BGPInspector: A Real-time Extensible Border Gateway Protocol Monitoring Framework

Mingwei Zhang
University of Oregon, Eugene, OR, USA
Email: mingwei@cs.uoregon.edu

***Abstract*—The Internet often experiences disruptions that affect its overall performance. Disruptive events include global-scale incidents such as large-scale power outages, undersea cable cuts, or Internet worms. They also include IP-prefix level anomalies such as prefix hijacking or route leak events. All such events could cause the Internet to deviate from its normal state of operation. It is therefore important to monitor and detect the abnormal events, and do so from both granularities. Current solutions mostly focus on detecting certain types of events or anomalies and ignoring the others. There is not yet a generic framework that can perform different monitoring tasks under one system. In this report, we present our work on improving the two monitors, I-seismograph and Buddyguard, and introduce our new extensible Internet monitoring framework that ties them together. Each component of the framework works independently, allowing our system to perform multiple monitoring tasks at the same time and to integrate new components without disrupting the currently active ones.**

***Index Terms*—BGP monitoring, system architecture, real-time system, anomaly detection**

I. Introduction

Routing is the fundamental function of the Internet. To make routing scalable, the Internet consists of smaller networks called Autonomous Systems (ASes), each of which has its own internal routing policies. The ASes are connected to enable hosts from each AS to reach any other place on the Internet. Internet routing is the routing decisions and actions among the ASes.

The Border Gateway Protocol (BGP) [1] is the *de-facto* inter-domain routing protocol for the Internet, and is commonly used to determine the route for any traffic that will cross ASes. BGP routers can announce, withdrawal, or change the AS-level path toward any IP prefix (i.e., a block of IP addresses). The messages to convey these changes are called BGP updates.

Due to its dominating role in Internet routing, any disruption of BGP and BGP routers can cause severe problems across the whole Internet. Regardless of its importance, BGP was not designed with security in mind, and it has become the tool that has been utilized by multiple types of malicious activities. Assuming every actor is benign, a BGP router by default will trust all messages (such as prefix path changes) from its connected peers. This allows prefix-level BGP anomalies to happen, such as prefix hijackings or route leaks [2]. Security solutions such as BGPSec [3] have been proven ineffective until they reach full deployment [4], which will be years away even if people are willing to push the deployment forward.

Because the anomalies and large-scale events on the Internet are happening increasingly often, it is crucial to be able to monitor the Internet routing status timely and accurately. With the data provided by BGP archiving projects like Route-Views [5] and RIPE Raw Data [6], people can monitor the Internet routing by examining the BGP update stream. Projects like [7], [8], [9], [10], [11], [12], [13], [14] monitor BGP at the prefix granularity, focusing on detecting anomalies for individual prefixes. Other types of BGP monitoring projects like [15], [16] look at the Internet as a single entity and monitor the dynamics of the BGP updates instead.

Despite many previous projects in this area, gaps still exist that prevent us from better monitoring, visualizing, and understanding the anomaly events happening on the Internet. For example, we cannot determine a prefix hijacking event belongs to an accidental route leak or indeed an intentional attack. On the other hand, the projects that monitor BGP dynamics cannot easily reason about the root-cause behind the unusual changes or further help resolve large-scale anomalies. Besides, there is rarely a system the can monitor the Internet from multiple granularities with extensibility on different further analysis tasks.

Looking at all these existing BGP monitoring projects, we believe that a good monitoring system should have the following features:

1) accuracy – able to detect Internet events with high accuracy, low false-positive and false-negative
2) multiple-granularity – able to monitor the Internet from multiple granularities
3) agility – able to perform fast, ideally real-time, monitoring
4) extensibility – able to incorporate new functionalities without interfering with existing ones.

In this report, we present a comprehensive framework called ”BGP Inspector”, aimed at providing fast, accurate, extensible, and multi-granular BGP monitoring. Our contributions can be summarized as follows:

1) we extensively improved our prefix-level and global-level monitors, and fit them in the new architecture,
2) we designed a pipe-lining BGP monitoring architecture that enables close-to-real-time monitoring,
3) we designed a novel plugin-system that supports adding new monitoring capabilities quickly and independent to the existing monitors.

The content of this report is organized as following. We will

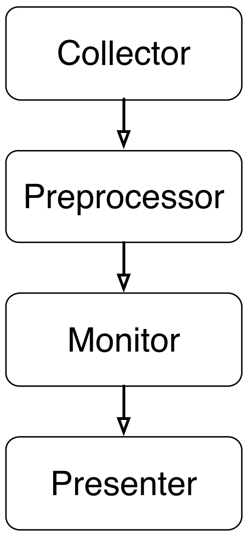


Fig. 1: **Components of the framework.**

first introduce our new design of the generic BGP monitoring framework and its architecture and components. Then we will introduce the current Internet level and prefix level monitoring mechanisms. After that, we will briefly discuss the related work and our future plans.

II. Generic BGP Monitoring Framework

In this section, we will introduce our generic BGP monitoring framework. When we design the framework, we follow the principle of modular, independent, and extensible. Each component should be independent from the design aspect, using the unified interface, and being extensible to further upgrades. As a result, our new framework incorporates the existing BGP monitoring mechanisms and enables accurate and agile monitoring of BGP.

A. *Framework Components*

Following our designing principle, we implemented our generic framework with four different types of components that work together to perform agile and accurate BGP monitoring tasks. In order to obtain BGP data, we first need components to collect data from different data sources. The obtained data may be stored in various file types, or contains artifacts such as BGP dumps caused by BGP session results. Therefore we also need to preprocess the data and produce a "clean" version of what we need for the monitoring and presenting tasks. The monitoring components are designed to be independent to each other, making the framework extensible for multiple purposes. In our case, we have monitors for both prefix-level and Internet-level monitoring tasks, running at the same time. For each monitor, we would also expect different types of presentation of the results. We designed presenter components that can meet different presentation requirement from the monitors. In the following, we will introduce each component and explain how they work together.

1) Collector:

As the first type of components, the collector is designed to collect, parse, and store the BGP data from different data sources. The collector is responsible for providing the parsed BGP update objects into the data preprocessing and monitoring components. The main data sources that we are using are RouteViews [5] and RIPE RIS raw data archive [6]. Both projects provide archived historical BGP updates data dating back to early 2000s. Besides historical data, our system can also incorporate real-time data, such as the XML stream provided by BPGmon [17]. In the report, we mainly use historical data to evaluate our system.

The collection process is a two-phase procedure includes selective downloading and encoded binary file parsing. Collector will first download the compressed Multi-threaded Routing Toolkit [18] (MRT), and then parse the MRT binary file into actual BGP updates. Both RIPE and RouteViews archives provide web-based file access (through HTTP or FTP), and the data is organized by fixed-length time slots (every 5 minutes for RIPE and 25 minutes for RouteViews). Because we are only interested in certain period of time, the collector will select only the relevant data archives to download, reducing the time and storage overhead. The compressed MRT files will then be stored and organized locally the same way as online archive.

With the MRT data file stored locally, the collector then will parse for each data file into BGP update objects in Java. We implemented the MRT parser based on the new MRT specification (RFC6396 [18]). For parsing BGP data, we focus on BGP4MP (the data type for BGPV4) and do not parse OSPF, TABLE_DUMP, ISIS or the types that MRT supports. Each MRT message contains a header that has the information about the router (local AS/IP, and its peer AS/IP), followed by the BGP message. The BGP message in MRT data file follows RFC4271 [1], the current specification of BGP version 4. We materialize the BGP updates into BGP "Update" objects, each of which contains the key fields that we care about for every BGP update message, including ORIGIN, AS_PATH, NEXT_HOP, NLRI, WITHDRAWN, etc. and useful functionalities. The Update objects will then be used as input for the data preprocessing and monitoring components. Depending on the parsing speed requirement, users may choose to store the parsed plain-text data for further usage. However, since the monitoring tasks usually require multiple weeks' data, the storage overhead of uncompressed data is much higher in this way. During our experiments, the parsing time overhead for two weeks of data is only around two minutes, while the storage overhead for plain-text data could be as high as hundreds of gigabytes.

2) Preprocessor:

As the second type of components, the preprocessor components will further clean the data and organize it into the desired format for the monitoring tasks. Recall that we use RIPE and RouteViews' data as our data sources. The collected data may contain duplicated information such as the duplicate updates caused by the connection resets between the data collecting routers. The duplicated information will confuse monitors and create false positives or negatives. To avoid these cases, we borrowed the algorithm discussed in [19] to remove the dumplicated BGP updates. The cleaned set of updates is then considered the *true* updates.

Besides data cleaning, we also need to organize the data into a specific format for different types of monitors. For example, the preprocessor needs to transform the BGP Update objects into one-minute data bins to be analyzed by the Internet seismograph monitor III. The modular nature of the processor's design enables our system to perform multiple monitoring tasks with one BGP Update input stream. We will discuss the workflow in Section II-B.

3) Monitor: Third type of components is the monitor, which are the key components of our framework. Each monitor component takes the preprocessed data as input, and performs the monitoring tasks. With different preprocessing requirements, the preprocessors and monitors always come in pairs. A monitor component contains the main logic of a monitoring task. In our framework, we have two types of monitors that monitor on Internet-level running status, and prefix-level anomalies. The Internet-level monitor is based on our previous work called "I-seismograph" [20]. We did multiple upgrades to the original I-seismograph and made corresponding changes to plug it into our framework. Similarly, we use the improved version of Buddyguard [2] as our prefix-level monitor. We will introduce the details of each monitor and the improvements we have done in Section III and Section IV. Briefly speaking, the Internet-level monitoring will look at the BGP dynamics for all one-minute data bins, and determine an abnormality value for each minute. The prefix-level monitoring will look at the AS-PATH toward each individual prefix and detect path anomalies. Combining the two monitors of different granularities, our system is able to comprehensively monitor the Internet.

4) Presenter:

Based on the monitor's results, our system should also present the results into different formats. The last type of components is presenter. We have developed different presenters to present the monitoring results for different goals. For example, the prefix-level anomaly detection must quickly determine the time and the severeness of any anomalies, therefore a time-series line graph is used to present the results. In order to understand the propagation of anomalies, we also adopted the path change animation technique (BGPlayJS) to show the progress of the propagations of any anomaly. Similarly for Internet seismograph, we use a time-serious line graph to show the value curves for all relevant attributes. In order to further reason about the events, we also developed a geographical information representation using Google Maps API for all the Internet events, and the demonstration is online for further review.

Note that, the presenter is also designed in a modular way, so that we can "plug-and-play" any kind of presentation methods, and we can apply multiple presenters on a same set of data results. The goal here is to give our monitoring framework the maximum flexibility to extend not only the monitoring capability, but also the visualization methods.

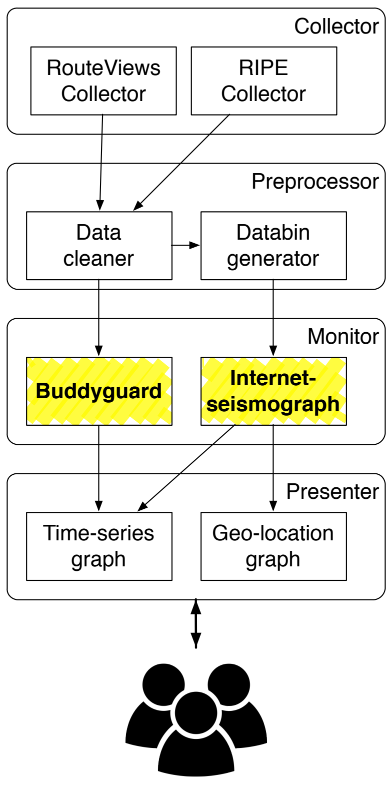


Fig. 2: **Work flow of the framework.**

B. Work Flow

In this section, we will introduce in detail how our integrated BGP monitoring system works, and how each components can be put into a real-time processing pipe-line that enables faster processing and scalable deployment.

We have introduced the four components, i.e., collector, preprocessor, monitor, and presenter, and how they work individually. The next question naturally becomes, how do we glue these components together into a generic real-time BGP monitoring framework that can perform different types of BGP monitoring tasks with high efficiency. Here we designed pipe-lining mechanism that take BGP updates feeds in sequence and generate results on the fly. The pipe-line is shown in Figure. 2, where the collectors will keep fetching the data into the pipe, after the preprocessing and different monitoring tasks, the results are eventually presented by the presenter.

