Buddyguard: A Buddy System for Fast and Reliable Detection of IP Prefix Anomalies

Jun Li, Toby Ehrenkranz, Paul Elliott
University of Oregon, Eugene, OR, USA
Email: {lijun, tehrenkr, pelliott}@cs.uoregon.edu

Abstract*—Due to operational malpractice or security attacks, an IP prefix (i.e., a block of IP addresses) can undergo many types of routing anomalies. Perhaps the most well-known of such anomalies is prefix hijacking, where an attacker hijacks traffic meant to reach the legitimate user of a prefix. Anomalies can also easily occur through route leaks, which can disrupt traffic for numerous prefixes at once. While various solutions have been proposed to detect such anomalies, these solutions are limited and susceptible to attacker countermeasures. In this paper we present Buddyguard, a new approach to detecting prefix anomalies including prefix hijacking and route leaks. Buddyguard compares the behavior of a monitored prefix with the behavior of a set of numerous buddy prefixes. The system detects anomalies when the behavior of the monitored prefix significantly diverges from that of its buddies. Our evaluation results show that Buddyguard provides fast, accurate and lightweight monitoring of IP prefix anomalies, and its introduction and use of buddy prefixes enables it to be resilient against resourceful attackers.

I. Introduction

A. Routing Anomalies with an IP Prefix

An **IP prefix** (i.e., a block of IP addresses) can be subjected to many types of routing anomalies. A prefix may suddenly become unreachable, reachable only through a path with poor routing performance, or it may experience pathological routing dynamics (e.g., oscillation between different paths). Whether a prefix is used by major online businesses (such as Google or YouTube) or ordinary end users (e.g., Alice and Bob), these prefix anomalies can cause loss of revenue, identity theft, or many other devastating consequences.

One of the most infamous of such anomalies is **prefix hijacking**, in which an attacker hijacks traffic meant to reach the legitimate user of a prefix. Real world cases of prefix hijacking have occurred repeatedly ([1], [2]), including the well-known Pakistan Telecom hijack of YouTube in 2008 [3]. These events have been reported as recently as January 2011, when INDOSAT-INP-AP hijacked traffic to nearly 3000 prefixes [4]. Furthermore, executing this kind of attack is not difficult. Theoretical studies show that a prefix can be hijacked by a tier 1 autonomous system (AS) with around a 50–80% probability, a tier 2 AS with around a 30–70% probability, and a tier 3 AS with around a 5–30% probability [5].

This material is based upon work supported by the USA National Science Foundation under Grant No. CNS-1118101. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Another common prefix anomaly are **route leaks**, where a misconfigured ISP advertises illegitimate routes for prefixes. In past history, such incidents have caused anomalies on a monumental scale [6], [7]. A well-publicized instance occurred on April 8, 2010, when an AS operated by China Telecom falsely originated nearly 37,000 prefixes, and mis-routed traffic to those prefixes for about 15 minutes [8]. Route leaks also continue to plague today's Internet, with events occurring as recently as February 2012 when Australia's Telstra ISP caused outages for nearly 1400 prefixes [9].

These anomalies are a real threat and *every* prefix on the Internet, whether commercial or private, is vulnerable. Even more disturbing, users or operators of a prefix cannot easily detect such incidents. The legitimate user of a prefix may not expect any incoming traffic at all while its traffic is being mis-routed; or in a more complex form of IP prefix hijacking called *prefix interception*, an attacker not only hijacks traffic, but also forwards that traffic to the victim—leaving the victim entirely unaware that its traffic is being hijacked.

Unfortunately, it is unlikely that these prefix anomalies will be resolved in the near future. While there exist proposals such as S-BGP [10] to secure Border Gateway Protocol (BGP), the *de facto* inter-domain routing protocol, their high overhead cost has prevented actual deployment. In fact, research has found that even these proposals are deployed, they will still fail in certain circumstances [11]. It is therefore critical to monitor prefixes and detect prefix anomalies as they occur.

But in the domain of monitoring prefixes and detecting anomalies, the outlook is still bleak. The current state-of-the-art monitoring schemes ([12], [13], [14], [15]) are narrowly focused on prefix hijacking alone, leaving other forms of prefix anomalies undetected. Furthermore, as we will detail in section II, a fundamental limitation with the current systems is that none of them fully address the range of countermeasures that an intelligent attacker could employ to avoid detection. For example, a hijacker might bypass monitoring schemes by performing *sub-prefix hijacking*, leaving the monitored prefix unaffected but hijacking traffic to its subspace. Even reference point monitoring approaches ([13], [14]), which address many deficiencies in earlier designs, are still vulnerable to resourceful attackers. Given these issues, there continues to be a pressing need for more flexible and resilient solutions than what are currently available.

B. Buddyguard: A Buddy-Based Prefix Monitoring Solution

In this work, we present a new approach to prefix monitoring that fills this missing gap. Dubbed **Buddyguard**, it surrounds a prefix with a buddy system composed of buddy prefixes, or **buddies**, and monitors the behavior of the prefix against that of its buddies. Not only does Buddyguard quickly and accurately detect various prefix anomalies including prefix hijacking and route leaks, but it is also lightweight to deploy and resilient against circumvention by attackers.

Key to monitoring an IP prefix is knowing what is normal behavior and what is not, and a buddy system makes this task feasible. When inspecting a prefix in isolation, it is difficult to know what behaviors are abnormal. For example, when the path to a prefix from a vantage point suddenly disappears or changes, it can be either a normal routing change, or an abnormal misconfiguration, or that an attacker has just misled routers to adopt a new path under the control of the attacker. In contrast, a buddy system provides a more reliable basis to determine if anything is abnormal with a prefix. By ensuring that there are enough buddies for the prefix, and under normal conditions a prefix is similar to most of its buddies (in terms of the behaviors being monitored) but no so if under abnormal situations, we can use these buddies to determine whether or not the prefix is experiencing anomalies.

This methodology has the following advantages:

(i) *It is flexible and extensible.* No matter what anomalous behavior of a prefix we want to monitor and detect, we can always first determine the type of behavior and how to measure it, and then select buddies in terms of that behavior for monitoring.

(ii) *It is resilient.* We depart from existent approaches by emphasizing the need to be resilient against attacker countermeasures. A key feature of Buddyguard is that a prefix is allowed to have *hundreds* or even *thousands* of buddies. As our results indicate, our system is capable of finding buddies from multiple different ASes, making it difficult to locate and simultaneously attack enough buddy prefixes to circumvent Buddyguard. Even if the attacker is successful in doing so, an attempt to attack that many prefixes at once would itself appear suspicious.

(iii) *It is scalable and easy to deploy.* Buddyguard requires only passive measurement using existing BGP collection systems and its input are publicly available routing data. Our results show that our system maintains very little overhead, so little that it may scale to monitor hundreds or even thousands of prefixes simultaneously.

