is a tree whose nodes are prefixes $P \in \mathcal{P}$, and whose leaves are each associated with a label, malicious or non-malicious. An IPtree thus serves as a classification function for the IP addresses $\mathcal{I}$. An IP address $i \in \mathcal{I}$ gets the label associated with its longest matching prefix in the tree. A k-IPtree is an IPtree with at most k leaves. By fixing the number of leaves, we get a constant-

[1] There may be situations where a prefix undergoes changes, but the change is not directly observed when traffic is aggregated at that prefix, e.g., a prefix could originate roughly the same amount of malicious and non-malicious traffic in s_{t+1} as it did in s_t, but misclassify both malicious and non-malicious activity on s_{t+1} (perhaps because some of its children prefixes have changed). We ignore such changes in this paper as they are not typically actionable.

[2] We could also define changes in terms of the relative shift in the malicious activity of the prefix. However, the definition we use above allows for a conceptually easier way to explain prefix behavior.

sized data structure. The *optimal* k-IPtree on a stream of IP address-label pairs is the k-IPtree that makes the smallest number of mistakes on the stream. Figure 2 shows an example IPtree of size 6.

We define prefix changes in terms of the IPTree: We define a Δ-*bad prefix* for an IPTree T as a prefix p that starts to originate malicious traffic when T labels traffic from p as legitimate. Likewise, a Δ-*good prefix* is a prefix p that starts to originate legitimate traffic when T labels traffic from p as malicious. In the example of Fig. 1, the /24s in the first and second scenarios are labeled as malicious and non-malicious respectively. The /25 in the second case sends traffic that differs from the tree's label. Fig. 1(b) shows an example Δ-bad prefix. Without loss of generality, we will use Δ-*change prefix* to refer to either Δ-good or a Δ-bad prefix. We of course report back to an operator whether a Δ-change prefix is Δ-bad or Δ-good.

In this paper we use TrackIPTree as a subroutine in our algorithms in order to infer decision trees from the data stream, as it meets all our algorithmic requirements for scalably building near-optimal decision trees over adversarial IP address data [29]. (Note TrackIPTree does not solve the problem of detecting the changed prefixes posed in this paper, even with a number of extensions, as we discuss in Section 2.2.) Conceptually, TrackIPTree keeps track of a large collection of closely-related decision trees, each of which is associated with a particular weight. It predicts the label for an IP address by a choosing a decision tree from this set in proportion to its relative weight in the set; when given labeled data to learn from, it increases the weights of the decision trees that make correct predictions, and decreases the weights of those that make incorrect predictions. TrackIPTree accomplishes this efficiently (from both space and computation perspectives) by keeping a single tree with the weights decomposed appropriately into the individual prefixes of the tree.

2.2 Alternate Approaches

We first discuss a few previous approaches that may appear to be simpler alternatives to our algorithms, and explain why they do not work.

BGP prefixes. A straightforward idea would be to use BGP prefixes such as *network-aware clusters* [19], a clustering that represents IP addresses that are close in terms of network topology. BGP prefixes have been a popular choice in measurement studies of spamming activity and spam-detection schemes [9, 13, 22, 27, 30], but have increasingly been shown to be far too coarse to model spamming activity accurately [22].

Unfortunately, BGP prefixes perform poorly because they do not model the address space at the appropriate granularity for malicious activity. BGP prefixes only reflect the granularity of the address space at which routing happens, but actual ownership (and corresponding security properties) may happen at finer or coarser prefix granularity. (Likewise, ASes are also not an appropriate representation because even though the Internet is clustered into ASes, there is no one-to-one mapping between service providers and ASes [3].) Our experiments in Sec. 4.1 demonstrate this, where network-aware clusters identify around 2.5 times fewer Δ-change prefixes than our algorithms. For example, such an algorithm fails to report Δ-changes in small to medium hosting providers. These hosting providers are located in different regions of the world; the provider manages small prefix blocks, but these prefix blocks do not appear in BGP prefixes. Any change in the hosting provider's behavior typically just disappears into the noise when observed at the owning BGP prefix, but can be the root cause of malicious activity that the operator should know about.

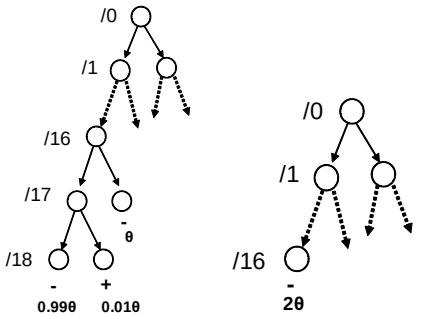
Strawman Approaches based on TrackIPTree. A second approach would be to learn IPTree snapshots that can classify the data accurately for different time intervals, and simply "diff" the IPTree snapshots to find the Δ-change prefixes. TrackIPTree [29] is a natural choice to construct these IPTree, as it can build a near-optimal classifier. However, even with near-optimal IPTrees, we cannot directly compare them to accurate find Δ-change prefixes.

Let s_a, s_b be two arbitrary input sequences of IPs on which we make no a priori assumptions, as described in Section 2.1. [3] Let T_a and T_b be the resulting IPtrees after learning over s_a and s_b respectively using TrackIPTree [29]. There are many immediate ways we could compare T_a and T_b, but when the trees are large, noisy and potentially error-prone, most of these lead to a lot of false positives. We use here small examples to illustrate how these differencing approaches fail, and in Section 4, we show that these lead to extremely high false positive rates on real IPTrees.

One possible approach to compare two decision trees is to compare the labels of their leaves. However, the two trees may assign different labels to a region even when there is not a (significant) difference in the relevant parts of s_a and s_b, e.g., both trees may be inaccurate in that region, making any differences found to be false positives.

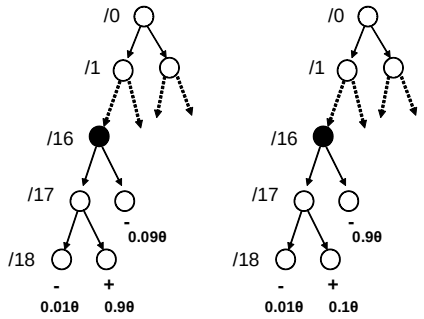
Even if we know which parts of the tree are accurate, and restrict ourselves to comparing only "mostly accurate" prefixes, we still cannot directly compare the trees. The trees may still appear different because the tree structure is different, even though they encode almost identical models of

[3] We make no assumption on the $< IP, label >$ pairs that are present in s_a and s_b. This means that there may be some IPs that are common to both s_a and s_b, and others that IPs are not present in s_a or s_b. The labels of the common IPs do not need to be identical in s_a and s_b; indeed, we expect that in real data, some of the common IPs will have the same labels in s_a and s_b, but others will differ. Even within a single sequence s_a, an IP i does not need to have the same label throughout, it may have different labels at different points in the sequence s_a.



(a) Learned T_a (b) Learned T_b

Figure 3. Comparing "Mostly Accurate" Prefixes. T_a and T_b classify 99% of traffic seen identically, but would be flagged different because of differences in the tree structure that affect very little traffic.



(a) Learned T_a (b) Learned T_b

Figure 4. Comparing prefixes that are accurate as well as have sufficient traffic. T_a and T_b are accurate, and share identical leaf labels; however, none of the leaves have enough traffic to be compared.

the malicious activity. We illustrate this with the example in Figure 3 (assume that each leaf shown in the figure classifies over 95% of its respective traffic accurately). The two trees T_a and T_b (learned over s_a and s_b respectively) then classify over 99% of IPs identically, yet would be flagged different if we simply compared their accurate prefixes. Such small deviations might just be caused by noise since the trees are learned over different sequences, e.g., s_a might have had more noise than s_b. It is of little use to identify such Δ-bad prefixes for operators.