At the beginning of the pipeline, the system will fetch the raw data from the data source. We currently use RIPE and RouteViews data as our data source. As described in Section II-A1, the RIPE and RouteViews data is organized by minutes, and archived in Multi-Threaded Routing Toolkit (MRT) format. As the system downloading the raw data, the collector starts parsing the downloaded MRT files, and translate them into actual BGP updates. In our implementation, the BGP updates are represented as Update objects, where all the values of the updates are stored as fields of the objects. Downloading and parsing in the same time, the collector keeps producing Update objects, forming a stream for the following components.

With the BGP update stream as input, our system will further preprocess the data, i.e., removing the updates generated from BGP reset sessions (cleaning the data). The data cleaning procedure will delay the pipeline by 15 minutes in order to find out the BGP reset sessions. After each 15 minutes, the BGP updates will be either directly fed into Buddyguard for prefix-level monitoring, or used to extract data bins and then fed into I-seismograph for global-scale monitoring. The preprocessing will change the "real-time" update stream into a "pseudo-real-time" update stream with 15 minutes delay. However, if we're using the real-time data source such as BGPMon, we can then safely skip the data-cleaning procedure and the update stream becomes real-time.

The presenter, who takes the results generated by the monitors, will further process and visualize the monitoring results, giving users a more direct visual feedback of the current BGP running status without digging into the text logs. This portion is important because the users (usually network operators) are often required to quickly response to the events happening on the Internet. The capability of understanding the source of the problem and proposing solutions is critical in this case. The presenter components of our system can serve this purpose.

Although the components work together using the pipeline stream, they run independently without requiring other components. In other words, the components for each phase can be replaced, removed, or extended individually without affecting the other components. For example, we can add more monitors of different purposes to the system, while keeps the original monitors running. We can also add or replace presenters to extend or upgrade the visualization part of our system. The "plug-and-play" system design enables us to keep updating our components and expanding the framework. The ultimate goal is to build a comprehensive framework that has multiple monitoring capabilities while still allowing users to redefine or add new functions to the framework.

III. Global-level Monitoring

We will first looking at the BGP monitoring tasks from global-level granularity. In this section, we will review the previous Internet monitoring project I-seismograph 1.0, and then introduce our work on improving the project as I-seismograph 2.0.

A. Problem Description

The Internet has become a critical infrastructure of our society, yet little has been studied on how to monitor the Internet as a whole and how to quantify the impact that disruptive events may have on it. However, the large-scale events such as natural disasters, undersea cable cuts or route leaks will disrupt the Internet routing infrastructure and slow down the connections. Although events such as security attacks, large-scale power outages, hurricanes, undersea cable cuts, and other kinds of natural disasters may cause observable disturbances to the normal operation of the Internet, we know little about the kind of impact each event might cause and how big it might be; the lack of such knowledge also makes it difficult to conduct effective network diagnosis, recovery, or other operation tasks.

In the following sub-sections, we first review the idea of I-seismograph, and then introduce the new functionalities that we brought to the new version of the I-seismograph, and evaluate the performance with demonstration of our visualization modules. At last, we will discuss how we can fit I-seismograph into our generic framework.

B. Basic Concepts

In the following, we will introduce the basic concepts and methodologies that have been used in both I-seismograph 1.0 and 2.0.

1) Definition of Impact:

We define an impact on BGP as any deviation from BGP's normal profile. The deviation consists of a *magnitude* and a *direction*. Assume we use a set of n distinct BGP attributes to inspect BGP (the attibutes we use are listed in Table I), A_1, $A_2, \cdots, A_n$. Also assume we have defined a normal profile of BGP by identifying the normal values of those attributes. At any time *t*, if the values of these attributes of BGP are $a_1(t)$, $a_2(t), \cdots, a_n(t)$, and they deviate from the normal profile as $\delta_1(t)$, $\delta_2(t), \cdots, \delta_n(t)$, the impact that BGP receives at *t* is then a vector as follows: $i(t) =< \delta_1(t), \delta_2(t), ..., \delta_n(t) >$. If looking at the impact on BGP over a time window, such as during the period of an event, we can define the impact during this window, say $[t_1, t_2]$, as: $I(t_1, t_2) = \int_{t_1}^{t_2} i(t)dt$ or $\sum_{t_1}^{t_2} i(t)$, depending on whether $i(t)$ is continuous or discrete.

2) Definition of Normality:

I-seismograph's basic data processing unit is BGP *databin*, which is simply a summary of the values of 10 distinct BGP attributes over a period of one minute (See table I). To measure the impact during a monitoring period, our basic idea is to check every databin from that period, and see whether it is associated with a **normal cluster** composed of a set of normal databins, or an **abnormal cluster** composed of a set of abnormal databins. At any point there is only one normal cluster but there can be multiple abnormal clusters. The normal cluster represents the normalcy of BGP, and the abnormal clusters represent different types of BGP abnormalities. Once we know every databin's associated cluster, we then can calculate the impact of the databin as well as the impact during the entire period. Our enhancements does not involve changing the basic algorithm, the details can be found in [20].

3) Impact Calculation:

Once we have the normal cluster from training period and the abnormal cluster from monitoring period, we can start calculate the impact for each databin in monitoring period. We apply the following concepts to measure the impact of a databin or the impact during a monitoring period:

- **Impact value (of a databin).** It measures the *distance* of a databin from the normal. We define every databin in the normal cluster has an impact value 0, and here we focus on those not in the normal cluster. Denote the databin as $d =< d_1, d_2, \cdots, d_n >$. We take the following steps: (1) For every attribute A_i $(i = 1, 2, ..., n)$ of d, we use all the

Attribute	Description
Announcement	# of BGP announcements
Withdrawal	# of BGP withdrawals
Update	# of BGP updates
WADiff	# of new-path announcements after withdrawing an old path to the same IP prefix
AADiff	# of new-path announcements to the same IP prefix (thus implicit withdrawals)
WWDup	# of duplicate withdrawals to the same IP prefix
AADupType1	# of duplicate announcements to the same IP prefix where all fields of the announcements are unchanged
AADupType2	# of duplicate announcements to the same IP prefix where only the AS-PATH and NEXT-HOP fields of the announcements are the same
WADup	# of re-announcements after withdrawing the same path
AW	# of withdrawals after announcing the same path

TABLE I: Names and descriptions of selected BGP attributes.

databins from the normal cluster to determine their mean μ_i and standard deviation σ_i of A_i. (2) We then calculate the difference between d_i and $(\mu_i \pm \sigma_i)$, denoted as δ_i. It is either $d_i - (\mu_i + \sigma_i)$ if d_i is greater than $(\mu_i + \sigma_i)$, or $(\mu_i - \sigma_i) - d_i$ if d_i is smaller than $(\mu_i - \sigma_i)$. (3) We normalize δ_i to be in the range of [0, 1] by dividing the maximum recorded value of δ_i. In the following δ_i always refers to a normalized value. (4) Then finally, we use the sum of the differences for all attributes, i.e., $\sum_{i=1}^{n} \delta_i$, as the distance of d from the normal. This distance is also called Manhattan distance. Since our study currently uses exactly 10 BGP attributes, every impact value will thus be between 0 and 10.

- **Impact curve (of a monitoring period).** This is the plot of the impact values of all the databins from a monitoring period over time.
- **Impact direction (of a databin).** It measures the *direction* that a databin deviates from the normal. Following the discussion of impact value above and using the same notations, we define the impact direction of a databin using the deviation vector $< \delta_1, \delta_2, \cdots, \delta_n >$.
- **Dominant and peak impact directions (of a monitoring period).** The abnormal cluster that has more databins from the monitoring period than any other abnormal clusters is what we call the *dominant abnormal cluster* for the period. We define the impact direction of this cluster's medoid (i.e., its most centrally located databin) as the *dominant impact direction* for the monitoring period in question. In addition, we define the impact directions of those databins from a monitoring period that have a peak impact value as the *peak impact directions* of the period. Note that those databins may or may *not* belong to the dominant abnormal cluster. The dominant direction represents the overall trend during a monitoring period, and the peak direction indicates the behavior during the maximum impact.

C. I-seismograph 1.0

In order to measure "Internet earthquakes.", Jun Li et al. designed an Internet seismograph, or *I-seismograph* [20]. Here we call it I-seismograph 1.0 as comparing to the current improved 2.0 version. It not only reports the magnitude of the impact during an event period, i.e., a "Richter scale" of an Internet earthquake, but also characterizes the nature of the earthquake. During a period when everything is normal, I-seismograph will simply report zero or close-to-zero impact; during a security attack, a natural disaster, or some other large-scale incident, if the regular operations of the Internet go awry, it can then indicate how badly the Internet got hit. Not only can we use I-seismograph to measure the impact over a period in the past, during which a disruptive event is suspected to have affected the Internet, but we also can use it to measure an Internet earthquake in real time.

The main design idea of I-seismograph is hinged upon discovering the "normal" state of the Internet, and then monitoring a given period to measure how the Internet activity deviates from it. I-seismograph uses BGP data to discover the normal and abnormal states. Due to the dynamic nature of BGP, I-seismograph applies a two-phase clustering method that can help defines the normal and abnormal states over a wide time span. I-seismograph will first divide the BGP update stream into one-minute databins, and use the databins as basic unit for the training and monitoring. After monitoring each databin should have a impact value and a impact direction. We could then plot the values to impact curves and present the seismograph results.

D. I-seismograph 2.0 - An Overview

The I-seismograph 1.0 gives users an idea of if there is an event happening on the Internet, and if so, what attribute deviates the most. We consider the result could be further enhanced and could convey more information for users to understand and the event and further mitigate the damage if there is any. Therefore we developed the multiple improvements for I-seismograph, added the root-cause analysis capability, and evaluated on the most recent Internet events.

Based on the I-seismograph's original design, we further improved the system by adding more in-depth root-cause analysis and thorough evaluation on most up-to-date Internet disruptive events. Here we call the improved version "I-seismograph 2.0." In this report, we will introduce the enhancement that we have done to help I-seismograph 2.0 to achieve more functionalities and being more insightful for the users.

In addition to the plain curve results, we also developed new types of visualization modules to visualize the Internet "earthquakes". The I-seismograph 2.0 with root-cause analysis and vivid visualization provides the users straightforward information about the Internet's routing status for any specific time period, and gives users further more information to reason about the potential causes and solutions for different types of the events.

E. Root Cause Analysis

One important feature that I-seismograph 1.0 missed is the capability to trace down the root cause of any events. The root cause of a large-scale disruptive event can be categorized into the following granularities:

1) the specific overall attributes that deviate from the normal status the most,

2) the autonomous systems that were involved in the unusual routing changes the most,
3) and the individual prefixes that contribute the most to the overall deviation of the attributes.

We developed the new components for I-seismograph 2.0 to conduct the root cause analysis on all three different level of granularities. In this section, we will present our methodology of the root cause analysis.

1) Internet-level root-cause analysis: From a coarser granularity, we can start by looking at the Internet from routing dynamics. Based on previous studies on BGP instability and dynamics, including those from [16], [21], we have identified ten distinct BGP attributes to summarize every minute of BGP activities (Table I). Among the 10 attributes, we can further categorize into 3 corresponding types of behaviors:

1) neutral behavior: Announcement, Wtihdrawal, and Update
2) forwarding dynamics: AADiff, WADiff, AADupType2, WADup, AW
3) pathological behavior: AADupType1, WADup, WWDup

By looking at the result, we can first identify the dominant impact direction and the peak impact attributes. Depending on the attributes' categories, different hypothesises can be proposed. Then we can go a step deeper to look at the prefix level data to confirm or deny the hypothesises.

2) Prefix-level root-cause analysis: In a more finer granularity, we would also like to learn that what prefixes contribute most to the impact. Whenever an update about prefix p causes attribute a to increase, we say prefix p contribute to a once. By this definition of prefix contribution, we can learn how much a prefix contributes to the deviation of certain prefixes. We use the principle of deviation where the more the contribution of a prefix p to attribute a deviates from the normal status, the more it adds to the impact of attribute a.

With the definition of contribution of a prefix to an attribute, we can then learn the normal states for prefixes and then later detect the abnormality. In order to learn the normal contribution of a prefix to an attribute, we main the statistics count for every hour for every prefix during the training period. We define the range of 3 standard deviation around the mean value to be the normal range. As we proceed to monitoring phase, we will gather the contribution statistics for each prefix for each hour. The monitoring statistics will be compared to the normal range, and the deviates outside the normal range will be used as our metric to measure how much a prefix adds to the impact. As a result, we will have a sorted list of prefixes for each hour each attribute with value to be the deviation of the contribution outside the normal range.

F. Apply I-seismograph 2.0 on Real-World Events

Beyond the cases tested in [20], we further evaluated I-seismograph in against some more recent events.