We demonstrate the efficacy of Buddyguard by testing our system on well-known prefix hijacks and route leaks. For these anomalies, the behavior in question is the routing paths to a given monitored prefix. With monitors being BGP speakers that peer with RouteViews [16] collectors, we train Buddyguard by observing routes from these monitors to the prefix and select buddies that best match these routes. Our evaluations show that monitoring prefixes with these buddies provides fast, accurate, and reliable detection with low false negatives and false positives.

In addition to the introduction and use of a buddy system for prefix anomaly detection, the major contributions of this work further include:

(i) A training algorithm for finding and selecting well-matched buddies from multiple ASes for a given prefix;

(ii) A buddy-based monitoring algorithm that provides fast, accurate, and reliable detection of prefix anomalies; and

(iii) A resilient and scalable system design that could easily be deployed for today's Internet.

The rest of the paper is organized as follows. In Section II we highlight the background and related work for this research. We then describe our design of Buddyguard in Section III and its resiliency to attacker countermeasures in Section IV. In Section V we detail how we evaluate Buddyguard through monitoring prefix hijacks and route leaks. In Section VI we discuss our results and future work, and present our conclusions in Section VII.

II. Related Work

A number of works have dealt with understanding BGP dynamics [17], [18], [19], [20], [21], as well as detecting BGP anomalies [22], [23], [24] and monitoring BGP in general [16], [25], [26], [27]. In this section, we focus on the limitations of current prefix monitoring systems, one of the primary motivations of our work. Although various approaches have been proposed to monitor routing behaviors of IP prefixes, the best we have are some solutions for detecting prefix hijacking, and even these solutions are very limited:

(i) They each can only detect certain prefix hijacking cases;

(ii) Intelligent attackers can circumvent them; and

(iii) These solutions are specific to prefix hijacking and cannot be easily extended to address other prefix anomalies.

Such limitations are in large part due to the underestimation or inadequate modeling of what prefix hijackers can do. Early solutions monitored prefix origin changes to detect prefix hijacking [22], [28], assuming that an attacker must claim itself as the new origin of a victim prefix in order to hijack it. But an attacker can hijack a prefix by merely stating it is *close* to the real origin of the victim prefix, invalidating this assumption. Later solutions recognized this fact, but they required the owner of a prefix to verify the paths to its prefix, putting a heavy burden on human users [29], [12]. Recent solutions set up monitors, probe them from a monitored prefix, and watch and analyze responses to determine if the prefix is hijacked [15]. However, in the case of prefix interception, the prefix will simply receive responses as usual.

An alternative monitoring solution, *reference point comparison*, addresses several of these deficiencies. This approach is evidenced in [5] and [13], as well as the current leading approach described in [14]. To detect whether a prefix is hijacked, it uses monitors distributed throughout the Internet to check whether each monitor's route to the prefix deviates significantly from its route to a topologically nearby *reference point*. This system has many advantages; it is both lightweight and capable of detecting prefix interception, since the prefix's route will still deviate from the reference point's route.

However, even these solutions fail to address a fundamental issue: the ability of prefix hijackers to circumvent defenses. An attacker can discover which IP prefix(es) likely contain the IP address of the reference point, and hijack these prefixes and the monitored prefix altogether, causing the hijack to go undetected. In particular, if the reference point shares the same origin AS as the monitored prefix, hijacking both in one fell swoop is trivial. Indeed, following the reference point selection process in [14], we found it very common for a monitor to use such a reference point. More specifically, using traceroute data from the iPlane project [30] for about 91,019 prefix atoms, where every **atom** is a group of prefixes that are reachable through the same routes from all locations [31], we found 54.6% of atoms share the same origin with their reference point, while 31.8% of atoms do so from *all* monitors and 45.5% of atoms do so from 90% or more of the monitors. Most current solutions are also susceptible to sub-prefix hijacking, where a hijacker advertises an invalid path to a *subspace* of a prefix to hijack only that subspace. Even if routers maintain a correct route to the prefix, due to the preference for more specific routes, they will adopt this invalid route to the subspace.

Lastly, if we inspect how these hijack detection approaches may translate to monitoring other prefix routing anomalies, the situation is still not optimistic. On one hand, due to the scale and complexity of Internet routing, the cause of routing anomalies is often very complicated and their symptoms are rarely predictable. Most of the time, network administrators have to handle them on an *ad hoc*, case by case basis—*if* an anomaly is even noticed or reported. Rather than focusing on specific cases of routing anomalies such as prefix hijacking or interception, a prefix monitoring system must be extensible enough to cover any prefix anomaly, both known and unknown. Given that the above approaches fail to do so, there still exists a significant gap between the state of the art and a truly effective prefix monitoring solution.

III. Design of Buddyguard

In this section, we present our design of Buddyguard, a control-plane prefix monitoring system that addresses the shortcomings of existing systems described in the previous section. We introduce the general architecture, cover our strategy for finding and selecting buddies, define how Buddyguard performs monitoring, and explain how the system is maintained. While the introductory use of a buddy system allows Buddyguard to handle any prefix anomaly, we use prefix hijacking in particular as means of illustrating how the system works.

A. System Overview

At its core, the Buddyguard architecture includes the monitored prefix, a set of **monitors**, and the **buddies** of every monitored prefix (Figure 1). The monitored prefix is any IP prefix whose owner requests the Buddyguard service for specific types of anomalies affecting that prefix. Buddyguard is able to monitor multiple prefixes in parallel (as we demonstrate in our evaluations, up to hundreds or thousands of prefixes at once). We define the remaining components, monitors and buddies, as follows.

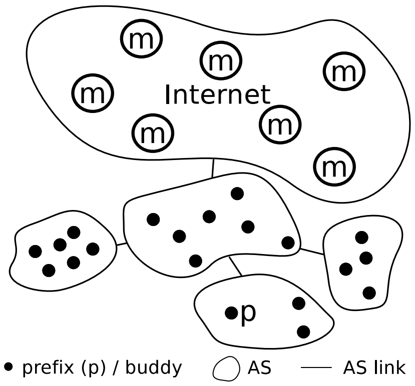


Fig. 1. **Buddyguard architecture.**

1) Monitors: A monitor is defined as a networked entity that can observe a prefix and its buddies in conjunction. Under normal conditions, the observed behavior of the monitored prefix and its buddies should match. Monitors detect anomalous conditions when the behavior of a prefix deviates significantly from that of its buddies. We leave the details of the monitoring algorithm for a later section.

What behaviors should monitors observe and compare? Since every prefix anomaly we are concerned with is within the domain of BGP, such behaviors should be the properties of any BGP operation related to a monitored prefix. In the case of prefix hijacking and route leaks, the related BGP operation is the announcement of a new path to that prefix, and the main property we inspect is the difference between this new path and the paths to the prefix's buddies. Different anomalies will have different behaviors, and therefore require a different set of buddies for the monitored prefix.