A third possible approach considers only prefixes that are both "mostly accurate" and have sufficient (at least θ) traffic, but even then, we cannot simply compare the trees. Consider Figure 4, where the true tree has a /16 prefix with two /17 children, and one /17 originates only malicious IPs, while the other /17 originates only legitimate traffic. In the two learned trees T_a and T_b, none of the leaves see sufficient (θ) traffic.[4] In this example, the highlighted /16 prefix is the longest parent prefix with θ traffic in both T_a and T_b. If we analyze the interior prefix's activity by the traffic it has seen, most of the traffic seen by the /16 is non-malicious in T_a and malicious in T_b. Thus, we would flag it as a Δ-bad prefix. However, this is once again a false positive – note that all leaf labels in T_a and T_b are identical (i.e., no region has actually changed its behaviour) – the only change is that a few leaves send less traffic in T_a and more in T_b (and vice versa). Such changes in traffic volume distribution occur routinely without malicious regions becoming benign. For example, some spam-bots may become quiet for a few days while they receive new spam templates, and then restart spamming activities. In Sec. 4.1, we show empirically that this third approach can lead to false positive rates of over 47%.

[4]We need to analyze the interior prefixes to ensure that we do not miss legitimate changes. For example, imagine a scenario where most of the leaves in T_a are negative, while most of the leaves in T_b are positive. The longest parent prefix with at least θ traffic is an interior prefix, and it has clearly undergone a change. If we do not analyze the interior prefix, we will miss such changes.

3 Our Algorithms

3.1 Overview of Our Approach

Our key insight is to use the classification error between the two trees in order to infer Δ-change prefixes. If a prefix has had low classification error in earlier time intervals with T_z, but now has high classification error (on substantial traffic), we can infer that it has undergone a Δ-change. The (earlier) low classification error (on sufficient traffic) implies our tree T_z used to model this region well in the past intervals, but and the current high classification error implies does not do so any longer. Thus, we infer that the prefix has changed its behavior – that it is sending traffic that is inconsistent with its past behaviour – and therefore, is a Δ-change region. As long as we are able to maintain a decision tree with high predictive accuracy for the sequences, our analysis can discover most prefixes changing between s_a and s_b. Further, by only selecting prefixes that have a high classification error on a substantial fraction of the traffic, we build some noise-tolerance into our approach.

This insight shows that we need to achieve the following three simultaneous goals to address the IPTree evolution problem: (1) keep track of a current model of the malicious activity; (2) measure the classification errors of the current sequence based on a *prior* accurate model; (3) keep track of a current model of the frequently changing regions. We keep multiple decision trees over the address to simultaneously achieve these goals. At a high-level, we let one IPtree learn over the current sequence, so it tracks the current malicious activity. We keep second set of IPtrees fixed (i.e., they cannot change its labels, weights, or structure), and use them to measure the classification accuracy on the current sequence. We then compare the classification errors of the second set of IPtrees (not the IPtrees themselves) on the different sequences to compute the specific changed prefixes (details in Section 3.2). For our third goal, we use our

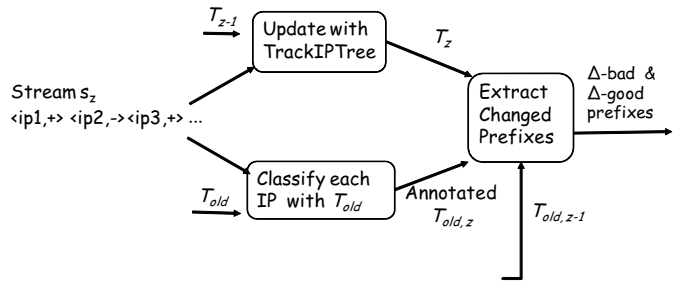


Figure 5. High-level sketch of Δ-Change Algorithm

learned IPtrees to discover which of the IP addresses in the current sequence have changed their labels. We then learn a third IPtree based on this information, partitions the address space into regions that are change frequently and those that do not. (To avoid confusion, we term this third IPtree as *change-IPTree*, and define it in Section 3.3).

3.2 The Δ-Change Algorithm

We now present our Δ-Change algorithm which addresses the question: can we identify the specific regions of changing malicious activity? Recall that a k-IPtree is a decision tree over the IP address space with at most k leaves. Let $s_1, s_2 \ldots$ denote the sequences of IP addresses at each time interval, and let T_z denote the IPtree built over the sequence s_z. For readability, we use T_{curr} to denote the tree that is being learned over the current sequence of IPs, and T_{old} to denote an older tree that is being kept fixed over the current sequence of IPs. We use $T_{old,z}$ to denote the tree T_{old} that is annotated with its classification errors on the sequence s_z.

At a high-level, we do the following: As each labelled IP i arrives in the sequence s_z, (i) we predict a label with the trees T_{curr} and $T_{old,z}$; (ii) we annotate the tree $T_{old,z}$ with its classification error on i, and (iii) we update the model of T_{curr} with i if necessary. At the end of the interval, we compare classification errors $T_{old,z-1}$ with $T_{old,z}$ and T_{curr} to discover the Δ-change prefixes, discarding redundant prefixes as appropriate (i.e., when multiple prefixes reflect the same change). T_{old} is then updated appropriately (we specify the details later), so it can used for measuring the classification errors of the IPs in the next sequence s_{z+1}. We sketch the high-level view of Δ-Change in Figure 5.

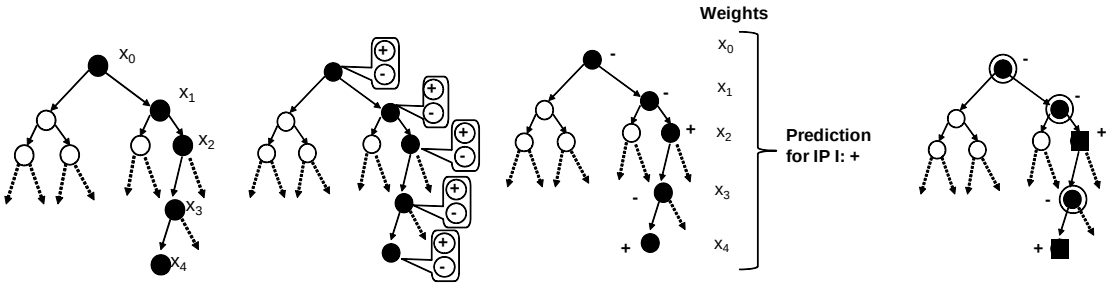
We describe the construction of T_{curr} and $T_{old,z}$ in Section 3.2.1, and the comparison between the trees in Section 3.2.2. We describe Δ-Change in terms of multiple trees for conceptual clarity. However, it may be implemented with just one IPTree (and additional counters) for efficiency.

3.2.1 Constructing the IPTrees

At a high-level, TrackIPTree involves all parent prefixes of an IP i in current IPtree in both in classifying IP i, as well

```
Δ-CHANGE Input: sequence s_z, T_old, T_curr;
    for IP-label pair < i, label > in s_z
        p_{i,curr} := label predicted on i using
          TrackIPTree on T_curr
        p_{i,old} := label predicted on i using
          TrackIPTree on T_old
        AnnotateTree(T_{old,z}, i)
        Update (T_curr, i, label)
    ExtractChangedPrefixes(T_old, T_curr);
sub ANNOTATETREE
    Input: IPTree T_old, IP i, label l
    for each parent prefix j of i in T_old
        IPs[j, label] += 1
        if p_{i,curr} ≠ p_{i,old}
            mistakes[j, label] += 1
sub EXTRACTCHANGEDPREFIXES
    Input: IPTree T_{old,z}, IPTree T_{old,z-1}, IPTree
      T_curr, error threshold γ, IP threshold θ
    Output: Set of Δ-changes C
    //Step 4: Isolate Candidate Prefixes
    Candidate Set of Δ-changes C = {}
    for each prefix j ∈ T_{old,z}
        error[j] = (mistakes[j, +] +
          mistakes[j, −])/(IPs[j, +] + IPs[j, −]);
        if error[j] > γ and IPs[j] > θ and
          state[j, T_old] ≠ state[j, T_curr]
            Add prefix j to candidate set C
    //Step 5: Prune Redundant Prefixes
    for each prefix c ∈ C
        for each parent j of c in T_old
            childMistakes[j] += mistakes[c];
            childIPs[j] += IPs[c]
        for each prefix c ∈ C
            if mistakes[c] - childMistakes[c] < θ
              || IPs[c] - childIPs[c] < γ
                discard c from C
        //Step 6: Discover New Children
        for each prefix c ∈ C
            for each child node c' of c in T_curr
            if c' ∉ T_old
                add subtree(c) to C
```