In order to evaluate our Internet seismograph, we choose the events that potentially have large influence on the Internet, such as route-leaks or large-scale prefix hijackings. In those cases, the Internet reacts by generating and propagating large amount of BGP updates to reroute the affected traffic, where the dynamics of BGP should deviate from the normal status.

In addition to our previous work, we evaluated our system on more recent events. Table II listed some large events happened in previous years, and the double line separated the events evaluated in our previous work and this report. We categorized the events by their source location and the known cause, The events could have regional impacts such as [2005_HurricaneKatrina] and [2012_Canada] where the events happened in certain regions, or global impacts such as [2001_CodeRedWorm] and [2014_CISCO512K] where the events happened across the Internet. We also categorized the types of the events by the reported root-causes. There are five types of events, which are:

1) Security attacks: large-scale attacks on the Internet such as Internet worms.
2) Blackouts: Internet blackouts caused by power outages or natural disasters.
3) Cable cuts: the undersea fiber cable cut events.
4) Route leaks: large-scale leaks of routing table entries.
5) Router failures: large-scale router failures caused by firmware bugs or other reasons.

Though the router failure type only has one instance in our event list, i.e., the [2014_CISCO512K] event, this type could happen more and more frequently because the average number of routing entries in BGP routers has grown really close to some CISCO router's limit of 512K. Here we will introduce the most recent events and the detection results.

TABLE II: Events table

Event name	Date	Location	Type
2001_CodeRedWorm	2001-07-21	Global	Security attack
2001_NimdaWorm	2001-09-20	Global	Security attack
2003_SlammerWorm	2003-01-25	Global	Security attack
2003_EastCoastBlackout	2003-08-16	US	Blackout
2005_HurricaneKatrina	2005-08-31	US	Blackout
2005_LABlackout	2005-09-13	US	Blackout
2006_TaiwanCableCut	2006-12-28	Taiwan	Cable Cut
2008_Mediterranean	2008-02-01	Mideast	Cable cut
2008_Mediterranean2	2008-12-21	Mideast	Cable cut
2010_China	2010-04-08	China	Route leak
2012_Canada	2012-08-08	Canada	Route leak
2014_Indosat	2014-04-02	Indonesia	Route leak
2014_Syria	2014-05-19	Syria	Blackout
2014_TimeWarner	2014-08-27	US	Blackout
2014_CISCO512K	2014-08-11	Global	Router failure

1) 2010 China Telecomm Route Leak:

The [2010_China] event happened on April 8, 2010, where China Telecom (AS 23724) falsely originated about 37,000 prefixes for about 15 minutes [22]. Shown in Figure 3, the events started at hour 15, where multiple attributes were deviates from normal state. Note that the high value of WADup (withdrawal-announce duplicate) attributes at the first 10 hours of April 8 is potentially correlates to another anomalies that were not recorded by any press articles. This anomaly most were contributed by the prefixes from AS36958 (CWSeychelles) of Seychelles and AS9534 (Binariang Berhad) of Malaysia, where none of the prefixes have contributed to WADup attribute during the training period, and contributes on average more than 50 WADup per hour. The frequent withdrawal and announce of the same prefixes were clearly not normal, however since the abnormality were limited within these two ASes, there was no global impacts reported.

2) 2012 Canada Dery Telecom Route Leak:

The [2012_Canada] event happened on August 8, 2012, when a Canadian ISP leaked its full routing table to its provider [23]. Unlike the China Telecom route leak where China Telecom claimed to be the origin of many of the affected prefixes, this event was more subtle such that the leaker simply claimed to be on the path to the affected prefixes. During the event, the Canadian ISP Dery Telecom Inc (AS 46618) leaked all its routes acquired from one of its provider (VideoTron, AS 5769) to its another provider (Bell, AS 577). Bell selected a large portion of these routes as best routes and then further propagated to its peers. This route leak event affected 107,409 prefixes from 14,391 different ASes across the Internet [23]. As we can see in Figure 4, at around hour 17, the attribute AADupType2, WADiff, WADup have high spikes. The AADupType2 (announce-announce duplicates with slight diffferences) attribute confirms the nature of the route leak events, where the announced prefixes were already in routing table with the same AS path, only some optional values were different such as community values. The maximum number of AADupType2 per minute reaches 57646 during the events, and 277,027 for number of updates. The high values of WADiff and WADup (withdrawal-announce different and duplicate) indicate that there were also a lot of the out-of-date updates during the event where the same prefixes have been withdrawn before.

3) 2014 Indosat Route Leak (Hijacks):

On 18:26 UTC April 2, 2014, Indosat (AS4761) one of the largest ISP in Indonesia started to originate 417,038 new prefixes [24], while normally it only originates about 300 prefixes. Figure 5 show the monitoring results from April 1 to April 2, where multiple attributes have spikes around hour 42 (18:00 of April 2). Among all the abnormal attributes, the attribute AADiff (announce-announce different) deviates the most from the normal range. AADiff represents the underline change of AS paths of prefixes, where in normal circumstances it shows the dynamics of the AS path change and should fall in normal range like it did in the first 40 hours of the monitoring period. However, under extreme unusual situation like the Indosat route leak, the value of different announcements surpass the limits of normal range and then showed up on our results.

4) 2014 Syria Internet Blackouts:

Since early 2014, the Internet connectivity to Syria has been suffering from instabilities. There were 6 events reported, each happened on 3/25, 5/19, 5/21, 7/10, 7/12, 7/13 in 2014 [25], of which the event on 5/19 lasted longest for almost 20 hours. The Figure 6 shows the monitoring results, where the instability starts from hour 20 (5/19) to almost hour 40 (5/20). During the instability period, the main attributes that deviated from normal are the number of withdrawals and the number of WADiff (withdrawal-announce different), indicating the frequent changes of the AS paths.

5) 2014 CISCO "512K" Router Failures:

The most infamous Internet routing incidence of the year 2014 would be the CISCO "512k" event [26], [27], in which on August 12, 2014 the global routing table size surpassed 512 thousands, exceeding some CISCO routers capability and causing large-scale reroutes of the Internet traffic. We started our monitoring from August 11, to confirm the normal running states the Internet routing infrastructure. Then at hour 30, all most all of the 10 attributes deviate from normal range. Shown as Figure 7, the sudden change sticks out around the "peaceful" periods before and after the event. Careful readers may also noticed that comparing to the route leak events such as [2010-China] or [2012-Canada], the attributes deviate less in this events, and the Internet seems suffer more. One reason behind this is that the route-leak events were usually not widespread because of the filtering mechanisms deployed on the upstream ASes, however the CISCO event was caused by router failure and without regional limits, therefore people would more likely to notice the influence.

6) 2014 Time Warner Internet Blackout:

Time Warner Cable, one of the US's largest cable and broadband providers, suffered an nationwide Internet outage on 8/27/2014. According to [28], the event was caused by "an erroneous configuration propagated throughout the backbone." Around hour 3 almost attributes again deviates from the normal value range, shown in Figure 8

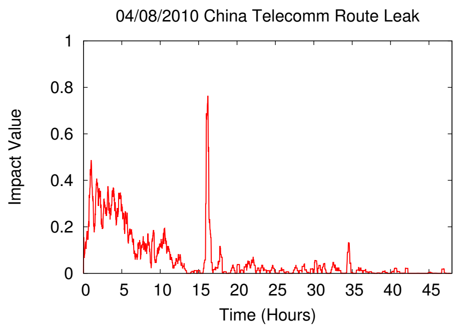
IV. Prefix-level Monitoring

In this section, we will look at the BGP monitoring from a more finer granularity, and introduce our prefix-level monitoring solution. Previously, Li et al. developed a prefix-level anomaly detection system called Buddyguard [2]. Based on that work, we then substantially improved the Buddyguard on multiple aspects. In this report, we will mainly focus on the enhancements we have done.

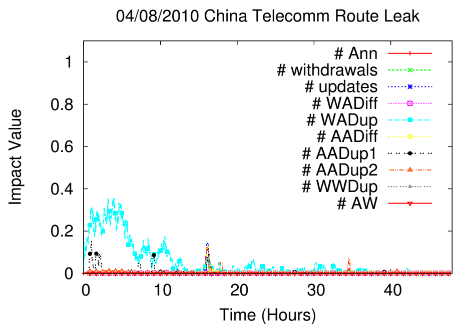
A. Problem Description

BGP is designed to exchange information on how to reach any IP prefixes (i.e., blocks of IP addresses), and the correctness of the update information is critical. An IP prefix can be subject 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 ([29], [30]), including the well-known Pakistan Telecom hijack of YouTube in 2008 [31] and the recent Icelandic ISP hijacking US traffic [32]. Another common prefix anomaly is **route leak**, where a misconfigured ISP advertises illegitimate routes for prefixes. In the past, such incidents have caused anomalies on a monumental scale [33], [34]. On April 8, 2010, an AS operated by China Telecom falsely originated nearly 37,000 prefixes, causing blackouts for those prefixes for about 15 minutes [22]. More recently in

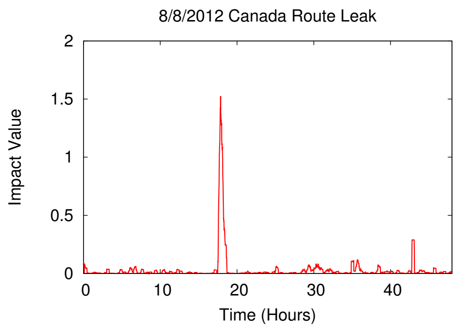


(a) 2010 China Telecom route leak, overall impact.

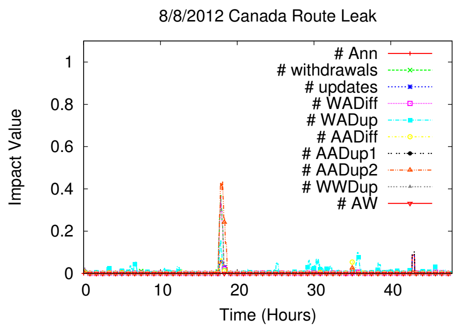


(b) 2010 China Telecom route leak, per-attribute impact.

Fig. 3: **I-seismograph monitoring results for 2010 China Telecomm route leak event.**

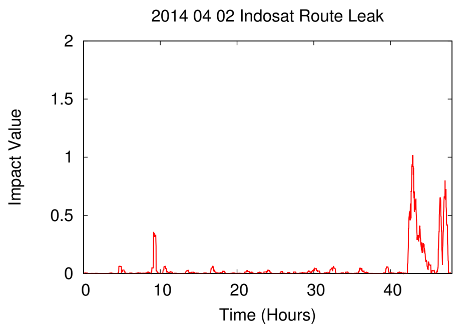


(a) 2012 Canada Dery Telecom route leak, overall impact.

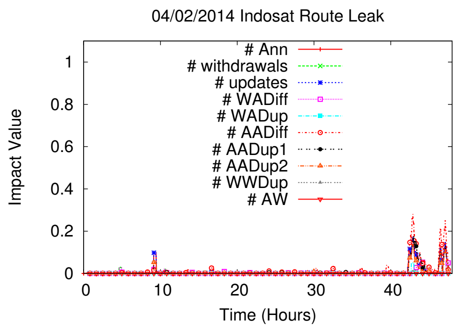


(b) 2012 Canada Dery Telecom route leak, per-attribute impact.

Fig. 4: **I-seismograph monitoring results for 2012 Canada Dery Telecom route leak event.**



(a) 2014 Indonesia Indosat route leak, overall impact.



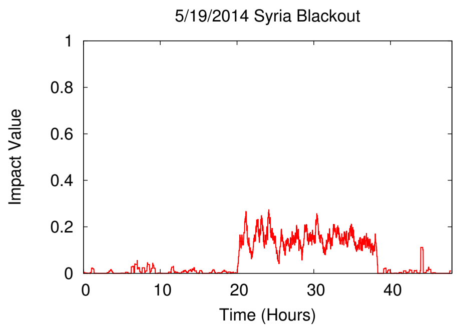
(b) 2014 Indonesia Indosat route leak, per-attribute impact.

Fig. 5: **I-seismograph monitoring results for three 2014 Indonesia Indosat route leak event.**

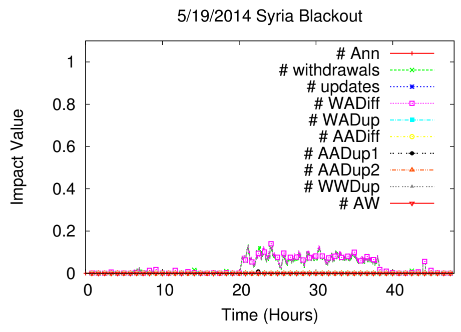
August 2012, Canadian ISP Dery Telecom Inc. leaked a large amount of routes that misrouted traffic to 107,409 prefixes in 14,391 ASes [23], or about one third of the entire Internet.