Since monitors must be able to measure these BGP operations, monitor placement is critical. Ideally, monitors must be able to hear conversations between BGP routers as close to real-time as possible. One solution involves peering monitors with existing BGP data collection systems such as RouteViews and RIPE [25] collectors or BGPMon [32], which collect real-time BGP updates from routers around the globe. This deployment scheme has the advantage of costing low overhead, as it does not require continuous data-plane queries like other solutions. We return to the efficacy of this monitor placement strategy in our discussion section.

2) Buddies: A buddy can be defined as IP prefix that behaves similarly to the monitored prefix under normal conditions, and diverges when anomalous conditions occur. Recall that for prefix hijacking, the behaviors we are concerned with are path updates associated with the prefix and its buddies. To detect whether a prefix is hijacked, we compare paths to a buddy b and a monitored prefix p such that:

(i) Under normal circumstances, the path from a monitor to b is similar to the path from that monitor to p;

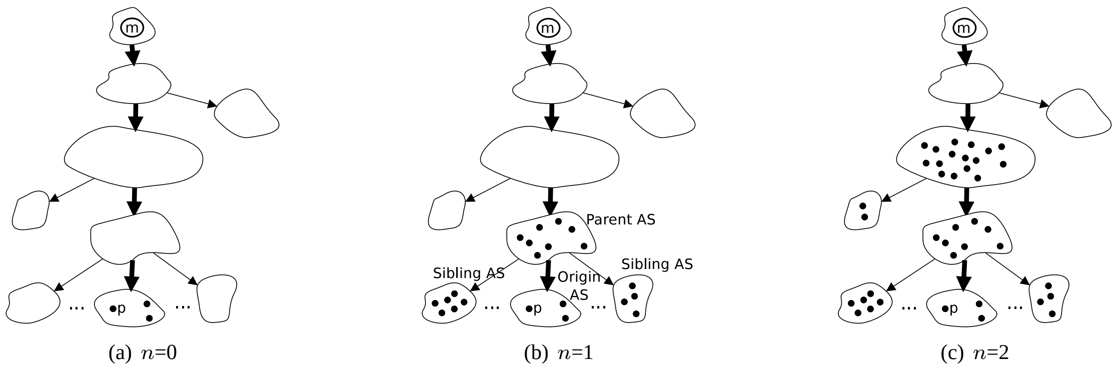


Fig. 2. **Finding buddies using path similarity principle.** n determines the number of shared AS hops between the two paths being compared.

(ii) If a legitimate routing change occurs so that the monitor has a new path to p, the monitor will also have a similar new path to b; and

(iii) If p is hijacked, the monitor will switch to a "bad" new path to p, but will still use the old path to b, causing the two paths to be dissimilar.

Clearly, if b perfectly meets these standards then p will only need that single buddy. However, in many situations buddies can only partially meet the above conditions. Sometimes a buddy may experience the same anomaly as the monitored prefix; for example, a hijacker could co-hijack a prefix and its buddy, leaving the anomaly undetected. Therefore, having one buddy for a prefix is typically not sufficient.

To solve this, we must obtain many buddies for a monitored prefix, where enough buddies are similar to the prefix when it is behaving normally, and at most a small number of buddies may experience the same anomaly together with the monitored prefix. Therefore, if a prefix deviates from enough of its buddies, we can determine that it is behaving abnormally. When detecting if a prefix is hijacked, for example, we modify the conditions (i)–(iii) above to:

(i) Each monitor m must have a set $B_m = \{b\}$ of buddies such that m will have similar paths to all of them, including p;

(ii) If a legitimate routing change occurs so that m has a new path to p, *enough* of its buddies will also switch to a similar new path from m; and

(iii) If the p is hijacked, m will switch to a "bad" new path to the p, but *enough* of p's buddies will not switch.

We leave the definition of *enough* for a later section.

Due to the decentralized nature of BGP, it is likely (but not necessary) that each monitor will have a distinct set of buddies for the monitored prefix. The specific location of a monitor will determine which BGP updates it is able to hear; indeed, certain updates may never reach a given monitor at all. The advantage of a per-monitor buddy system over a common buddy system (where a prefix has the same buddies for all monitors) is that each monitor need only be concerned with the BGP updates to which it is privy.

To build this architecture, we must employ three major classes of algorithms: training algorithms for discovering and selecting buddies for a monitored prefix, a buddy-based monitoring algorithm for detecting prefix anomalies, and a maintenance algorithm to ensure that a prefix always has good buddies. Without losing generality, we describe each with respect to prefix hijacking in the following sections.

B. Training: Finding and Selecting Good Buddies

The success of Buddyguard lies in having the best possible set of buddies for a given monitored prefix. To meet this objective, we define algorithms for finding buddy candidates that match the behavior of the monitored prefix and selecting the best matching candidates to be actual buddies. We call this bootstrapping phase: **training**.

1) Finding Good Buddy Candidates: Where should Buddyguard look for buddy candidates for a given monitored prefix? Ideally, we want buddies to be distributed across multiple ASes, making it difficult for a hijacker to co-hijack enough buddies to evade detection. Yet buddies must be well-matched in order for subsequent monitoring to be accurate. Therefore, our task becomes finding well-matching buddy candidates from a diverse set of ASes.

We can achieve this by using a simple path similarity principle. Consider the AS path update $u_p(t_i)$ from a monitor to a prefix p, which consists of an ordered list of autonomous systems (ASes) from the monitor to p. We can define an update u_i as similar to u_p ($u_i \sim u_p$) if both share the first $|u_p| - n$ AS hop. For definition purposes, we use $|x|$ to designate the length or size of x both here and in the remainder of this work. Figure 2 shows how $0 \leq n \leq 2$ may affect the distribution of buddy candidates, or which ASes may be eligible to offer buddies. Clearly, when n is 0, buddies can only be from the same AS (the origin AS) as the monitored prefix (Figure 2(a)). But when n is 1, buddies can be also from the so-called parent and sibling ASes (Figure 2(b)), and so on.

Using this notion of path similarity, we can find well-matching buddy candidates for a given prefix p through observation. Buddyguard observes p over a training period, during which each monitor listens for AS path updates regarding p. For a given monitor m and an AS path update $u_p(t_i)$ to p witnessed at time t_i, m checks for similar path updates $u_c(t_i \pm \Delta)$ to any candidate c that occur at roughly t_i. We

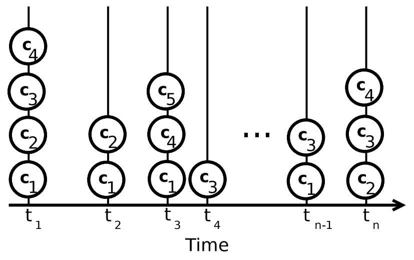


Fig. 3. **Buddy selection through the skewer mechanism.** A skewer represents a path change for the monitored prefix at time t_i. Each circle represents a buddy candidate c_j.

specify $t_i \pm \Delta$ to allow time for BGP convergence [33], and use $\Delta = 3\ minutes$ as a conservative measure for this work. After this period, each monitor m will have set $C_m = \{c\}$ of buddy candidates that matched paths from m to p.