Figure 6. Pseudocode for Δ-Change



(a) Parents of IP i in T, with relative importance weights

(b) Each parent chooses "+" or "-", based on its own label predictor weights

(c) Weight parent predictions by relative importance weights to get final prediction for the IP i

(d) Update T_{curr}: Penalize incorrect parents, and grow or prune T if needed

Figure 7. Δ-Change Algorithm: Illustrating Steps 1 & 3 for a single IP i. (a)-(c) show Step 1, and (d) shows Step 3. The shaded nodes indicate the parents of the current IP i in the tree.

as in learning from labeled IP i. Crucially, TrackIPTree decomposes the main prediction problem into 3 subproblems, which it treats independently: (a) deciding the prediction of each individual parent prefix, (b) combining the parent prefix predictions by deciding their relative importance, and (c) maintaining the tree structure so that the appropriate subtrees are grown and the unwanted subtrees are discarded. These subproblems are cast as instances of experts' problems [11, 20].

TrackIPTree uses the following data structure, shown in Figure 7(a) and (b). Each node in the IPtree maintains two sets of weights. One set, termed *label predictors*, decide whether an individual node should predict non-malicious or malicious, denoted $\{y_{j,+}, y_{j,-}\}$ for node j (Fig. 7(b)). The second set, termed *relative importance predictors*, keeps track of the weight of the prefix's prediction, in relation to the predictions of other prefixes – we use x_j to denote this weight for node j (Fig. 7(a)). In Δ-Change, we augment the basic IPTree data structure with a few additional counters, so that we can keep track of the classification error rates at the different parts of the address space on the current sequence s_z (we elaborate further in Step 2). Below we describe how we use this data structure in order to apply the relevant components of TrackIPTree to construct T_{curr} and $T_{old,z}$ and then discover the Δ-change prefixes.

Step 1: Predicting with T_{old} and T_{curr}. We compute predictions of T_{old} and T_{curr} using the prediction rules of TrackIPTree. We first allow each prefix of i in T_{old} (and likewise, T_{curr}) to make an individual prediction (biased by its label predictors). Then, we combine the predictions of the individual nodes (biased by its relative importance predictor). We illustrate these steps in Fig. 7(a)-(c).

Formally, let P denote the set of prefixes of i in a tree T. We compute each prefix $j \in P$'s prediction $p_{i,j}$, with a bias of $y_{j,+}$ to predict non-malicious, and a bias of $y_{j,-}$ to predict malicious. We then combine all the predictions $p_{i,j}$ into one prediction p_i for the IP i, by choosing prediction $p_{i,j}$ of node j with probability x_j, its relative importance predictor. The p_i is our final output prediction.

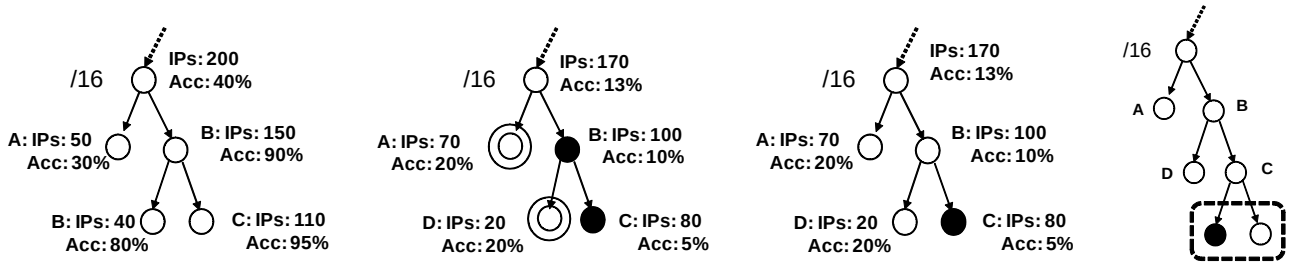
Step 2: Annotating T_{old} with Classification Errors. We next describe how we annotate the IPTree T_{old} to produce $T_{old,z}$ based on the errors that T_{old} makes on s_z. For this, we augment the basic IPTree data structure with four additional counters at each prefix of T_{old}: two counters that keep track of the number of malicious and non-malicious IPs on the sequence s_z, and two counters that keep track of the number of mistakes made on malicious and non-malicious IPs on s_z. (This is the only update to the IPtree data structure of [29] that Δ-Change requires.)

We do the following for each IP i: if the output prediction p_i of T_{old} (obtained from Step 1) does not match the input label of IP i, we increment the number of mistakes made at each parent prefix of i in tree $T_{old,z}$. We track mistakes on malicious and non-malicious IPs separately. We also update the number of malicious and non-malicious IPs seen at the prefix in the sequence s_z. Thus, for example, in Figure 7, if the input label of IP i is "-" and T_{old} has predicted "+", we update the errors for malicious IPs at each highlighted parent prefix of i, and we also update the number of malicious IPs seen at each highlighted parent prefix.

Step 3: Learning T_{curr}. Finally, we update T_{curr} with the labeled IP. This ensures that our model of the stream is current. This step directly applies the update rules of TrackIPTree to T_{curr} [29], as a subroutine. Effectively, the learning procedure penalizes the incorrect prefixes and rewards the correct prefixes by changing their weights appropriately; it then updates the tree's structure by growing or pruning as necessary. Due to space limits, we omit a detailed description of TrackIPTree's update rules here.

3.2.2 Extracting Changes from IPTrees

At this point, we have measured the classification error of T_{old} over the sequence s_z (denoted by $T_{old,z}$), and we have allowed T_{curr} to learn over s_z. Now, our goal is to extract the appropriate changes between s_z and s_{z-1} by comparing



(a) $T_{old,z-1}$: prior IPTree, annotated with errors on s_{z-1}

(b) $T_{old,z}$: prior IPTree, annotated with errors on s_z

(c) $T_{old,z}$: final Δ-change prefix (shaded)

(d) T_{curr}: new children found for Δ-change prefix

Figure 8. Δ-Change Algorithm: Steps 4-6. Running example illustrates how Δ-change prefixes are extracted by comparing $T_{old,z-1}$, $T_{old,z}$ and T_{curr}

the trees $T_{old,z}$, $T_{old,z-1}$ and T_{curr}.

At a high-level, we do this in three steps: (1) first, we isolate candidate Δ-change prefixes, by comparing their classification errors and states between $T_{old,z}$ and $T_{old,z-1}$; (2) we prune redundant prefixes, which ensures that we do not include prefixes as Δ-change if their children already account for bulk of the change; (3) finally, we discover new children that may have been grown in relation to these changes by comparing T_{curr} and T_{old}.

Specifically, let D be the set of states that prefixes can be assigned to. Let θ minimum number of IPs that a prefix needs to originate to be considered potentially a Δ-change. Let τ denote the maximum error a prefix may have to still be considered an accurate model of the region's traffic.[5] Let γ as the minimum increase in a prefix's error that guarantees that the prefix is indeed Δ-change. We derive γ from the set D, as we describe later. We will use Figure 8 to illustrate these three steps with a running example, where we set $\theta = 50$, $\tau = 0.1$, and $\gamma = 0.5$. In the example, we allow a prefix to have one of two states: "good", which corresponds to 0-50% of its traffic being malicious, and "bad", which corresponds to 51-100% of its traffic being malicious.

Step 4: Isolate Candidate Prefixes. We isolate as a candidate set C all prefixes in T_{old} satisfying the following conditions: (1) its classification error in $T_{old,z}$ exceeds γ, and its classification error in $T_{old,z-1}$ is below τ; (2) at least θ instances have been seen at the prefix. We then compare the prediction labels of each prefix in C to the corresponding prefix in T_{curr} to check for a state change. If there is no change in the prefix's state, the change is not localized, and we discard the prefix from C.