These anomalies threaten every prefix on the Internet, whether commercial or private. Even more disturbing, users or operators of a prefix cannot easily detect such incidents. The legitimate user of a prefix may not expect any traffic at all while its incoming traffic is being misrouted; 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. Given that traffic still arrives at victim prefix(es), this type of attack is more discrete than normal prefix hijacking or large-scale route leaks, and is becoming more frequent [32].

Unfortunately, it is unlikely that these prefix anomalies will be resolved in the near future. While there exist proposals such as BGPSEC [3] to secure the Border Gateway Protocol (BGP), their high overhead cost has prevented actual deployment. In fact, even these proposals are deployed, they will still fail in certain critical circumstances [35], [4]. It is therefore critical to monitor prefixes and detect prefix anomalies as they occur within the current BGP environment.

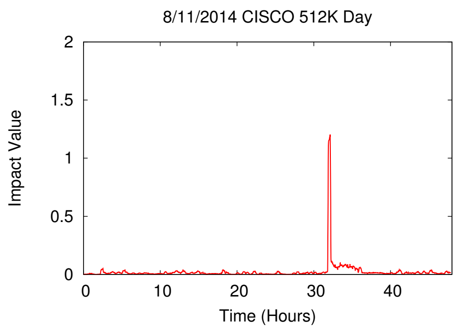


(a) 2014 Syria blackout, overall impact.

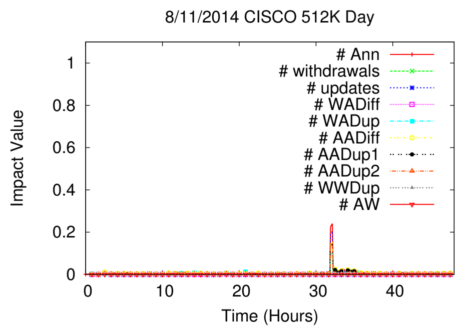


(b) 2014 Syria blackout, per-attribute impact.

Fig. 6: **I-seismograph monitoring results for 2014 Syria blackout event.**

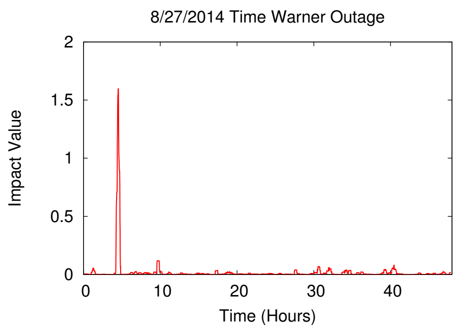


(a) 2014 CISCO ”512K” day large-scale equipment malfunction, overall impact.

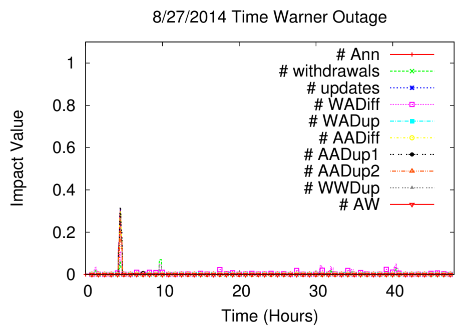


(b) 2014 CISCO ”512K” day large-scale equipment malfunction, per-attribute impact.

Fig. 7: **I-seismograph monitoring results for 2014 CISCO ”512” day event.**



(a) 2014 Time Warner large-scale outage, overall impact.



(b) 2014 Time Warner large-scale outage, per-attribute impact.

Fig. 8: **I-seismograph monitoring results for 2014 Time Warner outage event.**

B. Buddyguard 1.0 - An Early Solution

In order to better conduct the prefix level BGP monitoring, [2] presents the Buddyguard system. The goal of Buddyguard is to fast and accurately detect unusual changes of the BGP paths toward individual prefixes. 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 detect various prefix anomalies including prefix hijacking and route leaks, but it is also lightweight to deploy and resilient against circumvention by attackers.

The key to monitoring an IP prefix is to know 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 which 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 a 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. After ensuring that there are enough buddies for a prefix, we can use these buddies to determine whether or not the prefix is experiencing anomalies. Under normal conditions a prefix is similar to most of its buddies (in terms of the behaviors being monitored) but not so if under abnormal situations.

At its core, the Buddyguard architecture includes the monitored prefix, a set of **monitors**, and the **buddies** of every monitored prefix (Figure 9). 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.

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 [36] collectors or BGPMon [17], 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;
(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.

In [2], we demonstrate the efficacy of Buddyguard by testing our system on well-known prefix hijackings 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 [5] collectors, we train Buddyguard by observing routes from these monitors to the prefix and select buddies that best match these routes. The evaluations show that monitoring prefixes with these buddies provides fast, accurate, and reliable detection with low false negatives and false positives.

C. Buddyguard 2.0 - Overview

Based on the early solution, we seek for further improvement to make Buddyguard even better and more helpful. In specific, we would like to see our system to have better analysis on the detected events, more capabilities, and more thoroughly evaluated. Accordingly, we made multiple enhancements that improves the Buddyguard from both architecture and performance. We hence call it "Buddyguard 2.0". The enhancements include:

(i) *Better architecture and more capabilities.* From the architecture aspect, we added a central controller that aggregates and analysis the individual monitor's results. In addition to detecting the prefix anomalies, we added the functionality to trace the likely origin of an anomaly.

For each detected event, we also analyze the scope of its impact.

(ii) *An overhauled set of algorithms.* We have improved the algorithms for training and monitoring procedures. For training phase, we introduced adaptive training and relative prefixes training, and improved the previous training algorithm for robustness. For monitoring phase, we improved the algorithm and significantly improved the sensitivity toward prefix anomalies without increasing the false positive rate.

(iii) *Thorough evaluation against most recent events.* We had also conducted a significantly more thorough evaluation of Buddyguard, including performance, overhead, and new results in tracing the anomaly origins and scope of the event's impact.

(iv) *Low overhead.* We also showed that Buddyguard has low overhead and can scale to monitor all prefixes on the Internet with relatively low overhead.

Based on our previous work on Buddyguard 1.0 [2], we have made multiple enhancements on its architecture, functionalities, core algorithms, and evaluations. Together with all the enhancements, the Buddyguard system can accurately detect prefix-level routing anomalies with root-cause analysis for each incidence.

D. Smart Controller and New Capabilities

At its core, the Buddyguard 2.0 architecture includes monitored prefixes, buddies of every monitored prefix, a set of monitors, and a dedicated controller. For every monitored prefix, a Buddyguard monitor first runs a training process and then applies the training result to the monitoring phase. During training, it employs a set of training algorithms to discover and select best-matching buddies for the prefix. During monitoring, it runs the monitoring algorithm to detect if the prefix experiences anomalies using its buddies. Beyond conducting monitoring tasks by individual monitors, we introduced the *controller* component. The monitor will periodically update information to the controller. The updates convey monitoring results and suspicious activities. If an anomaly occurs, Buddyguard controller can then gather information from all monitors, and analyze the event for more information, including the event's scope of impact and possible origins of the anomaly.

1) A Smart Controller:

Whereas each monitor can detect anomalous situations from its vantage point on its own, Buddyguard achieves better detection accuracy by further aggregating the detection results from all monitors. It does so by asking every monitor to feed the controller a warning signal if the monitor detects that a monitored prefix is experiencing an anomaly type that the customer is interested in, and then counting how many monitors issued a warning to determine whether or not to issue a system-wide alert that the prefix is indeed experiencing an anomaly. With further details provided from monitors, the controller can further assess the scope of an anomaly and even trace the origin of the anomaly.

A concern may raise that the controller may become a performance bottleneck or a single point of security failure.

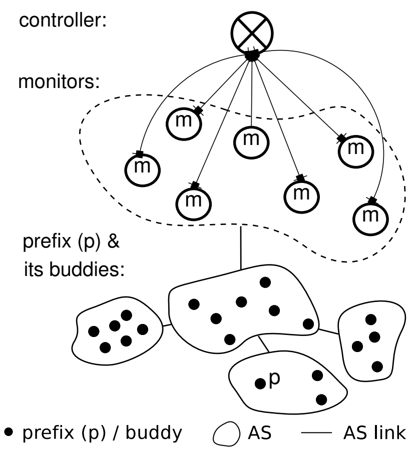


Fig. 9: **Buddyguard architecture.**

However, the controller will mostly be idle unless it begins to hear warning messages from monitors. Even then, the controller will only incur a light overhead of messages between itself and *all* monitors, and light storage and computational overhead when processing the messages. It is true that the controller may become a single point of failure, but the controller can be well-protected and can also have one or multiple back-up controllers. Note even without a controller, every monitor can still independently operate.

2) Prefix Anomaly Analysis at the Controller:

Based on our new architecture, we can then add new functionalities to Buddyguard to further help users for decision making and root cause analysis. Besides the detection results for the prefixes of interests, understanding the cause of the event and the propagation of the anomalies could largely help a user of Buddyguard to decide the actions to react upon the anomalies. Therefore, we brought new functionalities to Buddyguard and enhance its usabilities, including (1) global impact scope estimating, and (2) root-cause tracing capability.

While every monitor can independently check whether an update toward a monitored prefix is anomalous, Buddyguard relies on its controller to aggregate the information from monitors to determine whether or not an anomaly is occurring. In monitoring prefix hijacking and route leaks, once a monitor detects an update is likely a hijacking or route leak, it sends a warning to the controller. The controller can then count how many warnings it hears regarding a monitored prefix, denoted as $N_{warning}$. To decide whether an anomaly is indeed happening, the controller further checks how many monitors have heard an update about the monitored prefix at approximately the same time, denoted as N_{heard}, and see what fraction of these monitors issue warnings. If $\frac{N_{warning}}{N_{heard}}$ is more than the alert threshold ρ, the controller will regard the monitored prefix is experiencing an anomaly, and issue an alert. Note while the controller may hear a warning from a monitor about a prefix, later the same monitor may report to the controller that the prefix is back to normal, allowing the controller to discount the warning from this monitor.

The controller obtains the value of N_{heard} by communicating with every monitor using the **query** and **report** messages over an encrypted channel that is protected with a preset secret key. The **query** message **(prefix=p, timestamp=t, offset=Δ)** allows the controller to ask every monitor whether it has heard a new path to prefix **p** in the time window $[t \pm \Delta]$. The **report** message **(monitor_id=m, prefix=p, AS_path=AP, timestamp=t, suspicious=yes/no)** allows monitor **m** to report to the controller that at time **t** it heard a new AS path **AP** to prefix **p**, and whether or not the monitor views the path as suspicious. Clearly, a monitor can also use the report message to send a warning to the controller where the **suspicious** field is **yes**, or to notify the controller that it no longer treats a prefix in the "warning" state where the **suspicious** field is **no**. Now, to obtain the value of N_{heard}, as soon as the controller receives the first warning from a monitor regarding a monitored prefix, say **(monitor_id=m_x, prefix=p_x, AS_path=AP_x, timestamp=t_x, suspicious=yes)**, it sends a query message **(prefix=p_x, timestamp=t_x, offset=Δ)** to every other monitor. If a monitor does not hear a new path to prefix p_x in $[t_x \pm \Delta]$, it does nothing. Otherwise, the monitor checks if the new path is suspicious, and sends back a report message to the controller accordingly. (If prior to receiving the query message a monitor has already sent a warning to the controller, it does not have to resend the warning.) With all the report messages it receives, the controller can then calculate the value of N_{heard}.

Once the controller determines an anomaly is occurring, in addition to issuing an alert about the anomaly, the controller can further trace the origin of the anomaly and analyze the scope of its impact. We now look at how the controller can trace the origin of a prefix hijacking or route leak event and calculate the scope of impact from the event. While $N_{warning}$ monitors heard an anomalous path and raised a warning, the router that initiates the illegitimate routing update—either a hijacking update or an update that leaks the path to a prefix—is located at a specific AS, i.e., the origin AS of the anomaly. The illegitimate update then gets propagated throughout ASes on the Internet, including reaching the $N_{warning}$ monitors that raised warnings. The controller can then aggregate the warnings and deduce the origin AS of the anomaly. Recall every warning message from a monitor also includes an AS path from the monitor to the prefix that the monitor regards as illegitimate. The key point is that the origin AS of the anomaly must be on every illegitimate path. The controller thus can find out the intersection of all the AS paths reported in warning messages, i.e., ASes that exist on all these paths, and the origin AS of the anomaly must be one of them. Fig. 10 shows an example: Based on the monitors reporting warnings (not showing all monitors), the controller can determine that the illegitimate updates originated from either AS 3491 (PCCW [37]) or AS 17557 (Pakistan Telecom [38]). The customer (such as the owner of the victim prefix) can then contact out-of-band each AS in the set for further investigation.