2) Buddy Selection: The most frequently matching candidates found during this training period are clearly the best-matching, but how can Buddyguard select buddies such that (1) enough buddies always match the monitored prefix, and (2) the buddies are distributed across multiple ASes, and therefore resilient to co-hijacking? For this task, we employ a **skewering mechanism**. When monitor m hears a path update $u_p(t_i)$, Buddyguard creates a **skewer** data structure for time t_i. By the end of training, we have a set of skewers $S_m = \{s_{t_i}\}$ for every $u_p(t_i)$ that m witnessed (Figure 3). We then "skewer" candidates by sorting them by frequency of matching (best to worst), and place the best-matching candidate c on each skewer s_{t_i} where $u_p(t_i) \sim u_c(t_i \pm \Delta)$.

The skewering mechanism enables Buddyguard to select buddies according to the above criteria. We continue to skewer candidates until all of the skewers are full, or more formally:

$$\forall s_{t_i} \in S_m,\ |s_{t_i}| \geq \omega$$

for some lower bound capacity ω (we will wait to define ω until section V-B). This ensures that buddies can account for the full range of normal behavior for the monitored prefix. We can also ensure that buddies are widely distributed by skewering candidates until they cover multiple ASes. Once these conditions are met, Buddyguard selects all skewered candidates as buddies. The training methodology described thus far can be summed up in Figure 4.

C. Monitoring: Detecting Prefix Anomalies

Buddyguard provides accurate and fast detection of prefix anomalies using the buddies selected from training. The detection procedure is straightforward—again, using prefix hijacking as an example. If a monitor hears a new path to a prefix p, it will check whether this is one of the "good" paths that is has already seen. If not, the monitor will wait for a short period, and then check if more than $\alpha/|B_m|$ (for some value of α) of p's buddies also switch to a similar new path. If so, the monitor can record that path as a good path for future reference. If not, the monitor raises a warning flag, indicating that the prefix may be hijacked. In this way, buddies help the system determine both normal behavior and anomalous events. Warning flags can be lowered in two ways: (1) more than $\alpha/|B_m|$ of the buddies eventually switch over to the new path, or (2) p later switches to a good path.

Defining this per-monitor warning threshold α would seem to be a difficult task. How can Buddyguard know how many buddies will typically match the monitored prefix? The answer lies in using data from training, and here again the skewering mechanism becomes exceedingly useful. Consider a set of skewers S_m from training, which correspond to paths from monitor m to p. If buddies are normally distributed across these skewers, then we can define $\alpha = \mu - 3\sigma$ where μ and σ are the mean and standard deviation of buddies per skewer, respectively. In other words, based on p's behavior witnessed by m, the probability of having $\alpha/|B_m|$ or less buddies match p is roughly 0.1%. More plainly, the set of p's buddies for a given monitor should match well enough for the majority of p's normal behavior. Of course, it may be the case that buddies are not normally distributed across S_m. For these instances, we define a conservative base case threshold: at least a third of the buddies must match p or a warning is raised.

However, it is not enough to say that the prefix was hijacked if only one monitor raises a warning. The entire Buddyguard system must correlate warning flags from all monitors to decide whether the prefix was hijacked. If more than $X\%$ of the monitors have a warning flag raised at any given time, the system issues a hijacking alert. The alert state can only be dropped if the percent of monitors with warning flags raised drops below threshold X. Given a set of well-matched and widely distributed buddies, this system-wide alert threshold now becomes the deciding factor in determining prefix anomalies. The precision of this threshold is critical—too high values will result in false negatives, where Buddyguard fails to detect an anomaly; and too low values will result in false positives, or false alarms. We discuss the proper calibration of this threshold in our evaluation section.

Finally, the accuracy of a buddy-based monitoring scheme is dependent upon the ability to acquire the right set of buddies during our training period. A careful reader might ask, what if an anomaly occurs during this training period? Clearly this will reduce the well-matching quality of the selected buddies. We must therefore ensure that our training period is "clean"—no anomalies can occur for the monitored prefix during this time window. While this seems like a difficult task, we can exploit a simple principle to make it manageable. If Buddyguard finds buddies for the monitored prefix on the basis of anomalous behavior during training, then subsequent monitoring with those buddies will trigger false alarms for clearly valid path updates. When this occurs, we simply re-train the prefix until we obtain a clean training period. In this manner, we can ensure that our system uses buddies that best match the *normal* behavior of a monitored prefix.

D. Buddy Maintenance

After a monitor selects a set of buddies for a prefix, it is not guaranteed every buddy will always stay a good buddy. How can we maintain a good buddy system after the initial selection? Yet again, the information that Buddyguard collects

Algorithm III.1: TRAININGALGORITHM(p, m)

```
local $C_m$ candidates, $B_m$ buddies, $S_m$ skewers
for each $u_p(t_i)$ seen by $m$
  do {
    find candidates $c$ where
      $u_c(t_i \pm \Delta) \sim u_p(t_i)$
    append $c$ to $C_m$
    create skewer $s_{t_i}$
    append $s_{t_i}$ to $S_m$
  }
sort $C_m$ by frequency of matching
while $|s_{t_i}| < \omega \ \forall s_{t_i} \in S_m$
 and $B_m$ is not diverse
  do {
    place top matching $c$ on every $s_{t_i} \in S_m$
      where $u_c(t_i \pm \Delta) \sim u_p(t_i)$
    append $c$ to $B_m$
  }
return $(B_m)$
```

Fig. 4. **Training algorithm.** p is the monitored prefix and m is a monitor.

during training is invaluable for this task. Consider a given monitor m with buddies $b \in B_m$ who matched S_m skewers for the monitored prefix p during training. We can define the minimum quality $min(B_m)$ to be the minimum number of p's path updates matched during training by any buddy $b \in B_m$, and $|S_m|$ to be the number of skewers. During monitoring, we can re-evaluate B_m after $|S_m|$ path updates, and if any $b \in B_m$ matched less paths than $min(B_m)$, we drop b. Similarly, at any point during monitoring we can re-train p, replacing poorer quality buddies with better buddies found during re-training. In this way, we can maintain and improve the quality of B_m to ensure that Buddyguard can effectively monitor p.

Another concern that one might raise is the occurrence of policy changes that cause the monitored prefix to legitimately deviate from its buddies. For example, a prefix may become multihomed or enter a new peering agreement with another AS, while its buddies do not. However, even though these changes may occur at arbitrary times, such changes do not pose a serious problem for Buddyguard. Given that our system would be run as a service, we can simply require prefix owners to report such events when they occur. At that time, we can re-train the prefix according to its newly defined behavior, and continue monitoring with a new set of buddies.