Figure 8(a) shows the original $T_{old,z-1}$ (the states for each node are not shown for readability, assume they are all "good" in $T_{old,z-1}$) (b) shows $T_{old,z}$ (again, the states of each node are not shown, here assume all are "bad"). The shaded prefixes (nodes B & C) have both sufficient IPs and the necessary change in the classification error. Node A gets discarded because it is not accurate enough in $T_{old,z-1}$, and node D gets discarded because it has too few IPs.

Step 5: Prune Redundant Changes. Not all candidate prefixes isolated in C will represent a change in a distinct part of the IP address space. Every parent of a Δ-change prefix will also have made the same mistakes. In some cases, these mistakes may cause the parent prefix to also have a high overall classification error (e.g., when no other child of that parent originates substantial traffic). However, some parents of a Δ-change prefix may have a high classification error due to changes in a different part of the address space. To avoid over-counting, we include a prefix in C only if the following two conditions hold: (1) it accounts for an additional θ IPs after the removal of all children Δ-change prefixes; (2) there is at least γ classification error on the IPs from the remaining prefixes in $T_{old,z}$, and at most τ error in $T_{old,z}$. Figure 7(c) shows this step: we discard the parent /17 prefix (node B) from the list of Δ-change prefixes, because it does not account for an additional $\theta = 50$ IPs after we remove the contribution of its Δ-change /18 child (node C).

Step 6: Discover New Children. The tree T_{curr} may have grown new subtrees (with multiple children) to account for the changes in the input IP stream. We compare each prefix $c \in C$ with the corresponding prefixes in T_{curr} to see if new subtrees of c have been grown. If these subtrees differ from c in their prediction labels, we annotate c with these subtrees to report to the operator. Figure 8(d) shows T_{curr} and the corresponding subtrees (of depth 1 in this example) of the Δ-change prefix (node C) in T_{old}, which are annotated with node C for output.

To wrap up the Δ-Change algorithm, we discuss the two parameters we did not specify earlier. First, we need to define how we obtain T_{old} from T_{curr}. Since T_{old} needs to have its accuracy measured in interval $z-1$, it needs to be learnt no later than $z-2$. So, $T_{old} = T_{z-2}$ for the sequence s_z, and at every interval, we update T_{old} to be the tree learnt 2 intervals before the current one. We also need to derive γ from the set of states D. Since each state is defined by

[5] τ is typically set to a small value such as 0.01%, but cannot be set to 0% because of noisy data.

an interval in $[0, 1]$, we can use the interval boundaries to derive D. Thus, for example, if each state D has the same interval length, then $\gamma = \frac{1}{|D|}$.

Properties of Δ-Change: Efficiency and Correctness The Δ-Change algorithm meets our goals of operating online on streaming data and our computational requirements as each step involves only the following operations: learning an IPtree, comparing two IPtrees or applying an IPtree on the input data. The first operation is performed by the TrackIPTree algorithm, which is an online learning algorithm, and the remaining operations require storing only the IPtrees themselves, thus requiring only constant additional storage.

More precisely, the key data structures of Δ-Change are three IPtrees, i.e., $T_{old,z-1}, T_{old,z}, T_{curr}$. The basic IPTree data structure has a space complexity of $O(k)$ for a tree with k leaves [29], as TrackIPTree only stores a number of weights and counters per prefix. For Δ-Change algorithm, as described in Step 2, we need to augment the basic IPtree structure with four additional counters for each prefix. Thus, the space complexity for Δ-Change remains $O(k)$.

Next, we describe the run-time complexity of Δ-Change. Steps 1, 2 and 3 are applied to every IP in the input sequence, but each step has at most $O(\log k)$ complexity: for each IP, we examine all its parents in T_{curr} and T_{old} a constant number of times (only once in Step 2, 2-4 times in Steps 1 & 3 as part of the subroutines of TrackIPTree), so the run-time per IP is $O(logk)$. In Step 4, we compare each pair of prefixes in $T_{old,z}$ and $T_{old,z-1}$, so our run-time for Step 4 is $O(k)$. In Step 5, we examine potential Δ-change prefix together with its parents, so our run-time is bounded by $O(k \log k)$. For Step 6, we examine each potential Δ-change prefix together with its children subtrees in T_{curr}, and since T_{curr} is a k-IPtree, the run-time is bounded by $O(k)$. Thus, the total run-time of Δ-Change, for an input sequence of length n, becomes $O(n \log k + k \log k)$.

We conclude with a note about accuracy: by design, every Δ-change prefix discovered is guaranteed to reflect a change in the IPs between the sequences s_z and s_{z-1}. If a prefix has had high classification error in $T_{old,z}$ and low classification error in $T_{old,z-1}$, then that prefix is indeed originating a different kind of traffic in s_{z-1} than it did in s_z. Thus, the Δ-Change algorithm will have no false positives (though it may not find all Δ-change prefixes, since the T_{curr} and T_{old} are approximate).

3.3 The Δ-Motion Algorithm

In this section, we address the second question posed in our problem: What regions of the Internet are prone to frequent changes? The answer to this helps us pinpoint structural properties of the Δ-change prefixes.

A straightforward approach might be to use the Δ-change prefixes output by Δ-Change, but as just described

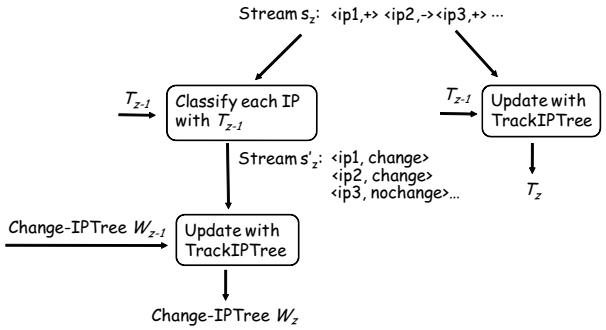


Figure 9. High-level approach of Δ-Motion

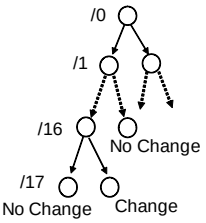


Figure 10. Example Change-IPTree: Partitioning the IP address space into "change" and "no-change" regions. This is just like the regular IPTree in Figure 1, but with the leaf labels denoting "change" or "no-change".

in Section 3.2, this list of Δ-change prefixes may be incomplete: Δ-Change can only ensure that every identified prefix is truly a Δ-change prefix (i.e., there are no false positives), but not that every Δ-change prefix is discovered (i.e., there may be false negatives). However, there is additional information in the structure of the learned IPtree T_{old} as well as the input data sequence s_z that we can exploit.

To answer our question, we need to partition the IP address space into regions that change quickly and regions that change slowly. We first observe that this problem may be modeled as another instance of the problem of learning an IPtree – we need simply to learn a decision tree over the IP address space *where the leaf labels denote "change" or "no change"*, rather than "malicious" or "non-malicious". For clarity, we define this IPTree as a *change-IPTree*; Figure 10 shows an example of such an IPtree. Therefore, if we get access to IPs labelled with "change" or "no change" (rather than our usual sequences of IPs labelled with "malicious" or "non-malicious"), and we directly use TrackIPTree to solve this problem.

Recall that we denote s_z to be the part of the stream that appears in interval z. Δ-Motion uses the IPtree T_{z-1} to annotate each IP i in s_z. If the label of IP i matches the prediction of T_{z-1}, it pairs IP i with label "no change", and if they do not match, it pairs the IP with a label "change". We thus have a new stream s'_z derived from s_z, where the label of each IP is "change" or "no change". Next, we apply TrackIPTree on this new stream, and the resulting change-IPtree

differentiates prefixes that change frequently from those do not change frequently. We use W_z to denote this change-IPtree built on the stream s'_z of IPs labeled with "change" or "no change". Even though the IPtree T_{z-1} we use to generate the new labels is approximate, it typically has a very high accuracy and so the new stream will typically have only a little noise. We note that the space and runtime complexity of Δ-Motion is identical to TrackIPTree: its data structure uses only three IPTrees (a change-IPTree and two regular IPTrees); each step of Δ-Motion applies a part of TrackIPTree, and the different parts of TrackIPTree are applied three times in Δ-Motion.