Among all "quiet" monitors who heard an AS path to the monitored prefix but did not report a warning (totally N_{heard} - $N_{warning}$ of them), the tracing procedure above can also help identify which monitors actually also received an illegitimate path (false negative) and which did not (true negative). Let N_{F-} denote the number of the former, and N_{T-} the number of the latter, where N_{heard} = $N_{warning}$ + N_{F-} + N_{T-}. The procedure is simple: if the intersection above also appears in a monitor's AS path, the path is then illegitimate, and the monitor had a false negative. For example, in Fig. 10, all quiet monitors located in grey ASes also heard a hijacking update since they all came from the ASes in the intersection.

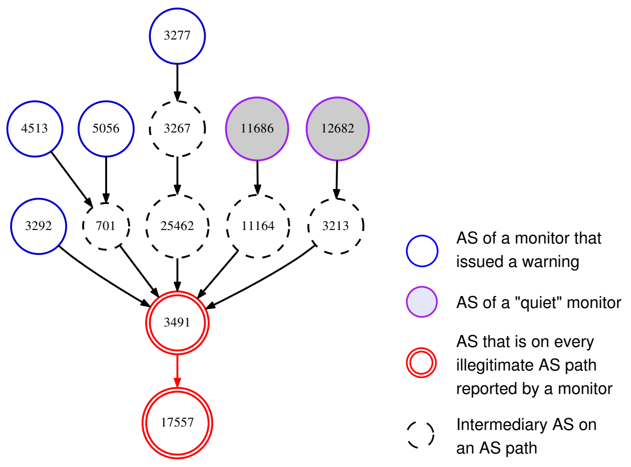

3277
4513
5056
3267
11686
12682
3292
701
25462
11164
3213
3491
17557
AS of a monitor that issued a warning
AS of a "quiet" monitor
AS that is on every illegitimate AS path reported by a monitor
Intermediary AS on an AS path

Fig. 10: **Tracing the origin AS of YouTube prefix hijacking using the AS paths reported in warnings from monitors** (not all monitors are shown).

The controller can then calculate the scope of impact from a prefix hijacking or route leak event. Among all the monitors, it can easily calculate the percentage of monitors who have heard an illegitimate update, which is the sum of $N_{warning}$ and N_{F-}. Because Buddyguard monitors are distributed among a diverse set of ASes on the Internet, the percentage value provides a good-faith estimation of what portion of the Internet suffered from the event.

E. Training and Monitoring Algorithms

The success of Buddyguard lies in having the best possible set of buddies for every monitored prefix. We meet this objective through the training process, in which we define algorithms for (1) collecting buddy candidates that potentially match the behavior of the monitored prefix , (2) selecting the best matching candidates to be actual buddies. We then use the best selection of the buddies to detect the anomalies during monitoring phase.

We have improved the existing algorithms for both training and monitoring algorithms. In specific, we introduced

1) an adaptive buddy candidate search algorithm to help obtaining potential buddies in various situations,
2) a better buddy selection algorithm with formally defined parameters,
3) a relative prefix training algorithm to obtain buddies even when there were only updates for super and sub-prefixes of the monitored prefix.
4) an improved monitoring algorithm with significantly more sensitivity to anomalies without increasing the false positive rate.

In this section, we will introduce these enhanced algorithms.

1) Adaptive Candidate Searching:

In Buddyguard 1.0, we only search for the potential buddies in a fixed-size **buddy candidate zone** (the definition will be introduced in the following section) and a fixed-size time period. Now with the consideration of more generic cases, where the size of each buddy candidate zone various in terms of the number of prefixes it contains, and the frequency of the updates for each prefix various, the Buddyguard 2.0 introduced the *adaptive candidate searching algorithm.*

- From the space dimension, it automatically selects the best fit buddy candidate zone for selecting potential buddies.
- From the time dimension, it automatically expands its training time period depending on the needs.

We will introduce the details in the following content.

A monitor obtains buddy candidates for a monitored prefix by applying a "fate-sharing" principle. Whenever the monitor hears a new path toward the monitored prefix, it finds prefixes toward which the monitor has also heard a *similar* path at roughly the same time, and adds these prefixes as buddy candidates for the monitored prefix. Consider the AS path u_p from a monitor to a prefix p, which consists of an ordered list of $|u_p|$ ASes from the monitor to p. (We designate $|x|$ as the length or size of x both here and in the remainder of this work.) We can define another AS path u_c from the monitor to a prefix c as similar to u_p—which we denote $u_c \sim u_p$—if two paths share the first $|u_p| - n$ AS hops (starting from the monitor's AS) where n is an integer indicating that the last n hops of u_p are ignored for similarity comparison. Furthermore, if $u_c \sim u_p$ and the monitor heard u_c and u_p almost simultaneously, say within time Δ from each other, the monitor can then use prefix c as a buddy candidate of prefix p.

The value of n directly affects the distribution of buddy candidates, or which ASes may be eligible to offer buddies. We define the set of all the ASes from which a monitor can choose buddy candidates for prefix p as p's **buddy candidate zone**, denoted by $Z_p(n)$. Fig. 11 shows three examples with n being 0, 1, or 2, respectively. When n is 0, buddies can only be from the same AS as the monitored prefix, i.e., the **origin AS** (Fig. 11(a)), or its descendent ASes (not shown); in other words, when n is 0, $Z_p(n)$includes only the origin AS and its descendent ASes. When n is 1, buddies can also be from—i.e., the zone can also include—the so-called **parent AS** and **sibling AS**es (Fig. 11(b)), or their descendant ASes (not shown). Clearly, n=2 will further expand $Z_p(n)$.

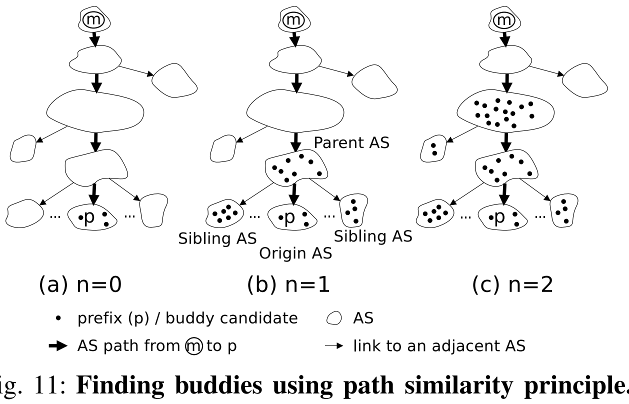


Fig. 11: **Finding buddies using path similarity principle.** Last n hops on the AS path from m to p are ignored for path similarity comparison.

Algorithm IV.1: CANDIDATESEARCH(u_p, t_{u_p})

```
output ($C_{u_p}$: set of buddy candidates)
local $n$; $fromASes$: ASes of buddy candidates
      $Z_p(n)$: p's buddy candidate zone
$C_{u_p} = \phi$; $fromASes = \phi$; $Z_p(-1) = \phi$; $n = 0$;
while $|C_{u_p}| \le \alpha$ or $|fromASes| \le \beta$
  do
    for each AS $x$ in $Z_p(n)$-$Z_p(n$-1)
      do
        for each prefix $c$ in AS $x$
          do
            if $u_c \sim u_p$ and $t_{u_c} \sim t_{u_p}$
              then
                add $c$ to $C_{u_p}$;
                add $x$ to $fromASes$;
    if $|u_p| - n \le \lambda$ or $Z_p(n)$ includes any tier 1 ASes
      then return ($C_{u_p}$);
    ++$n$;
return ($C_{u_p}$);
```

Fig. 12: **Buddy candidate searching algorithm.** u_p is the update of prefix p, and t_{u_p} is the time stamp of u_p. Thresholds α, β, λ are the minimum number of buddy candidates, the minimum number of ASes of buddy candidates, and the minimum number of shared hops in path comparison.

The monitor aims to find enough buddy candidates from many different ASes for *any* IP prefix on the Internet, but some prefixes may be surrounded by many ASes and thus many buddy candidates in its close neighborhood (e.g., a prefix from a tier-1 or tier-2 ISP usually has many sibling ASes), and some may be in a sparse area (e.g., a prefix whose ancestor AS in the recent several generations only has very few descendant ASes). The monitor thus adopts an *adaptive* buddy candidate search strategy with a varying value of n, instead of a fixed value. The strategy works as follows, as depicted in Fig. 12. During the training period, every monitor listens to new path announcements for the monitored prefix p. Whenever a monitor hears a new AS path to prefix p, say u_p at time t_{u_p}, it does the following. First, the monitor sets n to be 0, and follows the fate-sharing principle above to search all eligible buddy candidates from the current buddy candidate zone (the origin AS and its descendant ASes). If the monitor finds many buddy candidates *and* these candidates are from many different ASes, it is done obtaining buddy candidates. Otherwise, the monitor increases n by 1, thus increasing the size of p's buddy candidate zone, and searches from the newly added ASes to add all the new eligible buddy candidates. If the monitor still does not obtain enough buddy candidates from a diverse set of ASes, it will increase n by 1 and repeats this process again until it does. Note that a larger buddy candidate zone may imply a higher false negative; if a hijacker in the zone announces a fake path to a monitored prefix to hijack it, the monitor will not detect the hijacking.

Every monitor ensures it has enough skewers to compare the frequency of buddy candidates in matching a monitored prefix and every relative prefix. It begins with a one-week window, but if it does not have at least ξ BGP updates of the monitored prefix, it then expands the training period to search for more updates regarding the prefix. If by expanding its training period to more than four weeks the monitor still cannot obtain enough updates, it will simply give up and stop being a monitor for the prefix in question. Because Buddyguard relies on the collective efforts of its monitors, whereas every monitor does its best effort, it does not require every monitor to be able to monitor a given prefix.

2) Buddy Selection Algorithm:

Once a monitor has obtained the buddy candidates for a monitored prefix during the training period, it needs to decide which candidates to select to be the buddies for the prefix. The most frequently matching candidates found during this training period are clearly the best-matching, but in selecting buddies, the monitor further needs to consider the criteria that require (1) there are always enough buddies matching the monitored prefix, and (2) the buddies are distributed across multiple ASes (thus resilient to co-hijacking). Comparing to 1.0 version, Buddyguard 2.0 version introduced a more formal algorithm with all parameters clearly defined. The details of the algorithm is described as follows.

We employ a **skewering algorithm**, with its pseudo code in Fig. 13. During the training period, whenever monitor m hears a path update u_p at time t_i, it creates a **skewer** data structure for time t_i. By the end of training, the monitor will have a set of skewers $S = \{s_{t_i}\}$, each skewer corresponding to an update $u_p(t_i)$ that m witnessed (Fig. 14). We then "skewer" candidates by sorting them based on frequency of matching (best to worst), and place the best-matching candidate c on each skewer s_{t_i} where $u_c(t_i \pm \Delta) \sim u_p(t_i)$.

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

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

for some lower bound capacity ω. This ensures that buddies can account for the full range of normal behavior for the monitored prefix; in other words, every time the monitor hears a new path to a prefix, there are at least ω buddies of the prefix all of which will also have a new, similar path. We can also ensure that buddies are widely distributed by skewering candidates until they cover at least ψ ASes (we use ψ=10 in our current configuration). Through experiments we find that more than 70% of the monitors would have buddies cover at least 10 ASes, and more than half of them will cover at least 100 ASes. Once these conditions are met, Buddyguard selects all skewered candidates as buddies.

Algorithm IV.2: SKEWERINGALGORITHM(p)

```
output ($B_p$: buddies of prefix $p$)
local $C$: candidates; $S$: skewers; $A$: ASes seen already
$C = \phi$; $S = \phi$; $A = \phi$; $B_p = \phi$;
for each update $u_p(t_i)$ seen at time $t_i$
    do { $C = C \cup \{\text{CANDIDATESEARCH}(u_p(t_i), t_i)\}$;
         create skewer $s_{t_i}$ for $u_p(t_i)$ : $s_{t_i} = \phi$;
         $S = S \cup \{s_{t_i}\}$;

while $\exists s_{t_i} \in S$ $s.t.$ $|s_{t_i}| < \omega$
    do { choose the most frequent $c$ in $C$ where
         $u_c(t_i \pm \Delta) \sim u_p(t_i)$ :
             $s_{t_i} = s_{t_i} \cup \{c\}$;
         $\forall$ skewer $s_{t_j(j \neq i)}$ where $u_c(t_j \pm \Delta) \sim u_p(t_j)$ :
             $s_{t_j} = s_{t_j} \cup \{c\}$;
         $B_p = B_p \cup \{c\}$;
         $A = A \cup \{AS(c)\}$;    ($AS(c)$ is the AS of $c$.)

while $|A| < \psi$ (i.e., $B_p$ is not diverse)
    do { find the most frequent $c_{new} \in C$
             where $AS(c_{new}) \notin A$;
         $B_p = B_p \cup \{c_{new}\}$;
         $A = A \cup \{AS(c_{new})\}$;

return ($B_p$);
```

Fig. 13: **The skewering algorithm at monitor m, where p is a monitored prefix.** Thresholds: ω is the minimum number of buddies per skewer; ψ is the minimum number of ASes of the finally selected buddies.