IV. Resiliency of Buddyguard

As we alluded to previously, one of the primary advantages of the buddy system is that it allows Buddyguard to withstand intelligent attacks. With respect to prefix hijacking, there are several measures that a hijacker could employ to avoid detection by monitoring schemes. We now discuss some countermeasures that remain unaddressed in current systems, and explain how our system handles them.

A. Prefix Interception

When a prefix is intercepted, the hijacked traffic is forwarded on to the victim prefix. Since traffic is still routed to the victim, such an attack can easily go undetected. But from the perspective of a Buddyguard monitor, the path to the victim prefix still changes, and the path comparison with the victim's buddies will reveal the hijack and enable our system to detect the attack.

B. Sub-prefix Hijack

Buddyguard is particularly good at handling sub-prefix hijacking, a case that thwarts most existing solutions. If a monitor has never heard a sub-prefix of a monitored prefix, this monitor will use exactly the same path to reach the sub-prefix and the prefix all the time. Therefore, Buddyguard can view the buddies of the monitored prefix as the buddies of the sub-prefix, and use these buddies and the same detection procedure to determine whether the sub-prefix is hijacked.

C. Targeted attacks

What would happen if an attacker was aware of Buddyguard, and specifically attempted to thwart our monitoring scheme? For example, an attacker could try to co-hijack all or most of a prefix's buddies to eliminate the effectiveness of path comparison. However, each prefix we monitor has numerous buddies distributed across multiple ASes, and discovering enough buddies to co-hijack would be an enormous undertaking. An attacker might try to guess enough surrounding ASes and hijack all prefixes within those ASes, but a hijack on that scale would itself be blatantly suspicious. Furthermore, if Buddyguard is monitoring numerous prefixes each with their own buddies, then it is possible that some buddies of a target prefix are also being monitored. An attacker would have to hijack that prefix's buddies plus their own buddies, and possibly their buddies' buddies, resulting in a tedious recursive process. In effect, our buddy-based monitoring scheme is resilient against attackers who know how Buddyguard works.

An attacker could also try to exploit BGP routing policies to limit the visibility of the hijack. For example, if a hijacker knew which ASes contained Buddyguard monitors, it might insert those ASes into illegitimate path announcements. Since BGP routers discard path updates that contain their AS to avoid routing cycles, these attacks would not be noticed by our monitors. But to do so, a hijacker would have to insert the ASes of many or all of our monitors into an illegitimate path, which would greatly increase its length. Since BGP routers also prefer shorter routes, it is statistically improbable that this path would ever be adopted.

V. Evaluation

We evaluate Buddyguard in terms of the effectiveness of training (how well it can find and select good buddies), and the accuracy and performance of monitoring. For the latter we look specifically for cases of false negatives (where the system fails to detect an event) and false positives (false alarms). Our results demonstrate that using a system-wide alert threshold $X = 20\%$—recall that this means if 20% or more monitors have a warning raised, issue an alert—gives a proper balance for minimizing both false positives and false negatives. Again, without losing generality, we use prefix hijacking and route leaks as a means of demonstrating our system.

A. Setup

To test Buddyguard, we developed an initial testbed of monitors, prefixes, and all routing path updates concerning those prefixes during various time periods of interest. We built this testbed by processing real world BGP updates and RIB table dumps from the RouteViews collection [16], which allowed us to generate a set of over 600,000 prefixes and their associated BGP routing paths. For monitors, we used BGP speakers peered with RouteViews collectors, as these entities would have the same view of BGP routing data as seen in our test data (the coverage of this monitor placement strategy is discussed in section VI). This generated set of prefixes, monitors, and BGP updates enables us to evaluate Buddyguard on today's Internet topology and routing infrastructure.

B. Defining Thresholds

Our next step was to define threshold values for our training algorithms. To recap, these key thresholds are:

(i) *Path similarity* n. The number of AS hops to compare ($|u_p(t_i)| - n$) when checking if $u_p(t_i) \sim u_c(t_i \pm \Delta)$.
(ii) *Skewer lower bound capacity* ω. The number of buddy candidates required to fill a skewer.
(iii) *Resiliency factors*. The number of buddies needed for a monitored prefix and the minimum number of ASes covered by these buddies.

To demonstrate the efficacy of our system, we take a conservative approach toward defining the training thresholds. For path similarity, we use $n = 1$ such that Buddyguard may only find buddy candidates from the so-called origin, parent, and sibling ASes. We use the terms **origin, parent, and sibling buddies** to reference buddies from these respective ASes. Restricting our search algorithm in this way forces Buddyguard to rely on a minimal search space when looking for well-matched candidates ($n = 0$ excludes all but the origin AS, and therefore is too strict for our purposes). For the remaining two thresholds, we set $\omega = 30$ and require at least 90 buddies for a given monitor. By intuition, this stipulates that at least a third of the buddies should match the behavior of the monitored prefix during training, and that each monitor has a reasonably large set of buddies. Lastly, we require at least 30 of the buddies for a given monitor to be from sibling ASes, which ensures that the buddies are widely distributed—again, a key contribution to the resiliency of our system. We evaluate these conservative benchmarks in the next section, and discuss in section VI how they might be better calibrated.

Given these values, we proceeded with our tests accordingly. First we selected prefixes randomly from diverse locations of the network topology, including tier 1, tier 2, and tier 3+ ASes. For testing a specific event, we used the prefix(es) involved in that event. Each prefix we monitored was trained over a period of one week, during which we tried to find origin, parent, and sibling buddy candidates for each legitimate path update to the prefix. We define a legitimate update as an announced path that is not part of a routing path oscillation (for an explanation of this phenomenon, see [21]). We then used the skewering mechanism to select the best buddies from these candidates, and after verifying that the training period was clean, we monitored the prefix with the selected buddies. In order to prove the efficacy of using buddies from outside of the prefix's origin ASes, we show the results of monitoring a prefix with origin, parent, and sibling buddies separately. The following sections summarize our test results.

C. Evaluating Training

An important criteria for the effectiveness of training is whether Buddyguard can find numerous well-matched buddies across multiple ASes. Figure 5 depicts our training results across all the prefixes that we tested. With respect to buddies selected per monitor for a given prefix, we see that in most cases Buddyguard is able to find hundreds or thousands of eligible buddies (Figure 5(a)). However, we note that a very small portion of our samples (about 5%) have insufficient buddies. Fortunately, these edge cases can be attributed to our conservative definition of $n = 1$ for path similarity. Most of them come from very small ASes (tier 3+), where there is a deficiency of potential origin, parent, and sibling candidates. Increasing n adaptively would enable Buddyguard to search for candidates beyond those small ASes, allowing our system to find sufficient buddies for all cases. We discuss in section VI why we did not use this strategy, and for now are satisfied that $n = 1$ gives us enough buddies in the majority of cases.