4 Experimental Results

Data. Our first data set uses spam as our source of malicious activity. Our data is collected from the operational mailservers of a tier-1 ISP which handle mail for total of over 8 million subscribers. We collected data in two periods: from mid-April to mid-August 2010 over 120 days, and from mid June to late July 2012, over 41 days. Our data set includes the IP addresses of the senders' mail servers and the number of spam and legitimate messages that each mail server sends in a 5-minute interval; we do not collect any other information. We use the mailserver's spam-filtering system (Brightmail) as labels for IP addresses in our learning algorithm; a single IP address can thus be labeled malicious at one point in time and non-malicious at a different point, as it may send legitimate messages at some points and spam at others. In total, the IP addresses in our data have sent over 5.3 billion spam and 310 million legitimate messages. While our data may have some noise in labeling (due to Brightmail mislabeling spam as legitimate mail and vice-versa), because the algorithm is adaptive and noise-tolerant, a small fraction of inaccurate labels in the data will not have a significant long-term impact on the tree.

Our second data set is based on botnet activity from October 2011 to January 2012. For this data set, we first obtain a distribution of the active IP addresses across the Internet by collecting daily snapshots of flows sampled from IP backbone traffic. All together, our monitoring points cover 80% of the traffic carried by the IP backbone. On any given day, our data includes 24-28 million unique IP addresses. We use botnet activity to label these IP addresses as malicious or non-malicious for our algorithms. In particular, we obtain a daily snapshot of IP addresses within a tier-1 ISP that are part of a botnet, as identified by the ISP's security vendors. These security vendors employ a combination of monitoring algorithms, sinkholes, spam traps and malware binary analysis to identify and track bot IP addresses, and the daily snapshot includes all the bot IPs observed by the vendors on that particular day – specifically, a bot IP is included in the list for a particular day only if it has generated activity matching a signature on that particular day [6] The botnet feed contains around 30,000-100,000 unique IP addresses daily (these include drones as well as the responsible C&C servers), and the feed includes over 2.64 million unique bot IP addresses in total across 94 days of data. We label an IP address as malicious on day i if it appears in the botnet feed on day i. As in the spam data set, any noise in the input data stream will be carried over to our results; however, if there is a only small amount of noise in the labeling, the adaptive nature of the algorithm ensures that there will not be a long-term impact on the tree.

Our results demonstrate that our algorithms are able to discover many changes in the Internet's malicious activity, and do so substantially better than alternate approaches. The exact Δ-change prefixes we detect are, of course, specific to our data sets, and for confidentiality reasons, we anonymize the owning entities of all the prefixes in the results. Our results show two examples of how our algorithm can be applied on real data sets from operational networks, and discover changes that operators were unaware of.

Experiment Setup. Throughout our experiments, we keep the algorithm parameters fixed. We set $\epsilon = 0.05$, following [29]. We use IPtrees of size $k = 100,000$ for spam data and $k = 50,000$ for the botnet data, as they make accurate predictions on the input stream, and a further increase in k does not substantially increase the tree's accuracy. We measure the accuracy of our algorithms on a per-IP basis (following [29]), and the accuracy of our constructed IPtrees are similar to [29]. All our change-detection experiments are performed on day-length intervals, i.e., each of the three trees is built, tested and compared across different days. We use three states for the prefixes, split by legitimate-ratio thresholds: $[0, 0.33)$, $[0.33, 0.75)$, and $[0.75, 1]$. We term these states *bad*, *neutral* and *good* states respectively, and this means that a prefix state is assigned as "good" if it sends at least 75% non-malicious traffic, "neutral" if it sends $33\% - 75\%$ non-malicious traffic, and "bad" if it sends less than 33% non-malicious traffic. With the thresholds of the set of states, we derive $\gamma = 33\%$. We set allowable error $\tau = 5\%$ throughout, and the minimum traffic needed $\theta = 0.01\%$ and 0.05%. We chose these values for τ and θ because in our experiments, we are able to obtain a list of Δ-change prefixes that is small enough to be manually analyzed, and yet large enough for us to discover interesting trends across our data sets. Our parameters remain stable throughout our data set when we seek to analyze changes across day-long intervals. As operator resources allow, these parameters can be changed to allow for the discovery of either more fine-grained changes (say, with smaller of θ or larger values of k) or more coarse-grained

[6] While there are bound to be inaccuracies – both false positives and false negatives – in this dataset due to the difficulty of identifying botnets, our results demonstrate that our algorithms are able to highlight those prefixes where significant changes occur as a function of the input data.

changes. Our experiments were run on a on a 2.4GHz Sparc64-VI core. Our current (unoptimized) implementation takes 20-22 minutes to process a day's trace (around 30-35 million IP addresses) and requires less than 2-3 MB of memory storage.

We note that the ground truth in our data provides labels for the individual IP addresses, but does not tell us the prefixes that have changed. Thus, our ground truth allows us to confirm that the learned IPTree has high accuracy, but we cannot directly measure false positive rate and false negative rate of the change-detection algorithms. Thus, our experimental results instead demonstrate that our algorithm can find small changes in prefix behaviour very early on real data, and can do so substantially better than competing approaches. Our operators were previously unaware of most of these Δ-change prefixes, and as a consequence, our summarization makes it easy for operators to both note changes in behaviour of specific entities, as well as observe trends in malicious activity. [7]

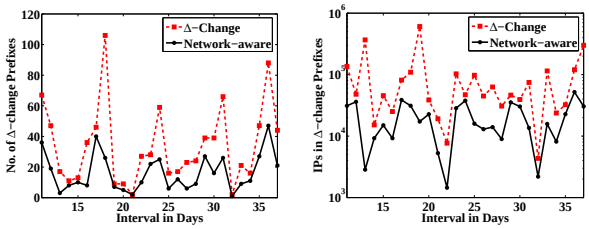
4.1 Comparisons with Alternate Approaches

We first compare Δ-Change with previous approaches and direct extensions to previous work. We compare two different possible alternate approaches with Δ-Change: (1) using a fixed set of network-based prefixes (i.e., network-aware clusters, see Sec. 2.2) instead of a customized IP-Tree, (2) directly differencing the IPTrees instead of using Δ-Change. We focus here on only spam data for space reasons.

Network-aware Clusters. As we described in Section 3.2, our change-detection approach has no false positives – every change we find will indeed be a change in the input data stream. Thus, we only need to demonstrate that Δ-Change finds substantially more Δ-changes than network-aware clusters (i.e., has a lower false negative rate), and therefore, is superior at summarizing changes in malicious activity to the appropriate prefixes for operator attention.

We follow the methodology of [29] for labeling the prefixes of the network-aware clusters optimally (i.e., we choose the labeling that minimizes errors), so that we can test the best possible performance of network-aware clusters against Δ-Change. We do this allowing the network-aware clusters multiple passes over the IP addresses (even though Δ-Change is allowed only a single pass), as detailed in [29]. We then use these clusters in place of the learned IPTree in our change-detection algorithms.

We first compare Δ-change prefixes identified by the network-aware clustering and Δ-Change. This comparison cannot be directly on the prefixes output by the two approaches, as slightly different prefixes may reflect the same underlying change in the data stream, e.g., network-aware clusters might identify a /24 while Δ-Change identifies a /25. In order to account for such differences, we group together prefixes into distinct subtrees, and match a group from the network-aware clustering to the appropriate group from Δ-Change if at least 50% of the volume of changed IPs in network-aware clustering was accounted for in Δ-Change. In our results, network-aware clustering identified no Δ-change prefixes that were *not* identified by Δ-Change; otherwise, we would have do the reverse matching as well. Furthermore, this is what allows us to compare the number of Δ-changes that were identified by both algorithms, otherwise we would not be able to make this comparison.