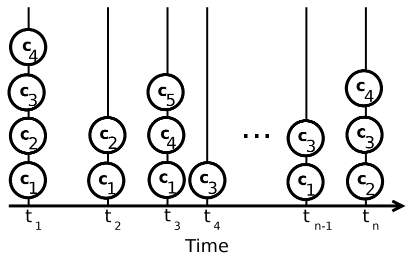


Fig. 14: **Skewers in the training process.** A skewer represents a path change for a monitored prefix at time t_i. Each circle represents a buddy c_j.

3) Relative Prefixes Training: In Buddyguard 1.0, we only takes the monitored prefixes into consideration when training, where in some cases the prefixes may be aggregated or only the sub-prefixes were announced during the training period, and therefore Buddyguard cannot hear any updates to the monitored prefixes. In order to cover the situations like this, we introduced the *relative prefixes training* algorithm, where we take the "relative prefixes" of the monitored prefix also into consideration when training. The detail is derived as follows.

During our study, we note that when training a prefix, a monitor not only needs to train the prefix itself, but also its *relative* prefixes, and learn the buddies of every relative prefix using the same procedure described above. The relative prefixes of a monitored prefix p includes (i) all sub-prefixes of p that are heard by the monitor during the training period where every sub-prefix is a subset of p, and (ii) the longest super-prefix of p, which is a superset of p, that the monitor

Algorithm IV.3: RELATIVEPREFIXTRAINING(p)

```
local $R$: relative prefixes of $p$; $B_x$: buddies of prefix $x$;
output ($B_R$: all relative prefixes of $p$ and their buddies)
$R = \phi$; $B_R = \phi$;
for each update $u_x(t)$ seen at time $t$ for prefix $x$
  do
    if $x \subset p$ and $x \notin R$ ($x$ is a sub-prefix of $p$)
      then $R = R \cup \{x\}$;
    if $x \supseteq p$ and $x \notin R$ ($x$ is a super-prefix of $p$)
      then
        if $\nexists\ y \in R\ s.t.\ y \supseteq p$
          then $R = R \cup \{x\}$;
        else if $\exists\ y \in R\ s.t.\ y \supset x$
          then replace $y$ with $x$ in $R$;
for each prefix $x$ in $R$
  do
    $B_x$ = SKEWERINGALGORITHM($x$);
    $B_R = B_R \cup \{\langle x, B_x \rangle\}$;
return ($B_R$);
```

Fig. 15: **Algorithm for collecting and training the relative prefixes of a monitored prefix p.**

hears *if* the monitor does not hear p itself during training. For example, 208.65.153.128/25 and 208.65.153.0/25 are two sub-prefixes of 208.65.153.0/24 (each /25 prefix is a subset of the /24 prefix) and 208.65.152.0/22 is a super-prefix of 208.65.153.0/24 (the /22 prefix is a superset of IP addresses of the /24 prefix). Recall every monitor is also a BGP router and follows the longest prefix matching principle. Once a monitor has trained a prefix x, i.e., the monitor has processed every AS path it has toward x and obtained buddies for x, it can use the buddies of x to monitor not only x, but also any sub-prefix s of x if the monitor has not heard anything about another prefix y such that i.e., $s \subseteq y \subset x$. If the monitor has heard about y, because y is more specific than x—note y could also be exactly s—it then uses the buddies of y to monitor s. Therefore, if the monitor has not heard anything about the monitored prefix p itself, it will train the longest super-prefix of p that the monitor has heard. Furthermore, the monitor will train all sub-prefixes of p—if any— that the monitor has heard during training. Doing so, the monitor can monitor p using the buddies of p or the buddies of the super-prefix (if it has not heard p), and it can also monitor *any* sub-prefix s of p whether it has heard s or not during training (by using the buddies of the longest super-prefix of s). Fig. 15 describes how a monitor runs training for all its relative prefixes.

4) Monitoring Algorithm:

While in Buddyguard 1.0 the monitoring tasks are conducted by only the monitors, the monitoring process of Buddyguard 2.0 happens at two levels instead: the monitor level and the controller level. At the monitor level, every monitor independently processes BGP updates toward every monitored prefix and their buddies to determine whether or not the prefix is experiencing an anomaly, and if so, sends a warning signal to the controller. At the controller level, the controller aggregates warning signals from all monitors to determine whether an anomaly occurs and if so issue an alert about the anomaly. The controller can further analyze an anomaly's scope of impact and possible origin. Like what we did with the training process description, to better illustrate how monitoring works, below we assume the monitored prefix anomalies are prefix hijacking and route leaks.

For every monitored prefix, after a monitor has identified buddies of the prefix through the training process, during the monitoring process it uses these buddies to detect if an anomaly occurs to the prefix. Assume we are monitoring prefix hijacking and route leaks. Whenever a monitor hears a new path to a prefix p, say at time t, it will check whether this is one of the "good paths" that it has already seen during the training phase. If not, the monitor will wait for a short period of Δ time, and then check if more than φ buddies of p also switch to a similar new path within the time period $t \pm \Delta$. If not, the monitor sends a warning report to the controller, indicating that the prefix may be hijacked. The monitor will also switch from "normal" state to "warning" state in terms of this prefix.

Defining the warning threshold φ would seem to be a difficult task. How can a monitor know how many buddies will typically match a 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 from training, which correspond to legitimate paths from monitor m to p, and look at the number of buddies on each skewer as a random variable. This random variable shows every time when p had a legitimate path change, how many buddies also switched to a new similar path. (We would expect many buddies would also switch since that is why they are considered buddies.) If the number of buddies on each skewer follows the normal distribution, then we can define $\varphi = \mu - 3\sigma$ where μ and σ are the mean and standard deviation of buddies per skewer, respectively, and the probability of only having φ or less buddies matching p is almost zero (roughly 0.0026). If the monitor hears a new path to a prefix but no more than φ of its buddies switching to a similar new path, the monitor can then treat the new path as very likely illegitimate.

If the monitor did not hear the prefix during the training, the monitor will use the buddies of its super-prefix, a relative prefix the monitor learned during training, to decide whether a newly heard path to the prefix (or the super-prefix) is illegitimate. Furthermore, if the monitor hears a new path to a sub-prefix of the monitored prefix, it will find the most specific relative prefix that includes the sub-prefix, and use the buddies of this relative prefix to verify the new path.

A monitor may switch from the "warning" state back to the "normal" state. After it sends a warning to the controller that it heard an illegitimate path to a prefix, after some time it may hear a new routing update about the prefix. The monitor will verify this update, using the same aforementioned procedure. If it does not find the route advertised in this update illegitimate, it will switch back to the "normal" state, and send a report to the controller that this prefix is normal now.

F. Apply Buddyguard 2.0 on Real-World Events

To evaluate Buddyguard 2.0 on today's Internet, we used 273 BGP speakers that peer with RouteViews collectors [5] as BGP monitors. To run the training and monitoring process, we used the BGP updates from these BGP speakers for 490,000 prefixes of interest during various time periods (totaling 125 days) from 2005 to 2013.

For evaluating Buddyguard's accuracy in detecting prefix hijackings, we first run the training process for every prefix that was later hijacked, and then run the monitoring process to see if Buddyguard can detect the hijacking. For accuracy in detecting route leaks, if hundreds or even thousands of prefixes may be affected, we select prefixes randomly from diverse locations, including tier 1, tier 2, and tier 3+ ASes.

We evaluated Buddyguard's ability to detect prefix hijacking by testing it on a wide manner of hijackings. For brevity, we show the results for four well-known hijacking events:

- [05-Cogent-Google] Cogent's hijack of one of Google's prefixes [30].
- [06-ConEd-MStewart] Con Edison's hijack of 30+ prefixes, including some belonging to their customers [29].
- [08-Pakistan-YouTube] Pakistan Telecom's hijack of a sub-prefix of YouTube's prefix [31].
- [13-Valitor-CenturyTel] A prefix interception event, Icelandic ISP Valitor hf intercepted traffic to a CenturyLink prefix [32], [39].

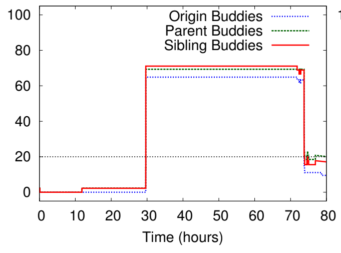
Our results from these hijacking events (Fig. 16) show that Buddyguard is well-suited to detecting prefix hijackings. For these events we found it sufficient to only use buddies from origin, parent, and sibling ASes, which we also call origin, parent, and sibling buddies, respectively. We show the results with origin, parent, and sibling buddies separately. For all hijacking events, to detect them it suffices to use $\rho = 25\%$ as the threshold of the minimum percentage of monitors raising warnings.

We also evaluated Buddyguard's accuracy in detecting route leaks (shown in Fig. 17). We picked three recent and well-known route leaks:

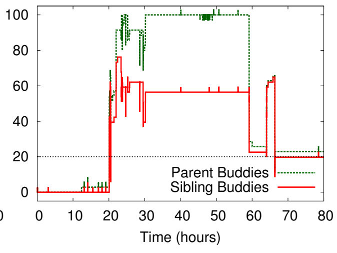
- [10-China-Routeleak] China Telecom route leak [22];
- [12-Canada-Routeleak] Canada Dery Telecom route leak [23]; and
- [12-Indonesia-Routeleak] Indonesia Moratel route leak [40].

Based on the details reported from each monitor, we can further trace the origin of a hijacking or route leak event and investigate its scope of impact. For every event we studied in this section, Table III shows which ASes are likely the origin of a hijacking or leaking event (we choose multiple victim prefixes in a route leak event). These ASes are on the intersection of all illegitimate AS paths from monitors to a victim prefix. For a route leak event, we can further narrow down the possible origin ASes of the leak to be the intersection of every victim prefix's possible origin ASes. In 12-Canada-Routeleak, for example, this would be $577 \rightarrow 46618 \rightarrow 5769$.

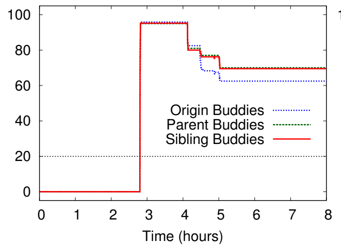
Calculating the scope of the impact is straightforward. Given a total of 273 monitors, Table III further shows the percentage of monitors that heard the hijacking or route leak



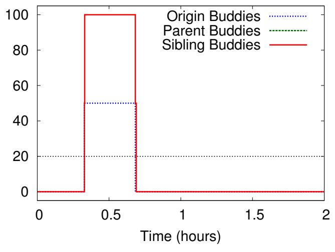
(a) Cogent hijacked a Google prefix (64.233.161.0/24). Time 0 is 5/6/05 at 09:00:00 UTC. The hijack began 5/7/05 at 14:37:56 UTC [30].



(b) Con Edison hijacked Martha Stewart Living (12.173.227.0/24). Time 0 is 1/21/06 at 09:00:00 UTC. The hijack began 1/22/06 at 05:05:33 UTC [29].



(c) Pakistan Telecom hijacked YouTube (208.65.152.0/22). Time 0 is 2/24/08 at 16:00:00 UTC. The hijack began at 18:47:57 UTC [31].



(d) Valitor hf hijacked a CenturyTel prefix (206.51.64.0/20). Time 0 is 8/19/2013 at 14:00:00 UTC. The hijack began at 14:19:40 UTC [32].

Fig. 16: **Detection results of prefix hijacking events.**

(the sum of $N_{warning}$ and N_{F-} divided by 273), i.e., the scope of the impact from an event. Clearly, some events have a larger impact than others (e.g., 08-Pakistan-YouTube), and some only propagate to a small region of the Internet (e.g., 12-Indonesia-Routeleak).

G. Overhead Estimation

Given that Buddyguard aims to be an Internet-scale monitoring system, in this section we demonstrate that Buddyguard scales with the number of monitored prefixes in terms of the memory, storage, and network overhead.