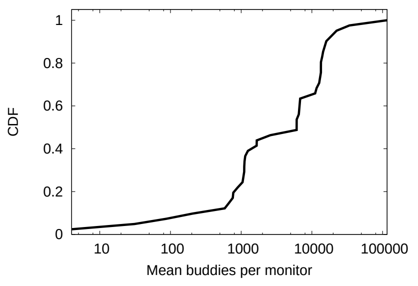
Our results are very similar when we look at the mean number of buddy or candidate ASes per monitor, as shown in Figure 5(b). Though we restrict candidate searching to origin, parent, and sibling ASes, this is still a large number of eligible ASes. In many cases Buddyguard had hundreds or thousands of ASes to choose from, and again, we can fix the few exceptions by relaxing n for path similarity. Therefore, our initial evaluations show that even under conservative restrictions, Buddyguard is able to find numerous and widely distributed buddies for monitoring prefixes.

It is also worth noting the general quality of the buddies selected. Figure 5(c) shows the CDF of buddy quality with respect to the percentage of prefix paths matched during training. The results are not exceptional—less than half of the samples match their respective prefix with any degree of regularity. While this may seem problematic, we demonstrate in the following sections that Buddyguard is able to accurately detect anomalies even when some buddies are mediocre.

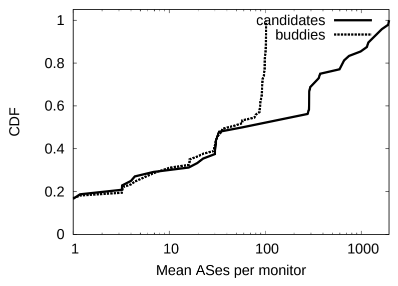
D. Accuracy

1) Detecting Prefix Hijacking: We evaluated Buddyguard's ability to detect prefix hijacking by testing our prototype on a wide manner of known hijacks. For brevity, we only show the results for three well-known hijacking events:

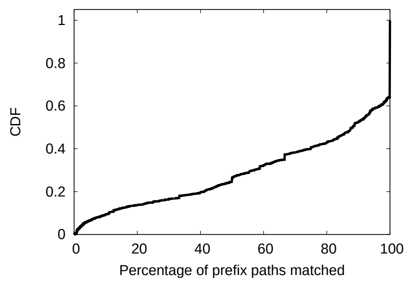
- Cogent's hijack of one of Google's prefixes [2].
- Con Edison's hijack of 30+ prefixes, including some belonging to their customers [1].



(a) Mean value of buddies selected per monitor (shown in log scale).

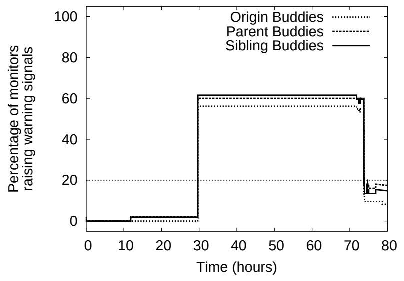


(b) Mean value of buddy and candidate ASes during training (shown in log scale).

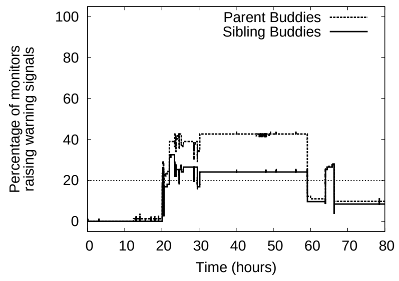


(c) Buddy quality in terms of % number of monitored prefix paths matched during training.

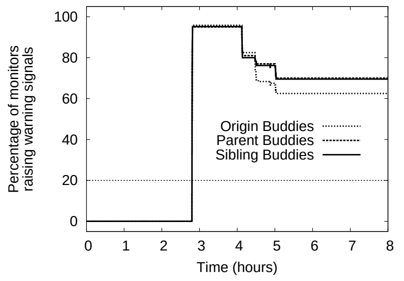
Fig. 5. **CDFs for finding buddies and selecting candidates across all prefixes sampled.**



(a) Warnings for when Cogent hijacked a Google prefix. Time 0 is 5/6/05 at 09:00:00 UTC. The hijack began 5/7/05 at 14:37:56 UTC [2].



(b) Warnings for Martha Stewart Living. (Note there are no results for origin buddies— the hijacked prefix was the only prefix in its origin AS.)



(c) Warnings for when Pakistan Telecom hijacked YouTube. Time 0 is 2/24/08 at 16:00:00 UTC. The hijack began at 18:47:57 UTC [3].

Fig. 6. **Percentage of monitors raising warning signals for Google, Martha Stewart Living, and YouTube hijacking events.**

- Pakistan Telecom's hijack of a sub-prefix of YouTube's prefix [3].

Our results from these three hijacking events (Figure 6) show that Buddyguard is well-suited to detecting prefix hijacks. For all tested events, a system-wide alert threshold $X = 20\%$ monitors with warnings raised (represented in our figures by the dashed horizontal line) suffices for detecting these hijacks. While in many cases X could be a smaller percentage, we point out that this conservative measure works for all tested events. We will also show that this observed threshold is effective for route leaks *and* maintains low false positives when monitoring normal prefix behavior.

When we inspect results from individual events, the efficacy of our monitoring system is clearly demonstrated. Looking at the Cogent event (Figure 6(a)), we see that the percentage of monitors with raised warnings dramatically increases for the duration of the event, and each monitor detects the hijack within 5 seconds of the event. The Con Edison event, which involved more than 30 hijacked prefixes, was also easily detected. One particularly interesting case here is the hijack of Martha Stewart Living's prefix (Figure 6(b)), the sole prefix within its origin AS. Even without the aid of origin buddies, Buddyguard was able to raise an alert for this hijack. This demonstrates that our system can successfully detect prefix hijacks even when using buddies outside of the origin AS.

Our monitoring scheme is also proven to be effective on sub-prefix hijacks. Using the YouTube hijacking as a case study (Figure 6(c)), we see that when the attacking AS announced an invalid path to a sub-prefix of YouTube's prefix, our system was able to detect the hijack within *one* second of seeing the first invalid path announcement. Taken together, the results show our system's accurate and resilient monitoring across a variety of prefix hijacking scenarios.

2) Detecting Route Leaks: We also tested Buddyguard on another well-known recent event: the April 4, 2010 China Telecom route leaks. We randomly selected 100 prefixes from those affected by the route leaks, and monitored them from 15:30 UTC to 16:30 UTC (the route leaks began at roughly 15:54 UTC and lasted until about 16:10 UTC). Figure 7 illustrates the percentage of monitor warnings raised across the aggregated sample. Once again, an alert threshold $X = 20\%$ monitors proves to be effective, detecting 90% of the (100) route leaks within seconds of the event onset. False negatives were largely due to not having enough buddies for monitoring, and as stated before such cases can be fixed by relaxing our training thresholds. As such, we see that a well-calibrated Buddyguard is adept at handling a variety of prefix anomalies, including prefix hijacking and route leaks.