(a) Δ-change Prefixes (b) IPs in Δ-change prefixes

Figure 11. Comparing Δ-Change algorithm with network-aware clusters on the spam data: Δ-Change always finds more prefixes and covers more IPs

Fig. 11(a) shows the results of our comparison for 37 days. Network-aware clustering typically finds only a small fraction of the Δ-change prefixes discovered by Δ-Change, ranging from $10\% - 50\%$. On average, Δ-Change finds over 2.5 times as many Δ-change prefixes as network-aware clusters. We compare also the number of IPs in Δ-change prefixes identified by the network-aware clustering and Δ-Change in Fig. 11(b). The Δ-change prefixes discovered by Δ-Change typically account for a factor of 3-5$\times$ IP addresses as those discovered by the network-aware clustering. It indicates that network-aware clustering does not discover many changes that involve a substantial volume of the input data. On many days, especially on days with changes, the fraction of IP addresses not identified by network-aware clusters, however, is still smaller than the fraction of prefixes that it does not identify. This indicates that network-aware clustering identifies the larger, coarser changes, but misses the fine-grained changes.

Network-aware clusters perform so poorly because the prefix granularity required to identify Δ-changes typically does not appear at all in routing tables. Indeed, as our analysis in Section 4.2 shows, a large number of Δ-change prefixes come from hosting providers, many of which do not even appear in BGP prefix tables.

Possible Differencing of IPTrees. We now show that the possible differencing approach described in Section 2.2

[7] As discussed in Section 1, our evaluation focuses exclusively on changes in prefix behaviour, since prior work [28, 29] already finds persistent malicious behaviour.

produces an extremely high false positive rate. For this experiment, we learn two trees (denoted T_x and T_y) over two consecutive day-long intervals, i, $i+1$ respectively. We calculate the differing common prefixes in the trees, and then use a basic mathematical argument to prove that there must be a very high false positive rate among these prefixes.

Each tree T_x and T_y has an overall accuracy rate exceeding 95.2% on each of the days i and $i+1$ (we measure this separately across all IPs in each day i and $i+1$). Since each tree makes less than 5% error, the two trees can differ on at most 10% of the IPs on each day i and $i+1$ (e.g., the trees may make errors on disjoint sets of IPs on each day); denote this set of IPs where the trees differ as M. Now, consider the set of prefixes that appear in both trees, and contain at least 0.01% of the data (and discard the redundant parents from this set that account for the same traffic). In order for a prefix to qualify as Δ-change, at least 33% of the IPs it sees must be from the set M. However, by the pigeonhole principle, there can be at most 3400 prefixes can (1) account for at least 0.01% of the IPs, and (2) have at least 33% of their IPs come from the set M. However, when we measured the number of the prefixes present in these two trees that were different, based either on leaf label or on traffic volume for interior nodes (ensuring we discard redundant parents), we found 5021 prefixes present in both T_x and T_y, with at least 0.01% of the traffic. Thus, at least 1621 of the prefixes have to be incorrect, giving a 47% false positive rate.

4.2 Characterization: Spam Data

Summary. We present a summary of Δ-changes discovered in the 2010 spam data, as it covers a longer period (120 days) compared to the 2012 data. Table 1(a) (Fig. 12) summarizes the Δ-change prefixes discovered by Δ-Change, categorized by the kind of behavioral change that they have undergone. The table shows results for different values of the threshold $\theta = 0.05\%, 0.01\%$. As we expect, when θ decreases, the number of prefixes identified as Δ-change increases, since there are more prefixes with at least θ IPs. Note that the majority of the changes come from prefixes that progressively originate more spam, i.e., nearly 75% Δ-change prefixes are Δ-bad. Further, regardless of θ, very few spamming prefixes actually change for the better. These observations are consistent with the earlier studies on spam origin and spammer behavior – while spammers tend to move around the address space, perhaps dependent on the bots they own, legitimate mail servers tend to remain stable. Further, when a region stops spamming, it are much more likely to stop sending mail traffic altogether, rather than start sending substantial volumes of legitimate mail. Since Δ-Change does not detect a prefix that simply stops originating traffic, we see very few Δ-good prefixes in Table 1(a). [8]

Table 2 (Fig. 13) shows the Δ-change prefixes split by access type of the prefix (in this analysis, we include a prefix only once even if it has appeared as a Δ-change prefix multiple times) for $\theta = 0.05\%$. The majority of the Δ-change prefixes come from small ISPs and hosting providers, although there are also a few large (tier-1) ISPs. As Table 1 shows, most of these prefixes are identified because they start to send spam. In Fig. 15(a) we also show the distribution of prefix lengths of the Δ-change prefixes: over 60% of prefixes have lengths between /16 and /26, which matches the prefix ranges expected of hosting providers and small ISPs. Obviously, many of these small ISPs and hosting providers obtain their IP address ranges from large ISPs, but Δ-Change identifies the small ISPs distinctly from their respective owning larger ISP only because their spamming activity differs significantly from the spamming activity of their respective owning larger ISP. DHCP effects also influence the prefixes that are discovered – they force the change in spamming activity to be identified at the granularity of the owning prefix, rather than the individual IP addresses, and this is likely another factor in the predominance of small ISPs and hosting providers as frequent Δ-changes. Indeed, the predominance of small regional ISPs and hosting providers as frequent Δ-changes emphasizes the need for techniques that can automatically infer changed malicious activity – these providers tend to be substantially more volatile and transient than large ISPs, making it much harder to track them with pre-compiled lists.

Case Study 1: Individual Provider Spamming Activity. Fig. 14 illustrates the spamming activity of three different providers that we identified as Δ-bad at $\theta = 0.05\%$. Provider A is a hosting provider (with a /19 prefix) based in south-eastern US, provider B is a virtual web-hosting company in Netherlands (with a /26 prefix), and provider C is a small ISP in mid-western US. (with a /22 prefix). Note that each one of these providers starts and stops spamming *multiple* times over 4 months. Δ-Change identifies all of these changes, as we highlight in Fig 14 with arrows. Further, we note that Δ-Change identifies each Δ-bad prefix early on, *before* their peak spamming activity. None of these three prefixes are detected when BGP prefixes are used, as they are much too small to appear in routing tables. Further, our mail operators were unaware that these specific providers were engaging in spamming activity, and would not have found them without exhaustive manual analysis.

These three providers are just examples of the many that were not detected by BGP prefixes and of which our operators were previously unaware. [9] We highlighted these to il-

[8] Note also the design of TrackIPTree ensures that such prefixes eventually get discarded from IPtree, and thus after a period of time, these prefixes will not be labeled malicious in the tree forever.

[9] Maintaining a list of hosting providers and using the list to track their

Original State	New State	$\theta = 0.01\%$	$\theta = 0.05\%$
Bad	Good	31	11
	Neutral	28	1
Good	Neutral	122	24
	Bad	205	33
Neutral	Good	66	9
	Bad	146	13

Table 1(a) Spam Data Set

Original State	New State	$\theta = 0.01\%$	$\theta = 0.05\%$
Bad	Good	134	23
	Neutral	189	16
Good	Neutral	42	17
	Bad	78	14
Neutral	Good	201	98
	Bad	285	43

Table 1(b) Botnet Data Set

Figure 12. Characterizing the Δ-change prefixes discovered for spam and botnet data sets.

ISP Type	# Identified
Large ISPs	4
Small ISPs	11
Hosting Providers	9
Others	2

Figure 13. Table 2: Spam Data: ISP Types of Δ-change prefixes

lustrate spamming activity from these smaller providers that repeatedly starts and stops. Our case study also illustrates how difficult it is to ensure that systems are configured to never spam, especially for hosting providers, since hosting providers typically allow their customers to easily establish new mail servers on their physical or virtual infrastructure, and can repeatedly get caught into a cycle of accidentally hosting spammers and cleaning up spamming activity.

Case Study 2: Drop in Internet-wide Spamming Activity. In our next case study, we examine the Δ-good prefixes discovered by Δ-Change during the Grum botnet takedown in July 2012. The Grum botnet was considered the third largest spamming botnet and responsible for around 17% of all the spam on the Internet. [12]. This case study illustrates what an operator would see with the Δ-Change algorithm during such a large event, with no a priori knowledge that the event was happening.