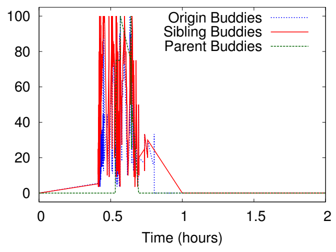
1) Affordable Memory and Disk Storage Overhead:

We now analyze the memory and disk storage overhead at every monitor. (We skip the analysis for the controller due to the very low overhead at the controller.) We show Buddyguard incurs a reasonable overhead during both the training phase and the monitoring phase.

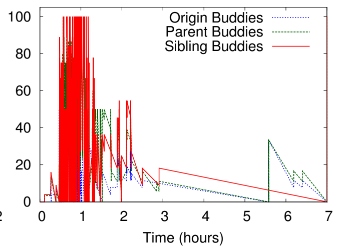
During the training process, a monitor can adopt a sliding time window mechanism to run Algorithm IV.1 (Fig. 12) in order to discover the buddy candidates. Assume we are monitoring *all* the IP prefixes on the Internet. As a monitor sequentially processes BGP updates during the training period, every time a monitor hears a BGP update to a monitored prefix, say at time t, it only needs to use updates from a time window of $[t-\Delta, t+\Delta]$. The primary memory cost for Algorithm IV.1 is thus to store these updates, particularly the AS path and the prefix of each update. The total is thus $\sum_i (1 + |u_i| * 4 + 4)$ bytes, where every update u_i has $|u_i|$ AS hops and we use 1 byte to record AS hop count, every hop is a 4-byte AS

TABLE III: **Tracing and scope of the impact of prefix hijacking and route leaks events.**

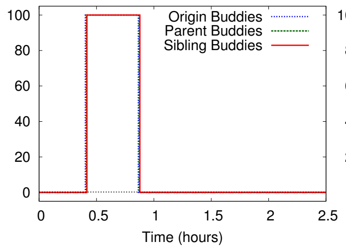
Events	Victim prefix	N_{heard}	$N_{warning}$	N_{F-}	N_{T-}	Scope of impact (%)	Possible origin ASes of the event
05-Cogent-Google	64.233.161.0/24	45	32	4	9	13.2	174
06-ConEd-MStewart	12.173.227.0/24	35	24	0	11	8.8	27507
08-Pakistan-YouTube	208.65.152.0/22	120	115	0	5	42.1	3491 $\rightarrow$ 17557
13-Valitor-CenturyTel	206.51.64.0/20	2	2	0	0	0.7	6677 $\rightarrow$ 47828
10-China-Routeleak	175.111.114.0/23	27	27	0	0	9.9	4134 $\rightarrow$ 23724
	83.139.14.0/24	19	19	0	0	7.0	4134 $\rightarrow$ 23724
12-Canada-Routeleak	197.159.80.0/20	30	30	0	0	11.0	577 $\rightarrow$ 46618 $\rightarrow$ 5769 $\rightarrow$ 8657
	186.2447.192.0/19	30	26	0	4	9.5	577 $\rightarrow$ 46618 $\rightarrow$ 5769 $\rightarrow$ 2119
	8.8.8.0/24	2	2	0	0	0.7	577 $\rightarrow$ 46618 $\rightarrow$ 5769 $\rightarrow$ 15169
12-Indonesia-Routeleak	8.8.8.0/24	2	2	0	0	0.7	3491 $\rightarrow$ 23947



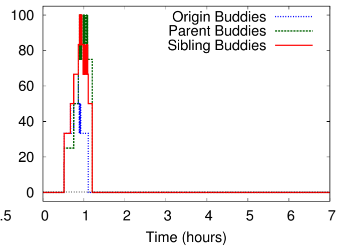
(a) Aggregated detection result of China Telecom route leak (100 randomly selected affected prefixes).



(b) Aggregated detection result of Canada Dery Telecom route leak (30 randomly selected affected prefixes).



(c) Detection result of Indonesia Moratel route leak (with 8.8.8.0/24 as the affected prefix).



(d) Detection result of Canada Dery Telecom leaking route of prefix 8.8.8.0/24.

Fig. 17: **Detection results of route leak events.**

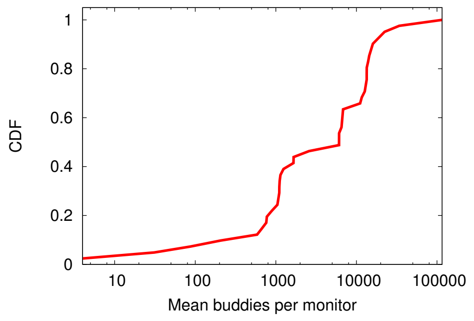


Fig. 18: **CDF of mean number of buddies per monitor**

TABLE IV: **Threshold parameters used in training.**

	Definition	Section	Value
Δ	maximum interval of "simultaneous" updates	IV-E1	40s
α	minimum # of buddy candidates per skewer	IV-E1	50
β	minimum # of ASes of buddy candidates per skewer	IV-E1	30
λ	minimum # of shared hops in path comparison	IV-E1	3
ω	minimum # of buddies per skewer	IV-E2	30
ψ	minimum # of ASes of a prefix's buddies	IV-E2	10
ξ	minimum # of skewers in training	IV-E2	10

number, plus the prefix of the update is 4 bytes. Assuming $|u_i|$ is 4.3 according to RIPE [41], and using the maximum number of updates that a monitor receives in Δ (40) seconds, i.e., 182,440 updates in 2012 from our measurement, the memory requirement will be only 4.05 MB.

Once a monitor discovers the set of buddy candidates corresponding to an update, it stores the set to its hard disk. With $|C_{u_p}|$ buddy candidates per update (Algorithm IV.1), where each candidate a 4-byte prefix, and $|S|$ updates during training for every monitored prefix (Algorithm IV.2), the total storage cost per prefix is thus $4*|C_{u_p}|*|S|$ bytes. In fact, the monitor can store all buddy candidates related to a given monitored prefix in the same file, with each file of $4*|C_{u_p}|*|S|$ bytes. Suppose we are monitoring all N prefixes on the Internet, the total hard disk storage would then be $4*|C_{u_p}|*|S|*N$ bytes. Fig. 18 shows the CDF of $|C_{u_p}|$, with its maximal value at 4619. As we limit $|S|$ to be no more than 100 and we know N is 490,000 according to [42], the worst case of the total storage cost will be 4*4619*100*490,000=905.3 GB.

When the monitor needs to run the skewering algorithm for a prefix (Algorithm IV.3), it needs to read the candidate file for the prefix into memory. In addition, it needs memory space for ω buddies per skewer, as well as ψ 4-byte AS numbers, leading to a total of $4|C_{u_p}|$*$|S|$+4ω*$|S|$+4ψ bytes. With ω=30 and ψ=10 (Table IV), this would then be only 4*4619*100+4*30*100+4*10=1,859,640 bytes. Even with all N prefix on the Internet to handle, Buddyguard can sequentially handle them, keeping memory cost always low.

The memory cost during the monitoring process is also low. For every monitored prefix, when hearing an update about it, the monitor will need to check its buddies' updates. Assume it has $|B_p|$ buddies and the AS path from each update u_b is $4*|u_b|$ bytes. The memory cost per monitored prefix is then 1 byte for tracking the state of the prefix ("warning" or "normal"), $4*|B_p|$ bytes for tracking buddies, plus $(|u_b|*4+1)$*$|B_p|$ bytes for recording AS paths of buddies (1 byte for AS hop count), with a total of $|B_p|*(4*|u_b|+5)+1$ bytes. If $|B_p|$ is 200, $|u_b|$ is 4.3, this would be only 4,441 bytes per prefix, and 4,441*490,000=2.176 GB for all 490,000 monitored prefixes.

2) Low network overhead:

The network overhead in Buddyguard happens between every monitor and the controller. Recall every monitor may send a report message regarding a monitored prefix to the controller, and the controller may query all monitors to learn whether or not they have heard a new path to a prefix (Section IV-D2). Based on the format of these two types of messages, we calculate that a query message has a payload of 10 bytes and a report has a payload of 32 bytes. Assuming all messages are carried using TCP with a TCP/IP header of 40 bytes, the length of an entire query message is then 50 bytes and the length of an entire report message is 72 bytes. Now, for one suspicious update, assume the controller queries all m monitors and every monitor reports to the controller, the total networking overhead will be then $50 * m + 72 * m = 122m$. At the controller, it will then incur $50m$ bytes outbound traffic (query) and $72m$ bytes inbound traffic (report). If every second there are κ suspicious updates, it will then be $50m\kappa$ bytes of query traffic and $72m\kappa$ bytes of report traffic. In our current Buddyguard architecture, m=273. Even in an extreme case where κ is 100, the bandwidth cost will be only 1.365 MB/s for query traffic and 1.966 MB/s for report traffic.

V. System Implementation and Demonstration

In order to achieve a near real-time process speed, we upgraded our architecture and enables multi-thread processing for each monitor component. In real development schema, each monitor component of I-seismograph should run on separate machines independently, with results sent to a central result collector. In our simulated environment, we use data collectors like RouteViews and RIPE RIS as our data source, where the BGP updates from all peers were mixed by the collector. In this case, we first group the updates by the original receiving peers. We then create a monitor instance for each peer, and dispatch the corresponding updates to this monitor instance for processing. When running I-seismograph on the test cases, we usually will have a data source with 20 to 25 peers. We start a new thread for each monitor instance, and manged to finish processing the total BGP updates 6 to 7 times faster than before. After all the monitor instance threads finish, we then aggregate the results for the presenters and statistical analysis. Figure. 19 shows the work flow of the parallel stream line processing, and Figure 20 shows the demonstration of current deployed I-seismograph monitoring instance on the web.

During our development, we targeted to create high-quality deployable code for real deployment. Our system I-seismograph 2.0 and Buddyguard 2.0 both have a working version within our generic framework ready for deployment.

VI. Related Work

A. Global-level Monitoring

Monitoring the routing infrastructure, especially BGP, has largely focused on its dynamics such as instability or pathological behavior. Researchers have not only measured BGP dynamics (e.g., [15], [16]), but have also attempted to investigate their origin (e.g., [43], [44]). There are also tools such as BGPlay [45], iBGPlay [46], and LinkRank [47] to visualize BGP dynamics. However, none of these studies or tools can help quantify how much the routing infrastructure, or BGP in particular, deviates from its normal state when certain dynamics happen. In fact, because most previous BGP measurement work focuses on a specific period, they do not even offer what normal might be in a long-term sense.

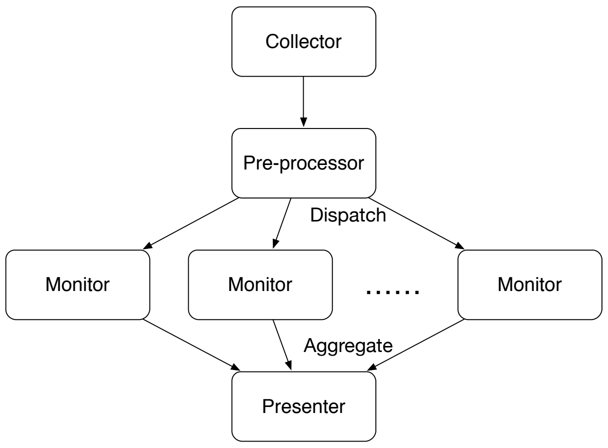


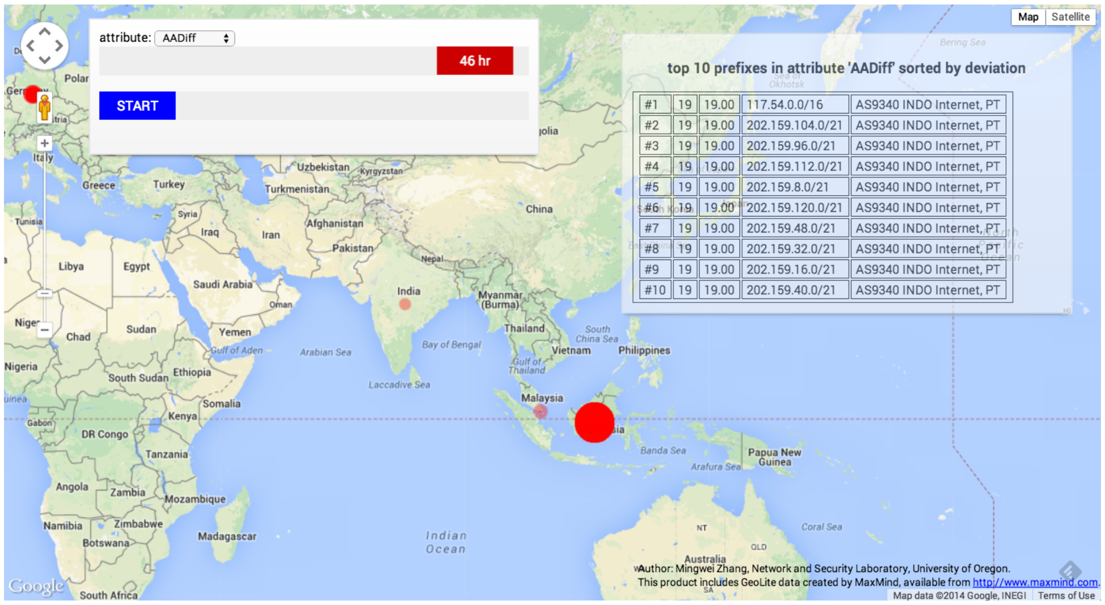
Fig. 19: **Parrallel stream line work flow.**

Many works (e.g., [48], [49], [50]) also investigated the effects of certain Internet worms, electricity outage, or undersea cable cut and other events on BGP. These works discovered that the Internet could experience a much higher level of dynamics under severe conditions. As every investigation is specific to a specific event, these studies cannot be unified to provide a uniform approach to measuring the impact on BGP.