It is also worth mentioning that for this event, many of the route leaks co-hijacked multiple origin buddies for several of the prefixes in our sample set. These co-hijacked origin buddies do not trigger warnings, as their (hijacked) routes match that of the monitored prefix. Here we see that such events cannot be detected by origin buddies or reference points alone; a topologically diverse set of buddies outside the origin AS was needed to detect the event. By maintaining enough valid points of comparison to the monitored prefix, Buddyguard remains resilient to such large scale route leaks. This

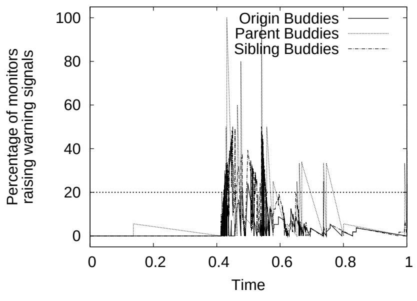


Fig. 7. **Aggregated warnings for 100 randomly selected affected prefixes during China Telecom route leaks. Time 0 is 4/8/10 at 15:30:00 UTC. Route leaks began at approx. 15:54:00 UTC.**

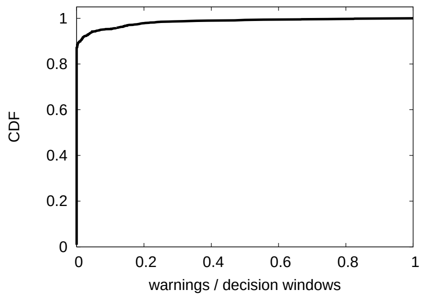


Fig. 8. **Mean value of false warnings over decision windows across all monitors and prefixes.**

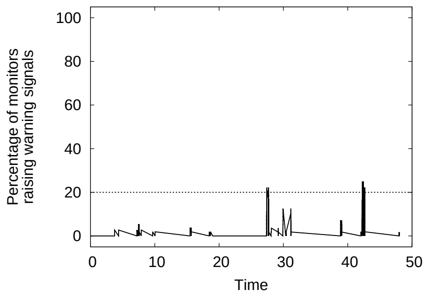


Fig. 9. **False warnings raised for all prefixes in our false positive tests.**

underscores the importance of topologically diverse buddy selection, a critical component of our system.

3) Monitoring Normal Prefix Behavior: While it is important that Buddyguard can quickly and accurately detect prefix anomalies, we also must ensure that our system raises minimal false alerts for normally behaving prefixes. To test false positives, we randomly selected 10 prefixes each from tier 1, tier 2, and tier 3 ASes. Each prefix was trained over the week of March 29-April 5, 2010, after which we verified that training was clean and training thresholds were met. For cases where Buddyguard was able to find enough buddies, we then monitored prefixes during a clean two-day period from April 5-7, 2010. We can reasonably ensure that this time period was clean from an absence of known hijacking events and path mismatches from origin buddies that matched perfectly during training. Our reasoning for the latter is that attackers do not yet know about Buddyguard, thus the probability that an event occurred which co-hijacked all origin buddies is minimal.

Our metric for false positives is the number of warnings raised by a given monitor divided by the number of decision windows, or distinct times when that monitor decided whether to raise a warning. Figure 8 shows the distribution of false positives across all samples. We see immediately that with sufficient buddies, our system can monitor IP prefixes with very low false warnings. In fact, nearly 90% of our monitors raise no warnings at all.

Furthermore, a warning raised by a single monitor does not translate directly to a false alert. Figure 9 shows the percent of monitors with warnings raised over the two day monitoring period across all samples. If we use the previously observed system-wide alert threshold $X = 20\%$, only about 6% of the decision windows actually peak above this threshold. Moreover, all of these instances occurred *for a single prefix*—in other words, only one prefix would have raised false alerts. While we discuss later how this observed threshold might be better calibrated, for now it is enough to say that $X = 20\%$ works to minimize both false negatives and false positives. Therefore, our monitoring tests on prefix hijacking, route leaks, and normal prefix behavior show that our system provides accurate and resilient monitoring of IP prefixes.

E. Performance: Latency and Overhead

Finally, we evaluate Buddyguard with respect to its overall performance—namely, the latency and overhead involved. Since Buddyguard analyzes BGP updates in close to real time, detecting anomalies can be achieved with little latency. In the hijack and misconfiguration cases described above, detection occurs within minutes (and often seconds) of the event. This is a crucial point; prefix anomalies must be detected and addressed immediately. Our system provides fast detection of anomalous events, giving prefix owners the chance to minimize the damage done.

Our system design also maintains low storage cost, an important measure when considering how Buddyguard might scale to monitor numerous IP prefixes. During the monitoring process, the storage cost for every monitored prefix is mainly (1) the current AS path from the monitor to the prefix and (2) a set of buddy prefixes. One bit of auxiliary data that the monitor may also store for the monitored prefix is (3) its set of good AS paths to that prefix. Assuming the average length of an AS path is 4, a prefix has 1,000 buddies on average, and a prefix has on average 10 AS paths known to be legitimate, the total storage cost per monitored prefix will be approximately 4.2 KB. Therefore, monitoring 100 prefixes would only cost about 400 KB, and 1,000 prefixes only about 4 MB. Thus, it is very feasible for Buddyguard to monitor even every prefix on the Internet. When coupled with pre-existing BGP monitoring systems like RouteViews, Buddyguard is lightweight, scalable, and easily deployable for today's Internet.

VI. Discussion and Future Work

Using our conservative approach toward defining the path similarity threshold $n = 1$, we were not able to find enough buddies for a small number of prefixes. However, these edge cases can be addressed by defining n adaptively: if $n = 1$ is not sufficient, continue incrementing n and searching for candidates. Proceeding in this manner, we can easily find enough buddy candidates for all prefixes. However, n has a direct effect on how well buddy candidates match the monitored prefix. For a given prefix p and two buddies b_0 and b_5 that are 0 and 5 AS hops away from p respectively, the path comparison between p and b_0 is much more strict; the paths to p and b_0 will need to share 5 more AS hops than the paths to p and b_5. Furthermore, if an attacker is close to p, what is the chance that the path to b_5 will even change if p is hijacked? Investigating the intricacies of searching for buddy candidates adaptively is beyond the scope of this work and is a topic for future study.

In addition, many of the other metrics that we use for this work could also be further optimized. The thresholds used for training are by no means absolute; we use them to demonstrate Buddyguard's effectiveness under constrained conditions. Moreover, while the observed system-wide alert threshold $X = 20\%$ works to minimize false negatives and false positives during monitoring, it is certainly not the only or best threshold for this purpose. A better design might involve using a "gradient" alert scale, where we determine system-wide alert status using ranges for inert, suspicious, and anomalous monitor warning levels. An entire study could be dedicated to optimizing this threshold, making such calibrations beyond the scope of this work.