Figure 15(b) shows the number of Δ-good prefixes discovered each day by Δ-Change and network-aware clusters, and the start of the botnet takedown is indicated (with an arrow). (As in Sec. 4.1, we count only Δ-good prefixes that correspond to distinct regions of the address space, in order to have a fair comparison between Δ-Change and the network aware clusters.) Our first observation is that there is sudden increase in the number of Δ-good prefixes right after the botnet takedown, showing that a number of prefixes have suddenly changed their spamming activity. The number of Δ-good prefixes discovered every day remains high for a number of days after the takedown – this happens because our algorithm discovers prefixes as Δ-changes when they actively generate traffic (e.g., by sending legitimate mail instead of spam in this case). Thus, whenever a

spamming activity would be less effective, since hosting providers start and shut down frequently.

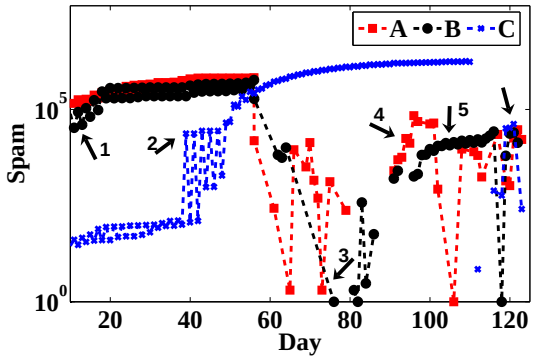


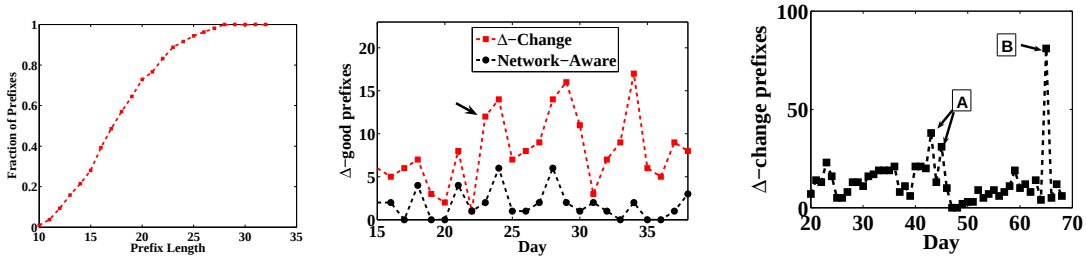
Figure 14. Case Study 1: Spamming Activity in Small Providers A, B, C. Δ-Change discovers spamming activity early in small ISPs and hosting providers (arrows indicate every time the prefixes are discovered).

(previously) infected region become active after the botnet takedown, its prefix blocks are identified as Δ-good.

We also observe that Δ-Change discovers far more Δ-good prefixes than the network-aware clusters (anywhere between a factor of 3-10). Further analysis showed that these prefixes had previously sent $0.01\% - 0.1\%$ of the daily spam volume in our data, and a few of them contained over two thousand spamming IP addresses. Most of these prefixes range are allocated to small regional ISPs (ranging from $/15$ to $/26$), and many of them do not appear in BGP routing tables, and so they cannot be detected with network-aware clusters. Thus, Δ-Change highlights to operators where on the Internet a drop in spamming activity took place.

4.3 Characterization: Botnet Data

Next, we examine the results of Δ-Change on the botnet data. Recall that our data only identifies botnet activity within a single large tier-1 ISP, and thus, Δ-change only detects changes internal to this ISP. This is especially useful since large ISPs often allocate prefix blocks to many smaller ISPs and other customers, many of which typically are are managed independently and change over time as business requirements change, and thus are likely to have very different security properties. In this scenario, Δ-Change was useful for highlighting to the operators a network-wide view



(a) Sizes of Δ-change Prefixes (b) Case Study 2: Drop in Internet-wide Spamming Activity (c) Case Study 3: New Botnet Activity

Figure 15. (a) shows sizes of Δ-change prefixes. (b) shows Case Study 2: Δ-good prefixes with drop in spamming activity during the Grum takedown (arrow indicates when the takedown started). There is a sharp increase in Δ-good prefixes after the takedown. (c) shows Case Study 3: Δ-change Prefixes in New Botnet Activity. A and B mark the Δ-bad prefixes discovered when over 22,000-36,000 new bot IPs appeared in the feed.

of the changing malicious activity, since there is likely to be a diversity of malicious activity when new threats emerge.

Summary. Table 1(b) (Fig. 12) summarizes the different prefixes for $\theta = 0.05\%, 0.01\%$, categorized by the type of change they have undergone. As in Section 4.2, the prefixes discovered increases sharply when θ is increased. However, note that in this experiment, there are very significant numbers of Δ-good prefixes discovered as well – over 56% of all the prefixes discovered are Δ-good, unlike the spam data. This is primarily because the active IP address space changes very little, while bot IP addresses appear in the feed for much shorter durations (e.g., this may be as bots get cleaned, or bot signatures get outdated). A former bot IP would then generate mostly legitimate traffic (its malicious traffic would drop, but its legitimate activity remains the same, and so it would get labelled as legitimate), and the corresponding IP regions thus become Δ-good.

Case Study 3: New Botnet Activity. Our case study illustrates the value of discovering Δ-bad prefixes internal to a large ISP's prefix blocks. Figure 15(c) shows the time-series of the Δ-change prefixes discovered over two months of our data set. The highlighted days (A and B) mark two sharp increases in the number of Δ-change prefixes discovered. These correspond to days with dramatic increases in the number of new bot IPs seen in the data feed – 22.1 & 28.6 thousand at the two days marked as A and 36.8 thousand at B Further analysis showed that on days marked A, nearly all of of these new bot IPs are from the DNSChanger botnet [8], and are responsible for 19 & 31 Δ-bad prefixes. On day B, these new bot IPs are from Sality [25] and Conficker [6], and 66 Δ-bad prefixes correspond to the new IPs from Sality and Conficker. By contrast, network-aware clusters were only able to discover 5-12 prefix blocks as Δ-bad during these events. These Δ-bad prefixes come from smaller regional ISPs, the tier-1 ISP's dial-up and DSL blocks; most of these prefixes had little to botnet activity (as identified by the vendor) earlier. Thus, in these two instances, Δ-Change effectively reduces the workload for operator from manually investigating over 22,000-36,000 new bot IPs to investigating 19-66 new IP prefixes, a drop of two orders of magnitude.

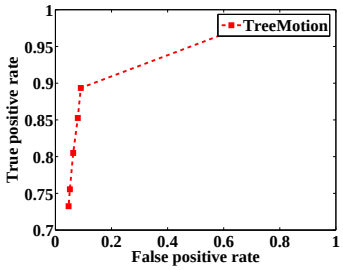


Figure 16. ROC curve for Δ-Motion's accuracy

4.4 Structural Analysis of IP Dynamics

Our earlier results demonstrate that there is constant change in the Internet's malicious activity. We now explore the structure underlying these changes with the Δ-Motion algorithm, focusing our analysis on spam dataset due to space. We use a snapshot of the change-IPtree W generated by Δ-Motion, 60 days into the dataset; W's high predictive accuracy indicates it can distinguish frequently-changing regions well, as shown by the ROC curve in Fig. 16. We use W to classify every IP in our data set as "change" or "non-change", and then aggregate the IPs by country and owning company. We define *freq-ratio* to be the fraction of the total IPs of that entity that are marked as change IPs, and analyze the freq-ratio of different aggregations.

Table 3 (Fig. 17) shows a breakdown for the origin of the frequently changing IPs. Together, these countries account for 90% of the data seen at our mail servers. We note that countries like China, Korea, Russia [23], which are known to harbor lot of spammers actually change very infrequently, while countries like US and Canada change 3-4 times more frequently. This makes sense, as countries where ISPs aggressively fight spammer infestations are likely to experience a more frequent change in malicious activity. Table 4 shows a breakdown by ISP type. Once again, hosting providers have a substantially higher ratio than the other cat-

Country	freq-ratio
USA	6.9%
W. Europe	2.6%
Brazil	0.8%
Canada	9.1%
Russia	2.2%
Estonia	1.1%
Poland	1.5%
Argentina	3.9%
Korea	1.1%
Colombia	3.4%
China	2.3%

Table 3: Country

ISP Type	freq-ratio
Large ISPs	6.6%
Small ISPs	4.9%
Hosting Providers	12.2%
Others	1.1%

Table 4: ISP type

Figure 17. Analyzing the IPtree learnt by Δ-Motion: the tables show frequently changing regions

egories, consistent with our results in Section 4.2, since it is much easier to spam out of a hosting provider. We see both large and small ISPs seem to have roughly the same frequency of change, and that businesses (which constitute the most of the "other" category) have a tiny ratio, as expected.