B. Prefix-level Monitoring

Early solutions monitored prefix origin changes to detect prefix hijacking [7], [8], 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 administrators [9], [10]. Recent solutions set up monitors, probe them from a monitored prefix, and watch and analyze responses to determine if the prefix is hijacked [11]. 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 used in [12] 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

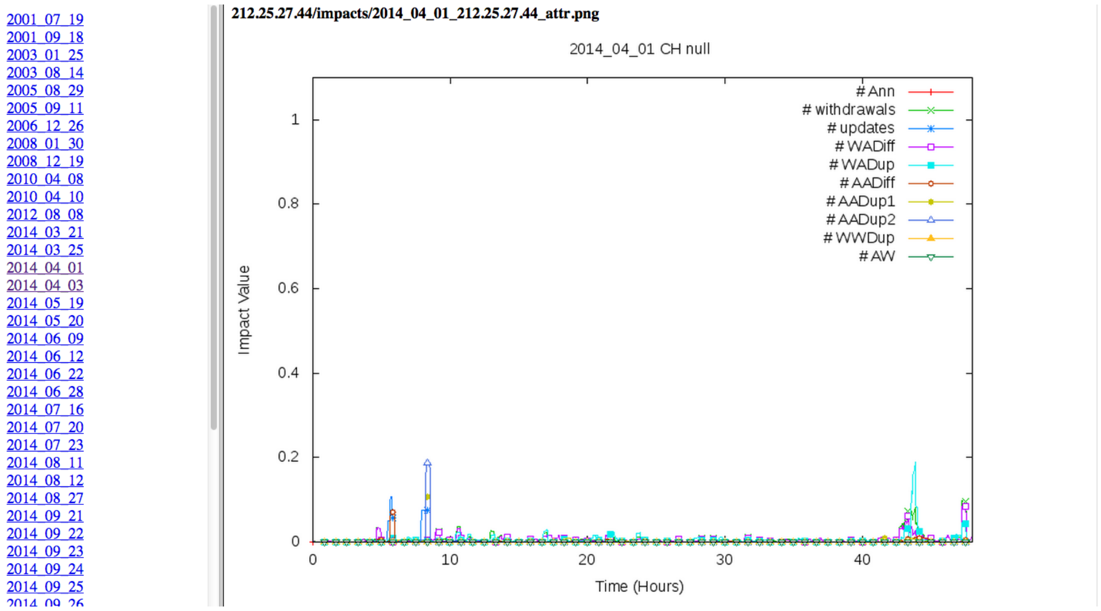


top 10 prefixes in attribute 'AADiff' sorted by deviation

#1	19	19.00	117.54.0.0/16	AS9340 INDO Internet, PT
#2	19	19.00	202.159.104.0/21	AS9340 INDO Internet, PT
#3	19	19.00	202.159.96.0/21	AS9340 INDO Internet, PT
#4	19	19.00	202.159.112.0/21	AS9340 INDO Internet, PT
#5	19	19.00	202.159.8.0/21	AS9340 INDO Internet, PT
#6	19	19.00	202.159.120.0/21	AS9340 INDO Internet, PT
#7	19	19.00	202.159.48.0/21	AS9340 INDO Internet, PT
#8	19	19.00	202.159.32.0/21	AS9340 INDO Internet, PT
#9	19	19.00	202.159.16.0/21	AS9340 INDO Internet, PT
#10	19	19.00	202.159.40.0/21	AS9340 INDO Internet, PT

Author: Mingwei Zhang, Network and Security Laboratory, University of Oregon.
This product includes GeoLite data created by MaxMind, available from http://www.maxmind.com.

(a) An map visualization for Indosat 2014 event. The event started at 4/2/2014 18:26 UTC.



212.25.27.44/impacts/2014_04_01_212.25.27.44_attr.png

2001_07_19
2001_09_18
2003_01_25
2003_08_14
2005_08_29
2005_09_11
2006_12_26
2008_01_30
2008_12_19
2010_04_08
2010_04_10
2012_08_08
2014_03_21
2014_03_25
2014_04_01
2014_04_03
2014_05_19
2014_05_20
2014_06_09
2014_06_12
2014_06_22
2014_06_28
2014_07_16
2014_07_20
2014_07_23
2014_08_11
2014_08_12
2014_08_27
2014_09_21
2014_09_22
2014_09_23
2014_09_24
2014_09_25
2014_09_26

(b) I-seismograph events impact results, organized by dates of individual events.

Fig. 20: **Running system demonstration of I-seismograph.**

the reference point, and hijack these prefixes and the monitored prefix simultaneously, 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.

Lastly, while many types of prefix anomalies could happen to an IP prefix, these hijack detection approaches cannot easily translate to monitoring other prefix routing anomalies. Due to the scale and complexity of Internet routing, the cause of various routing anomalies is often 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, a prefix monitoring system should be extensible enough to cover various known or unknown prefix anomalies.

VII. Conclusion

In this report, we introduced a generic BGP monitoring framework that can incorporate both monitoring systems and provide extensibility for any types of BGP monitoring tasks. We design and implemented the components of the framework and created a generic workflow for BGP monitoring. Our framework aims to provide a general framework that users can design and implement new components independently, and be able to "plug-and-play" the components to perform any monitoring tasks. Comparing to the existing projects like

BGPMon [51] or Cyclops [52], our system provides monitoring functionalities from different granularities, also provides more types of visualization techniques. Based on that, we introduced our work on the improvements for the Internet-level monitoring system "I-seismograph", and prefix-level monitoring system "Buddyguard". For I-seismograph, we designed and implemented new root-cause analysis functions and new approaches for result visualization. We also implemented I-seismograph as one monitor component of the framework. For Buddyguard, we introduced our newly designed algorithms for training and monitoring, new architecture, new functionalities. For both projects, we showed the evaluation results for the most recent real-world anomalous events.

In the future, we plan to push our system to industrial deployment, and provide help to the community. Users of our system look at the Internet with low overhead and requirement, and get a better understanding of the Internet routing's running status. More advanced users such as network operators can also use our system as an approach to conduct root-cause analysis on any disruptive events on the Internet.

REFERENCES

[1] Y. Rekhter and T. Li, "A border gateway protocol 4 (BGP-4)," 1995.

[2] J. Li, T. Ehrenkranz, and P. Elliott, "Buddyguard: A buddy system for fast and reliable detection of ip prefix anomalies," in *Network Protocols (ICNP), 2012 20th IEEE International Conference on*. IEEE, 2012, pp. 1–10.

[3] N. W. Group *et al.*, "BGPSEC protocol specification," 2011.

[4] R. Lychev, S. Goldberg, and M. Schapira, "BGP security in partial deployment: is the juice worth the squeeze?" in *ACM SIGCOMM*, 2013, pp. 171–182.

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

[6] RIPE NCC, "RIPE routing information service raw data," http://data.ris.ripe.net/.

[7] 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," 2003.

[8] C. Krügel, D. Mutz, W. K. Robertson, and F. Valeur, "Topology-based detection of anomalous BGP messages," pp. 17–35, 2003.

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

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

[11] 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.

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

[13] X. Hu and Z. M. Mao, "Accurate real-time identification of IP prefix hijacking," in *Proc. 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] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," in *ACM SIGCOMM*, 1997, pp. 115–126.

[16] 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.

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

[18] C. Labovitz, "Multithreaded routing toolkit," Merit Technical Report to the National Science Foundation, Tech. Rep., 1996.

[19] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S. Wu, and L. Zhang, "Observation and analysis of BGP behavior under stress," November 2002.

[20] J. Li and S. Brooks, "I-seismograph: Observing and measuring internet earthquakes," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 2624–2632.

[21] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," vol. 6, no. 5, pp. 515–528, 1998.

[22] A. Toonk, "Chinese ISP hijacks the internet," http://bgpmon.net/blog/?p=282, April 2010.

[23] ——, "A BGP leak made in Canada," http://www.bgpmon.net/a-bgp-leak-made-in-canada/, 2012.

[24] E. Zmijewski, "Indonesia Hijacks the World," http://research.dyn.com/2014/04/indonesia-hijacks-world/, 2014.

[25] L. Franceschi-Bicchierai, "Internet blackout sweeps syria, again," http://mashable.com/2014/03/20/syria-goes-almost-completely-offline-again/, 2014.

[26] A. Toonk, "What caused today's Internet hiccup," http://www.bgpmon.net/what-caused-todays-internet-hiccup/, 2014.

[27] Y. Shen, "Global Internet routing table reaches 512k milestone," http://blogs.cisco.com/sp/global-internet-routing-table-reaches-512k-milestone/, 2014.

[28] D. Reisinger, "Time Warner Cable suffers massive outage nationwide," http://www.cnet.com/news/time-warner-cable-suffers-massive-outage-nationwide/, 2014.

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

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

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

[32] J. Cowie, "The new threat: Targeted Internet traffic misdirection," http://www.renesys.com/2013/11/mitm-internet-hijacking/, 2013.

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

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

[35] S. Goldberg, M. Schapira, P. Hummon, and J. Rexford, "How secure are secure interdomain routing protocols?" in *ACM SIGCOM*, 2010.

[36] RIPE NCC, "RIPE routing information service," http://www.ris.ripe.net/.

[37] "PCCW Limited," http://www.pccw.com.

[38] "Pakistan Telecom," http://www.ptcl.com.pk.

[39] K. Zetter, "Someone's been siphoning data through a huge security hole in the Internet," http://www.wired.com/threatlevel/2013/12/bgp-hijacking-belarus-iceland/, 2013.

[40] T. Paseka, "Why Google went offline today and a bit about how the Internet works," http://blog.cloudflare.com/why-google-went-offline-today-and-a-bit-about, 2012.

[41] M. Kühne, "Update on AS path lengths over time," https://labs.ripe.net/Members/mirjam/update-on-as-path-lengths-over-time, 2012.

[42] T. Bates, P. Smith, and G. Huston, "CIDR report," http://www.cidr-report.org/as2.0/, 2014.

[43] D. Chang, R. Govindan, and J. Heidemann, "The temporal and topological characteristics of BGP path changes," in *Proceedings of the International Conference on Network Protocols*, November 2003, pp. 190–199.

[44] A. Feldmann, O. Maennel, Z. Mao, A. Berger, and B. Maggs, "Locating Internet routing instabilities," in *ACM SIGCOMM*, August 2004.

[45] L. Colitti, G. Battista, I. Marinis, F. Mariani, M. Pizzonia, and M. Patrignani, "BGPlay," http://www.ris.ripe.net/bgplay.

[46] "iBGPlay," http://www.ibgplay.org.

[47] M. Lad, L. Zhang, and D. Massey, "Link-rank: A graphical tool for capturing bgp routing dynamics," in *IEEE/IFIP NOMS*, 2004.

[48] J. Cowie, A. Ogielski, B. Premore, and Y. Yuan, "Internet worms and global routing instabilities," in *Proc. of SPIE International symposium on Convergence of IT and Communication*, July 2002.

[49] J. Cowie, A. Ogielski, B. Premore, E. Smith, and T. Underwood, "Impact of the 2003 blackouts on Internet communications," http://www.renesys.com/news/2003-11-21/Renesys_BlackoutReport.pdf, November 2003.

[50] S. LaPerrire, "Taiwan earthquake fiber cuts: a service provider view," in *NANOG 39*, February 2007.

[51] "BGPMon", "BGPMon," http://bgpmon.net, 2012.

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