Another important question rising out of this work is the extent of coverage provided by RouteViews collectors. A previous work by Zhang et al. [34] examined this issue, though not with respect to a buddy-based monitoring system. A follow up study to our work would be to compare coverage of other monitoring systems such as RIPE [25] and BGPMon [32], or investigating other monitor placement options such as peering with BGP speakers directly.

VII. Conclusions

The current state of prefix-level monitoring leaves much to be desired; today's systems are limited in scope and underestimate the capacity of intelligent attackers. While the eventual deployment of S-BGP could address many of these problems, widespread adoption is years away and we will still need tools for understanding and diagnosing prefix routing behavior and routing anomalies.

In this work we introduce Buddyguard, a lightweight prefix monitoring system that provides fast and accurate detection of various prefix anomalies and is resilient to targeted countermeasures. Using our buddy-based monitoring scheme, Buddyguard is able to detect all manner of prefix hijacks and route leaks, and remains highly extensible to new anomalies as they emerge. We rigorously evaluated Buddyguard against known hijacking events and route leaks, and demonstrate the accuracy and efficacy of our buddy system design. This work represents a major step forward in the accurate and resilient monitoring of IP prefixes and prefix-level anomalies.

References

[1] T. Underwood, "Con-Ed steals the 'Net - Renesys blog," 2006. [Online]. Available: http://renesys.com/blog/2006/01/coned-steals-the-net.shtml

[2] T. Wan and P. C. van Oorschot, "Analysis of BGP prefix origins during Google's May 2005 outage," in *Proceedings of IPDPS*, 2006.

[3] RIPE NCC, "YouTube hijacking: A RIPE NCC RIS case study," http://www.ripe.net/news/study-youtube-hijacking.html.

[4] "BGPMon", "'Hijack' by AS4761 - Indosat, a quick report," 2011. [Online]. Available: http://bgpmon.net/blog/?p=400

[5] H. Ballani, P. Francis, and X. Zhang, "A study of prefix hijacking and interception in the Internet," in *ACM SIGCOMM*, 2007, pp. 265–276.

[6] "BGPmon", "BGP prefix hijack by AS16735," November 2008, http://bgpmon.net/blog/?p=80?

[7] T. Underwood, "Internet-wide catastrophe—last year," December 2005. [Online]. Available: http://www.renesys.com/blog/2005/12/internetwide_nearcatastrophela.shtml

[8] "BGPmon", "Chinese ISP hijacked 10% of the internet," April 2010, http://bgpmon.net/blog/?p=282.

[9] "BGPMon", "How the Internet in Australia went down under," February 2012. [Online]. Available: http://bgpmon.net/blog/?p=554

[10] S. Kent, C. Lynn, and K. Seo, "Secure border gateway protocol (S-BGP)," *IEEE Journal of Selected Areas in Communications*, vol. 18, no. 4, pp. 582–592, April 2000.

[11] S. Goldberg, M. Schapira, P. Hummon, and J. Rexford, "How secure are secure interdomain routing protocols," in *ACM SIGCOMM*, 2010, pp. 87–98.

[12] M. Lad, D. Massey, D. Pei, Y. Wu, B. Zhang, and L. Zhang, "PHAS: a prefix hijack alert system," in *Proceedings of the USENIX Security Symposium*, 2006.

[13] X. Hu and Z. M. Mao, "Accurate real-time identification of IP prefix hijacking," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2007, pp. 3–17.

[14] C. Zheng, L. Ji, D. Pei, J. Wang, and P. Francis, "A light-weight distributed scheme for detecting IP prefix hijacks in real-time," in *ACM SIGCOMM*, 2007, pp. 277–288.

[15] Z. Zhang, Y. Zhang, Y. C. Hu, Z. M. Mao, and R. Bush, "iSPY: Detecting IP prefix hijacking on my own," in *ACM SIGCOMM*, 2008, pp. 327–338.

[16] Univ. of Oregon, "Route Views Project," http://www.routeviews.org/.

[17] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," in *ACM SIGCOMM*, 1997, pp. 115–126.

[18] C. Labovitz, G. Malan, and F. Jahanian, "Origins of Internet routing instability," in *Proceedings of IEEE INFOCOM*, March 1999.

[19] R. Mahajan, D. Wetherall, and T. Anderson, "Understanding BGP misconfiguration," in *Proceedings of ACM SIGCOMM*, August 2002.

[20] J. Li, D. Dou, Z. Wu, S. Kim, and V. Agarwal, "An Internet routing forensics framework for discovering rules of abnormal BGP events," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 55–66, October 2005.

[21] J. Li, M. Guidero, Z. Wu, E. Purpus, and T. Ehrenkranz, "BGP routing dynamics revisited," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 7–16, April 2007.

[22] S. Teoh, K. Ma, S. Wu, D. Massey, X. Zhao, D. Pei, L. Wang, L. Zhang, and R. Bush, "Visual-based anomaly detection for BGP origin AS change (OASC) events," in *Proceedings of the Distributed Systems, Operations, and Management Workshop*, 2003.

[23] K. Zhang, A. Yen, X. Zhao, D. Massey, F. S. Wu, and L. Zhang, "On detection of anomalous routing dynamics in BGP," in *Proceedings of the International IFIP-TC6 Networking Conference*, 2004, pp. 259–270.

[24] J. Li, Z. Wu, and E. Purpus, "Toward understanding the behavior of BGP during large-scale power outages," in *Proceedings of IEEE GLOBECOM*, November 2006.

[25] "RIPE Routing Information Service," http://www.ris.ripe.net/.

[26] "Cyclops," http://cyclops.cs.ucla.edu/.

[27] B. A. Prakash, N. Valler, D. Andersen, M. Faloutsos, and C. Faloutsos, "BGP-lens: Patterns and anomalies in Internet routing updates," in *Proc. of ACM SIGKDD*, 2009.

[28] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur, "Topology-based detection of anomalous BGP messages," in *RAID*, 2003.

[29] RIPE NCC, "RIPE MyASN service," http://ris.ripe.net/myasn.html.

[30] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane: An information plane for distributed services," in *Proceedings of the Symposium on Operating System Design and Implementation (OSDI)*, November 2006.

[31] A. Broido and k. claffy, "Analysis of RouteViews BGP data: policy atoms," in *Network Resource Data Management Workshop*, 2001.

[32] H. Yan, R. Oliveira, K. Burnett, D. Matthews, L. Zhang, and D. Massey, "BGPmon: A real-time, scalable, extensible monitoring system," in *Proceedings of Cybersecurity Applications and Technologies Conference for Homeland Security*, 2009.

[33] T. Griffin and G. Wilfong, "An analysis of BGP convergence properties," in *ACM SIGCOMM*, August 1999.

[34] Y. Zhang, Z. Zhang, Z. Mao, C. Hu, and B. MacDowell Maggs, "On the impact of route monitor selection," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, 2007.