The set of hosting providers discovered by Δ-Motion (which are the same as those that Δ-Change identifies repeatedly as Δ-bad prefixes) are of particular interest to mail operators. As discussed in Section 4.2, hosting providers are especially vulnerable to changes because they see a wide variety of users, who sometimes take any opportunity to spam. However, because these providers also have many legitimate clients, they cannot be entirely blacklisted, and therefore need to be closely monitored so that they do not cause a significant performance impact. Indeed, this is likely true of all new hosting providers as they appear on the market, and it is this kind of structural insight about malicious activity that Δ-Motion could discover, which may help operators prioritize their resources.

5 Related Work

Spam. There has recently been a lot of interest in designing non-content based approaches to spam-filtering. Of these, most closely related to our work are the IP-based spam filtering approaches. These have included studies on individual IP addresses, AS numbers and /24 prefixes [23], BGP prefixes [27, 30], prefixes with dynamic IP assignment [31], highly predictive blacklists [33], using a combination of DNS clusters and BGP prefixes [22], and using well-defined properties of spammers to discover IP address ranges used by spam gangs [9]. Our work differs from all of these as we are concerned with automatically discovering the prefixes that change their malicious behavior, using only a stream of IP addresses labelled spammer or legitimate; we do not use a priori fixed clusters that originating from network-based properties. There have also been behavior-based spam filtering approaches [13, 24], and analysis and identification of spam campaigns [2,18] and spamming botnets [15, 32]; these take a very different angle, complementary to ours, for analyzing shifting malicious activity. Lastly, there have been a number of studies showing the relative inaccuracy of DNS-based blacklists [16, 26] Again, our results are complementary to (and consistent with) all these analyses, as we show that even with a near-optimal partitioning of the IP address space, there are still a large number of changes in spamming behavior.

Other Related Work. Xie et al [31] consider the problem of discovering IP addresses that are dynamically assigned. Our problem is different from this work, as we are interested in dynamic of malicious activity, not of IP address assignment. Soldo et al. [28] study the problem of filtering malicious activity but their algorithms only operate on offline data, not streaming data. Finally, note also that our problem differs from work on identifying hierarchical heavy-hitters [7, 10, 34], and discovering significant changes in the multi-dimensional aggregates [1, 4, 14, 17]: these problems are concerned with volumetric changes on a hierarchy, not on changes in classification of decision tree.

6 Conclusion

In this paper, we formulated and addressed the problem of discovering changes in malicious activity across the Internet. Our evaluations using a large corpus of mail data and botnet activity indicate that our algorithms are fast, can keep up with Internet scale traffic data, and can extract changes in sources of spam activity substantially better (a factor of 2.5) than approaches based on using predetermined levels of aggregation such as BGP-based network-aware clusters. Using our algorithms, we find that some regions of the Internet are prone to much faster changes than others, such as a set of hosting providers that are of particular interest to mail operators.

References

[1] D. Agarwal, D. Barman, D. Gunopulous, F. Korn, D. Srivastava, and N. Young. Efficient and effective explanations of change in hierarchical summaries. In *Proceedings of KDD'07*, 2007.

[2] D. Anderson, C. Fleizach, S. Savage, and G. Voelker. Spamscatter: Characterizing the internet scam hosting infrastructure. In *Proceedings of Usenix Security '07*, 2007.

[3] B. Augustin, B. Krishnamurthy, and W. Willinger. Ixps: mapped? In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, 2009.

[4] D. Barman, F. Korn, D. Srivastava, D. Gunopulos, N. Yong, and D. Agarwal. Parsimonious explanations of change in hierarchical data. In *Proceedings of ICDE 2007*, 2007.

[5] M. P. Collins, T. J. Shimeall, S. Faber, J. Naies, R. Weaver, and M. D. Shon. Using uncleanliness to predict future botnet addresses. In *Proceedings of the Internet Measurement Conference*, 2007.

[6] http://www.confickerworkinggroup.org/wiki/.

[7] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Diamond in the rough: finding hierarchical heavy hitters in multi-dimensional data. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 155–166, New York, NY, USA, 2004. ACM.

[8] http://www.fbi.gov/news/stories/2011/november/malware_110911/.

[9] H. Esquivel, T. Mori, A. Akella, and A. Mutapcic. On the effectiveness of IP reputation for spam filtering. In *Proceedings of COMSNETS'10*, 2010.

[10] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *SIGCOMM*, 2003.

[11] Y. Freund, R. E. Schapire, Y. Singer, and M. K. Warmuth. Using and combining predictors that specialize. In *STOC*, 1997.

[12] http://blog.fireeye.com/research/2012/07/grum-botnet-no-longer-safe-havens.html.

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

[14] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of KDD'01*, 2001.

[15] J. P. John, A. Moshchuk, S. Gribble, and A. Krishnamurthy. Studying spamming botnets using botlab. In *Proceedings of NSDI '09*, 2009.

[16] J. Jung and E. Sit. An empirical study of spam traffic and the use of DNS black lists. In *Proceedings of Internet Measurement Conference (IMC)*, 2004.

[17] D. Kifer, S. Ben-David, and J. Gehrke. Detecting changes in data streams. In *Proceedings of VLDB 2004*, 2004.

[18] M. Konte, J. Jung, and N. Feamster. Dynamics of online scam hosting infrastructure. In *Proceedings of PAM '09*, 2009.

[19] B. Krishnamurthy and J. Wang. On network-aware clustering of web clients. In *Proceedings of ACM SIGCOMM*, 2000.

[20] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–251, 1994.

[21] Z. M. Mao, V. Sekar, O. Spatscheck, J. van der Merwe, and R. Vasudevan. Analyzing large ddos attacks using multiple data sources. In *ACM SIGCOMM Workshop on Large Scale Attack Defense*, 2006.

[22] Z. Qian, Z. Mao, Y. Xie, and F. Yu. On network-level clusters for spam detection. In *Proceedings of NDSS 2010*, 2010.

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

[24] A. Ramachandran, N. Feamster, and S. Vempala. Filtering spam with behavioral blacklisting. In *Proceedings of ACM CCS*, 2007.

[25] http://www.symantec.com/security_response/writeup.jsp?docid=2006-011714-3948-99.

[26] S. Sinha, M. Bailey, and F. Jahanian. Shades of grey: On the effectiveness of reputation-based "blacklists". In *Proceedings of Malware 2008*, 2008.

[27] S. Sinha, M. Bailey, and F. Jahanian. Improving spam blacklisting through dynamic thresholding and speculative aggregation. In *Proceedings of NDSS 2010*, 2010.

[28] F. Soldo, A. Markopoulo, and K. Argyraki. Optimal filtering of source address prefixes: Models and algorithms. In *INFOCOM*, 2009.

[29] S. Venkataraman, A. Blum, D. Song, S. Sen, and O. Spatscheck. Tracking dynamic sources of malicious activity at internet-scale. In *NIPS*, 2009.

[30] S. Venkataraman, S. Sen, O. Spatscheck, P. Haffner, and D. Song. Exploiting network structure for proactive spam mitigation. In *Proceedings of Usenix Security'07*, 2007.

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

[32] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming botnets: signatures and characteristics. In *Proceedings of SIGCOMM 2008*, 2008.

[33] J. Zhang, P. Porras, and J. Ulrich. Highly predictive blacklists. In *Proceedings of Usenix Security'08*, 2008.

[34] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund. Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 101–114, New York, NY, USA, 2004. ACM.